LALE Documentation

Release 0.8.0-dev

IBM AI Research

Feb 21, 2024

CONTENTS

These pages show API documentation for Lale, which is auto-generated from docstrings, many of which are themselves auto-generated from schemas. For an introductory overview, installation instructions, examples, talks, code, a paper, etc., please check the Lale github page: https://github.com/IBM/lale .

CHAPTER

ONE

CORE MODULES

- lale.operators Classes for operators including pipelines.
- lale.datasets Schema-augmented datasets.
- lale.grammar Pipeline topology search syntax.
- lale.json_operator Includes a from_json() function.
- lale.schemas Python API for writing JSON schemas.

CHAPTER

TWO

OPERATOR LIBRARIES

- lale.lib.sklearn based on scikit-learn
- lale.lib.lale containing Lale-specific operators
- lale.lib.aif360 based on AI Fairness 360
- lale.lib.autogen auto-generated based on scikit-learn
- lale.lib.category_encoders based on category_encoders
- lale.lib.imblearn based on imbalanced-learn
- lale.lib.lightgbm based on LightGBM
- lale.lib.rasl relational algebra in scikit-learn
- lale.lib.snapml based on Snap ML
- lale.lib.xgboost based on XGBoost

CHAPTER

THREE

INDICES AND TABLES

- genindex
- search

3.1 Frequently Asked Questions

- Can I work with other modalities besides tables?
  - Besides tables, we have successfully used Lale for text, images, and time-series. In fact, Lale even works for multi-modal data, using the `&` combinator to specify different preprocessing paths per modality.
- Can I work with other tasks besides classification?
  - Besides classification, we mostly use Lale for regression. That said, you can define your own scoring metrics for evaluation, and pass them to automation tools to guide their search. The following notebook includes an example `disparate_impact_scorer`: https://github.com/IBM/lale/blob/master/examples/demo_aif360.ipynb
- I get an error when I instantiate an operator imported from Lale. What's wrong?
  - Lale raises errors on invalid hyperparameter values or combinations. This ensures that the operators are used correctly. So don't be surprised if you get any errors when you initialize Lale operators with some hyperparameter values. Chances are that those hyperpameters or combinations of hyperparameters are invalid. If not, please contact us.
- The algorithm I want to use is not present in Lale. Can I still use it?
  - Some of the features of Lale can be used if the algorithm implementation follows the scikit-learn conventions of fit/predict or fit/transform. You can turn any such operator into a Lale operator using `lale_op = lale.operators.make_operator(non_lale_op)`. If you want to get full Lale support for your own operator, we have a separate guide for how to do that: https://github.com/IBM/lale/blob/master/examples/docs_new_operators.ipynb
- Can I use Lale for deep learning?
  - There are multiple facets to this question. The Lale library already includes a few DL operators such as a BERT transformer, a ResNet classifier, and an MLP classifier. Lale can perform joint algorithm selection and hyperparameter optimization over pipelines involving these operators. Furthermore, users can wrap additional DL operators as described. On the other hand, Lale does not currently support full-fledged neural architecture search (NAS). It can only perform architecture search when that is exposed through hyperparameters, such as `hidden_layer_sizes` for MLP.
- How does the search space generation work?

  - Lale includes a search space generator that takes in a planned pipeline and the schemas of the operators in that pipeline, and returns a search space for your auto-ML tool of choice. Our arXiv paper describes how that works in detail: https://arxiv.org/abs/1906.03957

- Does Lale optimize for computational performance?

  - While Lale focuses mostly on types and automation, we have also done a little bit of work on computational performance. However, it has not been a major focus. If you encounter pain-points, please reach out to us.

- What is the relationship between Lale and IBM products?

  - Lale is free and open-source and does not depend on any commercial products. It is available under the Apache license and only requires the other open-source packages listed in setup.py. Lale is used by IBM's AutoAI SDK. The AutoAI SDK provides API access to various services on IBM cloud, including advanced pipeline search optimizers, cloud-hosted notebooks in Watson Studio, storage for datasets in Cloud Object Storage, storage for historical pipelines, deploying trained pipelines as a scoring service in Watson Machine Learning, etc. Lale does not require the AutoAI SDK to run, you can use it without the AutoAI SDK.

3.2 Installation

3.2.1 Installing from PyPI

Lale is easy to install. Assuming you already have a Python 3.7+ environment, all you need is the following:

```
pip install lale
```

This will install the **Lale Core** setup target, which includes many operators, pipelines, and search space generation targeting hyperopt and scikit-learn's GridSearchCV. It has a smaller set of dependencies than the **Lale Full** setup target, which also includes search space generation for SMAC, support for loading OpenML datasets in ARFF format, and some deep-learning operators. You can install it as follows:

```
pip install "lale[full]"
```

Now you should be ready to start using Lale, for instance, in a Jupyter notebook.

3.2.2 Installing from Source

As an alternative to installing Lale directly from the online github repository, you can also first clone the repository and then install Lale from your local clone. For the **Lale Core** setup target:

```
git clone https://github.com/IBM/lale.git
cd lale
pip install .
```

For the **Lale Full** and **Lale Test** setup targets:

```
pip install ".[full,test]"
```

Now, you are ready to run some tests. For a quick check, do the following in the `lale` directory:

```
export PYTHONPATH=`pwd`
python -m unittest test.test_core_classifiers.TestLogisticRegression
```

The output should look like:

```
Ran 20 tests in 105.201s
OK
```

3.2.3 Setting up the Environment

For the full functionality of Lale, you will need a Python 3.7+ environment, as well as g++, graphviz, make, and swig. You can use Lale on Linux, Windows 10, or Mac OS X. Depending on your operating system, you can skip ahead to the appropriate section below.

On Windows 10

First, you should enable the Windows Subsystem for Linux (WSL). At this point, you can continue with the instructions in section *On Ubuntu Linux*.

On Ubuntu Linux

Start by making sure your Ubuntu installation is up-to-date and check the version. In a command shell, type:

```
sudo apt update
sudo apt upgrade
lsb_release -a
```

This should output something like "Description: Ubuntu 16.04.4 LTS".

Also, make sure you have g++, make, graphviz, and swig installed. Otherwise, you can install them:

```
sudo apt install g++
sudo apt install graphviz
sudo apt install make
sudo apt install swig
```

Next, set up a Python virtual environment with Python 3.7.

```
sudo add-apt-repository ppa:deadsnakes/ppa
sudo apt-get install python3.7
sudo apt-get install python3-virtualenv
sudo apt-get install python3.7-distutils
virtualenv -p /usr/bin/python3.7 ~/python3.7venv
source ~/python3.7venv/bin/activate
```

At this point, you can continue with the Lale *Installation* instructions at the top of this file.

On Mac OS X

Assuming you already have a Python 3.7+ virtual environment, you will need to install swig using brew before you can install Lale.

If you encounter any issues in installing SMAC:

MacOS 10.14

```
open /Library/Developer/CommandLineTools/Packages/macOS_SDK_headers_for_macOS_10.14.pkg
```

Then

```
export CPATH=/Library/Developer/CommandLineTools/usr/include/c++/v1
```

MacOS 10.15 Catalina:

```
CFLAGS=-stdlib=libc++  pip install smac
```

3.3 Papers

"Pipeline Combinators for Gradual AutoML". Guillaume Baudart, Martin Hirzel, Kiran Kate, Parikshit Ram, Avraham Shinnar, and Jason Tsay. Conference on Neural Information Processing Systems (NeurIPS), pages 19705-19718, December 2021. https://proceedings.neurips.cc/paper/2021/file/a3b36cb25e2e0b93b5f334ffb4e4064e-Paper.pdf

This is the preferred citation for the Lale project.

```
@InProceedings{baudart_et_al_2021,
  title = "Pipeline Combinators for Gradual {AutoML}",
  author = "Baudart, Guillaume and Hirzel, Martin and Kate, Kiran and Ram, Parikshit and␣
↪Shinnar, Avraham and Tsay, Jason",
  booktitle = "Advances in Neural Information Processing Systems (NeurIPS)",
  year = 2021,
  month = dec,
  pages = "19705--19718",
  url = "https://proceedings.neurips.cc/paper/2021/file/a3b36cb25e2e0b93b5f334ffb4e4064e-
↪Paper.pdf" }
```

"Searching for Fairer Machine Learning Ensembles, Michael Feffer, Martin Hirzel, Samuel C. Hoffman, Kiran Kate, Parikshit Ram, and Avraham Shinnar. Conference on Automated Machine Learning (AutoML), September 2023.

```
@InProceedings{feffer_et_al_2023,
  title = "Searching for Fairer Machine Learning Ensembles",
  author = "Feffer, Michael and Hirzel, Martin and Hoffman, Samuel C. and Kate, Kiran␣
↪and Ram, Parikshit and Shinnar, Avraham",
  booktitle = "Conference on Automated Machine Learning (AutoML)",
  year = 2023,
  month = sep }
```

"A Suite of Fairness Datasets for Tabular Classification", Martin Hirzel and Michael Feffer. arXiv:2308.00133 [cs.LG], July 2023. https://arxiv.org/abs/2308.00133

```
@Misc{hirzel_feffer_2023,
  title = "A Suite of Fairness Datasets for Tabular Classification",
  author = "Hirzel, Martin and Feffer, Michael",
  year = 2023,
  month = jul,
  url = "https://arxiv.org/abs/2308.00133" }
```

"AI for Low-Code for AI", Nikitha Rao, Jason Tsay, Kiran Kate, Vincent J. Hellendoorn, and Martin Hirzel. arXiv:2305.20015 [cs.SE], May 2023. https://arxiv.org/abs/2305.20015

```
@Misc{rao_et_al_2023,
  title = "{AI} for Low-Code for {AI}",
  author = "Rao, Nikitha and Tsay, Jason and Kate, Kiran and Hellendoorn, Vincent J. and␣
↪Hirzel, Martin",
  year = 2023,
  month = may,
  url = "https://arxiv.org/abs/2305.20015" }
```

"Gradual AutoML using Lale". Kiran Kate, Martin Hirzel, Parikshit Ram, Avraham Shinnar, and Jason Tsay. Tutorial at Conference on Knowledge Discovery and Data Mining (KDD-Tutorial), August 2022. https://doi.org/10.1145/3534678.3542630

```
@InProceedings{kate_et_al_2022,
  author = "Kate, Kiran and Hirzel, Martin and Ram, Parikshit and Shinnar, Avraham and␣
↪Tsay, Jason",
  title = "Gradual {AutoML} using {Lale}",
  booktitle = "Tutorial at the Conference on Knowledge Discovery and Data Mining (KDD-
↪Tutorial)",
  year = 2022,
  month = aug,
  pages = "4794--4795",
  url = "https://doi.org/10.1145/3534678.3542630" }
```

"An Empirical Study of Modular Bias Mitigators and Ensembles". Michael Feffer, Martin Hirzel, Samuel C. Hoffman, Kiran Kate, Parikshit Ram, and Avraham Shinnar. Workshop on Benchmarking Data for Data-Centric AI (DataPerf@ICML), July 2022. http://hirzels.com/martin/papers/dataperf22-fair-ensembles.pdf

```
@InProceedings{feffer_et_al_2022,
  title = "An Empirical Study of Modular Bias Mitigators and Ensembles",
  author = "Feffer, Michael and Hirzel, Martin and Hoffman, Samuel C. and Kate, Kiran␣
↪and Ram, Parikshit and Shinnar, Avraham",
  booktitle = "Workshop on Benchmarking Data for Data-Centric AI (DataPerf@ICML)",
  year = 2022,
  month = jul,
  url = "http://hirzels.com/martin/papers/dataperf22-fair-ensembles.pdf" }
```

"The Raise of Machine Learning Hyperparameter Constraints in Python Code". Ingkarat Rak-amnouykit, Ana Milanova, Guillaume Baudart, Martin Hirzel, and Julian Dolby. International Symposium on Software Testing and Analysis (ISSTA), pages 580-592, July 2022. https://doi.org/10.1145/3533767.3534400

Winner of a Distinguished Paper Award at ISSTA 2022.

```
@InProceedings{rakamnouykit_et_al_2022,
  title = "The Raise of Machine Learning Hyperparameter Constraints in {Python} Code",
```

(continues on next page)

(continued from previous page)

```
  author = "Rak-amnouykit, Ingkarat and Milanova, Ana and Baudart, Guillaume and Hirzel,␣
↪Martin and Dolby, Julian",
  booktitle = "International Symposium on Software Testing and Analysis (ISSTA)",
  year = 2022,
  pages = "580--592",
  month = jul,
  url = "https://doi.org/10.1145/3533767.3534400" }
```

"Automatically Debugging AutoML Pipelines Using Maro: ML Automated Remediation Oracle". Julian Dolby, Jason Tsay, and Martin Hirzel. Symposium on Machine Programming (MAPS), pages 60-69, June 2022.

```
@InProceedings{dolby_tsay_hirzel_2022,
  title = "Automatically Debugging {AutoML} Pipelines Using {Maro}: {ML} Automated␣
↪Remediation Oracle",
  author = "Dolby, Julian and Tsay, Jason and Hirzel, Martin",
  booktitle = "Symposium on Machine Programming (MAPS)",
  year = 2022,
  month = jun,
  pages = "60--69",
  url = "https://dl.acm.org/doi/10.1145/3520312.3534868" }
```

"RASL: Relational Algebra in Scikit-Learn Pipelines". Chirag Sahni, Kiran Kate, Avraham Shinnar, Hoang Thanh Lam, and Martin Hirzel. Workshop on Databases and AI (DBAI@NeurIPS), December 2021. https://openreview.net/forum?id=u9ct1gjoDcn

```
@InProceedings{sahni_et_al_2021,
  title = "{RASL}: Relational Algebra in Scikit-Learn Pipelines",
  author = "Sahni, Chirag and Kate, Kiran and Shinnar, Avraham and Lam, Hoang Thanh and␣
↪Hirzel, Martin",
  booktitle = "Workshop on Databases and AI (DBAI@NeurIPS)",
  year = 2021,
  month = dec,
  url = "https://openreview.net/forum?id=u9ct1gjoDcn" }
```

"Finding Data Compatibility Bugs with JSON Subschema Checking". Andrew Habib, Avraham Shinnar, Martin Hirzel, and Michael Pradel. International Symposium on Software Testing and Analysis (ISSTA), pages 620-632, July 2021. https://doi.org/10.1145/3460319.3464796

Winner of a Distinguished Artifact Award at ISSTA 2021.

```
@InProceedings{habib_et_al_2021,
  title = "Finding Data Compatibility Bugs with {JSON} Subschema Checking",
  author = "Habib, Andrew and Shinnar, Avraham and Hirzel, Martin and Pradel, Michael",
  booktitle = "International Symposium on Software Testing and Analysis (ISSTA)",
  year = 2021,
  pages = "620--632",
  url = "https://doi.org/10.1145/3460319.3464796" }
```

"Engineering Fair Machine Learning Pipelines". Martin Hirzel, Kiran Kate, and Parikshit Ram. ICLR Workshop on Responsible AI (RAI@ICLR), May 2021. http://hirzels.com/martin/papers/rai21-fairness.pdf

```
@InProceedings{hirzel_kate_ram_2021,
  title = "Engineering Fair Machine Learning Pipelines",
```

(continues on next page)

(continued from previous page)

```
  author = "Hirzel, Martin and Kate, Kiran and Ram, Parikshit",
  booktitle = "ICLR Workshop on Responsible AI (RAI@ICLR)",
  year = 2021,
  month = may,
  url = "http://hirzels.com/martin/papers/rai21-fairness.pdf" }
```

"Extracting Hyperparameter Constraints from Code". Ingkarat Rak-amnouykit, Ana Milanova, Guillaume Baudart, Martin Hirzel, and Julian Dolby. ICLR Workshop on Security and Safety in Machine Learning Systems (SecML@ICLR), May 2021. https://aisecure-workshop.github.io/aml-iclr2021/papers/18.pdf

```
@InProceedings{rakamnouykit_et_al_2021-secml,
  title = "Extracting Hyperparameter Constraints from Code",
  author = "Rak-amnouykit, Ingkarat and Milanova, Ana and Baudart, Guillaume and Hirzel,␣
↪Martin and Dolby, Julian",
  booktitle = "ICLR Workshop on Security and Safety in Machine Learning Systems␣
↪(SecML@ICLR)",
  year = 2021,
  month = may,
  url = "https://aisecure-workshop.github.io/aml-iclr2021/papers/18.pdf" }
```

"Lale: Consistent Automated Machine Learning". Guillaume Baudart, Martin Hirzel, Kiran Kate, Parikshit Ram, and Avraham Shinnar. KDD Workshop on Automation in Machine Learning (AutoML@KDD), August 2020. https://arxiv.org/abs/2007.01977

```
@InProceedings{baudart_et_al_2020-automl_kdd,
  title = "Lale: Consistent Automated Machine Learning",
  author = "Baudart, Guillaume and Hirzel, Martin and Kate, Kiran and Ram, Parikshit and␣
↪Shinnar, Avraham",
  booktitle = "KDD Workshop on Automation in Machine Learning (AutoML@KDD)",
  year = 2020,
  month = aug,
  url = "https://arxiv.org/abs/2007.01977" }
```

"Mining Documentation to Extract Hyperparameter Schemas". Guillaume Baudart, Peter Kirchner, Martin Hirzel, and Kiran Kate. ICML Workshop on Automated Machine Learning (AutoML@ICML), July 2020. https://arxiv.org/abs/2006.16984

```
@InProceedings{baudart_et_al_2020_automl_icml,
  title = "Mining Documentation to Extract Hyperparameter Schemas",
  author = "Baudart, Guillaume and Kirchner, Peter and Hirzel, Martin and Kate, Kiran",
  booktitle = "ICML Workshop on Automated Machine Learning (AutoML@ICML)",
  month = jul,
  year = 2020,
  url = "https://arxiv.org/abs/2006.16984" }
```

"A Semi-supervised Deep Learning Algorithm for Abnormal EEG Identification". Subhrajit Roy, Kiran Kate, and Martin Hirzel. Machine Learning for Health Workshop at NeurIPS (ML4H), December 2019. https://arxiv.org/abs/1903.07822v2

```
@InProceedings{roy_kate_hirzel_2019,
  title = "A Semi-supervised Deep Learning Algorithm for Abnormal {EEG} Identification",
  author = "Roy, Subhrajit and Kate, Kiran and Hirzel, Martin",
  booktitle = "Machine Learning for Health Workshop at NeurIPS (ML4H)",
```

(continues on next page)

(continued from previous page)

```
  month = dec,
  year = 2019,
  url = "https://arxiv.org/abs/1903.07822v2" }
```

"Type Safety with JSON Subschema". Andrew Habib, Avraham Shinnar, Martin Hirzel, and Michael Pradel. arXiv:1911.12651 [cs.PL], November 2019. https://arxiv.org/abs/1911.12651

```
@Article{habib_et_al_2019,
  title = "Type Safety with {JSON} Subschema",
  author = "Habib, Andrew and Shinnar, Avraham and Hirzel, Martin and Pradel, Michael",
  journal = "CoRR",
  volume = "abs/1911.12651",
  year = 2019,
  month = nov,
  url = "https://arxiv.org/abs/1911.12651" }
```

"Type-Driven Automated Learning with Lale". Martin Hirzel, Kiran Kate, Avraham Shinnar, Subhrajit Roy, and Parikshit Ram. arXiv:1906.03957 [cs.PL], May 2019. https://arxiv.org/abs/1906.03957

```
@Article{hirzel_et_al_2019,
  author = "Hirzel, Martin and Kate, Kiran and Shinnar, Avraham and Roy, Subhrajit and␣
↪Ram, Parikshit",
  title = "Type-Driven Automated Learning with {Lale}",
  journal = "CoRR",
  volume = "abs/1906.03957",
  year = 2019,
  month = may,
  url = "https://arxiv.org/abs/1906.03957" }
```

3.4 lale

3.4.1 lale package

Subpackages

lale.datasets package

Subpackages

lale.datasets.multitable package

Submodules

lale.datasets.multitable.fetch_datasets module

lale.datasets.multitable.fetch_datasets.**fetch_creditg_multitable_dataset**(*datatype: Literal['pandas', 'spark'] = 'pandas'*)

Fetches credit-g dataset from OpenML, but in a multi-table format. It transforms the [credit-g](https://www.openml.org/d/31) dataset from OpenML to a multi-table format. We split the dataset into 3 tables: *loan_application*, *bank_account_info* and *existing_credits_info*. The table *loan_application* serves as our primary table, and we treat the other two tables as providing additional information related to the applicant's bank account and existing credits. As one can see, this is very close to a real life scenario where information is present in multiple tables in normalized forms. We created a primary key column *id* as a proxy to the loan applicant's identity number.

Parameters

`datatype` (*string, optional, default 'pandas'*) – If 'pandas', Returns a list of singleton dictionaries (each element of the list is one table from the dataset) after reading the downloaded CSV files. The key of each dictionary is the name of the table and the value contains a pandas dataframe consisting of the data.

Returns

dataframes_list

Return type

list of singleton dictionary of pandas dataframes

`lale.datasets.multitable.fetch_datasets.`**`fetch_go_sales_dataset`**(*datatype: Literal['pandas', 'spark'] = 'pandas'*)

Fetches the Go_Sales dataset from IBM's Watson's ML samples. It contains information about daily sales, methods, retailers and products of a company in form of 5 CSV files. This method downloads and stores these 5 CSV files under the 'lale/lale/datasets/multitable/go_sales_data' directory. It creates this directory by itself if it does not exists.

Dataset URL: https://github.com/IBM/watson-machine-learning-samples/raw/master/cloud/data/go_sales/

Parameters

`datatype` (*string, optional, default 'pandas'*) – If 'pandas', Returns a list of singleton dictionaries (each element of the list is one table from the dataset) after reading the downloaded CSV files. The key of each dictionary is the name of the table and the value contains a pandas dataframe consisting of the data.

If 'spark', Returns a list of singleton dictionaries (each element of the list is one table from the dataset) after reading the downloaded CSV files. The key of each dictionary is the name of the table and the value contains a spark dataframe consisting of the data extended with an index column.

Else, Throws an error as it does not support any other return type.

Returns

go_sales_list

Return type

list of singleton dictionary of pandas / spark dataframes

`lale.datasets.multitable.fetch_datasets.`**`fetch_imdb_dataset`**(*datatype: Literal['pandas', 'spark'] = 'pandas'*)

Fetches the IMDB movie dataset from Relational Dataset Repo. It contains information about directors, actors, roles and genres of multiple movies in form of 7 CSV files. This method downloads and stores these 7 CSV files under the 'lale/lale/datasets/multitable/imdb_data' directory. It creates this directory by itself if it does not exists.

Dataset URL: https://relational.fit.cvut.cz/dataset/IMDb

Parameters

`datatype` (*string, optional, default 'pandas'*) – If 'pandas', Returns a list of singleton dictionaries (each element of the list is one table from the dataset) after reading the already existing CSV files. The key of each dictionary is the name of the table and the value contains a pandas dataframe consisting of the data.

If 'spark', Returns a list of singleton dictionaries (each element of the list is one table from the dataset) after reading the downloaded CSV files. The key of each dictionary is the name of the table and the value contains a spark dataframe consisting of the data extended with an index column.

Else, Throws an error as it does not support any other return type.

Returns
imdb_list

Return type
list of singleton dictionary of pandas / spark dataframes

Raises
`jsonschema.ValueError` – dataset not found

`lale.datasets.multitable.fetch_datasets.get_data_from_csv`(*datatype: Literal['pandas', 'spark'], data_file_name*)

lale.datasets.multitable.util module

`lale.datasets.multitable.util.multitable_train_test_split`(*dataset: List[Any], main_table_name: str, label_column_name: str, test_size: float = 0.25, random_state: Optional[Union[RandomState, int]] = None*) → Tuple

Splits X and y into random train and test subsets stratified by labels and protected attributes.

Behaves similar to the train_test_split function from scikit-learn.

Parameters

- **dataset** (*list of either Pandas or Spark dataframes*) – Each dataframe in the list corresponds to an entity/table in the multi-table setting.
- **main_table_name** (*string*) – The name of the main table as the split is going to be based on the main table.
- **label_column_name** (*string*) – The name of the label column from the main table.
- **test_size** (*float or int, default=0.25*) – If float, should be between 0.0 and 1.0 and represent the proportion of the dataset to include in the test split. If int, represents the absolute number of test samples.
- **random_state** (*int, RandomState instance or None, default=None*) – Controls the shuffling applied to the data before applying the split. Pass an integer for reproducible output across multiple function calls.
  - None

    RandomState used by numpy.random
  - numpy.random.RandomState

    Use the provided random state, only affecting other users of that same random state instance.

– integer

Explicit seed.

Returns

result –

- item 0: train_X, List of datasets corresponding to the train split
- item 1: test_X, List of datasets corresponding to the test split
- item 2: train_y
- item 3: test_y

Return type
tuple

Raises
`jsonschema.ValueError` – Bad configuration. Either the table name was not found, or te provided list does not contain spark or pandas dataframes

Module contents

Functions:

- multitable_train_test_split

lale.datasets.openml package

Submodules

lale.datasets.openml.openml_datasets module

`lale.datasets.openml.openml_datasets.add_schemas`(*schema_orig*, *target_col*, *train_X*, *test_X*, *train_y*, *test_y*)

`lale.datasets.openml.openml_datasets.download_if_missing`(*dataset_name*, *verbose=False*)

`lale.datasets.openml.openml_datasets.fetch`(*dataset_name*, *task_type*, *verbose=False*, *preprocess=True*, *test_size=0.33*, *astype=None*, *seed=0*)

Module contents

lale.datasets.uci package

Submodules

lale.datasets.uci.uci_datasets module

`lale.datasets.uci.uci_datasets.download`(*dataset_id*, *zip_name*, *contents_files*) lale.datasets.uci.uci_datasets.**fetch_drugscom**()

lale.datasets.uci.uci_datasets.**fetch_household_power_consumption**()

lale.datasets.uci.uci_datasets.**tsv_to_Xy**(*file_name*, *target_col*, *schema_orig*)

Module contents

Submodules

lale.datasets.data_schemas module

class lale.datasets.data_schemas.**DataFrameWithSchema**(*data=None*, *index: Axes | None = None*, *columns: Axes | None = None*, *dtype: Dtype | None = None*, *copy: bool | None = None*)

Bases: DataFrame

class lale.datasets.data_schemas.**NDArrayWithSchema**(*shape*, *dtype=<class 'float'>*, *buffer=None*, *offset=0*, *strides=None*, *order=None*, *json_schema=None*, *table_name=None*)

Bases: ndarray

class lale.datasets.data_schemas.**SeriesWithSchema**(*data=None*, *index=None*, *dtype: Dtype | None = None*, *name=None*, *copy: bool | None = None*, *fastpath: bool | lib.NoDefault = _NoDefault.no_default*)

Bases: Series

class lale.datasets.data_schemas.**SparkDataFrameWithIndex**(*df*, *index_names=None*)

Bases: object

property index_name: Optional[str]

property index_names: List[str]

property schema: Any

toPandas(**args*, ***kwargs*) → DataFrame

lale.datasets.data_schemas.**add_schema**(*obj*, *schema=None*, *raise_on_failure=False*, *recalc=False*) → Any

lale.datasets.data_schemas.**add_schema_adjusting_n_rows**(*obj*, *schema*)

lale.datasets.data_schemas.**add_table_name**(*obj*, *name*) → Any

lale.datasets.data_schemas.**csr_matrix_to_schema**(*matrix*) → Dict[str, Any]

lale.datasets.data_schemas.**dataframe_to_schema**(*df*) → Dict[str, Any]

lale.datasets.data_schemas.**dtype_to_schema**(*typ*) → Dict[str, Any]

lale.datasets.data_schemas.**forward_metadata**(*old*, *new*)

lale.datasets.data_schemas.**get_index_name**(*obj*)

lale.datasets.data_schemas.**get_index_names**(*obj*) `lale.datasets.data_schemas.`**`get_table_name`**`(`*obj*`)`

`lale.datasets.data_schemas.`**`is_liac_arff`**`(`*obj*`)` → bool

`lale.datasets.data_schemas.`**`is_list_tensor`**`(`*obj*`)` → bool

`lale.datasets.data_schemas.`**`liac_arff_to_schema`**`(`*larff*`)` → Dict[str, Any]

`lale.datasets.data_schemas.`**`list_tensor_to_schema`**`(`*ls*`)` → Optional[Dict[str, Any]]

`lale.datasets.data_schemas.`**`list_tensor_to_shape_and_dtype`**`(`*ls*`)` → Optional[Tuple[Tuple[int, ...], Type]]

`lale.datasets.data_schemas.`**`make_optional_schema`**`(`*schema: Dict[str, Any]*`)` → Dict[str, Any]

`lale.datasets.data_schemas.`**`ndarray_to_schema`**`(`*array*`)` → Dict[str, Any]

`lale.datasets.data_schemas.`**`series_to_schema`**`(`*series*`)` → Dict[str, Any]

`lale.datasets.data_schemas.`**`shape_and_dtype_to_schema`**`(`*shape*, *dtype*`)` → Dict[str, Any]

`lale.datasets.data_schemas.`**`spark_df_to_schema`**`(`*df*`)` → Dict[str, Any]

`lale.datasets.data_schemas.`**`strip_schema`**`(`*obj*`)`

`lale.datasets.data_schemas.`**`to_schema`**`(`*obj*`)` → Dict[str, Any]

`lale.datasets.data_schemas.`**`torch_tensor_to_schema`**`(`*tensor*`)` → Dict[str, Any]

lale.datasets.movie_review module

`lale.datasets.movie_review.`**`load_movie_review`**`()`

Loads the sentiment classification from a movie reviews dataset. Read the readme from data/movie_review for more details.

lale.datasets.sklearn_to_pandas module

`lale.datasets.sklearn_to_pandas.`**`boston_housing_df`**`(`*test_size=0.2*, *random_state=42*`)`

`lale.datasets.sklearn_to_pandas.`**`california_housing_df`**`(`*test_size=0.2*, *random_state=42*`)`

`lale.datasets.sklearn_to_pandas.`**`covtype_df`**`(`*test_size=0.2*, *random_state=42*`)`

`lale.datasets.sklearn_to_pandas.`**`digits_df`**`(`*test_size=0.2*, *random_state=42*`)`

`lale.datasets.sklearn_to_pandas.`**`load_iris_df`**`(`*test_size=0.2*`)`

lale.datasets.util module

lale.datasets.util.**load_boston**(*return_X_y: Literal[True]*) → Tuple[Any, Any]
lale.datasets.util.**load_boston**(*return_X_y: Literal[False] = False*) → Bunch

lale.datasets.util.**pandas2spark**(*pandas_df*)

Module contents

lale.lib package

Subpackages

lale.lib.aif360 package

Submodules

lale.lib.aif360.adversarial_debiasing module

class lale.lib.aif360.adversarial_debiasing.**AdversarialDebiasing**(**, favorable_labels, protected_attributes, unfavorable_labels=None, redact=True, preparation=None, scope_name='adversarial_debiasing', sess=None, seed=None, adversary_loss_weight=0.1, num_epochs=50, batch_size=128, classifier_num_hidden_units=200, debias=True, verbose=0*)

Bases: `PlannedIndividualOp`

AdversarialDebiasing in-estimator fairness mitigator. Learns a classifier to maximize prediction accuracy and simultaneously reduce an adversary's ability to determine the protected attribute from the predictions (Zhang et al. 2018). This approach leads to a fair classifier as the predictions cannot carry any group discrimination information that the adversary can exploit. Implemented based on TensorFlow.

This documentation is auto-generated from JSON schemas.

Parameters

- **favorable_labels** (*array, >=1 items, not for optimizer*) – Label values which are considered favorable (i.e. "positive").
  - items : union type
    - float

      Numerical value.
    - *or* string

      Literal string value.

        * *or* boolean

          Boolean value.

        * *or* array, >=2 items, <=2 items *of* items : float

          Numeric range [a,b] from a to b inclusive.

- **protected_attributes** (*array, >=1 items, not for optimizer*) – Features for which fairness is desired.
  - items : dict
    * feature : union type

      Column name or column index.

      · string

      · *or* integer

    * reference_group : array, >=1 items

      Values or ranges that indicate being a member of the privileged group.

      · items : union type

      · string

        Literal value.

      · *or* float

        Numerical value.

      · *or* array, >=2 items, <=2 items *of* items : float

        Numeric range [a,b] from a to b inclusive.

    * monitored_group : union type, default None

      Values or ranges that indicate being a member of the unprivileged group.

      · None

        If *monitored_group* is not explicitly specified, consider any values not captured by *reference_group* as monitored.

      · *or* array, >=1 items

      · items : union type

      · string

        Literal value.

      · *or* float

        Numerical value.

      · *or* array, >=2 items, <=2 items *of* items : float

        Numeric range [a,b] from a to b inclusive.

- **unfavorable_labels** (*union type, not for optimizer, default None*) – Label values which are considered unfavorable (i.e. "negative").
  - None

If *unfavorable_labels* is not explicitly specified, consider any labels not captured by *favorable_labels* as unfavorable.

- *or* array, >=1 items
  - items : union type
    - float

      Numerical value.
    - *or* string

      Literal string value.
    - *or* boolean

      Boolean value.
    - *or* array, >=2 items, <=2 items *of* items : float

      Numeric range [a,b] from a to b inclusive.

- **redact** (*boolean, not for optimizer, default True*) – Whether to redact protected attributes before data preparation (recommended) or not.
- **preparation** (*union type, not for optimizer, default None*) – Transformer, which may be an individual operator or a sub-pipeline.
  - operator
  - *or* None

    lale.lib.lale.NoOp
- **scope_name** (*string, not for optimizer, default 'adversarial_debiasing'*) – Scope name for the tenforflow variables. A unique alpha-numeric suffix is added to this value.
- **sess** (*union type, not for optimizer, default None*) – TensorFlow session.
  - Any

    User-provided session object.
  - *or* None

    Create a session for the user.
- **seed** (*union type, not for optimizer, default None*) – Seed to make *predict* repeatable.
  - integer
  - *or* None
- **adversary_loss_weight** (*float, >=0.03125 for optimizer, <=32768 for optimizer, loguniform distribution, default 0.1*) – Hyperparameter that chooses the strength of the adversarial loss.
- **num_epochs** (*integer, >=1, >=5 for optimizer, <=500 for optimizer, loguniform distribution, default 50*) – Number of training epochs.
- **batch_size** (*integer, >=1, >=4 for optimizer, <=512 for optimizer, loguniform distribution, default 128*) – Batch size.

- **classifier_num_hidden_units** (*integer, >=1, >=16 for optimizer, <=1024 for optimizer, loguniform distribution, default 200*) – Number of hidden units in the classifier model.
- **debias** (*boolean, not for optimizer, default True*) – Learn a classifier with or without debiasing.
- **verbose** (*integer, not for optimizer, default 0*) – If zero, then no output.

`fit`(*X*, *y=None*, ***fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

- **X** (*array*) – Features; the outer array is over samples.
  - items : array
    - items : union type
      - float
      - *or* string
- **y** (*union type*) – Target class labels; the array is over samples.
  - array *of* items : float
  - *or* array *of* items : string

`predict`(*X*, ***predict_params*)

Make predictions.

Note: The predict method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (*array*) – Features; the outer array is over samples.
- items : array
  - items : union type
    - float
    - *or* string

Returns

result – Predicted class label per sample.
- array *of* items : float
- *or* array *of* items : string

Return type

union type

`predict_proba`(*X*)

Probability estimates for all classes.

Note: The predict_proba method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (*array*) – Features; the outer array is over samples.
- items : array
  - items : union type
    - float

* *or* string

Returns

result – The class probabilities of the input samples

- array *of* items : Any
- *or* array *of* items : array *of* items : Any

Return type

union type

lale.lib.aif360.bagging_orbis_classifier module

class `lale.lib.aif360.bagging_orbis_classifier.BaggingOrbisClassifier`(***, *favorable_labels*, *protected_attributes*, *unfavorable_labels=None*, *redact=True*, *preparation=None*, *estimator=None*, *n_estimators=10*, *imbalance_repair_level=0.8*, *bias_repair_level=0.8*, *combine='keep_separate'*, *sampling_strategy='mixed'*, *replacement=False*, *n_jobs=1*, *random_state=None*)

Bases: *PlannedIndividualOp*

Experimental BaggingOrbisClassifier in-estimator fairness mitigator.

This documentation is auto-generated from JSON schemas.

Work in progress and subject to change; only supports pandas DataFrame so far. Bagging ensemble classifier, where each inner classifier gets trained on a subset of the data that has been balanced with Orbis. Unlike other mitigators in *lale.lib.aif360*, this mitigator does not come from AIF360.

Parameters

- **favorable_labels** (*array, >=1 items, not for optimizer*) – Label values which are considered favorable (i.e. "positive").
  - items : union type
    * float

      Numerical value.
    * *or* string

      Literal string value.
    * *or* boolean

      Boolean value.
    * *or* array, >=2 items, <=2 items *of* items : float

      Numeric range [a,b] from a to b inclusive.
- **protected_attributes** (*array, >=1 items, not for optimizer*) – Features for which fairness is desired.
  - items : dict
    * feature : union type

Column name or column index.

· string

· *or* integer

∗ reference_group : array, >=1 items

Values or ranges that indicate being a member of the privileged group.

· items : union type

· string

Literal value.

· *or* float

Numerical value.

· *or* array, >=2 items, <=2 items *of* items : float

Numeric range [a,b] from a to b inclusive.

∗ monitored_group : union type, default None

Values or ranges that indicate being a member of the unprivileged group.

· None

If *monitored_group* is not explicitly specified, consider any values not captured by *reference_group* as monitored.

· *or* array, >=1 items

· items : union type

· string

Literal value.

· *or* float

Numerical value.

· *or* array, >=2 items, <=2 items *of* items : float

Numeric range [a,b] from a to b inclusive.

- **unfavorable_labels** (*union type, not for optimizer, default None*) – Label values which are considered unfavorable (i.e. "negative").
  - None

    If *unfavorable_labels* is not explicitly specified, consider any labels not captured by *favorable_labels* as unfavorable.
  - *or* array, >=1 items
    ∗ items : union type
      · float
        Numerical value.
      · *or* string
        Literal string value.
      · *or* boolean
        Boolean value.
      · *or* array, >=2 items, <=2 items *of* items : float
        Numeric range [a,b] from a to b inclusive.

- **redact** (*boolean, optional, not for optimizer, default True*) – Whether to redact protected attributes before data preparation (recommended) or not.
- **preparation** (*union type, optional, not for optimizer, default None*) – Transformer, which may be an individual operator or a sub-pipeline.
  - operator
  - *or* None

    NoOp
- **estimator** (*union type, optional, not for optimizer, default None*) – The nested classifier to fit on balanced subsets of the data.
  - operator
  - *or* None

    DecisionTreeClassifier
- **n_estimators** (*integer, >=10 for optimizer, <=100 for optimizer, uniform distribution, optional, default 10*) – The number of base estimators in the ensemble.
- **imbalance_repair_level** (*float, >=0.0, <=1.0, optional, default 0.8*) – How much to repair for class imbalance (0 means original imbalance, 1 means perfect balance).

  See also *constraint-1*.
- **bias_repair_level** (*float, >=0.0, <=1.0, optional, default 0.8*) – How much to repair for group bias (0 means original bias, 1 means perfect fairness).

  See also *constraint-1*.
- **combine** ('keep_separate', 'and', 'or', *or* 'error', optional, not for optimizer, default 'keep_separate') – How to handle the case when there is more than one protected attribute.
- **sampling_strategy** ('under', 'over', 'mixed', 'minimum', *or* 'maximum', optional, not for optimizer, default 'mixed') –

  How to change the intersection sizes.

  Possible choices are:
  - `'under'`: under-sample large intersections to desired repair levels;
  - `'over'`: over-sample small intersection to desired repair levels;
  - `'mixed'`: mix under- with over-sampling while keeping sizes similar to original;
  - `'minimum'`: under-sample everything to the size of the smallest intersection;
  - `'maximum'`: over-sample everything to the size of the largest intersection.

  See also *constraint-1*.
- **replacement** (*boolean, optional, not for optimizer, default False*) – Whether under-sampling is with or without replacement.
- **n_jobs** (*integer, optional, not for optimizer, default 1*) – The number of threads to open if possible.
- **random_state** (*union type, optional, not for optimizer, default None*) – Control the randomization of the algorithm.
  - None

    RandomState used by np.random
  - *or* integer

    The seed used by the random number generator
  - *or* numpy.random.RandomState

    Random number generator instance.

Notes

constraint-1 : union type
    When sampling_strategy is minimum or maximum, both repair levels must be 1.
    - sampling_strategy : negated type *of* 'minimum' *or* 'maximum'
    - *or* dict
        - imbalance_repair_level : 1
        - bias_repair_level : 1

fit(*X*, *y=None*, ***fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

- **X** (*array*) – Features; the outer array is over samples.
    - items : array
        - items : union type
            - float
            - *or* string
- **y** (*union type*) – Target class labels; the array is over samples.
    - array *of* items : float
    - *or* array *of* items : string

predict(*X*, ***predict_params*)

Make predictions.

Note: The predict method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (*array*) – Features; the outer array is over samples.
- items : array
    - items : union type
        - float
        - *or* string

Returns

result – Predicted class label per sample.
- array *of* items : float
- *or* array *of* items : string

Return type

union type

predict_proba(*X*)

Probability estimates for all classes.

Note: The predict_proba method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (*array*) – Features; the outer array is over samples.
- items : array
    - items : union type
        - float
        - *or* string

Returns

result – The class probabilities of the input samples
- array *of* items : Any
- *or* array *of* items : array *of* items : Any

Return type
union type

lale.lib.aif360.calibrated_eq_odds_postprocessing module

class `lale.lib.aif360.calibrated_eq_odds_postprocessing.`**`CalibratedEqOddsPostprocessing`**(**, favorable_labels, protected_attributes, unfavorable_labels=None, estimator, redact=True, cost_constraint='weig seed=None*)

Bases: *PlannedIndividualOp*

Calibrated equalized odds postprocessing post-estimator fairness mitigator. Optimizes over calibrated classifier score outputs to find probabilities with which to change output labels with an equalized odds objective (Pleiss et al. 2017).

This documentation is auto-generated from JSON schemas.

Parameters
- **favorable_labels** (*array, >=1 items, not for optimizer*) – Label values which are considered favorable (i.e. "positive").
  - items : union type
    - float
      Numerical value.
    - *or* string
      Literal string value.
    - *or* boolean
      Boolean value.
    - *or* array, >=2 items, <=2 items *of* items : float
      Numeric range [a,b] from a to b inclusive.
- **protected_attributes** (*array, >=1 items, not for optimizer*) – Features for which fairness is desired.
  - items : dict
    - feature : union type
      Column name or column index.
      - string
      - *or* integer
    - reference_group : array, >=1 items
      Values or ranges that indicate being a member of the privileged group.

· items : union type
· string
Literal value.
· *or* float
Numerical value.
· *or* array, >=2 items, <=2 items *of* items : float
Numeric range [a,b] from a to b inclusive.
* monitored_group : union type, default None
Values or ranges that indicate being a member of the unprivileged group.
· None
If *monitored_group* is not explicitly specified, consider any values not captured by *reference_group* as monitored.
· *or* array, >=1 items
· items : union type
· string
Literal value.
· *or* float
Numerical value.
· *or* array, >=2 items, <=2 items *of* items : float
Numeric range [a,b] from a to b inclusive.

- **unfavorable_labels** (*union type, not for optimizer, default None*) – Label values which are considered unfavorable (i.e. "negative").
  - None
    If *unfavorable_labels* is not explicitly specified, consider any labels not captured by *favorable_labels* as unfavorable.
  - *or* array, >=1 items
    * items : union type
      · float
      Numerical value.
      · *or* string
      Literal string value.
      · *or* boolean
      Boolean value.
      · *or* array, >=2 items, <=2 items *of* items : float
      Numeric range [a,b] from a to b inclusive.
- **estimator** (*operator, not for optimizer*) – Nested supervised learning operator for which to mitigate fairness.
- **redact** (*boolean, not for optimizer, default True*) – Whether to redact protected attributes before data preparation (recommended) or not.
- **cost_constraint** ('fpr', 'fnr', *or* 'weighted', default 'weighted') –
- **seed** (*union type, not for optimizer, default None*) – Seed to make *predict* repeatable.
  - integer
  - *or* None

fit(*X*, *y=None*, ***fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

- **X** (*array*) – Features; the outer array is over samples.
  - items : array

  * items : union type
    - float
    - *or* string
- **y** (*union type*) – Target class labels; the array is over samples.
  - array *of* items : float
  - *or* array *of* items : string

`predict`(*X*, ***predict_params*)

Make predictions.

Note: The predict method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (*array*) – Features; the outer array is over samples.
- items : array
  - items : union type
    * float
    * *or* string

Returns

result – Predicted class label per sample.
- array *of* items : float
- *or* array *of* items : string

Return type

union type

`predict_proba`(*X*)

Probability estimates for all classes.

Note: The predict_proba method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (*array*) – Features; the outer array is over samples.
- items : array
  - items : union type
    * float
    * *or* string

Returns

result – The class probabilities of the input samples
- array *of* items : Any
- *or* array *of* items : array *of* items : Any

Return type

union type

lale.lib.aif360.datasets module

Fetcher methods to load fairness datasets and provide fairness_info for them.

See the notebook demo_fairness_datasets for an example for using the functions, along with some tables and figures about them. There is also an arxiv paper about these datasets. Some of the fetcher methods have a *preprocess* argument that defaults to False. The notebook does not use that argument, instead demonstrating how to do any required preprocessing in the context of a Lale pipeline. Most of the datasets are from OpenML, a few are from meps.ahrq or ProPublica, and most of the datasets have been used in various papers. The Lale library does not distribute the datasets themselves, it only provides methods for downloading them.

`lale.lib.aif360.datasets.fetch_adult_df`(*preprocess: bool = False*)

Fetch the adult dataset from OpenML and add *fairness_info*. It contains information about individuals from the 1994 U.S. census. The prediction task is a binary classification on whether the income of a person exceeds 50K a year. Without preprocessing, the dataset has 48,842 rows and 14 columns. There are two protected attributes, sex and race, and the disparate impact is 0.23. The data includes both categorical and numeric columns, and has some missing values.

Parameters

preprocess (*boolean, optional, default False*) – If True, impute missing values; encode protected attributes in X as 0 or 1 to indicate privileged groups; encode labels in y as 0 or 1 to indicate favorable outcomes; and apply one-hot encoding to any remaining features in X that are categorical and not protected attributes.

Returns

result –

- item 0: pandas Dataframe
  Features X, including both protected and non-protected attributes.
- item 1: pandas Series
  Labels y.
- item 3: fairness_info
  JSON meta-data following the format understood by fairness metrics and mitigation operators in *lale.lib.aif360*.

Return type

tuple

`lale.lib.aif360.datasets.fetch_bank_df`(*preprocess: bool = False*)

Fetch the bank-marketing dataset from OpenML and add *fairness_info*.

It contains information from marketing campaigns of a Portuguise bank. The prediction task is a binary classification on whether the client will subscribe a term deposit. Without preprocessing, the dataset has 45,211 rows and 16 columns. There is one protected attribute, age, and the disparate impact of 0.84. The data includes both categorical and numeric columns, with no missing values.

Parameters

preprocess (*boolean, optional, default False*) – If True, encode protected attributes in X as 0 or 1 to indicate privileged groups; encode labels in y as 0 or 1 to indicate favorable outcomes; and apply one-hot encoding to any remaining features in X that are categorical and not protected attributes.

Returns

result –

- item 0: pandas Dataframe
  Features X, including both protected and non-protected attributes.
- item 1: pandas Series
  Labels y.
- item 3: fairness_info

JSON meta-data following the format understood by fairness metrics and mitigation operators in *lale.lib.aif360*.

Return type
tuple

`lale.lib.aif360.datasets.fetch_compas_df(preprocess: bool = False)`

Fetch the compas-two-years dataset, also known as ProPublica recidivism, from GitHub and add *fairness_info*.

It contains information about individuals with a binary classification for recidivism, indicating whether they were re-arrested within two years after the first arrest. Without preprocessing, the dataset has 6,172 rows and 51 columns. There are two protected attributes, sex and race, and the disparate impact is 0.75. The data includes numeric and categorical columns, with some missing values.

Parameters
`preprocess` (*boolean, optional, default False*) – If True, encode protected attributes in X as 0 or 1 to indicate privileged groups (1 if Female or Caucasian for the corresponding sex and race columns respectively); and apply one-hot encoding to any remaining features in X that are categorical and not protected attributes.

Returns

result –
- item 0: pandas Dataframe
  Features X, including both protected and non-protected attributes.
- item 1: pandas Series
  Labels y.
- item 3: fairness_info
  JSON meta-data following the format understood by fairness metrics and mitigation operators in *lale.lib.aif360*.

Return type
tuple

`lale.lib.aif360.datasets.fetch_compas_violent_df(preprocess: bool = False)`

Fetch the compas-two-years-violent dataset, also known as ProPublica violent recidivism, from GitHub and add *fairness_info*.

It contains information about individuals with a binary classification for violent recidivism, indicating whether they were re-arrested within two years after the first arrest. Without preprocessing, the dataset has 4,020 rows and 51 columns. There are three protected attributes, sex, race, and age, and the disparate impact is 0.85. The data includes numeric and categorical columns, with some missing values.

Parameters
`preprocess` (*boolean, optional, default False*) – If True, encode protected attributes in X as 0 or 1 to indicate privileged groups (1 if Female, Caucasian, or at least 25 for the corresponding sex, race, and age columns respectively); and apply one-hot encoding to any remaining features in X that are categorical and not protected attributes.

Returns

result –
- item 0: pandas Dataframe
  Features X, including both protected and non-protected attributes.
- item 1: pandas Series
  Labels y.
- item 3: fairness_info
  JSON meta-data following the format understood by fairness metrics and mitigation operators in *lale.lib.aif360*.

Return type
tuple

`lale.lib.aif360.datasets.fetch_creditg_df(preprocess: bool = False)`

Fetch the credit-g dataset from OpenML and add *fairness_info*.

It contains information about individuals with a binary classification into good or bad credit risks. Without preprocessing, the dataset has 1,000 rows and 20 columns. There are two protected attributs, personal_status/sex and age, and the disparate impact is 0.75. The data includes both categorical and numeric columns, with no missing values.

Parameters

preprocess (*boolean, optional, default False*) – If True, encode protected attributes in X as 0 or 1 to indicate privileged groups; encode labels in y as 0 or 1 to indicate favorable outcomes; and apply one-hot encoding to any remaining features in X that are categorical and not protected attributes.

Returns

result –

- item 0: pandas Dataframe
  Features X, including both protected and non-protected attributes.
- item 1: pandas Series
  Labels y.
- item 3: fairness_info
  JSON meta-data following the format understood by fairness metrics and mitigation operators in *lale.lib.aif360*.

Return type

tuple

`lale.lib.aif360.datasets.fetch_default_credit_df()`

Fetch the Default of Credit Card Clients Dataset from OpenML and add *fairness_info*. It is a binary classification to predict whether the customer suffers a default in the next month (1) or not (0). The dataset has 30,000 rows and 24 columns, all numeric. The protected attribute is sex and the disparate impact is 0.957.

Returns

result –

- item 0: pandas Dataframe
  Features X, including both protected and non-protected attributes.
- item 1: pandas Series
  Labels y.
- item 3: fairness_info
  JSON meta-data following the format understood by fairness metrics and mitigation operators in *lale.lib.aif360*.

Return type

tuple

`lale.lib.aif360.datasets.fetch_heart_disease_df()`

Fetch the heart-disease dataset from OpenML and add *fairness_info*. It is a binary classification to predict heart disease from the Cleveland database, with 303 rows and 13 columns, all numeric. The protected attribute is age and the disparate impact is 0.589.

Returns

result –

- item 0: pandas Dataframe
  Features X, including both protected and non-protected attributes.
- item 1: pandas Series
  Labels y.
- item 3: fairness_info
  JSON meta-data following the format understood by fairness metrics and mitigation operators in *lale.lib.aif360*.

Return type
tuple

`lale.lib.aif360.datasets.fetch_law_school_df()`

Fetch the law school dataset from OpenML and add *fairness_info*. This function returns both X and y unchanged, since the dataset was already binarized by the OpenML contributors, with the target of predicting whether the GPA is greater than 3. The protected attributes is race1 and the disparate impact is 0.704. The dataset has 20,800 rows and 11 columns (5 categorical and 6 numeric columns).

Returns

result –

- item 0: pandas Dataframe
  Features X, including both protected and non-protected attributes.
- item 1: pandas Series
  Labels y.
- item 3: fairness_info
  JSON meta-data following the format understood by fairness metrics and mitigation operators in *lale.lib.aif360*.

Return type
tuple

`lale.lib.aif360.datasets.fetch_meps_panel19_fy2015_df`(*preprocess: bool = False*)

Fetch a subset of the MEPS dataset from aif360 and add fairness info.

It contains information collected on a nationally representative sample of the civilian noninstitutionalized population of the United States, specifically reported medical expenditures and civilian demographics. This dataframe corresponds to data from panel 19 from the year 2015. Without preprocessing, the dataframe contains 16578 rows and 1825 columns. (With preprocessing the dataframe contains 15830 rows and 138 columns.) There is one protected attribute, race, and the disparate impact is 0.496 if preprocessing is not applied and 0.490 if preprocessing is applied. The data includes numeric and categorical columns, with some missing values.

Note: in order to use this dataset, be sure to follow the instructions found in the AIF360 documentation and accept the corresponding license agreement.

Parameters

preprocess (*boolean, optional, default False*) – If True, encode protected attribute in X corresponding to race as 0 or 1 to indicate privileged groups; encode labels in y as 0 or 1 to indicate faborable outcomes; rename columns that are panel or round-specific; drop columns such as ID columns that are not relevant to the task at hand; and drop rows where features are unknown.

Returns

result –

- item 0: pandas Dataframe
  Features X, including both protected and non-protected attributes.
- item 1: pandas Series
  Labels y.
- item 3: fairness_info
  JSON meta-data following the format understood by fairness metrics and mitigation operators in *lale.lib.aif360*.

Return type
tuple

`lale.lib.aif360.datasets.fetch_meps_panel20_fy2015_df`(*preprocess: bool = False*)

Fetch a subset of the MEPS dataset from aif360 and add fairness info.

It contains information collected on a nationally representative sample of the civilian noninstitutionalized population of the United States, specifically reported medical expenditures and civilian demographics. This dataframe corresponds to data from panel 20 from the year 2015. Without preprocessing, the dataframe contains 18849 rows and 1825 columns. (With preprocessing the dataframe contains 17570 rows and 138 columns.) There is one protected attribute, race, and the disparate impact is 0.493 if preprocessing is not applied and 0.488 if preprocessing is applied. The data includes numeric and categorical columns, with some missing values.

Note: in order to use this dataset, be sure to follow the instructions found in the AIF360 documentation and accept the corresponding license agreement.

Parameters

`preprocess` (*boolean, optional, default False*) – If True, encode protected attribute in X corresponding to race as 0 or 1 to indicate privileged groups; encode labels in y as 0 or 1 to indicate faborable outcomes; rename columns that are panel or round-specific; drop columns such as ID columns that are not relevant to the task at hand; and drop rows where features are unknown.

Returns

result –

- item 0: pandas Dataframe
  Features X, including both protected and non-protected attributes.
- item 1: pandas Series
  Labels y.
- item 3: fairness_info
  JSON meta-data following the format understood by fairness metrics and mitigation operators in *lale.lib.aif360*.

Return type

tuple

`lale.lib.aif360.datasets.`**`fetch_meps_panel21_fy2016_df`**(*preprocess: bool = False*)

Fetch a subset of the MEPS dataset from aif360 and add fairness info.

It contains information collected on a nationally representative sample of the civilian noninstitutionalized population of the United States, specifically reported medical expenditures and civilian demographics. This dataframe corresponds to data from panel 20 from the year 2016. Without preprocessing, the dataframe contains 17052 rows and 1936 columns. (With preprocessing the dataframe contains 15675 rows and 138 columns.) There is one protected attribute, race, and the disparate impact is 0.462 if preprocessing is not applied and 0.451 if preprocessing is applied. The data includes numeric and categorical columns, with some missing values.

Note: in order to use this dataset, be sure to follow the instructions found in the AIF360 documentation and accept the corresponding license agreement.

Parameters

`preprocess` (*boolean, optional, default False*) – If True, encode protected attribute in X corresponding to race as 0 or 1 to indicate privileged groups; encode labels in y as 0 or 1 to indicate faborable outcomes; rename columns that are panel or round-specific; drop columns such as ID columns that are not relevant to the task at hand; and drop rows where features are unknown.

Returns

result –

- item 0: pandas Dataframe
  Features X, including both protected and non-protected attributes.
- item 1: pandas Series
  Labels y.
- item 3: fairness_info
  JSON meta-data following the format understood by fairness metrics and mitigation operators in *lale.lib.aif360*.

Return type

tuple

`lale.lib.aif360.datasets.fetch_nlsy_df()`

Fetch the National Longitudinal Survey for the Youth (NLSY) (also known as "University of Michigan Health and Retirement Study (HRS)") dataset from OpenML and add *fairness_info*.

It is a binary classification to predict whether the income at a certain time exceeds a threshold, with 4,908 rows and 15 columns (comprising 6 categorical and 9 numerical columns). The protected attributes are age and gender and the disparate impact is 0.668.

Returns

result –

- item 0: pandas Dataframe
  Features X, including both protected and non-protected attributes.
- item 1: pandas Series
  Labels y.
- item 3: fairness_info
  JSON meta-data following the format understood by fairness metrics and mitigation operators in *lale.lib.aif360*.

Return type

tuple

`lale.lib.aif360.datasets.fetch_nursery_df(preprocess: bool = False)`

Fetch the nursery dataset from OpenML and add *fairness_info*.

It contains data gathered from applicants to public schools in Ljubljana, Slovenia during a competitive time period. Without preprocessing, the dataset has 12960 rows and 8 columns. There is one protected attribute, parents, and the disparate impact is 0.46. The data has categorical columns (with numeric ones if preprocessing is applied), with no missing values.

Parameters

preprocess (*boolean, optional, default False*) – If True, encode protected attributes in X as 0 or 1 to indicate privileged groups and apply one-hot encoding to any remaining features in X that are categorical and not protected attributes.

Returns

result –

- item 0: pandas Dataframe
  Features X, including both protected and non-protected attributes.
- item 1: pandas Series
  Labels y.
- item 3: fairness_info
  JSON meta-data following the format understood by fairness metrics and mitigation operators in *lale.lib.aif360*.

Return type

tuple

`lale.lib.aif360.datasets.fetch_ricci_df(preprocess: bool = False)`

Fetch the ricci_vs_destefano dataset from OpenML and add *fairness_info*.

It contains test scores for 2003 New Haven Fire Department promotion exams with a binary classification into promotion or no promotion. Without preprocessing, the dataset has 118 rows and 5 columns. There is one protected attribute, race, and the disparate impact is 0.50. The data includes both categorical and numeric columns, with no missing values.

Parameters

preprocess (*boolean, optional, default False*) – If True, encode protected attributes in X as 0 or 1 to indicate privileged groups; encode labels in y as 0 or 1 to indicate favorable outcomes; and apply one-hot encoding to any remaining features in X that are categorical and not protected attributes.

Returns

result –
- item 0: pandas Dataframe
  Features X, including both protected and non-protected attributes.
- item 1: pandas Series
  Labels y.
- item 3: fairness_info
  JSON meta-data following the format understood by fairness metrics and mitigation operators in *lale.lib.aif360*.

Return type
tuple

`lale.lib.aif360.datasets.fetch_speeddating_df(preprocess: bool = False)`

Fetch the SpeedDating dataset from OpenML and add *fairness_info*.

It contains data gathered from participants in experimental speed dating events from 2002-2004 with a binary classification into match or no match. Without preprocessing, the dataset has 8378 rows and 122 columns. There are two protected attributes, whether the other candidate has the same race and importance of having the same race, and the disparate impact is 0.85. The data includes both categorical and numeric columns, with some missing values.

Parameters

preprocess (*boolean, optional, default False*) – If True, encode protected attributes in X as 0 or 1 to indicate privileged groups; encode labels in y as 0 or 1 to indicate favorable outcomes; and apply one-hot encoding to any remaining features in X that are categorical and not protected attributes.

Returns

result –
- item 0: pandas Dataframe
  Features X, including both protected and non-protected attributes.
- item 1: pandas Series
  Labels y.
- item 3: fairness_info
  JSON meta-data following the format understood by fairness metrics and mitigation operators in *lale.lib.aif360*.

Return type
tuple

`lale.lib.aif360.datasets.fetch_student_math_df()`

Fetch the Student Performance (Math) dataset from OpenML and add *fairness_info*.

The original prediction target is a integer math grade from 1 to 20. This function returns X unchanged but with a binarized version of the target y, using 1 for values >=10 and 0 otherwise. The two protected attributes are sex and age and the disparate impact is 0.894. The dataset has 395 rows and 32 columns, including both categorical and numeric columns.

Returns

result –
- item 0: pandas Dataframe
  Features X, including both protected and non-protected attributes.
- item 1: pandas Series
  Labels y.
- item 3: fairness_info
  JSON meta-data following the format understood by fairness metrics and mitigation operators in *lale.lib.aif360*.

Return type
tuple

`lale.lib.aif360.datasets.fetch_student_por_df()`

Fetch the Student Performance (Portuguese) dataset from OpenML and add *fairness_info*.

The original prediction target is a integer Portuguese grade from 1 to 20. This function returns X unchanged but with a binarized version of the target y, using 1 for values >=10 and 0 otherwise. The two protected attributes are sex and age and the disparate impact is 0.858. The dataset has 649 rows and 32 columns, including both categorical and numeric columns.

Returns

result –

- item 0: pandas Dataframe
  Features X, including both protected and non-protected attributes.
- item 1: pandas Series
  Labels y.
- item 3: fairness_info
  JSON meta-data following the format understood by fairness metrics and mitigation operators in *lale.lib.aif360*.

Return type
tuple

`lale.lib.aif360.datasets.fetch_tae_df`(*preprocess: bool = False*)

Fetch the tae dataset from OpenML and add *fairness_info*.

It contains information from teaching assistant (TA) evaluations. at the University of Wisconsin–Madison. The prediction task is a classification on the type of rating a TA receives (1=Low, 2=Medium, 3=High). Without preprocessing, the dataset has 151 rows and 5 columns. There is one protected attributes, "whether_of_not_the_ta_is_a_native_english_speaker" [sic], and the disparate impact of 0.45. The data includes both categorical and numeric columns, with no missing values.

Parameters

preprocess (*boolean or "y", optional, default False*) – If True, encode protected attributes in X as 0 or 1 to indicate privileged group ("native_english_speaker"); encode labels in y as 0 or 1 to indicate favorable outcomes; and apply one-hot encoding to any remaining features in X that are categorical and not protecteded attributes. If "y", leave features X unchanged and only encode labels y as 0 or 1. If False, encode neither features X nor labels y.

Returns

result –

- item 0: pandas Dataframe
  Features X, including both protected and non-protected attributes.
- item 1: pandas Series
  Labels y.
- item 3: fairness_info
  JSON meta-data following the format understood by fairness metrics and mitigation operators in *lale.lib.aif360*.

Return type
tuple

`lale.lib.aif360.datasets.fetch_titanic_df`(*preprocess: bool = False*)

Fetch the Titanic dataset from OpenML and add *fairness_info*.

It contains data gathered from passengers on the Titanic with a binary classification into "survived" or "did not survive". Without preprocessing, the dataset has 1309 rows and 13 columns. There is one protected attribute, sex, and the disparate impact is 0.26. The data includes both categorical and numeric columns, with some missing values.

Parameters

`preprocess` (*boolean, optional, default False*) – If True, encode protected attributes in X as 0 or 1 to indicate privileged groups; and apply one-hot encoding to any remaining features in X that are categorical and not protected attributes.

Returns

result –

- item 0: pandas Dataframe

  Features X, including both protected and non-protected attributes.
- item 1: pandas Series

  Labels y.
- item 3: fairness_info

  JSON meta-data following the format understood by fairness metrics and mitigation operators in *lale.lib.aif360*.

Return type

tuple

`lale.lib.aif360.datasets.`**`fetch_us_crime_df`**`()`

Fetch the us_crime (also known as "communities and crime") dataset from OpenML and add *fairness_info*. The original dataset has several columns with a large number of missing values, which this function drops. The binary protected attribute is blackgt6pct, which is derived by thresholding racepctblack > 0.06 and dropping the original racepctblack. The binary target is derived by thresholding its original y > 0.70. The disparate impact is 0.888. The resulting dataset has 1,994 rows and 102 columns, all but one of which are numeric.

Returns

result –

- item 0: pandas Dataframe

  Features X, including both protected and non-protected attributes.
- item 1: pandas Series

  Labels y.
- item 3: fairness_info

  JSON meta-data following the format understood by fairness metrics and mitigation operators in *lale.lib.aif360*.

Return type

tuple

lale.lib.aif360.disparate_impact_remover module

class `lale.lib.aif360.disparate_impact_remover.`**`DisparateImpactRemover`**`(*`, *favorable_labels*, *protected_attributes*, *unfavorable_labels=None*, *redact=True*, *preparation=None*, *repair_level=1*`)`

Bases: *`PlannedIndividualOp`*

Disparate impact remover pre-estimator fairness mitigator. Edits feature values to increase group fairness while preserving rank-ordering within groups (Feldman et al. 2015). In the case of multiple protected attributes, the combined reference group is the intersection of the reference groups for each attribute.

This documentation is auto-generated from JSON schemas.

Parameters

- **`favorable_labels`** (*`array, >=1 items, not for optimizer`*) – Label values which are considered favorable (i.e. "positive").
  - items : union type
    - float
      Numerical value.
    - *or* string
      Literal string value.
    - *or* boolean
      Boolean value.
    - *or* array, >=2 items, <=2 items *of* items : float
      Numeric range [a,b] from a to b inclusive.
- **`protected_attributes`** (*`array, >=1 items, not for optimizer`*) – Features for which fairness is desired.
  - items : dict
    - feature : union type
      Column name or column index.
      - string
      - *or* integer
    - reference_group : array, >=1 items
      Values or ranges that indicate being a member of the privileged group.
      - items : union type
      - string
        Literal value.
      - *or* float
        Numerical value.
      - *or* array, >=2 items, <=2 items *of* items : float
        Numeric range [a,b] from a to b inclusive.
    - monitored_group : union type, default None
      Values or ranges that indicate being a member of the unprivileged group.
      - None
        If *monitored_group* is not explicitly specified, consider any values not captured by *reference_group* as monitored.
      - *or* array, >=1 items
      - items : union type
      - string
        Literal value.
      - *or* float
        Numerical value.
      - *or* array, >=2 items, <=2 items *of* items : float
        Numeric range [a,b] from a to b inclusive.
- **`unfavorable_labels`** (*`union type, not for optimizer, default None`*) – Label values which are considered unfavorable (i.e. "negative").
  - None
    If *unfavorable_labels* is not explicitly specified, consider any labels not captured by *favorable_labels* as unfavorable.
  - *or* array, >=1 items
    - items : union type
      - float
        Numerical value.
      - *or* string
        Literal string value.

- · *or* boolean
  Boolean value.
- · *or* array, >=2 items, <=2 items *of* items : float
  Numeric range [a,b] from a to b inclusive.

- **redact** (*boolean, not for optimizer, default True*) – Whether to redact protected attributes before data preparation (recommended) or not.
- **preparation** (*union type, not for optimizer, default None*) – Transformer, which may be an individual operator or a sub-pipeline.
  - operator
  - *or* None
    lale.lib.lale.NoOp
- **repair_level** (*float, >=0, <=1, default 1*) – Repair amount from 0 = none to 1 = full.

fit(*X*, *y=None*, ***fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

- **X** (*array*) – Features; the outer array is over samples.
  - items : array
    - * items : union type
      - · float
      - · *or* string
- **y** (*union type*) – Target class labels; the array is over samples.
  - array *of* items : float
  - *or* array *of* items : string

transform(*X*, *y=None*)

Transform the data.

Note: The transform method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (*array*) – Features; the outer array is over samples.

- items : array
  - items : union type
    - * float
    - * *or* string

Returns

result – Output data schema for reweighted features.

Return type

array *of* items : array *of* items : float

lale.lib.aif360.eq_odds_postprocessing module

class `lale.lib.aif360.eq_odds_postprocessing.`**`EqOddsPostprocessing`**(*, *favorable_labels*, *protected_attributes*, *unfavorable_labels=None*, *estimator*, *redact=True*, *seed=None*)

Bases: `PlannedIndividualOp`

Equalized odds postprocessing post-estimator fairness mitigator. Solves a linear program to find probabilities with which to change output labels to optimize equalized odds (Hardt et al. 2016, Pleiss et al. 2017).

This documentation is auto-generated from JSON schemas.

Parameters

- **favorable_labels** (*array, >=1 items, not for optimizer*) – Label values which are considered favorable (i.e. "positive").
  - items : union type
    - float
      Numerical value.
    - *or* string
      Literal string value.
    - *or* boolean
      Boolean value.
    - *or* array, >=2 items, <=2 items *of* items : float
      Numeric range [a,b] from a to b inclusive.
- **protected_attributes** (*array, >=1 items, not for optimizer*) – Features for which fairness is desired.
  - items : dict
    - feature : union type
      Column name or column index.
      - string
      - *or* integer
    - reference_group : array, >=1 items
      Values or ranges that indicate being a member of the privileged group.
      - items : union type
      - string
        Literal value.
      - *or* float
        Numerical value.
      - *or* array, >=2 items, <=2 items *of* items : float
        Numeric range [a,b] from a to b inclusive.
    - monitored_group : union type, default None
      Values or ranges that indicate being a member of the unprivileged group.
      - None
        If *monitored_group* is not explicitly specified, consider any values not captured by *reference_group* as monitored.
      - *or* array, >=1 items
      - items : union type
      - string
        Literal value.
      - *or* float

Numerical value.
- *or* array, >=2 items, <=2 items *of* items : float
  Numeric range [a,b] from a to b inclusive.

- **unfavorable_labels** (*union type, not for optimizer, default None*) – Label values which are considered unfavorable (i.e. "negative").
  - None
    If *unfavorable_labels* is not explicitly specified, consider any labels not captured by *favorable_labels* as unfavorable.
  - *or* array, >=1 items
    - items : union type
      - float
        Numerical value.
      - *or* string
        Literal string value.
      - *or* boolean
        Boolean value.
      - *or* array, >=2 items, <=2 items *of* items : float
        Numeric range [a,b] from a to b inclusive.
- **estimator** (*operator, not for optimizer*) – Nested supervised learning operator for which to mitigate fairness.
- **redact** (*boolean, not for optimizer, default True*) – Whether to redact protected attributes before data preparation (recommended) or not.
- **seed** (*union type, not for optimizer, default None*) – Seed to make *predict* repeatable.
  - integer
  - *or* None

fit(*X*, *y=None*, ***fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

- **X** (*array*) – Features; the outer array is over samples.
  - items : array
    - items : union type
      - float
      - *or* string
- **y** (*union type*) – Target class labels; the array is over samples.
  - array *of* items : float
  - *or* array *of* items : string

predict(*X*, ***predict_params*)

Make predictions.

Note: The predict method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (*array*) – Features; the outer array is over samples.

- items : array
  - items : union type
    - float
    - *or* string

Returns

result – Predicted class label per sample.

- array *of* items : float
- *or* array *of* items : string

Return type
union type

predict_proba(*X*)

Probability estimates for all classes.

Note: The predict_proba method is not available until this operator is trained.

Once this method is available, it will have the following signature:

lale.lib.aif360.gerry_fair_classifier module

class lale.lib.aif360.gerry_fair_classifier.**GerryFairClassifier**(**, favorable_labels, protected_attributes, unfavorable_labels=None, redact=True, preparation=None, C=10, printflag=False, heatmapflag=False, heatmap_iter=10, heatmap_path='.', max_iters=10, gamma=0.01, fairness_def='FP', predictor=None*)

Bases: `PlannedIndividualOp`

GerryFairClassifier in-estimator fairness mitigator. Attempts to learn classifiers that are fair with respect to rich subgroups (Kearns et al. 2018, Kearns et al. 2019). Rich subgroups are defined by (linear) functions over the sensitive attributes, and fairness notions are statistical: false positive, false negative, and statistical parity rates. This implementation uses a max of two regressions as a cost-sensitive classification oracle, and supports linear regression, support vector machines, decision trees, and kernel regression.

This documentation is auto-generated from JSON schemas.

Parameters

- **favorable_labels** (*array, >=1 items, not for optimizer*) – Label values which are considered favorable (i.e. "positive").
  - items : union type
    - float
      Numerical value.
    - *or* string
      Literal string value.
    - *or* boolean
      Boolean value.
    - *or* array, >=2 items, <=2 items *of* items : float
      Numeric range [a,b] from a to b inclusive.
- **protected_attributes** (*array, >=1 items, not for optimizer*) – Features for which fairness is desired.
  - items : dict
    - feature : union type
      Column name or column index.
      - string
      - *or* integer
    - reference_group : array, >=1 items

Values or ranges that indicate being a member of the privileged group.

· items : union type

· string

Literal value.

· *or* float

Numerical value.

· *or* array, >=2 items, <=2 items *of* items : float

Numeric range [a,b] from a to b inclusive.

∗ monitored_group : union type, default None

Values or ranges that indicate being a member of the unprivileged group.

· None

If *monitored_group* is not explicitly specified, consider any values not captured by *reference_group* as monitored.

· *or* array, >=1 items

· items : union type

· string

Literal value.

· *or* float

Numerical value.

· *or* array, >=2 items, <=2 items *of* items : float

Numeric range [a,b] from a to b inclusive.

- **unfavorable_labels** (*union type, not for optimizer, default None*) – Label values which are considered unfavorable (i.e. "negative").
  - None

    If *unfavorable_labels* is not explicitly specified, consider any labels not captured by *favorable_labels* as unfavorable.
  - *or* array, >=1 items
    - ∗ items : union type

      · float

      Numerical value.

      · *or* string

      Literal string value.

      · *or* boolean

      Boolean value.

      · *or* array, >=2 items, <=2 items *of* items : float

      Numeric range [a,b] from a to b inclusive.
- **redact** (*boolean, not for optimizer, default True*) – Whether to redact protected attributes before data preparation (recommended) or not.
- **preparation** (*union type, not for optimizer, default None*) – Transformer, which may be an individual operator or a sub-pipeline.
  - operator
  - *or* None

    lale.lib.lale.NoOp
- **C** (*float, >=0.03125 for optimizer, <=32768 for optimizer, default 10*) – Maximum L1 norm for the dual variables.
- **printflag** (*boolean, not for optimizer, default False*) – Print output flag.
- **heatmapflag** (*boolean, not for optimizer, default False*) – Save heatmaps every heatmap_iter flag.
- **heatmap_iter** (*integer, >=1, not for optimizer, default 10*) – Save heatmaps every heatmap_iter.

- **heatmap_path** (*string, not for optimizer, default '.'*) – Save heatmaps path.
- **max_iters** (*integer, >=1, <=1000 for optimizer, loguniform distribution, default 10*) – Time horizon for the fictitious play dynamic.
- **gamma** (*float, >0.0, >=0.001 for optimizer, <=1.0 for optimizer, loguniform distribution, default 0.01*) – Fairness approximation parameter.
- **fairness_def** ('FP' *or* 'FN', default 'FP') – Fairness notion.
- **predictor** (*union type, not for optimizer, default None*) – Hypothesis class for the learner.
  - operator

    Supports LR, SVM, KR, Trees.
  - *or* None

    sklearn.linear_model.LinearRegression

fit(*X*, *y=None*, ***fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

- **X** (*array*) – Features; the outer array is over samples.
  - items : array
    - items : union type
      - float
      - *or* string
- **y** (*union type*) – Target class labels; the array is over samples.
  - array *of* items : float
  - *or* array *of* items : string

predict(*X*, ***predict_params*)

Make predictions.

Note: The predict method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (*array*) – Features; the outer array is over samples.

- items : array
  - items : union type
    - float
    - *or* string

Returns

result – Predicted class label per sample.

- array *of* items : float
- *or* array *of* items : string

Return type

union type

predict_proba(*X*)

Probability estimates for all classes.

Note: The predict_proba method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (*array*) – Features; the outer array is over samples.

- items : array
  - items : union type
    - float
    - *or* string

Returns

result – The class probabilities of the input samples

- array *of* items : Any
- *or* array *of* items : array *of* items : Any

Return type

union type

lale.lib.aif360.lfr module

class `lale.lib.aif360.lfr.LFR`(*, *favorable_labels*, *protected_attributes*, *unfavorable_labels=None*, *redact=True*, *preparation=None*, *k=5*, *Ax=0.01*, *Az=1.0*, *Ay=50.0*, *print_interval=250*, *verbose=0*, *seed=None*)

Bases: `PlannedIndividualOp`

LFR (learning fair representations) pre-estimator fairness mitigator. Finds a latent representation that encodes the data well but obfuscates information about protected attributes (Zemel et al. 2013).

This documentation is auto-generated from JSON schemas.

Parameters

- **favorable_labels** (*array, >=1 items, not for optimizer*) – Label values which are considered favorable (i.e. "positive").
  - items : union type
    - float

      Numerical value.
    - *or* string

      Literal string value.
    - *or* boolean

      Boolean value.
    - *or* array, >=2 items, <=2 items *of* items : float

      Numeric range [a,b] from a to b inclusive.
- **protected_attributes** (*array, >=1 items, not for optimizer*) – Features for which fairness is desired.
  - items : dict
    - feature : union type

      Column name or column index.
      - string
      - *or* integer
    - reference_group : array, >=1 items

      Values or ranges that indicate being a member of the privileged group.
      - items : union type
      - string

        Literal value.
      - *or* float

        Numerical value.
      - *or* array, >=2 items, <=2 items *of* items : float

        Numeric range [a,b] from a to b inclusive.
    - monitored_group : union type, default None

Values or ranges that indicate being a member of the unprivileged group.

· None

If *monitored_group* is not explicitly specified, consider any values not captured by *reference_group* as monitored.

· *or* array, >=1 items

· items : union type

· string

Literal value.

· *or* float

Numerical value.

· *or* array, >=2 items, <=2 items *of* items : float

Numeric range [a,b] from a to b inclusive.

- **unfavorable_labels** (*union type, not for optimizer, default None*) – Label values which are considered unfavorable (i.e. "negative").
  - None

    If *unfavorable_labels* is not explicitly specified, consider any labels not captured by *favorable_labels* as unfavorable.
  - *or* array, >=1 items
    - items : union type

      · float

      Numerical value.

      · *or* string

      Literal string value.

      · *or* boolean

      Boolean value.

      · *or* array, >=2 items, <=2 items *of* items : float

      Numeric range [a,b] from a to b inclusive.
- **redact** (*boolean, not for optimizer, default True*) – Whether to redact protected attributes before data preparation (recommended) or not.
- **preparation** (*union type, not for optimizer, default None*) – Transformer, which may be an individual operator or a sub-pipeline.
  - operator
  - *or* None

    lale.lib.lale.NoOp
- **k** (*integer, >=1, <=20 for optimizer, default 5*) – Number of prototypes.
- **Ax** (*float, >=0.0, <=100.0 for optimizer, default 0.01*) – Input recontruction quality term weight.
- **Az** (*float, >=0.0, <=100.0 for optimizer, default 1.0*) – Fairness constraint term weight.
- **Ay** (*float, >=0.0, <=100.0 for optimizer, default 50.0*) – Output prediction error.
- **print_interval** (*integer, >=1, not for optimizer, default 250*) – Print optimization objective value every print_interval iterations.
- **verbose** (*integer, not for optimizer, default 0*) – If zero, then no output.
- **seed** (*union type, not for optimizer, default None*) – Seed to make *transform* repeatable.
  - integer
  - *or* None

fit(*X*, *y=None*, ***fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

- **X** (*array*) – Features; the outer array is over samples.
  - items : array
    - items : union type
      - float
      - *or* string
- **y** (*union type*) – Target class labels; the array is over samples.
  - array *of* items : float
  - *or* array *of* items : string

transform(*X*, *y=None*)

Transform the data.

Note: The transform method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (*array*) – Features; the outer array is over samples.

- items : array
  - items : union type
    - float
    - *or* string

Returns

result – Output data schema for reweighted features.

Return type

array *of* items : array *of* items : float

lale.lib.aif360.meta_fair_classifier module

class lale.lib.aif360.meta_fair_classifier.**MetaFairClassifier**(**, favorable_labels, protected_attributes, unfavorable_labels=None, redact=True, preparation=None, tau=0.8, type='fdr'*)

Bases: *PlannedIndividualOp*

MetaFairClassifier in-estimator fairness mitigator. Takes the fairness metric as part of the input and returns a classifier optimized with respect to that fairness metric (Celis et al. 2019).

This documentation is auto-generated from JSON schemas.

Parameters

- **favorable_labels** (*array, >=1 items, not for optimizer*) – Label values which are considered favorable (i.e. “positive”).
  - items : union type
    - float

      Numerical value.
    - *or* string

      Literal string value.
    - *or* boolean

      Boolean value.
    - *or* array, >=2 items, <=2 items *of* items : float

      Numeric range [a,b] from a to b inclusive.
- **protected_attributes** (*array, >=1 items, not for optimizer*) – Features for which fairness is desired.

- items : dict
  - feature : union type
    Column name or column index.
    - string
    - *or* integer
  - reference_group : array, >=1 items
    Values or ranges that indicate being a member of the privileged group.
    - items : union type
    - string
      Literal value.
    - *or* float
      Numerical value.
    - *or* array, >=2 items, <=2 items *of* items : float
      Numeric range [a,b] from a to b inclusive.
  - monitored_group : union type, default None
    Values or ranges that indicate being a member of the unprivileged group.
    - None
      If *monitored_group* is not explicitly specified, consider any values not captured by *reference_group* as monitored.
    - *or* array, >=1 items
    - items : union type
    - string
      Literal value.
    - *or* float
      Numerical value.
    - *or* array, >=2 items, <=2 items *of* items : float
      Numeric range [a,b] from a to b inclusive.

- **unfavorable_labels** (*union type, not for optimizer, default None*) – Label values which are considered unfavorable (i.e. "negative").
  - None
    If *unfavorable_labels* is not explicitly specified, consider any labels not captured by *favorable_labels* as unfavorable.
  - *or* array, >=1 items
    - items : union type
      - float
        Numerical value.
      - *or* string
        Literal string value.
      - *or* boolean
        Boolean value.
      - *or* array, >=2 items, <=2 items *of* items : float
        Numeric range [a,b] from a to b inclusive.
- **redact** (*boolean, not for optimizer, default True*) – Whether to redact protected attributes before data preparation (recommended) or not.
- **preparation** (*union type, not for optimizer, default None*) – Transformer, which may be an individual operator or a sub-pipeline.
  - operator
  - *or* None
    lale.lib.lale.NoOp
- **tau** (*float, >=0.0, <=1.0, default 0.8*) – Fairness penalty parameter.
- **type** (*union type, default 'fdr'*) – The type of fairness metric to be used.

- 'fdr'
  False discovery rate ratio.
- *or* 'sr'
  Statistical rate / disparate impact.

fit(*X*, *y=None*, ***fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

- **X** (*array*) – Features; the outer array is over samples.
  - items : array
    - items : union type
      - float
      - *or* string
- **y** (*union type*) – Target class labels; the array is over samples.
  - array *of* items : float
  - *or* array *of* items : string

predict(*X*, ***predict_params*)

Make predictions.

Note: The predict method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (*array*) – Features; the outer array is over samples.

- items : array
  - items : union type
    - float
    - *or* string

Returns

result – Predicted class label per sample.

- array *of* items : float
- *or* array *of* items : string

Return type

union type

predict_proba(*X*)

Probability estimates for all classes.

Note: The predict_proba method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (*array*) – Features; the outer array is over samples.

- items : array
  - items : union type
    - float
    - *or* string

Returns

result – The class probabilities of the input samples

- array *of* items : Any
- *or* array *of* items : array *of* items : Any

Return type
union type

lale.lib.aif360.optim_preproc module

class lale.lib.aif360.optim_preproc.**OptimPreproc**(**, favorable_labels, protected_attributes, unfavorable_labels=None, optimizer=None, optim_options={}, verbose=0, seed=None*)

Bases: *PlannedIndividualOp*

Work-in-progress, not covered in successful test yet: Optimized Preprocessing pre-estimator fairness mitigator. Learns a probabilistic transformation that edits the features and labels in the data with group fairness, individual distortion, and data fidelity constraints and objectives (Calmon et al. 2017).

This documentation is auto-generated from JSON schemas.

Parameters

- **favorable_labels** (*array, >=1 items, not for optimizer*) – Label values which are considered favorable (i.e. "positive").
  - items : union type
    - float
      Numerical value.
    - *or* string
      Literal string value.
    - *or* boolean
      Boolean value.
    - *or* array, >=2 items, <=2 items *of* items : float
      Numeric range [a,b] from a to b inclusive.
- **protected_attributes** (*array, >=1 items, not for optimizer*) – Features for which fairness is desired.
  - items : dict
    - feature : union type
      Column name or column index.
      - string
      - *or* integer
    - reference_group : array, >=1 items
      Values or ranges that indicate being a member of the privileged group.
      - items : union type
      - string
        Literal value.
      - *or* float
        Numerical value.
      - *or* array, >=2 items, <=2 items *of* items : float
        Numeric range [a,b] from a to b inclusive.
    - monitored_group : union type, default None
      Values or ranges that indicate being a member of the unprivileged group.
      - None
        If *monitored_group* is not explicitly specified, consider any values not captured by *reference_group* as monitored.
      - *or* array, >=1 items
      - items : union type
      - string

Literal value.
  · *or* float
    Numerical value.
  · *or* array, >=2 items, <=2 items *of* items : float
    Numeric range [a,b] from a to b inclusive.

- **unfavorable_labels** (*union type, not for optimizer, default None*) – Label values which are considered unfavorable (i.e. "negative").
  - None
    If *unfavorable_labels* is not explicitly specified, consider any labels not captured by *favorable_labels* as unfavorable.
  - *or* array, >=1 items
    - items : union type
      · float
        Numerical value.
      · *or* string
        Literal string value.
      · *or* boolean
        Boolean value.
      · *or* array, >=2 items, <=2 items *of* items : float
        Numeric range [a,b] from a to b inclusive.
- **optimizer** (*union type, not for optimizer, default None*) – Optimizer class.
  - Any
    User-provided.
  - *or* None
    Use *aif360.algorithms.preprocessing.optim_preproc_helpers.opt_tools.OptTools*.
- **optim_options** (*dict, not for optimizer, default {}*) – Options for optimization to estimate the transformation.
- **verbose** (*integer, not for optimizer, default 0*) – If zero, then no output.
- **seed** (*union type, not for optimizer, default None*) – Seed to make *transform* repeatable.
  - integer
  - *or* None

fit(*X*, *y=None*, ***fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

- **X** (*array*) – Features; the outer array is over samples.
  - items : array
    - items : union type
      · float
      · *or* string
- **y** (*union type*) – Target class labels; the array is over samples.
  - array *of* items : float
  - *or* array *of* items : string

transform(*X*, *y=None*)

Transform the data.

Note: The transform method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters
X (*array*) – Features; the outer array is over samples.
- items : array
  - items : union type
    - float
    - *or* string

Returns
result – Output data schema for transform.

Return type
array *of* items : array *of* items : float

lale.lib.aif360.orbis module

class lale.lib.aif360.orbis.**Orbis**(*, *favorable_labels*, *protected_attributes*, *unfavorable_labels=None*, *estimator*, *redact=True*, *imbalance_repair_level=0.8*, *bias_repair_level=0.8*, *combine='keep_separate'*, *sampling_strategy='mixed'*, *replacement=False*, *n_jobs=1*, *random_state=None*, *k_neighbors=5*)

Bases: *PlannedIndividualOp*

Experimental Orbis (Oversampling to Repair Bias and Imbalance Simultaneously) pre-estimator fairness mitigator.

This documentation is auto-generated from JSON schemas.

Work in progress and subject to change; only supports pandas DataFrame so far. Uses SMOTE and RandomUnderSampler to resample not only for repairing class imbalance, but also group bias. Internally, this works by replacing class labels by the cross product of classes and groups, then changing the sizes of the new intersections to achieve the desired repair levels. Unlike other mitigators in *lale.lib.aif360*, this mitigator does not come from AIF360.

Parameters
- **favorable_labels** (*array, >=1 items, not for optimizer*) – Label values which are considered favorable (i.e. "positive").
  - items : union type
    - float
      Numerical value.
    - *or* string
      Literal string value.
    - *or* boolean
      Boolean value.
    - *or* array, >=2 items, <=2 items *of* items : float
      Numeric range [a,b] from a to b inclusive.
- **protected_attributes** (*array, >=1 items, not for optimizer*) – Features for which fairness is desired.
  - items : dict
    - feature : union type
      Column name or column index.
      - string
      - *or* integer
    - reference_group : array, >=1 items
      Values or ranges that indicate being a member of the privileged group.
      - items : union type
      - string

Literal value.
· *or* float
Numerical value.
· *or* array, >=2 items, <=2 items *of* items : float
Numeric range [a,b] from a to b inclusive.
* monitored_group : union type, default None
Values or ranges that indicate being a member of the unprivileged group.
· None
If *monitored_group* is not explicitly specified, consider any values not captured by *reference_group* as monitored.
· *or* array, >=1 items
· items : union type
· string
Literal value.
· *or* float
Numerical value.
· *or* array, >=2 items, <=2 items *of* items : float
Numeric range [a,b] from a to b inclusive.

- **unfavorable_labels** (*union type, not for optimizer, default None*) – Label values which are considered unfavorable (i.e. "negative").
  - None
    If *unfavorable_labels* is not explicitly specified, consider any labels not captured by *favorable_labels* as unfavorable.
  - *or* array, >=1 items
    * items : union type
      · float
      Numerical value.
      · *or* string
      Literal string value.
      · *or* boolean
      Boolean value.
      · *or* array, >=2 items, <=2 items *of* items : float
      Numeric range [a,b] from a to b inclusive.
- **estimator** (*operator, not for optimizer*) – Nested classifier.
- **redact** (*boolean, optional, not for optimizer, default True*) – Whether to redact protected attributes before data preparation (recommended) or not.
- **imbalance_repair_level** (*float, >=0.0, <=1.0, optional, default 0.8*) – How much to repair for class imbalance (0 means original imbalance, 1 means perfect balance).

  See also *constraint-1*.
- **bias_repair_level** (*float, >=0.0, <=1.0, optional, default 0.8*) – How much to repair for group bias (0 means original bias, 1 means perfect fairness).

  See also *constraint-1*.
- **combine** ('keep_separate', 'and', 'or', *or* 'error', optional, not for optimizer, default 'keep_separate') – How to handle the case when there is more than one protected attribute.
- **sampling_strategy** ('under', 'over', 'mixed', 'minimum', *or* 'maximum', optional, not for optimizer, default 'mixed') –
  How to change the intersection sizes.
  Possible choices are:
  – `'under'`: under-sample large intersections to desired repair levels;

    - 'over': over-sample small intersection to desired repair levels;
    - 'mixed': mix under- with over-sampling while keeping sizes similar to original;
    - 'minimum': under-sample everything to the size of the smallest intersection;
    - 'maximum': over-sample everything to the size of the largest intersection.

  See also *constraint-1*.

- **replacement** (*boolean, optional, not for optimizer, default False*) – Whether under-sampling is with or without replacement.
- **n_jobs** (*integer, optional, not for optimizer, default 1*) – The number of threads to open if possible.
- **random_state** (*union type, optional, not for optimizer, default None*) – Control the randomization of the algorithm.
  - None

    RandomState used by np.random
  - *or* integer

    The seed used by the random number generator
  - *or* numpy.random.RandomState

    Random number generator instance.
- **k_neighbors** (*union type, optional, not for optimizer, default 5*) – Number of nearest neighbours to use to construct synthetic samples.
  - integer

    Number of nearest neighbours to use to construct synthetic samples.
  - *or* Any

    An estimator that inherits from `sklearn.neighbors.base.KNeighborsMixin` that will be used to find the *n_neighbors*.

Notes

constraint-1 : union type

When sampling_strategy is minimum or maximum, both repair levels must be 1.

- sampling_strategy : negated type *of* 'minimum' *or* 'maximum'
- *or* dict
  - imbalance_repair_level : 1
  - bias_repair_level : 1

fit(*X*, *y=None*, ***fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

- **X** (*array*) – Features; the outer array is over samples.
  - items : array
    - items : union type
      - float
      - *or* string
- **y** (*union type*) – Target class labels; the array is over samples.
  - array *of* items : float
  - *or* array *of* items : string

predict(*X*, ***predict_params*)

Make predictions.

Note: The predict method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (*array*) – Features; the outer array is over samples.

- items : array
  - items : union type
    - float
    - *or* string

Returns

result – Predicted class label per sample.

- array *of* items : float
- *or* array *of* items : string

Return type

union type

predict_proba(*X*)

Probability estimates for all classes.

Note: The predict_proba method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (*array*) – Features; the outer array is over samples.

- items : array
  - items : union type
    - float
    - *or* string

Returns

result – The class probabilities of the input samples

- array *of* items : Any
- *or* array *of* items : array *of* items : Any

Return type

union type

lale.lib.aif360.prejudice_remover module

class lale.lib.aif360.prejudice_remover.**PrejudiceRemover**(**, favorable_labels, protected_attributes, unfavorable_labels=None, redact=True, preparation=None, eta=1.0*)

Bases: *PlannedIndividualOp*

PrejudiceRemover in-estimator fairness mitigator. Adds a discrimination-aware regularization term to the learning objective (Kamishima et al. 2012).

This documentation is auto-generated from JSON schemas.

Parameters

- **favorable_labels** (*array, >=1 items, not for optimizer*) – Label values which are considered favorable (i.e. "positive").
  - items : union type
    - float

      Numerical value.
    - *or* string

      Literal string value.
    - *or* boolean

      Boolean value.

    * *or* array, >=2 items, <=2 items *of* items : float
      Numeric range [a,b] from a to b inclusive.

- **protected_attributes** (*array, >=1 items, not for optimizer*) – Features for which fairness is desired.
  - items : dict
    * feature : union type
      Column name or column index.
      · string
      · *or* integer
    * reference_group : array, >=1 items
      Values or ranges that indicate being a member of the privileged group.
      · items : union type
      · string
        Literal value.
      · *or* float
        Numerical value.
      · *or* array, >=2 items, <=2 items *of* items : float
        Numeric range [a,b] from a to b inclusive.
    * monitored_group : union type, default None
      Values or ranges that indicate being a member of the unprivileged group.
      · None
        If *monitored_group* is not explicitly specified, consider any values not captured by *reference_group* as monitored.
      · *or* array, >=1 items
      · items : union type
      · string
        Literal value.
      · *or* float
        Numerical value.
      · *or* array, >=2 items, <=2 items *of* items : float
        Numeric range [a,b] from a to b inclusive.
- **unfavorable_labels** (*union type, not for optimizer, default None*) – Label values which are considered unfavorable (i.e. "negative").
  - None
    If *unfavorable_labels* is not explicitly specified, consider any labels not captured by *favorable_labels* as unfavorable.
  - *or* array, >=1 items
    * items : union type
      · float
        Numerical value.
      · *or* string
        Literal string value.
      · *or* boolean
        Boolean value.
      · *or* array, >=2 items, <=2 items *of* items : float
        Numeric range [a,b] from a to b inclusive.
- **redact** (*boolean, not for optimizer, default True*) – Whether to redact protected attributes before data preparation (recommended) or not.
- **preparation** (*union type, not for optimizer, default None*) – Transformer, which may be an individual operator or a sub-pipeline.
  - operator

  - *or* None

    lale.lib.lale.NoOp

- **eta** (*float, >0.0, >=0.03125 for optimizer, <=32768 for optimizer, default 1.0*) – Fairness penalty parameter.

fit(*X*, *y=None*, ***fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

- **X** (*array*) – Features; the outer array is over samples.
  - items : array
    - items : union type
      - float
      - *or* string
- **y** (*union type*) – Target class labels; the array is over samples.
  - array *of* items : float
  - *or* array *of* items : string

predict(*X*, ***predict_params*)

Make predictions.

Note: The predict method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (*array*) – Features; the outer array is over samples.

- items : array
  - items : union type
    - float
    - *or* string

Returns

result – Predicted class label per sample.

- array *of* items : float
- *or* array *of* items : string

Return type

union type

predict_proba(*X*)

Probability estimates for all classes.

Note: The predict_proba method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (*array*) – Features; the outer array is over samples.

- items : array
  - items : union type
    - float
    - *or* string

Returns

result – The class probabilities of the input samples

- array *of* items : Any
- *or* array *of* items : array *of* items : Any

Return type
union type

lale.lib.aif360.protected_attributes_encoder module

class lale.lib.aif360.protected_attributes_encoder.**ProtectedAttributesEncoder**(**args*, *_lale_trained=False*, *_lale_impl=None*, ***kwargs*)

Bases: *TrainedIndividualOp*

Protected attributes encoder.

This documentation is auto-generated from JSON schemas.

The *protected_attributes* argument describes each sensitive column by a *feature* name or index and a *reference_group* list of values or ranges. This transformer encodes protected attributes with values of *0*, *0.5*, or *1* to indicate membership in the unprivileged, neither, or privileged group, respectively. That encoding makes the protected attributes suitable as input for downstream fairness mitigation operators. This operator does not encode the remaining (non-protected) attributes. A common usage is to encode non-protected attributes with a separate data preparation pipeline and to perform a feature union before piping the transformed data to downstream operators that require numeric data. This operator is used internally by various lale.lib.aif360 metrics and mitigators, so you often do not need to use it directly yourself.

Parameters

- **favorable_labels** (*union type, optional, not for optimizer, default None*) –
  - array, >=1 items
    Label values which are considered favorable (i.e. "positive").
    - items : union type
      - float
        Numerical value.
      - *or* string
        Literal string value.
      - *or* boolean
        Boolean value.
      - *or* array, >=2 items, <=2 items *of* items : float
        Numeric range [a,b] from a to b inclusive.
  - *or* None

  See also *constraint-1*.
- **protected_attributes** (*union type, not for optimizer, default None*) –
  - array, >=1 items
    Features for which fairness is desired.
    - items : dict
      - feature : union type
        Column name or column index.
        string
        or integer
      - reference_group : array, >=1 items
        Values or ranges that indicate being a member of the privileged group.
        items : union type
        string
        Literal value.

or float
Numerical value.
or array, >=2 items, <=2 items *of* items : float
Numeric range [a,b] from a to b inclusive.
· monitored_group : union type, default None
Values or ranges that indicate being a member of the unprivileged group.
None
If *monitored_group* is not explicitly specified, consider any values not captured by *reference_group* as monitored.
or array, >=1 items
items : union type
string
Literal value.
or float
Numerical value.
or array, >=2 items, <=2 items *of* items : float
Numeric range [a,b] from a to b inclusive.
   - *or* None

  See also *constraint-2*.

- **unfavorable_labels** (*union type, optional, not for optimizer, default None*) – Label values which are considered unfavorable (i.e. "negative").
  - None
    If *unfavorable_labels* is not explicitly specified, consider any labels not captured by *favorable_labels* as unfavorable.
  - *or* array, >=1 items
    * items : union type
      · float
        Numerical value.
      · *or* string
        Literal string value.
      · *or* boolean
        Boolean value.
      · *or* array, >=2 items, <=2 items *of* items : float
        Numeric range [a,b] from a to b inclusive.
- **remainder** ('passthrough' *or* 'drop', optional, not for optimizer, default 'drop') – Transformation for columns that were not specified in protected_attributes.
- **return_X_y** (*boolean, optional, not for optimizer, default False*) – Deprecated, use transform_X_y instead. If True, transform returns a tuple with X and y; otherwise, transform returns only X, not as a tuple.

  See also *constraint-1*.
- **combine** ('keep_separate', 'and', 'or', *or* 'error', optional, not for optimizer, default 'keep_separate') – How to handle the case when there is more than one protected attribute.

  See also *constraint-2*.

Notes

constraint-1 : union type

If returning y, need to know how to encode it.

- return_X_y : False
- *or* favorable_labels : negated type *of* None

constraint-2 : union type

If combine is error, must have only one protected attribute.

- combine : negated type *of* 'error'
- *or* protected_attributes : array, <=1 items

transform(*X*, *y=None*)

Transform the data.

Parameters

- **X** (*array*) – Features; the outer array is over samples.
  - items : array
    - items : union type
      - float
      - *or* string
- **y** (*union type, default None*) – Target labels.
  - array
    - items : union type
      - float
      - *or* string
  - *or* None

Returns

result –

- array

  If return_X_y is False, return X.
  - items : array

    This operator encodes protected attributes as *0*, *0.5*, or *1*. So if the remainder (non-protected attributes) is dropped, the output is numeric. Otherwise, the output may still contain non-numeric values.
    - items : union type
      - float
      - *or* string
- *or* tuple

  If return_X_y is True, return tuple of X and y.
  - item 0 : array

    X
    - items : array
      - items : union type
      - float
      - *or* string
  - item 1 : array

    y
    - items : union type
      - float
      - *or* string

Return type

union type

transform_X_y(*X*, *y*)

Transform the data and target.

Parameters

- **X** (*array*) – Features; the outer array is over samples.
  - items : array
    - items : union type
      - float
      - *or* string
- **y** (*array*) – Target labels.
  - items : union type
    - float
    - *or* string

Returns

result –

- item 0 : array

  X
  - items : array
    - items : union type
      - float
      - *or* string
- item 1 : array

  y
  - items : union type
    - float
    - *or* string

Return type

tuple

lale.lib.aif360.redacting module

class `lale.lib.aif360.redacting.Redacting`(*, *favorable_labels*, *protected_attributes*, *unfavorable_labels*)

Bases: *PlannedIndividualOp*

Redacting preprocessor for fairness mitigation.

This documentation is auto-generated from JSON schemas.

This sets all the protected attributes to constants, using the most frequent value in the column. This operator is used internally by various lale.lib.aif360 metrics and mitigators, so you often do not need to use it directly yourself.

Parameters

- **favorable_labels** (*Any, not for optimizer*) – Ignored.
- **protected_attributes** (*array, >=1 items, not for optimizer*) – Features for which fairness is desired.
  - items : dict
    - feature : union type

      Column name or column index.
      - string
      - *or* integer
    - reference_group : array, >=1 items

      Values or ranges that indicate being a member of the privileged group.
      - items : union type
      - string

Literal value.
- *or* float
  Numerical value.
- *or* array, >=2 items, <=2 items *of* items : float
  Numeric range [a,b] from a to b inclusive.
- monitored_group : union type, default None
  Values or ranges that indicate being a member of the unprivileged group.
  - None
    If *monitored_group* is not explicitly specified, consider any values not captured by *reference_group* as monitored.
  - *or* array, >=1 items
  - items : union type
  - string
    Literal value.
  - *or* float
    Numerical value.
  - *or* array, >=2 items, <=2 items *of* items : float
    Numeric range [a,b] from a to b inclusive.
- **unfavorable_labels** (*Any, not for optimizer*) – Ignored.

fit(*X*, *y=None*, ***fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters
- **X** (*array*) – Features; the outer array is over samples.
  - items : array
    - items : union type
      - float
      - *or* string
- **y** (*any type, optional*) – Target values; the array is over samples.

transform(*X*, *y=None*)

Transform the data.

Note: The transform method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (*array*) – Features; the outer array is over samples.
- items : array
  - items : union type
    - float
    - *or* string

Returns

result – Output data schema for reweighted features.
- items : array
  - items : union type
    - float
    - *or* string

Return type

array

lale.lib.aif360.reject_option_classification module

class lale.lib.aif360.reject_option_classification.**RejectOptionClassification**(***, *favorable_labels*, *protected_attributes*, *unfavorable_labels=None*, *estimator*, *redact=True*, *low_class_thresh=0.01*, *high_class_thresh=0.99*, *num_class_thresh=100*, *num_ROC_margin=50*, *metric_name='Statistical parity difference'*, *metric_ub=0.05*, *metric_lb=-0.05*, *repair_level=None*)

Bases: *PlannedIndividualOp*

Reject option classification post-estimator fairness mitigator. Gives favorable outcomes to unpriviliged groups and unfavorable outcomes to priviliged groups in a confidence band around the decision boundary with the highest uncertainty (Kamiran et al. 2012).

This documentation is auto-generated from JSON schemas.

Parameters

- **favorable_labels** (*array, >=1 items, not for optimizer*) – Label values which are considered favorable (i.e. "positive").
  - items : union type
    - float
      Numerical value.
    - *or* string
      Literal string value.
    - *or* boolean
      Boolean value.
    - *or* array, >=2 items, <=2 items *of* items : float
      Numeric range [a,b] from a to b inclusive.
- **protected_attributes** (*array, >=1 items, not for optimizer*) – Features for which fairness is desired.
  - items : dict
    - feature : union type
      Column name or column index.
      - string
      - *or* integer
    - reference_group : array, >=1 items
      Values or ranges that indicate being a member of the privileged group.
      - items : union type

· string
  Literal value.
· *or* float
  Numerical value.
· *or* array, >=2 items, <=2 items *of* items : float
  Numeric range [a,b] from a to b inclusive.

∗ monitored_group : union type, default None
  Values or ranges that indicate being a member of the unprivileged group.
  · None
    If *monitored_group* is not explicitly specified, consider any values not captured by *reference_group* as monitored.
  · *or* array, >=1 items
  · items : union type
  · string
    Literal value.
  · *or* float
    Numerical value.
  · *or* array, >=2 items, <=2 items *of* items : float
    Numeric range [a,b] from a to b inclusive.

- **unfavorable_labels** (*union type, not for optimizer, default None*) – Label values which are considered unfavorable (i.e. "negative").
  - None
    If *unfavorable_labels* is not explicitly specified, consider any labels not captured by *favorable_labels* as unfavorable.
  - *or* array, >=1 items
    ∗ items : union type
      · float
        Numerical value.
      · *or* string
        Literal string value.
      · *or* boolean
        Boolean value.
      · *or* array, >=2 items, <=2 items *of* items : float
        Numeric range [a,b] from a to b inclusive.
- **estimator** (*operator, not for optimizer*) – Nested supervised learning operator for which to mitigate fairness.
- **redact** (*boolean, not for optimizer, default True*) – Whether to redact protected attributes before data preparation (recommended) or not.
- **low_class_thresh** (*float, >=0.0, <=1.0, not for optimizer, default 0.01*) – Smallest classification threshold to use in the optimization.
- **high_class_thresh** (*float, >=0.0, <=1.0, not for optimizer, default 0.99*) – Highest classification threshold to use in the optimization.
- **num_class_thresh** (*integer, >=1, not for optimizer, default 100*) – Number of classification thresholds between low_class_thresh and high_class_thresh for the optimization search.
- **num_ROC_margin** (*integer, >=1, not for optimizer, default 50*) – Number of relevant ROC margins to be used in the optimization search.
- **metric_name** ('Statistical parity difference', 'Average odds difference', *or* 'Equal opportunity difference', default 'Statistical parity difference') – Name of the metric to use for the optimization.
- **metric_ub** (*float, >=0, <=1, not for optimizer, default 0.05*) – Upper bound of constraint on the metric value.

- **metric_lb** (*float*, *>=-1*, *<=0*, *not for optimizer*, *default -0.05*) – Lower bound of constraint on the metric value.
- **repair_level** (*union type*, *optional*, *not for optimizer*, *default None*) – Repair amount from 0 = none to 1 = full.
  - None

    Keep metric_lb and metric_ub unchanged.
  - *or* float, >=0, <=1

    Set metric_ub = 1 - repair_level and metric_lb = - metric_ub.

fit(*X*, *y=None*, ***fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

- **X** (*array*) – Features; the outer array is over samples.
  - items : array
    - items : union type
      - float
      - *or* string
- **y** (*union type*) – Target class labels; the array is over samples.
  - array *of* items : float
  - *or* array *of* items : string

predict(*X*, ***predict_params*)

Make predictions.

Note: The predict method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (*array*) – Features; the outer array is over samples.

- items : array
  - items : union type
    - float
    - *or* string

Returns

result – Predicted class label per sample.

- array *of* items : float
- *or* array *of* items : string

Return type

union type

predict_proba(*X*)

Probability estimates for all classes.

Note: The predict_proba method is not available until this operator is trained.

Once this method is available, it will have the following signature:

lale.lib.aif360.reweighing module

class `lale.lib.aif360.reweighing.`**`Reweighing`**(**, favorable_labels, protected_attributes, unfavorable_labels=None, estimator, redact=True*)

Bases: *PlannedIndividualOp*

Reweighing pre-estimator fairness mitigator. Weights the examples in each (group, label) combination differently to ensure fairness before classification (Kamiran and Calders 2012).

This documentation is auto-generated from JSON schemas.

Parameters

- **favorable_labels** (*array, >=1 items, not for optimizer*) – Label values which are considered favorable (i.e. "positive").
  - items : union type
    - float
      Numerical value.
    - *or* string
      Literal string value.
    - *or* boolean
      Boolean value.
    - *or* array, >=2 items, <=2 items *of* items : float
      Numeric range [a,b] from a to b inclusive.
- **protected_attributes** (*array, >=1 items, not for optimizer*) – Features for which fairness is desired.
  - items : dict
    - feature : union type
      Column name or column index.
      - string
      - *or* integer
    - reference_group : array, >=1 items
      Values or ranges that indicate being a member of the privileged group.
      - items : union type
      - string
        Literal value.
      - *or* float
        Numerical value.
      - *or* array, >=2 items, <=2 items *of* items : float
        Numeric range [a,b] from a to b inclusive.
    - monitored_group : union type, default None
      Values or ranges that indicate being a member of the unprivileged group.
      - None
        If *monitored_group* is not explicitly specified, consider any values not captured by *reference_group* as monitored.
      - *or* array, >=1 items
      - items : union type
      - string
        Literal value.
      - *or* float
        Numerical value.
      - *or* array, >=2 items, <=2 items *of* items : float
        Numeric range [a,b] from a to b inclusive.

- **unfavorable_labels** (*union type, not for optimizer, default None*) – Label values which are considered unfavorable (i.e. “negative”).
  - None
    If *unfavorable_labels* is not explicitly specified, consider any labels not captured by *favorable_labels* as unfavorable.
  - *or* array, >=1 items
    - items : union type
      - float
        Numerical value.
      - *or* string
        Literal string value.
      - *or* boolean
        Boolean value.
      - *or* array, >=2 items, <=2 items *of* items : float
        Numeric range [a,b] from a to b inclusive.
- **estimator** (*operator, not for optimizer*) – Nested classifier, fit method must support sample_weight.
- **redact** (*boolean, not for optimizer, default True*) – Whether to redact protected attributes before data preparation (recommended) or not.

fit(*X*, *y=None*, ***fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

- **X** (*array*) – Features; the outer array is over samples.
  - items : array
    - items : union type
      - float
      - *or* string
- **y** (*union type*) – Target class labels; the array is over samples.
  - array *of* items : float
  - *or* array *of* items : string

predict(*X*, ***predict_params*)

Make predictions.

Note: The predict method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (*array*) – Features; the outer array is over samples.

- items : array
  - items : union type
    - float
    - *or* string

Returns

result – Predicted class label per sample.

- array *of* items : float
- *or* array *of* items : string

Return type

union type

lale.lib.aif360.util module

class lale.lib.aif360.util.**FairStratifiedKFold**(**, favorable_labels: List[Union[float, str, bool, List[float]]], protected_attributes: List[Dict[str, Any]], unfavorable_labels: Optional[List[Union[float, str, bool, List[float]]]] = None, n_splits: int = 5, n_repeats: int = 1, shuffle: bool = False, random_state=None*)

Bases: `object`

Stratified k-folds cross-validator by labels and protected attributes.

Behaves similar to the StratifiedKFold and RepeatedStratifiedKFold cross-validation iterators from scikit-learn. This cross-validation object can be passed to the *cv* argument of the auto_configure method.

Parameters

- **favorable_labels** (*array*) – Label values which are considered favorable (i.e. "positive").
- **protected_attributes** (*array*) – Features for which fairness is desired.
- **unfavorable_labels** (*array or None, default None*) – Label values which are considered unfavorable (i.e. "negative").
- **n_splits** (*integer, optional, default 5*) – Number of folds. Must be at least 2.
- **n_repeats** (*integer, optional, default 1*) – Number of times the cross-validator needs to be repeated. When >1, this behaves like RepeatedStratifiedKFold.
- **shuffle** (*boolean, optional, default False*) – Whether to shuffle each class's samples before splitting into batches. Ignored when n_repeats>1.
- **random_state** (*union type, not for optimizer, default None*) – When shuffle is True, random_state affects the ordering of the indices.
  - None
    RandomState used by np.random
  - numpy.random.RandomState
    Use the provided random state, only affecting other users of that same random state instance.
  - integer
    Explicit seed.

get_n_splits(*X=None, y=None, groups=None*) → int

The number of splitting iterations in the cross-validator.

Parameters

- **X** (*Any*) – Always ignored, exists for compatibility.
- **y** (*Any*) – Always ignored, exists for compatibility.
- **groups** (*Any*) – Always ignored, exists for compatibility.

Returns
The number of splits.

Return type
integer

split(*X, y, groups=None*)

Generate indices to split data into training and test set.

X : array **of** items : array **of** items : Any
Training data, including columns with the protected attributes.

y : union type
Target class labels; the array is over samples.

- array **of** items : float
- array **of** items : string

groups : Any
Always ignored, exists for compatibility.

Returns

result –

- train
  The training set indices for that split.
- test
  The testing set indices for that split.

Return type
tuple

lale.lib.aif360.util.**accuracy_and_disparate_impact**(*favorable_labels: List[Union[float, str, bool, List[float]]], protected_attributes: List[Dict[str, Any]], unfavorable_labels: Optional[List[Union[float, str, bool, List[float]]]] = None, fairness_weight: float = 0.5*) → _AccuracyAndDisparateImpact

Create a scikit-learn compatible blended scorer for accuracy and symmetric disparate impact given the fairness info. The scorer is suitable for classification problems, with higher resulting scores indicating better outcomes. The result is a linear combination of accuracy and symmetric disparate impact, and is between 0 and 1. This metric can be used as the *scoring* argument of an optimizer such as Hyperopt, as shown in this demo.

Parameters

- **favorable_labels** (*array of union*) – Label values which are considered favorable (i.e. "positive").
  - string
    Literal value
  - *or* number
    Numerical value
  - *or* array of numbers, >= 2 items, <= 2 items
    Numeric range [a,b] from a to b inclusive.
- **protected_attributes** (*array of dict*) – Features for which fairness is desired.
  - feature : string or integer
    Column name or column index.
  - reference_group : array of union
    Values or ranges that indicate being a member of the privileged group.
    - string
      Literal value
    - *or* number
      Numerical value
    - *or* array of numbers, >= 2 items, <= 2 items
      Numeric range [a,b] from a to b inclusive.
  - monitored_group : union type, default None
    Values or ranges that indicate being a member of the unprivileged group.
    - None
      If *monitored_group* is not explicitly specified, consider any values not captured by *reference_group* as monitored.
    - *or* array of union
      - string
        Literal value
      - *or* number

Numerical value

· *or* array of numbers, >= 2 items, <= 2 items

Numeric range [a,b] from a to b inclusive.

- **unfavorable_labels** (*union type, default None*) – Label values which are considered unfavorable (i.e. "negative").
  - None

    If *unfavorable_labels* is not explicitly specified, consider any labels not captured by *favorable_labels* as unfavorable.
  - *or* array of union
    * string

      Literal value
    * *or* number

      Numerical value
    * *or* array of numbers, >= 2 items, <= 2 items

      Numeric range [a,b] from a to b inclusive.
- **fairness_weight** (*number, >=0, <=1, default=0.5*) – At the default weight of 0.5, the two metrics contribute equally to the blended result. Above 0.5, fairness influences the combination more, and below 0.5, fairness influences the combination less. In the extreme, at 1, the outcome is only determined by fairness, and at 0, the outcome ignores fairness.

Returns

result – Scorer that takes three arguments `(estimator, X, y)` and returns a scalar number. Furthermore, besides being callable, the returned object also has two methods, `score_data(y_true, y_pred, X)` for evaluating datasets and `score_estimator(estimator, X, y)` for evaluating estimators.

Return type

callable

`lale.lib.aif360.util.`**`average_odds_difference`**(*favorable_labels: List[Union[float, str, bool, List[float]]], protected_attributes: List[Dict[str, Any]], unfavorable_labels: Optional[List[Union[float, str, bool, List[float]]]] = None*) → _AverageOddsDifference

Create a scikit-learn compatible average odds difference scorer given the fairness info. Average of difference in false positive rate and true positive rate between unprivileged and privileged groups.

$$\tfrac{1}{2}\left[(\mathrm{FPR}_{D=\text{unprivileged}} - \mathrm{FPR}_{D=\text{privileged}}) + (\mathrm{TPR}_{D=\text{unprivileged}} - \mathrm{TPR}_{D=\text{privileged}})\right]$$

The ideal value of this metric is 0. A value of <0 implies higher benefit for the privileged group and a value >0 implies higher benefit for the unprivileged group. Fairness for this metric is between -0.1 and 0.1.

Parameters

- **favorable_labels** (*array of union*) – Label values which are considered favorable (i.e. "positive").
  - string

    Literal value
  - *or* number

    Numerical value
  - *or* array of numbers, >= 2 items, <= 2 items

    Numeric range [a,b] from a to b inclusive.
- **protected_attributes** (*array of dict*) – Features for which fairness is desired.
  - feature : string or integer

    Column name or column index.
  - reference_group : array of union

    Values or ranges that indicate being a member of the privileged group.
    * string

Literal value

- *or* number

  Numerical value

- *or* array of numbers, >= 2 items, <= 2 items

  Numeric range [a,b] from a to b inclusive.

- monitored_group : union type, default None

  Values or ranges that indicate being a member of the unprivileged group.

  - None

    If *monitored_group* is not explicitly specified, consider any values not captured by *reference_group* as monitored.

  - *or* array of union

    - string

      Literal value

    - *or* number

      Numerical value

    - *or* array of numbers, >= 2 items, <= 2 items

      Numeric range [a,b] from a to b inclusive.

- **unfavorable_labels** (*union type, default None*) – Label values which are considered unfavorable (i.e. "negative").

  - None

    If *unfavorable_labels* is not explicitly specified, consider any labels not captured by *favorable_labels* as unfavorable.

  - *or* array of union

    - string

      Literal value

    - *or* number

      Numerical value

    - *or* array of numbers, >= 2 items, <= 2 items

      Numeric range [a,b] from a to b inclusive.

Returns

result – Scorer that takes three arguments `(estimator, X, y)` and returns a scalar number. Furthermore, besides being callable, the returned object also has two methods, `score_data(y_true, y_pred, X)` for evaluating datasets and `score_estimator(estimator, X, y)` for evaluating estimators.

Return type

callable

`lale.lib.aif360.util.`**`balanced_accuracy_and_disparate_impact`**(*favorable_labels: List[Union[float, str, bool, List[float]]], protected_attributes: List[Dict[str, Any]], unfavorable_labels: Optional[List[Union[float, str, bool, List[float]]]] = None, fairness_weight: float = 0.5*) → _BalancedAccuracyAndDisparateImpact

Create a scikit-learn compatible blended scorer for balanced accuracy and symmetric disparate impact given the fairness info. The scorer is suitable for classification problems, with higher resulting scores indicating better outcomes. The result is a linear combination of accuracy and symmetric disparate impact, and is between 0 and 1. This metric can be used as the *scoring* argument of an optimizer such as Hyperopt, as shown in this demo.

Parameters

- **favorable_labels** (*array of union*) – Label values which are considered favorable (i.e. "positive").

  - string
    Literal value
  - *or* number
    Numerical value
  - *or* array of numbers, >= 2 items, <= 2 items
    Numeric range [a,b] from a to b inclusive.
- **protected_attributes** (*array of dict*) – Features for which fairness is desired.
  - feature : string or integer
    Column name or column index.
  - reference_group : array of union
    Values or ranges that indicate being a member of the privileged group.
    * string
      Literal value
    * *or* number
      Numerical value
    * *or* array of numbers, >= 2 items, <= 2 items
      Numeric range [a,b] from a to b inclusive.
  - monitored_group : union type, default None
    Values or ranges that indicate being a member of the unprivileged group.
    * None
      If *monitored_group* is not explicitly specified, consider any values not captured by *reference_group* as monitored.
    * *or* array of union
      · string
        Literal value
      · *or* number
        Numerical value
      · *or* array of numbers, >= 2 items, <= 2 items
        Numeric range [a,b] from a to b inclusive.
- **unfavorable_labels** (*union type, default None*) – Label values which are considered unfavorable (i.e. "negative").
  - None
    If *unfavorable_labels* is not explicitly specified, consider any labels not captured by *favorable_labels* as unfavorable.
  - *or* array of union
    * string
      Literal value
    * *or* number
      Numerical value
    * *or* array of numbers, >= 2 items, <= 2 items
      Numeric range [a,b] from a to b inclusive.
- **fairness_weight** (*number, >=0, <=1, default=0.5*) – At the default weight of 0.5, the two metrics contribute equally to the blended result. Above 0.5, fairness influences the combination more, and below 0.5, fairness influences the combination less. In the extreme, at 1, the outcome is only determined by fairness, and at 0, the outcome ignores fairness.

Returns

result – Scorer that takes three arguments `(estimator, X, y)` and returns a scalar number. Furthermore, besides being callable, the returned object also has two methods, `score_data(y_true, y_pred, X)` for evaluating datasets and `score_estimator(estimator, X, y)` for evaluating estimators.

Return type
callable

`lale.lib.aif360.util.`**`count_fairness_groups`**(*X: Union[DataFrame, ndarray], y: Union[Series, ndarray], favorable_labels: List[Union[float, str, bool, List[float]]], protected_attributes: List[Dict[str, Any]], unfavorable_labels: Optional[List[Union[float, str, bool, List[float]]]] = None*) → DataFrame

Count size of each intersection of groups induced by the fairness info.

Parameters

- **X** (*array*) – Features including protected attributes as numpy ndarray or pandas dataframe.
- **y** (*array*) – Labels as numpy ndarray or pandas series.
- **favorable_labels** (*array*) – Label values which are considered favorable (i.e. "positive").
- **protected_attributes** (*array*) – Features for which fairness is desired.
- **unfavorable_labels** (*array or None, default None*) – Label values which are considered unfavorable (i.e. "negative").

Returns
result – DataFrame with a multi-level index on the rows, where the first level indicates the binarized outcome, and the remaining levels indicate the binarized group membership according to the protected attributes. Column "count" specifies the number of instances for each group. Column "ratio" gives the ratio of the given outcome relative to the total number of instances with any outcome but the same encoded protected attributes.

Return type
pd.DataFrame

`lale.lib.aif360.util.`**`dataset_to_pandas`**(*dataset, return_only: Literal['X', 'y', 'Xy'] = 'Xy'*) → Tuple[Optional[Series], Optional[Series]]

Return pandas representation of the AIF360 dataset.

Parameters

- **dataset** (*aif360.datasets.BinaryLabelDataset*) – AIF360 dataset to convert to a pandas representation.
- **return_only** (*'Xy', 'X', or 'y'*) – Which part of features X or labels y to convert and return.

Returns

result –

- item 0: pandas Dataframe or None, features X
- item 1: pandas Series or None, labels y

Return type
tuple

`lale.lib.aif360.util.`**`disparate_impact`**(*favorable_labels: List[Union[float, str, bool, List[float]]], protected_attributes: List[Dict[str, Any]], unfavorable_labels: Optional[List[Union[float, str, bool, List[float]]]] = None*) → _DisparateImpact

Create a scikit-learn compatible disparate_impact scorer given the fairness info (Feldman et al. 2015). Ratio of rate of favorable outcome for the unprivileged group to that of the privileged group.

$$\frac{\Pr(Y = \text{favorable} | D = \text{unprivileged})}{\Pr(Y = \text{favorable} | D = \text{privileged})}$$

In the case of multiple protected attributes, *D=privileged* means all protected attributes of the sample have corresponding privileged values in the reference group, and *D=unprivileged* means all protected attributes of the sample have corresponding unprivileged values in the monitored group. The ideal value of this metric is

1. A value <1 implies a higher benefit for the privileged group and a value >1 implies a higher benefit for the unprivileged group. Fairness for this metric is between 0.8 and 1.25.

Parameters

- **favorable_labels** (*array of union*) – Label values which are considered favorable (i.e. "positive").
  - string
    Literal value
  - *or* number
    Numerical value
  - *or* array of numbers, >= 2 items, <= 2 items
    Numeric range [a,b] from a to b inclusive.
- **protected_attributes** (*array of dict*) – Features for which fairness is desired.
  - feature : string or integer
    Column name or column index.
  - reference_group : array of union
    Values or ranges that indicate being a member of the privileged group.
    * string
      Literal value
    * *or* number
      Numerical value
    * *or* array of numbers, >= 2 items, <= 2 items
      Numeric range [a,b] from a to b inclusive.
  - monitored_group : union type, default None
    Values or ranges that indicate being a member of the unprivileged group.
    * None
      If *monitored_group* is not explicitly specified, consider any values not captured by *reference_group* as monitored.
    * *or* array of union
      · string
        Literal value
      · *or* number
        Numerical value
      · *or* array of numbers, >= 2 items, <= 2 items
        Numeric range [a,b] from a to b inclusive.
- **unfavorable_labels** (*union type, default None*) – Label values which are considered unfavorable (i.e. "negative").
  - None
    If *unfavorable_labels* is not explicitly specified, consider any labels not captured by *favorable_labels* as unfavorable.
  - *or* array of union
    * string
      Literal value
    * *or* number
      Numerical value
    * *or* array of numbers, >= 2 items, <= 2 items
      Numeric range [a,b] from a to b inclusive.

Returns

result – Scorer that takes three arguments `(estimator, X, y)` and returns a scalar number. Furthermore, besides being callable, the returned object also has two methods, `score_data(y_true, y_pred, X)` for evaluating datasets and `score_estimator(estimator, X, y)` for evaluating estimators.

Return type
callable

`lale.lib.aif360.util.`**`equal_opportunity_difference`**(*favorable_labels: List[Union[float, str, bool, List[float]]], protected_attributes: List[Dict[str, Any]], unfavorable_labels: Optional[List[Union[float, str, bool, List[float]]]] = None*) → _EqualOpportunityDifference

Create a scikit-learn compatible equal opportunity difference scorer given the fairness info. Difference of true positive rates between the unprivileged and the privileged groups. The true positive rate is the ratio of true positives to the total number of actual positives for a given group.

$$\text{TPR}_{D=\text{unprivileged}} - \text{TPR}_{D=\text{privileged}}$$

The ideal value is 0. A value of <0 implies disparate benefit for the privileged group and a value >0 implies disparate benefit for the unprivileged group. Fairness for this metric is between -0.1 and 0.1.

Parameters

- **favorable_labels** (*array of union*) – Label values which are considered favorable (i.e. "positive").
  - string
    Literal value
  - *or* number
    Numerical value
  - *or* array of numbers, >= 2 items, <= 2 items
    Numeric range [a,b] from a to b inclusive.
- **protected_attributes** (*array of dict*) – Features for which fairness is desired.
  - feature : string or integer
    Column name or column index.
  - reference_group : array of union
    Values or ranges that indicate being a member of the privileged group.
    * string
      Literal value
    * *or* number
      Numerical value
    * *or* array of numbers, >= 2 items, <= 2 items
      Numeric range [a,b] from a to b inclusive.
  - monitored_group : union type, default None
    Values or ranges that indicate being a member of the unprivileged group.
    * None
      If *monitored_group* is not explicitly specified, consider any values not captured by *reference_group* as monitored.
    * *or* array of union
      · string
        Literal value
      · *or* number
        Numerical value
      · *or* array of numbers, >= 2 items, <= 2 items
        Numeric range [a,b] from a to b inclusive.
- **unfavorable_labels** (*union type, default None*) – Label values which are considered unfavorable (i.e. "negative").
  - None

If *unfavorable_labels* is not explicitly specified, consider any labels not captured by *favorable_labels* as unfavorable.

- *or* array of union
  - string
    Literal value
  - *or* number
    Numerical value
  - *or* array of numbers, >= 2 items, <= 2 items
    Numeric range [a,b] from a to b inclusive.

Returns
result – Scorer that takes three arguments `(estimator, X, y)` and returns a scalar number. Furthermore, besides being callable, the returned object also has two methods, `score_data(y_true, y_pred, X)` for evaluating datasets and `score_estimator(estimator, X, y)` for evaluating estimators.

Return type
callable

`lale.lib.aif360.util.`**`f1_and_disparate_impact`**(*favorable_labels: List[Union[float, str, bool, List[float]]], protected_attributes: List[Dict[str, Any]], unfavorable_labels: Optional[List[Union[float, str, bool, List[float]]]] = None, fairness_weight: float = 0.5*) → _F1AndDisparateImpact

Create a scikit-learn compatible blended scorer for f1 and symmetric disparate impact given the fairness info. The scorer is suitable for classification problems, with higher resulting scores indicating better outcomes. The result is a linear combination of F1 and symmetric disparate impact, and is between 0 and 1. This metric can be used as the *scoring* argument of an optimizer such as Hyperopt, as shown in this demo.

Parameters

- **`favorable_labels`** (*`array of union`*) – Label values which are considered favorable (i.e. "positive").
  - string
    Literal value
  - *or* number
    Numerical value
  - *or* array of numbers, >= 2 items, <= 2 items
    Numeric range [a,b] from a to b inclusive.
- **`protected_attributes`** (*`array of dict`*) – Features for which fairness is desired.
  - feature : string or integer
    Column name or column index.
  - reference_group : array of union
    Values or ranges that indicate being a member of the privileged group.
    - string
      Literal value
    - *or* number
      Numerical value
    - *or* array of numbers, >= 2 items, <= 2 items
      Numeric range [a,b] from a to b inclusive.
  - monitored_group : union type, default None
    Values or ranges that indicate being a member of the unprivileged group.
    - None
      If *monitored_group* is not explicitly specified, consider any values not captured by *reference_group* as monitored.

        * *or* array of union
            · string
                Literal value
            · *or* number
                Numerical value
            · *or* array of numbers, >= 2 items, <= 2 items
                Numeric range [a,b] from a to b inclusive.

- **unfavorable_labels** (*union type, default None*) – Label values which are considered unfavorable (i.e. "negative").
    - None
        If *unfavorable_labels* is not explicitly specified, consider any labels not captured by *favorable_labels* as unfavorable.
    - *or* array of union
        * string
            Literal value
        * *or* number
            Numerical value
        * *or* array of numbers, >= 2 items, <= 2 items
            Numeric range [a,b] from a to b inclusive.
- **fairness_weight** (*number, >=0, <=1, default=0.5*) – At the default weight of 0.5, the two metrics contribute equally to the blended result. Above 0.5, fairness influences the combination more, and below 0.5, fairness influences the combination less. In the extreme, at 1, the outcome is only determined by fairness, and at 0, the outcome ignores fairness.

Returns
    result – Scorer that takes three arguments `(estimator, X, y)` and returns a scalar number. Furthermore, besides being callable, the returned object also has two methods, `score_data(y_true, y_pred, X)` for evaluating datasets and `score_estimator(estimator, X, y)` for evaluating estimators.

Return type
    callable

`lale.lib.aif360.util.`**`fair_stratified_train_test_split`**(*X*, *y*, **arrays*, *favorable_labels: List[Union[float, str, bool, List[float]]]*, *protected_attributes: List[Dict[str, Any]]*, *unfavorable_labels: Optional[List[Union[float, str, bool, List[float]]]] = None*, *test_size: float = 0.25*, *random_state: Optional[Union[RandomState, int]] = None*) → Tuple

Splits X and y into random train and test subsets stratified by labels and protected attributes.

Behaves similar to the train_test_split function from scikit-learn.

Parameters
- **X** (*array*) – Features including protected attributes as numpy ndarray or pandas dataframe.
- **y** (*array*) – Labels as numpy ndarray or pandas series.
- ***arrays** (*array*) – Sequence of additional arrays with same length as X and y.
- **favorable_labels** (*array*) – Label values which are considered favorable (i.e. "positive").
- **protected_attributes** (*array*) – Features for which fairness is desired.
- **unfavorable_labels** (*array or None, default None*) – Label values which are considered unfavorable (i.e. "negative").
- **test_size** (*float or int, default=0.25*) – If float, should be between 0.0 and 1.0 and represent the proportion of the dataset to include in the test split. If int, repre-

sents the absolute number of test samples.

- **random_state** (*int, RandomState instance or None, default=None*) – Controls the shuffling applied to the data before applying the split. Pass an integer for reproducible output across multiple function calls.
  - None
    RandomState used by numpy.random
  - numpy.random.RandomState
    Use the provided random state, only affecting other users of that same random state instance.
  - integer
    Explicit seed.

Returns

result –
- item 0: train_X
- item 1: test_X
- item 2: train_y
- item 3: test_y
- item 4+: Each argument in **arrays*, if any, yields two items in the result, for the two splits of that array.

Return type
tuple

lale.lib.aif360.util.**r2_and_disparate_impact**(*favorable_labels: List[Union[float, str, bool, List[float]]], protected_attributes: List[Dict[str, Any]], unfavorable_labels: Optional[List[Union[float, str, bool, List[float]]]] = None, fairness_weight: float = 0.5*) → _R2AndDisparateImpact

Create a scikit-learn compatible blended scorer for R2 score and symmetric disparate impact given the fairness info. The scorer is suitable for regression problems, with higher resulting scores indicating better outcomes. It first scales R2, which might be negative, to be between 0 and 1. Then, the result is a linear combination of the scaled R2 and symmetric disparate impact, and is also between 0 and 1. This metric can be used as the *scoring* argument of an optimizer such as Hyperopt.

Parameters
- **favorable_labels** (*array of union*) – Label values which are considered favorable (i.e. “positive”).
  - string
    Literal value
  - *or* number
    Numerical value
  - *or* array of numbers, >= 2 items, <= 2 items
    Numeric range [a,b] from a to b inclusive.
- **protected_attributes** (*array of dict*) – Features for which fairness is desired.
  - feature : string or integer
    Column name or column index.
  - reference_group : array of union
    Values or ranges that indicate being a member of the privileged group.
    - string
      Literal value
    - *or* number
      Numerical value
    - *or* array of numbers, >= 2 items, <= 2 items
      Numeric range [a,b] from a to b inclusive.
  - monitored_group : union type, default None

Values or ranges that indicate being a member of the unprivileged group.

        * None

            If *monitored_group* is not explicitly specified, consider any values not captured by *reference_group* as monitored.

        * *or* array of union

            · string

                Literal value

            · *or* number

                Numerical value

            · *or* array of numbers, >= 2 items, <= 2 items

                Numeric range [a,b] from a to b inclusive.

- **unfavorable_labels** (*union type, default None*) – Label values which are considered unfavorable (i.e. "negative").
    - None

        If *unfavorable_labels* is not explicitly specified, consider any labels not captured by *favorable_labels* as unfavorable.
    - *or* array of union
        * string

            Literal value
        * *or* number

            Numerical value
        * *or* array of numbers, >= 2 items, <= 2 items

            Numeric range [a,b] from a to b inclusive.
- **fairness_weight** (*number, >=0, <=1, default=0.5*) – At the default weight of 0.5, the two metrics contribute equally to the blended result. Above 0.5, fairness influences the combination more, and below 0.5, fairness influences the combination less. In the extreme, at 1, the outcome is only determined by fairness, and at 0, the outcome ignores fairness.

Returns

result – Scorer that takes three arguments `(estimator, X, y)` and returns a scalar number. Furthermore, besides being callable, the returned object also has two methods, `score_data(y_true, y_pred, X)` for evaluating datasets and `score_estimator(estimator, X, y)` for evaluating estimators.

Return type

callable

`lale.lib.aif360.util.`**`statistical_parity_difference`**(*favorable_labels: List[Union[float, str, bool, List[float]]], protected_attributes: List[Dict[str, Any]], unfavorable_labels: Optional[List[Union[float, str, bool, List[float]]]] = None*) → _StatisticalParityDifference

Create a scikit-learn compatible statistical parity difference scorer given the fairness info. Difference of the rate of favorable outcomes received by the unprivileged group to the privileged group.

$$\Pr(Y = \text{favorable} | D = \text{unprivileged}) - \Pr(Y = \text{favorable} | D = \text{privileged})$$

The ideal value of this metric is 0. A value of <0 implies higher benefit for the privileged group and a value >0 implies higher benefit for the unprivileged group. Fairness for this metric is between -0.1 and 0.1. For a discussion of potential issues with this metric see (Dwork et al. 2012).

Parameters

- **favorable_labels** (*array of union*) – Label values which are considered favorable (i.e. "positive").
    - string

Literal value
- *or* number
  Numerical value
- *or* array of numbers, >= 2 items, <= 2 items
  Numeric range [a,b] from a to b inclusive.

• `protected_attributes` (*array of dict*) – Features for which fairness is desired.
  - feature : string or integer
    Column name or column index.
  - reference_group : array of union
    Values or ranges that indicate being a member of the privileged group.
    * string
      Literal value
    * *or* number
      Numerical value
    * *or* array of numbers, >= 2 items, <= 2 items
      Numeric range [a,b] from a to b inclusive.
  - monitored_group : union type, default None
    Values or ranges that indicate being a member of the unprivileged group.
    * None
      If *monitored_group* is not explicitly specified, consider any values not captured by *reference_group* as monitored.
    * *or* array of union
      · string
        Literal value
      · *or* number
        Numerical value
      · *or* array of numbers, >= 2 items, <= 2 items
        Numeric range [a,b] from a to b inclusive.

• `unfavorable_labels` (*union type, default None*) – Label values which are considered unfavorable (i.e. "negative").
  - None
    If *unfavorable_labels* is not explicitly specified, consider any labels not captured by *favorable_labels* as unfavorable.
  - *or* array of union
    * string
      Literal value
    * *or* number
      Numerical value
    * *or* array of numbers, >= 2 items, <= 2 items
      Numeric range [a,b] from a to b inclusive.

Returns
result – Scorer that takes three arguments `(estimator, X, y)` and returns a scalar number. Furthermore, besides being callable, the returned object also has two methods, `score_data(y_true, y_pred, X)` for evaluating datasets and `score_estimator(estimator, X, y)` for evaluating estimators.

Return type
callable

`lale.lib.aif360.util.symmetric_disparate_impact`(*favorable_labels: List[Union[float, str, bool, List[float]]], protected_attributes: List[Dict[str, Any]], unfavorable_labels: Optional[List[Union[float, str, bool, List[float]]]] = None*) → _SymmetricDisparateImpact

Create a scikit-learn compatible scorer for symmetric disparate impact given the fairness info. For disparate impact <= 1.0, return that value, otherwise return its inverse. The result is between 0 and 1. The higher this metric, the better, and the ideal value is 1. A value <1 implies that either the privileged group or the unprivileged group is receiving a disparate benefit.

Parameters

- **favorable_labels** (*array of union*) – Label values which are considered favorable (i.e. "positive").
  - string
    Literal value
  - *or* number
    Numerical value
  - *or* array of numbers, >= 2 items, <= 2 items
    Numeric range [a,b] from a to b inclusive.
- **protected_attributes** (*array of dict*) – Features for which fairness is desired.
  - feature : string or integer
    Column name or column index.
  - reference_group : array of union
    Values or ranges that indicate being a member of the privileged group.
    - string
      Literal value
    - *or* number
      Numerical value
    - *or* array of numbers, >= 2 items, <= 2 items
      Numeric range [a,b] from a to b inclusive.
  - monitored_group : union type, default None
    Values or ranges that indicate being a member of the unprivileged group.
    - None
      If *monitored_group* is not explicitly specified, consider any values not captured by *reference_group* as monitored.
    - *or* array of union
      - string
        Literal value
      - *or* number
        Numerical value
      - *or* array of numbers, >= 2 items, <= 2 items
        Numeric range [a,b] from a to b inclusive.
- **unfavorable_labels** (*union type, default None*) – Label values which are considered unfavorable (i.e. "negative").
  - None
    If *unfavorable_labels* is not explicitly specified, consider any labels not captured by *favorable_labels* as unfavorable.
  - *or* array of union
    - string
      Literal value
    - *or* number
      Numerical value

      * *or* array of numbers, >= 2 items, <= 2 items
        Numeric range [a,b] from a to b inclusive.

Returns
  result – Scorer that takes three arguments `(estimator, X, y)` and returns a scalar number. Furthermore, besides being callable, the returned object also has two methods, `score_data(y_true, y_pred, X)` for evaluating datasets and `score_estimator(estimator, X, y)` for evaluating estimators.

Return type
  callable

`lale.lib.aif360.util.`**`theil_index`**(*favorable_labels: List[Union[float, str, bool, List[float]]], protected_attributes: List[Dict[str, Any]], unfavorable_labels: Optional[List[Union[float, str, bool, List[float]]]] = None*) → _AIF360ScorerFactory

Create a scikit-learn compatible Theil index scorer given the fairness info (Speicher et al. 2018). Generalized entropy of benefit for all individuals in the dataset, with alpha=1. Measures the inequality in benefit allocation for individuals. With $b_i = \hat{y}_i - y_i + 1$:

$$\mathcal{E}(\alpha) = \begin{cases} \frac{1}{n\alpha(\alpha-1)} \sum_{i=1}^{n} \left[\left(\frac{b_i}{\mu}\right)^{\alpha} - 1\right], & \alpha \neq 0, 1, \\ \frac{1}{n} \sum_{i=1}^{n} \frac{b_i}{\mu} \ln \frac{b_i}{\mu}, & \alpha = 1, \\ -\frac{1}{n} \sum_{i=1}^{n} \ln \frac{b_i}{\mu}, & \alpha = 0. \end{cases}$$

A value of 0 implies perfect fairness. Fairness is indicated by lower scores, higher scores are problematic.

Parameters
- **`favorable_labels`** (*array of union*) – Label values which are considered favorable (i.e. "positive").
  - string
    Literal value
  - *or* number
    Numerical value
  - *or* array of numbers, >= 2 items, <= 2 items
    Numeric range [a,b] from a to b inclusive.
- **`protected_attributes`** (*array of dict*) – Features for which fairness is desired.
  - feature : string or integer
    Column name or column index.
  - reference_group : array of union
    Values or ranges that indicate being a member of the privileged group.
    * string
      Literal value
    * *or* number
      Numerical value
    * *or* array of numbers, >= 2 items, <= 2 items
      Numeric range [a,b] from a to b inclusive.
  - monitored_group : union type, default None
    Values or ranges that indicate being a member of the unprivileged group.
    * None
      If *monitored_group* is not explicitly specified, consider any values not captured by *reference_group* as monitored.
    * *or* array of union
      · string
        Literal value

        · *or* number
            Numerical value
        · *or* array of numbers, >= 2 items, <= 2 items
            Numeric range [a,b] from a to b inclusive.

- **unfavorable_labels** (*union type, default None*) – Label values which are considered unfavorable (i.e. "negative").
  - None
    If *unfavorable_labels* is not explicitly specified, consider any labels not captured by *favorable_labels* as unfavorable.
  - *or* array of union
    - string
      Literal value
    - *or* number
      Numerical value
    - *or* array of numbers, >= 2 items, <= 2 items
      Numeric range [a,b] from a to b inclusive.

Returns
result – Scorer that takes three arguments `(estimator, X, y)` and returns a scalar number. Furthermore, besides being callable, the returned object also has two methods, `score_data(y_true, y_pred, X)` for evaluating datasets and `score_estimator(estimator, X, y)` for evaluating estimators.

Return type
callable

Module contents

Scikit-learn compatible wrappers for several operators and metrics from AIF360 along with schemas to enable hyper-parameter tuning, as well as functions for fetching fairness dataset.

All operators and metrics in the Lale wrappers for AIF360 take two arguments, *favorable_labels* and *protected_attributes*, collectively referred to as *fairness info*. For example, the following code indicates that the reference group comprises male values in the *personal_status* attribute as well as values from 26 to 1000 in the *age* attribute.

```
creditg_fairness_info = {
    "favorable_labels": ["good"],
    "protected_attributes": [
        {
            "feature": "personal_status",
            "reference_group": [
                "male div/sep", "male mar/wid", "male single",
            ],
        },
        {"feature": "age", "reference_group": [[26, 1000]]},
    ],
}
```

See the following notebooks for more detailed examples:

- https://github.com/IBM/lale/blob/master/examples/demo_aif360.ipynb
- https://github.com/IBM/watson-machine-learning-samples/blob/master/cloud/notebooks/python_sdk/experiments/autoai/Use%20Lale%20AIF360%20scorers%20to%20calculate%20and%20mitigate%20bias%20for%20credit%20risk%20AutoAI%20model.ipynb

Pre-Estimator Mitigation Operators:

- DisparateImpactRemover
- LFR
- Orbis
- Reweighing

In-Estimator Mitigation Operators:

- AdversarialDebiasing
- BaggingOrbisClassifier
- GerryFairClassifier
- MetaFairClassifier
- PrejudiceRemover

Post-Estimator Mitigation Operators:

- CalibratedEqOddsPostprocessing
- EqOddsPostprocessing
- RejectOptionClassification

Datasets:

datasets module docstring

- fetch_adult_df
- fetch_bank_df
- fetch_compas_df
- fetch_compas_violent_df
- fetch_creditg_df
- fetch_default_credit_df
- fetch_heart_disease_df
- fetch_law_school_df
- fetch_meps_panel19_fy2015_df
- fetch_meps_panel20_fy2015_df
- fetch_meps_panel21_fy2016_df
- fetch_nlsy_df
- fetch_nursery_df
- fetch_ricci_df
- fetch_speeddating_df

- fetch_student_math_df
- fetch_student_por_df
- fetch_tae_df
- fetch_titanic_df
- fetch_us_crime_df

Metrics:

- accuracy_and_disparate_impact
- balanced_accuracy_and_disparate_impact
- average_odds_difference
- disparate_impact
- equal_opportunity_difference
- f1_and_disparate_impact
- r2_and_disparate_impact
- statistical_parity_difference
- symmetric_disparate_impact
- theil_index

Other Classes and Operators:

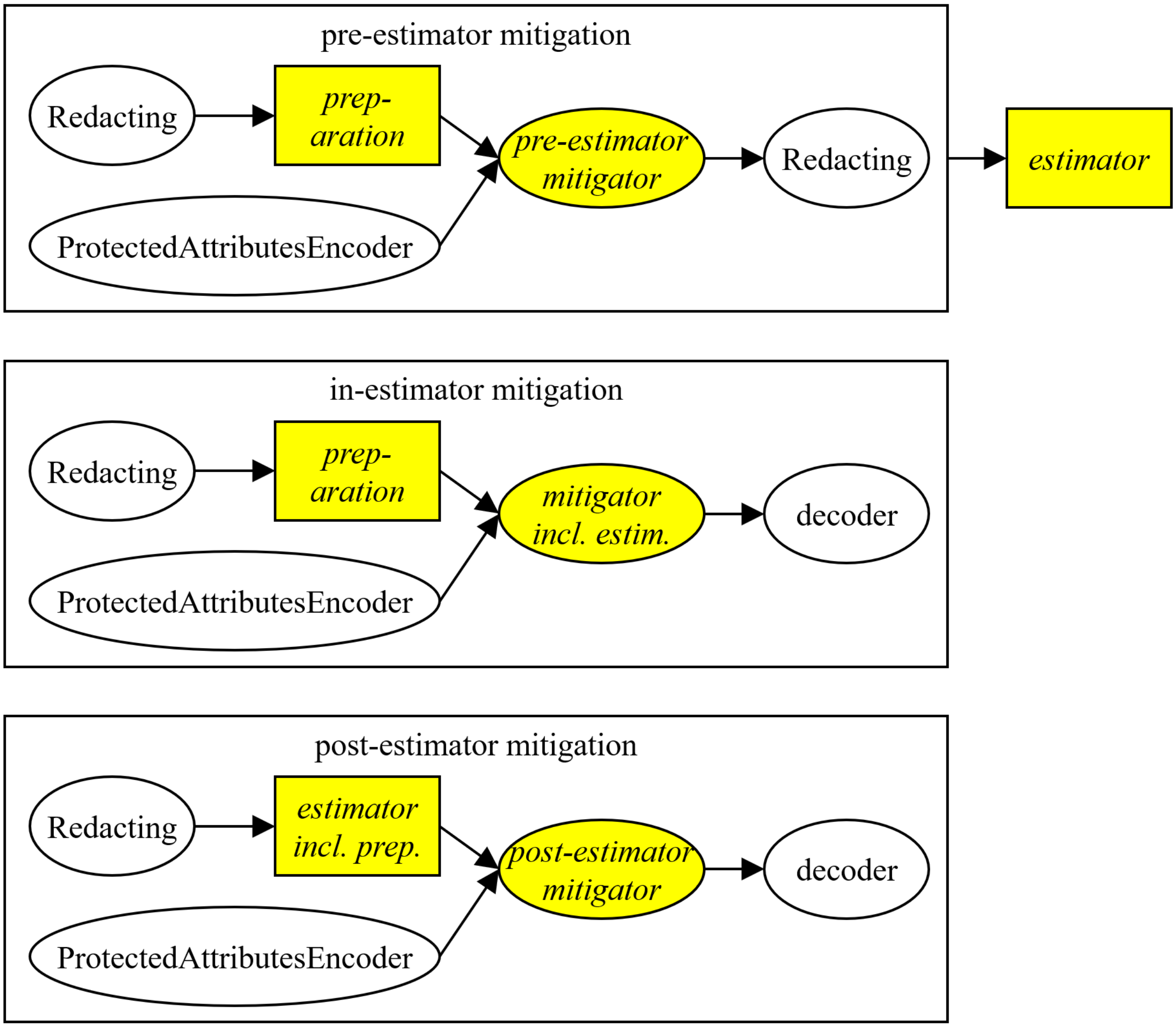
- FairStratifiedKFold
- ProtectedAttributesEncoder
- Redacting

Other Functions:

- count_fairness_groups
- dataset_to_pandas
- fair_stratified_train_test_split

Mitigator Patterns:

AIF360 provides three kinds of fairness mitigators, illustrated in the following picture. *Pre-estimator* mitigators transform the data before it gets to an estimator; *in-estimator* mitigators include their own estimator; and *post-estimator* mitigators transform predictions after those come back from an estimator.



In the picture, italics indicate parameters of the pattern. For example, consider the following code:

```
pipeline = LFR(
    **fairness_info,
    preparation=(
        (Project(columns={"type": "string"}) >> OneHotEncoder(handle_unknown="ignore"))
        & Project(columns={"type": "number"})
    )
    >> ConcatFeatures
) >> LogisticRegression(max_iter=1000)
```

In this example, the *mitigator* is LFR (which is pre-estimator), the *estimator* is LogisticRegression, and the *preparation* is a sub-pipeline that one-hot-encodes strings. If all features of the data are numerical, then the preparation can be omitted. Internally, the LFR higher-order operator uses two auxiliary operators, Redacting and ProtectedAttributesEncoder. Redacting sets protected attributes to a constant to prevent them from directly influencing fairness-agnostic data preparation or estimators. And the ProtectedAttributesEncoder encodes protected attributes and labels as zero or one to simplify the task for the mitigator.

lale.lib.autogen package

Submodules

lale.lib.autogen.additive_chi2_sampler module

class lale.lib.autogen.additive_chi2_sampler.**AdditiveChi2Sampler**(*, *sample_steps=2*, *sample_interval=None*)

Bases: *PlannedIndividualOp*

Combined schema for expected data and hyperparameters.

This documentation is auto-generated from JSON schemas.

Parameters

- **sample_steps** (*integer, >=1 for optimizer, <=5 for optimizer, uniform distribution, default 2*) – Gives the number of (complex) sampling points.

  See also *constraint-1*.
- **sample_interval** (*union type, default None*) – Sampling interval
  - float, >=0.1 for optimizer, <=1.0 for optimizer, uniform distribution
  - *or* None

  See also *constraint-1*.

Notes

constraint-1 : union type

From /kernel_approximation.py:AdditiveChi2Sampler:fit, Exception: raise ValueError('If sample_steps is not in [1, 2, 3], you need to provide sample_interval')

- sample_interval : negated type *of* None
- *or* sample_steps : 1, 2, *or* 3

fit(*X*, *y=None*, ***fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

X (array *of* items : array *of* items : float) – Training data, where n_samples in the number of samples and n_features is the number of features.

transform(*X*, *y=None*)

Transform the data.

Note: The transform method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (array *of* items : array *of* items : float) –

Returns

result – Whether the return value is an array of sparse matrix depends on the type of the input X.

Return type

Any

lale.lib.autogen.ard_regression module

class lale.lib.autogen.ard_regression.**ARDRegression**(**, n_iter='deprecated', tol=0.001, alpha_1=1e-06, alpha_2=1e-06, lambda_1=1e-06, lambda_2=1e-06, compute_score=False, threshold_lambda=10000.0, fit_intercept=True, copy_X=True, verbose=False, max_iter=None*)

Bases: `PlannedIndividualOp`

Combined schema for expected data and hyperparameters.

This documentation is auto-generated from JSON schemas.

Parameters

- **n_iter** (*union type, default 'deprecated'*) – Deprecated. Use *max_iter* instead.
  - integer, >=5 for optimizer, <=1000 for optimizer, uniform distribution, default 300

    Maximum number of iterations
  - *or* 'deprecated'
- **tol** (*float, >=1e-08 for optimizer, <=0.01 for optimizer, default 0.001*) – Stop the algorithm if w has converged
- **alpha_1** (*float, not for optimizer, default 1e-06*) – Hyper-parameter : shape parameter for the Gamma distribution prior over the alpha parameter
- **alpha_2** (*float, not for optimizer, default 1e-06*) – Hyper-parameter : inverse scale parameter (rate parameter) for the Gamma distribution prior over the alpha parameter
- **lambda_1** (*float, not for optimizer, default 1e-06*) – Hyper-parameter : shape parameter for the Gamma distribution prior over the lambda parameter
- **lambda_2** (*float, not for optimizer, default 1e-06*) – Hyper-parameter : inverse scale parameter (rate parameter) for the Gamma distribution prior over the lambda parameter
- **compute_score** (*boolean, default False*) – If True, compute the objective function at each step of the model
- **threshold_lambda** (*float, not for optimizer, default 10000.0*) – threshold for removing (pruning) weights with high precision from the computation
- **fit_intercept** (*boolean, default True*) – whether to calculate the intercept for this model
- **copy_X** (*boolean, default True*) – If True, X will be copied; else, it may be overwritten.
- **verbose** (*boolean, not for optimizer, default False*) – Verbose mode when fitting the model.
- **max_iter** (*union type, optional, not for optimizer, default None*) – Maximum number of iterations
  - integer, >=5 for optimizer, <=1000 for optimizer, uniform distribution
  - *or* None

    Corresponds to 300

fit(*X, y=None, **fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

- **X** (array *of* items : array *of* items : float) – Training vector, where n_samples in the number of samples and n_features is the number of features.
- **y** (array *of* items : float) – Target values (integers)

predict(*X*, ***predict_params*)

Make predictions.

Note: The predict method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

- **X** (array *of* items : array *of* items : float) – Samples.
- **return_std** (*union type, optional, default None*) – Whether to return the standard deviation of posterior prediction.
  - boolean
  - *or* None

Returns

result – Predict using the linear model.

Return type

Any

lale.lib.autogen.bayesian_ridge module

class lale.lib.autogen.bayesian_ridge.**BayesianRidge**(**, n_iter='deprecated', tol=0.001, alpha_1=1e-06, alpha_2=1e-06, lambda_1=1e-06, lambda_2=1e-06, compute_score=False, fit_intercept=True, copy_X=True, verbose=False, max_iter=None*)

Bases: *PlannedIndividualOp*

Combined schema for expected data and hyperparameters.

This documentation is auto-generated from JSON schemas.

Parameters

- **n_iter** (*union type, default 'deprecated'*) – Deprecated. Use *max_iter* instead.
  - integer, >=5 for optimizer, <=1000 for optimizer, uniform distribution, default 300

    Maximum number of iterations
  - *or* 'deprecated'
- **tol** (*float, >=1e-08 for optimizer, <=0.01 for optimizer, default 0.001*) – Stop the algorithm if w has converged
- **alpha_1** (*float, not for optimizer, default 1e-06*) – Hyper-parameter : shape parameter for the Gamma distribution prior over the alpha parameter
- **alpha_2** (*float, not for optimizer, default 1e-06*) – Hyper-parameter : inverse scale parameter (rate parameter) for the Gamma distribution prior over the alpha parameter
- **lambda_1** (*float, not for optimizer, default 1e-06*) – Hyper-parameter : shape parameter for the Gamma distribution prior over the lambda parameter
- **lambda_2** (*float, not for optimizer, default 1e-06*) – Hyper-parameter : inverse scale parameter (rate parameter) for the Gamma distribution prior over the lambda parameter
- **compute_score** (*boolean, default False*) – If True, compute the objective function at each step of the model

- **fit_intercept** (*boolean, default True*) – whether to calculate the intercept for this model
- **copy_X** (*boolean, default True*) – If True, X will be copied; else, it may be overwritten.
- **verbose** (*boolean, not for optimizer, default False*) – Verbose mode when fitting the model.
- **max_iter** (*union type, optional, not for optimizer, default None*) – Maximum number of iterations
  - integer, >=5 for optimizer, <=1000 for optimizer, uniform distribution
  - *or* None

    Corresponds to 300

fit(*X*, *y=None*, ***fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

- **X** (array *of* items : array *of* items : float) – Training data
- **y** (array *of* items : float) – Target values
- **sample_weight** (array, optional *of* items : float) – Individual weights for each sample

predict(*X*, ***predict_params*)

Make predictions.

Note: The predict method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

- **X** (array *of* items : array *of* items : float) – Samples.
- **return_std** (*union type, optional, default None*) – Whether to return the standard deviation of posterior prediction.
  - boolean
  - *or* None

Returns

result – Predict using the linear model.

Return type

Any

lale.lib.autogen.bernoulli_nb module

class lale.lib.autogen.bernoulli_nb.**BernoulliNB**(**, alpha=1.0, binarize=0.0, fit_prior=True, class_prior=None*)

Bases: *PlannedIndividualOp*

Combined schema for expected data and hyperparameters.

This documentation is auto-generated from JSON schemas.

Parameters

- **alpha** (*float, >=1e-10 for optimizer, <=1.0 for optimizer, loguniform distribution, default 1.0*) – Additive (Laplace/Lidstone) smoothing parameter (0 for no smoothing).
- **binarize** (*union type, default 0.0*) – Threshold for binarizing (mapping to booleans) of sample features
  - float, >=-1.0 for optimizer, <=1.0 for optimizer

    - *or* None

  See also *constraint-1*, *constraint-1*.
- **fit_prior** (*boolean, default True*) – Whether to learn class prior probabilities or not
- **class_prior** (*union type, not for optimizer, default None*) – Prior probabilities of the classes
    - array *of* items : float
    - *or* None

Notes

constraint-1 : union type

Cannot binarize a sparse matrix with threshold < 0

- binarize : None
- *or* negated type *of* 'X/isSparse'
- *or* binarize : float, >=0

fit(*X*, *y=None*, ***fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

- **X** (array *of* items : array *of* items : float) – Training vectors, where n_samples is the number of samples and n_features is the number of features.
- **y** (array *of* items : float) – Target values.
- **sample_weight** (*union type, optional, default None*) – Weights applied to individual samples (1
    - array *of* items : float
    - *or* None

predict(*X*, ***predict_params*)

Make predictions.

Note: The predict method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (array *of* items : array *of* items : float) –

Returns

result – Predicted target values for X

Return type

array *of* items : float

predict_proba(*X*)

Probability estimates for all classes.

Note: The predict_proba method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (array *of* items : array *of* items : float) –

Returns

result – Returns the probability of the samples for each class in the model

Return type

array *of* items : array *of* items : float

lale.lib.autogen.bernoulli_rbm module

class lale.lib.autogen.bernoulli_rbm.**BernoulliRBM**(**, n_components=256, learning_rate=0.1, batch_size=10, n_iter=10, verbose=0, random_state=33*)

Bases: *PlannedIndividualOp*

Combined schema for expected data and hyperparameters.

This documentation is auto-generated from JSON schemas.

Parameters

- **n_components** (*integer, >=2 for optimizer, <=256 for optimizer, uniform distribution, default 256*) – Number of binary hidden units.
- **learning_rate** (*float, not for optimizer, default 0.1*) – The learning rate for weight updates
- **batch_size** (*integer, >=3 for optimizer, <=128 for optimizer, uniform distribution, default 10*) – Number of examples per minibatch.
- **n_iter** (*integer, >=5 for optimizer, <=1000 for optimizer, uniform distribution, default 10*) – Number of iterations/sweeps over the training dataset to perform during training.
- **verbose** (*integer, not for optimizer, default 0*) – The verbosity level
- **random_state** (*union type, not for optimizer, default 33*) – A random number generator instance to define the state of the random permutations generator
  - integer
  - *or* numpy.random.RandomState

fit(*X, y=None, **fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

X (array *of* items : array *of* items : float) – Training data.

transform(*X, y=None*)

Transform the data.

Note: The transform method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (array *of* items : array *of* items : float) – The data to be transformed.

Returns

result – Latent representations of the data.

Return type

array *of* items : array *of* items : float

lale.lib.autogen.binarizer module

class `lale.lib.autogen.binarizer.Binarizer`(*, *threshold=0.0*, *copy=True*)

Bases: `PlannedIndividualOp`

Combined schema for expected data and hyperparameters.

This documentation is auto-generated from JSON schemas.

Parameters

- **threshold** (*float, not for optimizer, default 0.0*) – Feature values below or equal to this are replaced by 0, above it by 1
- **copy** (*boolean, default True*) – set to False to perform inplace binarization and avoid a copy (if the input is already a numpy array or a scipy.sparse CSR matrix).

fit(*X*, *y=None*, ***fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

X (array *of* items : Any) –

transform(*X*, *y=None*)

Transform the data.

Note: The transform method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

- **X** (array *of* items : array *of* items : float) – The data to binarize, element by element
- **y** (*Any, optional*) –
- **copy** (*boolean, optional*) – Copy the input X or not.

Returns

result – Binarize each element of X

Return type

Any

lale.lib.autogen.birch module

class `lale.lib.autogen.birch.Birch`(*, *threshold=0.5*, *branching_factor=50*, *n_clusters=3*, *compute_labels=True*, *copy=True*)

Bases: `PlannedIndividualOp`

Combined schema for expected data and hyperparameters.

This documentation is auto-generated from JSON schemas.

Parameters

- **threshold** (*float, not for optimizer, default 0.5*) – The radius of the subcluster obtained by merging a new sample and the closest subcluster should be lesser than the threshold
- **branching_factor** (*integer, >=50 for optimizer, <=51 for optimizer, uniform distribution, default 50*) – Maximum number of CF subclusters in each node

- **n_clusters** (*integer, >=2 for optimizer, <=8 for optimizer, uniform distribution, default 3*) – Number of clusters after the final clustering step, which treats the subclusters from the leaves as new samples
- **compute_labels** (*boolean, default True*) – Whether or not to compute labels for each fit.
- **copy** (*boolean, default True*) – Whether or not to make a copy of the given data

fit(*X*, *y=None*, ***fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

- **X** (array *of* items : array *of* items : float) – Input data.
- **y** (*any type*) –

predict(*X*, ***predict_params*)

Make predictions.

Note: The predict method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters
X (array *of* items : array *of* items : float) – Input data.

Returns
result – Labelled data.

Return type
Any

transform(*X*, *y=None*)

Transform the data.

Note: The transform method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters
X (array *of* items : array *of* items : float) – Input data.

Returns
result – Transformed data.

Return type
array *of* items : array *of* items : float

lale.lib.autogen.calibrated_classifier_cv module

class lale.lib.autogen.calibrated_classifier_cv.**CalibratedClassifierCV**(***, *method='sigmoid'*, *cv=None*, *n_jobs=None*, *ensemble=True*, *estimator=None*)

Bases: *PlannedIndividualOp*

Combined schema for expected data and hyperparameters.

This documentation is auto-generated from JSON schemas.

Parameters

- **method** ('sigmoid' *or* 'isotonic', default 'sigmoid') – The method to use for calibration
- **cv** (*union type, default None*) –

Cross-validation as integer or as object that has a split function.
The fit method performs cross validation on the input dataset for per trial, and uses the mean cross validation performance for optimization. This behavior is also impacted by handle_cv_failure flag. If integer: number of folds in sklearn.model_selection.StratifiedKFold. If object with split function: generator yielding (train, test) splits as arrays of indices. Can use any of the iterators from https://scikit-learn.org/stable/modules/cross_validation.html#cross-validation-iterators.

  - integer, >=1, >=3 for optimizer, <=4 for optimizer, uniform distribution, default 5
  - *or* Any, not for optimizer
  - *or* None
  - *or* 'prefit'

- **n_jobs** (*union type, optional, not for optimizer, default None*) – Number of jobs to run in parallel.
  - None
    1 unless in joblib.parallel_backend context.
  - *or* -1
    Use all processors.
  - *or* integer, >=1
    Number of jobs to run in parallel.
- **ensemble** (*boolean, optional, not for optimizer, default True*) – Determines how the calibrator is fitted when cv is not 'prefit'. Ignored if cv='prefit
- **estimator** (*union type, optional, not for optimizer, default None*) – The base estimator to fit on random subsets of the dataset.
  - operator
  - *or* None
    LinearSVC

fit(*X*, *y=None*, ***fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

- **X** (array *of* items : array *of* items : float) – Training data.
- **y** (array *of* items : float) – Target values.
- **sample_weight** (*union type, optional*) – Sample weights
  - array *of* items : float
  - *or* None

predict(*X*, ***predict_params*)

Make predictions.

Note: The predict method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters
X (array *of* items : array *of* items : float) – The samples.

Returns
result – The predicted class.

Return type
array *of* items : float

predict_proba(*X*)

Probability estimates for all classes.

Note: The predict_proba method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (array *of* items : array *of* items : float) – The samples.

Returns

result – The predicted probas.

Return type

array *of* items : array *of* items : float

lale.lib.autogen.cca module

class lale.lib.autogen.cca.**CCA**(*, *n_components=2*, *scale=True*, *max_iter=500*, *tol=1e-06*, *copy=True*)

Bases: *PlannedIndividualOp*

Combined schema for expected data and hyperparameters.

This documentation is auto-generated from JSON schemas.

Parameters

- **n_components** (*integer, >=2 for optimizer, <=256 for optimizer, uniform distribution, default 2*) – number of components to keep.
- **scale** (*boolean, default True*) – whether to scale the data?
- **max_iter** (*integer, >=10 for optimizer, <=1000 for optimizer, uniform distribution, default 500*) – the maximum number of iterations of the NIPALS inner loop
- **tol** (*float, >=1e-08 for optimizer, <=0.01 for optimizer, default 1e-06*) – the tolerance used in the iterative algorithm
- **copy** (*boolean, default True*) – Whether the deflation be done on a copy

fit(*X*, *y=None*, ***fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

- **X** (array *of* items : array *of* items : float) – Training vectors, where n_samples is the number of samples and n_features is the number of predictors.
- **Y** (array, optional *of* items : array *of* items : float) – Target vectors, where n_samples is the number of samples and n_targets is the number of response variables.

predict(*X*, ***predict_params*)

Make predictions.

Note: The predict method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

- **X** (array *of* items : array *of* items : float) – Training vectors, where n_samples is the number of samples and n_features is the number of predictors.
- **copy** (*boolean, optional, default True*) – Whether to copy X and Y, or perform in-place normalization.

Returns

result – Apply the dimension reduction learned on the train data.

Return type

Any

transform(*X*, *y=None*)

Transform the data.

Note: The transform method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

- **X** (array *of* items : array *of* items : float) – Training vectors, where n_samples is the number of samples and n_features is the number of predictors.
- **Y** (array, optional *of* items : array *of* items : float) – Target vectors, where n_samples is the number of samples and n_targets is the number of response variables.
- **copy** (*boolean, optional, default True*) – Whether to copy X and Y, or perform in-place normalization.

Returns

result – Apply the dimension reduction learned on the train data.

Return type

Any

lale.lib.autogen.complement_nb module

class lale.lib.autogen.complement_nb.**ComplementNB**(**, alpha=1.0, fit_prior=True, class_prior=None, norm=False*)

Bases: *PlannedIndividualOp*

Combined schema for expected data and hyperparameters.

This documentation is auto-generated from JSON schemas.

Parameters

- **alpha** (*float, not for optimizer, default 1.0*) – Additive (Laplace/Lidstone) smoothing parameter (0 for no smoothing).
- **fit_prior** (*boolean, not for optimizer, default True*) – Only used in edge case with a single class in the training set.
- **class_prior** (*union type, not for optimizer, default None*) – Prior probabilities of the classes
  - array *of* items : float
  - *or* None
- **norm** (*boolean, not for optimizer, default False*) – Whether or not a second normalization of the weights is performed

Notes

constraint-1 : any type

fit(*X*, *y=None*, ***fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

- **X** (array *of* items : array *of* items : float) – Training vectors, where n_samples is the number of samples and n_features is the number of features.
- **y** (array *of* items : float) – Target values.

- **sample_weight** (*union type, optional, default None*) – Weights applied to individual samples (1
  - array *of* items : float
  - *or* None

predict(*X*, ***predict_params*)

Make predictions.

Note: The predict method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (array *of* items : array *of* items : float) –

Returns

result – Predicted target values for X

Return type

array *of* items : float

predict_proba(*X*)

Probability estimates for all classes.

Note: The predict_proba method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (array *of* items : array *of* items : float) –

Returns

result – Returns the probability of the samples for each class in the model

Return type

array *of* items : array *of* items : float

lale.lib.autogen.dictionary_learning module

class lale.lib.autogen.dictionary_learning.**DictionaryLearning**(**, n_components=None, alpha=1, max_iter=1000, tol=1e-08, fit_algorithm='lars', transform_algorithm='omp', transform_n_nonzero_coefs=None, transform_alpha=None, n_jobs=1, code_init=None, dict_init=None, verbose=False, split_sign=False, random_state=None, positive_code=False, positive_dict=False, callback=None*)

Bases: *PlannedIndividualOp*

Combined schema for expected data and hyperparameters.

This documentation is auto-generated from JSON schemas.

Parameters

- **n_components** (*union type, default None*) – number of dictionary elements to extract
  - integer, >=2 for optimizer, <=256 for optimizer, uniform distribution
  - *or* None

- **alpha** (*float*, *>=1e-10 for optimizer*, *<=1.0 for optimizer*, *loguniform distribution*, *default 1*) – sparsity controlling parameter
- **max_iter** (*integer*, *>=10 for optimizer*, *<=1000 for optimizer*, *uniform distribution*, *default 1000*) – maximum number of iterations to perform
- **tol** (*float*, *>=1e-08 for optimizer*, *<=0.01 for optimizer*, *default 1e-08*) – tolerance for numerical error
- **fit_algorithm** ('lars' *or* 'cd', default 'lars') – lars: uses the least angle regression method to solve the lasso problem (linear_model.lars_path) cd: uses the coordinate descent method to compute the Lasso solution (linear_model.Lasso)
- **transform_algorithm** ('lasso_lars', 'lasso_cd', 'lars', 'omp', *or* 'threshold', default 'omp') – Algorithm used to transform the data lars: uses the least angle regression method (linear_model.lars_path) lasso_lars: uses Lars to compute the Lasso solution lasso_cd: uses the coordinate descent method to compute the Lasso solution (linear_model.Lasso)
- **transform_n_nonzero_coefs** (*None*, *not for optimizer*, *default None*) – Number of nonzero coefficients to target in each column of the solution
- **transform_alpha** (*union type*, *not for optimizer*, *default None*) – If *algorithm='lasso_lars'* or *algorithm='lasso_cd'*, *alpha* is the penalty applied to the L1 norm
  - float
  - *or* None
- **n_jobs** (*union type*, *not for optimizer*, *default 1*) – Number of parallel jobs to run
  - integer
  - *or* None
- **code_init** (*union type*, *not for optimizer*, *default None*) – initial value for the code, for warm restart
  - array *of* items : array *of* items : float
  - *or* None
- **dict_init** (*union type*, *not for optimizer*, *default None*) – initial values for the dictionary, for warm restart
  - array *of* items : array *of* items : float
  - *or* None
- **verbose** (*boolean*, *not for optimizer*, *default False*) – To control the verbosity of the procedure.
- **split_sign** (*boolean*, *not for optimizer*, *default False*) – Whether to split the sparse feature vector into the concatenation of its negative part and its positive part
- **random_state** (*union type*, *not for optimizer*, *default None*) – If int, random_state is the seed used by the random number generator; If RandomState instance, random_state is the random number generator; If None, the random number generator is the RandomState instance used by *np.random*.
  - integer
  - *or* numpy.random.RandomState
  - *or* None
- **positive_code** (*boolean*, *not for optimizer*, *default False*) – Whether to enforce positivity when finding the code
- **positive_dict** (*boolean*, *not for optimizer*, *default False*) – Whether to enforce positivity when finding the dictionary
- **callback** (*union type*, *optional*, *not for optimizer*, *default None*) – Callable that gets invoked every five iterations.
  - callable, not for optimizer
  - *or* None

Notes

constraint-1 : any type

fit(*X*, *y=None*, ***fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

- **X** (array *of* items : array *of* items : float) – Training vector, where n_samples in the number of samples and n_features is the number of features.
- **y** (*any type*) –

transform(*X*, *y=None*)

Transform the data.

Note: The transform method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (array *of* items : array *of* items : float) – Test data to be transformed, must have the same number of features as the data used to train the model.

Returns

result – Transformed data

Return type

array *of* items : array *of* items : float

lale.lib.autogen.elastic_net module

class lale.lib.autogen.elastic_net.**ElasticNet**(*, *alpha=1.0*, *l1_ratio=0.5*, *fit_intercept=True*, *precompute=False*, *max_iter=1000*, *copy_X=True*, *tol=0.0001*, *warm_start=False*, *positive=False*, *random_state=None*, *selection='cyclic'*)

Bases: *PlannedIndividualOp*

Combined schema for expected data and hyperparameters.

This documentation is auto-generated from JSON schemas.

Parameters

- **alpha** (*float, >=1e-10 for optimizer, <=1.0 for optimizer, loguniform distribution, default 1.0*) – Constant that multiplies the penalty terms
- **l1_ratio** (*float, not for optimizer, default 0.5*) – The ElasticNet mixing parameter, with `0 <= l1_ratio <= 1`
- **fit_intercept** (*boolean, default True*) – Whether the intercept should be estimated or not
- **precompute** (*union type, not for optimizer, default False*) – Whether to use a precomputed Gram matrix to speed up calculations
  - array *of* items : Any
  - *or* boolean
- **max_iter** (*integer, >=10 for optimizer, <=1000 for optimizer, uniform distribution, default 1000*) – The maximum number of iterations

- **copy_X** (*boolean, default True*) – If True, X will be copied; else, it may be overwritten.
- **tol** (*float, >=1e-08 for optimizer, <=0.01 for optimizer, default 0.0001*) – The tolerance for the optimization: if the updates are smaller than tol, the optimization code checks the dual gap for optimality and continues until it is smaller than tol.
- **warm_start** (*boolean, not for optimizer, default False*) – When set to True, reuse the solution of the previous call to fit as initialization, otherwise, just erase the previous solution
- **positive** (*boolean, default False*) – When set to True, forces the coefficients to be positive.
- **random_state** (*union type, not for optimizer, default None*) – The seed of the pseudo random number generator that selects a random feature to update
  - integer
  - *or* numpy.random.RandomState
  - *or* None
- **selection** ('random' *or* 'cyclic', default 'cyclic') – If set to 'random', a random coefficient is updated every iteration rather than looping over features sequentially by default

fit(*X*, *y=None*, ***fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

- **X** (*Any*) – Data
- **y** (*Any*) – Target
- **check_input** (*boolean, optional, default True*) – Allow to bypass several input checking

predict(*X*, ***predict_params*)

Make predictions.

Note: The predict method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (*union type*) – Samples.

- array *of* items : Any
- *or* array *of* items : array *of* items : float

Returns

result – Returns predicted values.

Return type

array *of* items : float

lale.lib.autogen.elastic_net_cv module

class lale.lib.autogen.elastic_net_cv.**ElasticNetCV**(**, l1_ratio=0.5, eps=0.001, n_alphas=100, alphas=None, fit_intercept=True, precompute='auto', max_iter=1000, tol=0.0001, cv, copy_X=True, verbose=0, n_jobs=1, positive=False, random_state=None, selection='cyclic'*)

Bases: *PlannedIndividualOp*

Combined schema for expected data and hyperparameters.

This documentation is auto-generated from JSON schemas.

Parameters

- **l1_ratio** (*float, >=0.0 for optimizer, <=1.0 for optimizer, uniform distribution, default 0.5*) – float between 0 and 1 passed to ElasticNet (scaling between l1 and l2 penalties)

  See also *constraint-1*.
- **eps** (*float, >=0.001 for optimizer, <=0.1 for optimizer, loguniform distribution, default 0.001*) – Length of the path
- **n_alphas** (*integer, >=100 for optimizer, <=101 for optimizer, uniform distribution, default 100*) – Number of alphas along the regularization path, used for each l1_ratio.
- **alphas** (*union type, not for optimizer, default None*) – List of alphas where to compute the models
  - array *of* items : Any
  - *or* None

  See also *constraint-1*.
- **fit_intercept** (*boolean, default True*) – whether to calculate the intercept for this model
- **precompute** (*union type, default 'auto'*) – Whether to use a precomputed Gram matrix to speed up calculations
  - array, not for optimizer *of* items : Any
  - *or* 'auto'
- **max_iter** (*integer, >=10 for optimizer, <=1000 for optimizer, uniform distribution, default 1000*) – The maximum number of iterations
- **tol** (*float, >=1e-08 for optimizer, <=0.01 for optimizer, default 0.0001*) – The tolerance for the optimization: if the updates are smaller than `tol`, the optimization code checks the dual gap for optimality and continues until it is smaller than `tol`.
- **cv** (*union type*) –

  Cross-validation as integer or as object that has a split function.

  The fit method performs cross validation on the input dataset for per trial, and uses the mean cross validation performance for optimization. This behavior is also impacted by handle_cv_failure flag. If integer: number of folds in sklearn.model_selection.StratifiedKFold. If object with split function: generator yielding (train, test) splits as arrays of indices. Can use any of the iterators from https://scikit-learn.org/stable/modules/cross_validation.html#cross-validation-iterators.

  - integer, >=1, >=3 for optimizer, <=4 for optimizer, uniform distribution, default 5
  - *or* Any, not for optimizer
- **copy_X** (*boolean, default True*) – If `True`, X will be copied; else, it may be overwritten.
- **verbose** (*union type, not for optimizer, default 0*) – Amount of verbosity.
  - boolean
  - *or* integer
- **n_jobs** (*union type, not for optimizer, default 1*) – Number of CPUs to use during the cross validation
  - integer
  - *or* None

- **positive** (*boolean, default False*) – When set to True, forces the coefficients to be positive.
- **random_state** (*union type, not for optimizer, default None*) – The seed of the pseudo random number generator that selects a random feature to update
  - integer
  - *or* numpy.random.RandomState
  - *or* None
- **selection** ('random' *or* 'cyclic', default 'cyclic') – If set to 'random', a random coefficient is updated every iteration rather than looping over features sequentially by default

Notes

constraint-1 : union type

From /linear_model/_coordinate_descent.py:None:_alpha_grid, Exception: raise ValueError(Automatic alpha grid generation is not supported for l1_ratio=0. Please supply a grid by providing your estimator with the appropriate *alphas*= argument.

- alphas : negated type *of* None
- *or* l1_ratio : negated type *of* 0

fit(*X*, *y=None*, ***fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

- **X** (*Any*) – Training data
- **y** (*union type*) – Target values
  - array *of* items : float
  - *or* array *of* items : array *of* items : float

predict(*X*, ***predict_params*)

Make predictions.

Note: The predict method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (*union type*) – Samples.

- array *of* items : Any
- *or* array *of* items : array *of* items : float

Returns

result – Returns predicted values.

Return type

array *of* items : float

lale.lib.autogen.factor_analysis module

class lale.lib.autogen.factor_analysis.**FactorAnalysis**(**, n_components=None, tol=0.01, copy=True, max_iter=1000, noise_variance_init=None, svd_method='randomized', iterated_power=3, random_state=0, rotation=None*)

Bases: *PlannedIndividualOp*

Combined schema for expected data and hyperparameters.

This documentation is auto-generated from JSON schemas.

Parameters

- **n_components** (*union type, not for optimizer, default None*) – Dimensionality of latent space, the number of components of X that are obtained after transform
  - integer, >=2 for optimizer, <='X/items/maxItems', <=256 for optimizer, uniform distribution
  - *or* None
- **tol** (*float, >=1e-08 for optimizer, <=0.01 for optimizer, default 0.01*) – Stopping tolerance for EM algorithm.
- **copy** (*boolean, default True*) – Whether to make a copy of X
- **max_iter** (*integer, >=10 for optimizer, <=1000 for optimizer, uniform distribution, not for optimizer, default 1000*) – Maximum number of iterations.
- **noise_variance_init** (*None, not for optimizer, default None*) – The initial guess of the noise variance for each feature
- **svd_method** ('lapack' *or* 'randomized', default 'randomized') – Which SVD method to use

  See also *constraint-3*, *constraint-3*.
- **iterated_power** (*integer, >=3 for optimizer, <=4 for optimizer, uniform distribution, default 3*) – Number of iterations for the power method
- **random_state** (*union type, not for optimizer, default 0*) – If int, random_state is the seed used by the random number generator; If RandomState instance, random_state is the random number generator; If None, the random number generator is the RandomState instance used by *np.random*
  - integer
  - *or* numpy.random.RandomState
  - *or* None

  See also *constraint-3*.
- **rotation** ('varimax', 'quartimax', *or* None, optional, not for optimizer, default None) – if not None, apply the indicated rotation. Currently, varimax and quartimax are implemented.

Notes

constraint-1 : any type

constraint-2 : any type

constraint-3 : union type

('random_state' only used when svd_method equals 'randomized') From /utils/validation.py:None:check_random_state, Exception: raise ValueError('%r cannot be used to seed a numpy.random.RandomState instance' % seed)

- svd_method : 'lapack'

- *or* svd_method : negated type *of* 'randomized'
- *or* random_state : None
- *or* any type
- *or* any type

constraint-4 : negated type *of* 'X/isSparse'
A sparse matrix was passed, but dense data is required. Use X.toarray() to convert to a dense numpy array.

fit(*X*, *y=None*, ***fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters
- **X** (array *of* items : array *of* items : float) – Training data.
- **y** (*any type*) –

transform(*X*, *y=None*)

Transform the data.

Note: The transform method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters
X (array *of* items : array *of* items : float) – Training data.

Returns
result – The latent variables of X.

Return type
array *of* items : array *of* items : float

lale.lib.autogen.fast_ica module

class lale.lib.autogen.fast_ica.**FastICA**(*, *n_components=None*, *algorithm='parallel'*, *whiten='arbitrary-variance'*, *fun='logcosh'*, *fun_args=None*, *max_iter=200*, *tol=0.0001*, *w_init=None*, *random_state=None*, *whiten_solver='svd'*)

Bases: *PlannedIndividualOp*

Combined schema for expected data and hyperparameters.

This documentation is auto-generated from JSON schemas.

Parameters
- **n_components** (*union type, default None*) – Number of components to use
  - integer, >=2 for optimizer, <=256 for optimizer, uniform distribution
  - *or* None
- **algorithm** ('parallel' *or* 'deflation', default 'parallel') – Apply parallel or deflational algorithm for FastICA.
- **whiten** (*union type, default 'arbitrary-variance'*) – Specify the whitening strategy to use.
  - False
    The data is already considered to be whitened, and no whitening is performed.
  - *or* 'arbitrary-variance'
    A whitening with variance arbitrary is used
  - *or* 'unit-variance'

The whitening matrix is rescaled to ensure that each recovered source has unit variance.

- **fun** ('cube', 'exp', *or* 'logcosh', default 'logcosh') – The functional form of the G function used in the approximation to neg-entropy
- **fun_args** (*None, not for optimizer, default None*) – Arguments to send to the functional form
- **max_iter** (*integer, >=1, >=10 for optimizer, <=1000 for optimizer, uniform distribution, default 200*) – Maximum number of iterations during fit.
- **tol** (*float, >=1e-08 for optimizer, <=0.01 for optimizer, default 0.0001*) – Tolerance on update at each iteration.
- **w_init** (*None, not for optimizer, default None*) – The mixing matrix to be used to initialize the algorithm.
- **random_state** (*union type, not for optimizer, default None*) – If int, random_state is the seed used by the random number generator; If RandomState instance, random_state is the random number generator; If None, the random number generator is the RandomState instance used by *np.random*.
  - integer
  - *or* numpy.random.RandomState
  - *or* None
- **whiten_solver** (*union type, optional, not for optimizer, default 'svd'*) – The solver to use for whitening.
  - 'eigh'

    Generally more memory efficient when n_samples >= n_features, and can be faster when n_samples >= 50 * n_features.
  - *or* 'svd'

    More stable numerically if the problem is degenerate, and often faster when n_samples <= n_features.

fit(*X*, *y=None*, ***fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

- **X** (array *of* items : array *of* items : float) – Training data, where n_samples is the number of samples and n_features is the number of features.
- **y** (*any type*) –

transform(*X*, *y=None*)

Transform the data.

Note: The transform method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

- **X** (array *of* items : array *of* items : float) – Data to transform, where n_samples is the number of samples and n_features is the number of features.
- **y** (*Any, optional*) –
- **copy** (*Any, optional*) – If False, data passed to fit are overwritten

Returns

result – Recover the sources from X (apply the unmixing matrix).

Return type

array *of* items : array *of* items : float

lale.lib.autogen.gaussian_process_classifier module

class lale.lib.autogen.gaussian_process_classifier.**GaussianProcessClassifier**(**, kernel=None, optimizer='fmin_l_bfgs_b', n_restarts_optimizer=0, max_iter_predict=100, warm_start=False, copy_X_train=True, random_state=None, multi_class='one_vs_rest', n_jobs=1*)

Bases: `PlannedIndividualOp`

Combined schema for expected data and hyperparameters.

This documentation is auto-generated from JSON schemas.

Parameters

- **kernel** (*None, not for optimizer, default None*) – The kernel specifying the covariance function of the GP
- **optimizer** (*union type, default 'fmin_l_bfgs_b'*) – Can either be one of the internally supported optimizers for optimizing the kernel's parameters, specified by a string, or an externally defined optimizer passed as a callable
  - callable, not for optimizer
  - *or* 'fmin_l_bfgs_b'
- **n_restarts_optimizer** (*integer, >=0 for optimizer, <=1 for optimizer, uniform distribution, default 0*) – The number of restarts of the optimizer for finding the kernel's parameters which maximize the log-marginal likelihood
- **max_iter_predict** (*integer, >=100 for optimizer, <=101 for optimizer, uniform distribution, default 100*) – The maximum number of iterations in Newton's method for approximating the posterior during predict
- **warm_start** (*boolean, not for optimizer, default False*) – If warm-starts are enabled, the solution of the last Newton iteration on the Laplace approximation of the posterior mode is used as initialization for the next call of _posterior_mode()
- **copy_X_train** (*boolean, not for optimizer, default True*) – If True, a persistent copy of the training data is stored in the object
- **random_state** (*union type, not for optimizer, default None*) – The generator used to initialize the centers
  - integer
  - *or* numpy.random.RandomState
  - *or* None
- **multi_class** ('one_vs_one' *or* 'one_vs_rest', default 'one_vs_rest') – Specifies how multi-class classification problems are handled
- **n_jobs** (*union type, not for optimizer, default 1*) – The number of jobs to use for the computation
  - integer
  - *or* None

fit(*X, y=None, **fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

- **X** (array *of* items : array *of* items : float) – Training data
- **y** (array *of* items : float) – Target values, must be binary

`predict`(*X*, ***predict_params*)

Make predictions.

Note: The predict method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (array *of* items : array *of* items : float) –

Returns

result – Predicted target values for X, values are from `classes_`

Return type

array *of* items : float

`predict_proba`(*X*)

Probability estimates for all classes.

Note: The predict_proba method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (array *of* items : array *of* items : float) –

Returns

result – Returns the probability of the samples for each class in the model

Return type

array *of* items : array *of* items : float

lale.lib.autogen.gaussian_process_regressor module

class `lale.lib.autogen.gaussian_process_regressor.GaussianProcessRegressor`(*, *kernel=None*, *alpha=1e-10*, *optimizer='fmin_l_bfgs_b'*, *n_restarts_optimizer=0*, *normalize_y=False*, *copy_X_train=True*, *random_state=None*, *n_targets=None*)

Bases: `PlannedIndividualOp`

Combined schema for expected data and hyperparameters.

This documentation is auto-generated from JSON schemas.

Parameters

- **kernel** (*None, not for optimizer, default None*) – The kernel specifying the covariance function of the GP
- **alpha** (*union type, default 1e-10*) – Value added to the diagonal of the kernel matrix during fitting
  - float, >=1e-10 for optimizer, <=1.0 for optimizer, loguniform distribution
  - *or* array, not for optimizer *of* items : Any

- **optimizer** (*union type, default 'fmin_l_bfgs_b'*) – Can either be one of the internally supported optimizers for optimizing the kernel's parameters, specified by a string, or an externally defined optimizer passed as a callable
  - callable, not for optimizer
  - *or* 'fmin_l_bfgs_b'
- **n_restarts_optimizer** (*integer, >=0 for optimizer, <=1 for optimizer, uniform distribution, default 0*) – The number of restarts of the optimizer for finding the kernel's parameters which maximize the log-marginal likelihood
- **normalize_y** (*boolean, default False*) – Whether the target values y are normalized, i.e., the mean of the observed target values become zero
- **copy_X_train** (*boolean, not for optimizer, default True*) – If True, a persistent copy of the training data is stored in the object
- **random_state** (*union type, not for optimizer, default None*) – The generator used to initialize the centers
  - integer
  - *or* numpy.random.RandomState
  - *or* None
- **n_targets** (*union type, optional, not for optimizer, default None*) – The number of dimensions of the target values. Used to decide the number of outputs when sampling from the prior distributions (i.e. calling sample_y before fit). This parameter is ignored once fit has been called.
  - integer, >=0, uniform distribution
  - *or* None

`fit`(*X*, *y=None*, ***fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

- **X** (array *of* items : array *of* items : float) – Training data
- **y** (*Any*) – Target values

`predict`(*X*, ***predict_params*)

Make predictions.

Note: The predict method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

- **X** (array *of* items : array *of* items : float) – Query points where the GP is evaluated
- **return_std** (*boolean, optional, default False*) – If True, the standard-deviation of the predictive distribution at the query points is returned along with the mean.
- **return_cov** (*boolean, optional, default False*) – If True, the covariance of the joint predictive distribution at the query points is returned along with the mean

Returns

result – Predict using the Gaussian process regression model

Return type

Any

lale.lib.autogen.gaussian_random_projection module

class lale.lib.autogen.gaussian_random_projection.**GaussianRandomProjection**(**, n_components='auto', eps=0.1, random_state=None*)

Bases: *PlannedIndividualOp*

Combined schema for expected data and hyperparameters.

This documentation is auto-generated from JSON schemas.

Parameters

- **n_components** (*union type, default 'auto'*) – Dimensionality of the target projection space
  - integer, >=2 for optimizer, <=256 for optimizer, uniform distribution
  - *or* 'auto'

  See also *constraint-1*.
- **eps** (*float, >=0.001 for optimizer, <=0.1 for optimizer, loguniform distribution, default 0.1*) – Parameter to control the quality of the embedding according to the Johnson-Lindenstrauss lemma when n_components is set to 'auto'
- **random_state** (*union type, not for optimizer, default None*) – Control the pseudo random number generator used to generate the matrix at fit time
  - integer
  - *or* numpy.random.RandomState
  - *or* None

Notes

constraint-1 : union type

eps=%f and n_samples=%d lead to a target dimension of %d which is larger than the original space with n_features=%d' % (self.eps, n_samples, *self.n_components_*, n_features)

- n_components : negated type *of* 'auto'
- *or* any type
- *or* any type

fit(*X, y=None, **fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

- **X** (*union type*) – Training set: only the shape is used to find optimal random matrix dimensions based on the theory referenced in the afore mentioned papers.
  - array *of* items : Any
  - *or* array *of* items : array *of* items : float
- **y** (*Any*) – Ignored

transform(*X, y=None*)

Transform the data.

Note: The transform method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (*union type*) – The input data to project into a smaller dimensional space.

- array *of* items : Any

- *or* array *of* items : array *of* items : float

Returns

result – Projected array.

- array *of* items : Any
- *or* array *of* items : array *of* items : float

Return type
union type

lale.lib.autogen.huber_regressor module

class lale.lib.autogen.huber_regressor.**HuberRegressor**(*, *epsilon=1.35*, *max_iter=100*, *alpha=0.0001*, *warm_start=False*, *fit_intercept=True*, *tol=1e-05*)

Bases: *PlannedIndividualOp*

Combined schema for expected data and hyperparameters.

This documentation is auto-generated from JSON schemas.

Parameters

- **epsilon** (*float, >=1.0 for optimizer, <=2.0 for optimizer, uniform distribution, default 1.35*) – The parameter epsilon controls the number of samples that should be classified as outliers
- **max_iter** (*integer, >=10 for optimizer, <=1000 for optimizer, uniform distribution, default 100*) – Maximum number of iterations that scipy.optimize.fmin_l_bfgs_b should run for.
- **alpha** (*float, >=1e-10 for optimizer, <=1.0 for optimizer, loguniform distribution, default 0.0001*) – Regularization parameter.
- **warm_start** (*boolean, not for optimizer, default False*) – This is useful if the stored attributes of a previously used model has to be reused
- **fit_intercept** (*boolean, default True*) – Whether or not to fit the intercept
- **tol** (*float, >=1e-08 for optimizer, <=0.01 for optimizer, default 1e-05*) – The iteration will stop when `max{|proj g_i | i = 1, ..., n}` <= `tol` where pg_i is the i-th component of the projected gradient.

fit(*X*, *y=None*, ***fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

- **X** (array *of* items : array *of* items : float) – Training vector, where n_samples in the number of samples and n_features is the number of features.
- **y** (array *of* items : float) – Target vector relative to X.
- **sample_weight** (array, optional *of* items : float) – Weight given to each sample.

predict(*X*, ***predict_params*)

Make predictions.

Note: The predict method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (*union type*) – Samples.

- array *of* items : Any
- *or* array *of* items : array *of* items : float

Returns
result – Returns predicted values.
Return type
array *of* items : float

lale.lib.autogen.incremental_pca module

class lale.lib.autogen.incremental_pca.**IncrementalPCA**(***, *n_components=None*, *whiten=False*, *copy=True*, *batch_size=None*)

Bases: *PlannedIndividualOp*

Combined schema for expected data and hyperparameters.

This documentation is auto-generated from JSON schemas.

Parameters

- **n_components** (*union type, default None*) – Number of components to keep
  - integer, >=2 for optimizer, <=256 for optimizer, uniform distribution
  - *or* None
- **whiten** (*boolean, default False*) – When True (False by default) the components_ vectors are divided by n_samples times components_ to ensure uncorrelated outputs with unit component-wise variances
- **copy** (*boolean, default True*) – If False, X will be overwritten
- **batch_size** (*union type, default None*) – The number of samples to use for each batch
  - integer, >=3 for optimizer, <=128 for optimizer, uniform distribution
  - *or* None

Notes

constraint-1 : any type

fit(*X*, *y=None*, ***fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

- **X** (array *of* items : array *of* items : float) – Training data, where n_samples is the number of samples and n_features is the number of features.
- **y** (*any type*) –

transform(*X*, *y=None*)

Transform the data.

Note: The transform method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters
X (array *of* items : array *of* items : float) – New data, where n_samples is the number of samples and n_features is the number of features.
Returns
result – Apply dimensionality reduction to X.
Return type
array *of* items : array *of* items : float

lale.lib.autogen.k_bins_discretizer module

class lale.lib.autogen.k_bins_discretizer.**KBinsDiscretizer**(*, *n_bins=5*, *encode='onehot'*, *strategy='quantile'*, *dtype=None*, *subsample='warn'*)

Bases: *PlannedIndividualOp*

Combined schema for expected data and hyperparameters.

This documentation is auto-generated from JSON schemas.

Parameters

- **n_bins** (*union type, not for optimizer, default 5*) – The number of bins to produce
  - integer
  - *or* array *of* items : float
- **encode** ('onehot', 'onehot-dense', *or* 'ordinal', default 'onehot') – Method used to encode the transformed result
- **strategy** ('uniform', 'quantile', *or* 'kmeans', default 'quantile') – Strategy used to define the widths of the bins
- **dtype** (*Any, optional, not for optimizer, default None*) –
- **subsample** (*union type, optional, not for optimizer, default 'warn'*) – Maximum number of samples, used to fit the model, for computational efficiency. Defaults to 200_000 when strategy='quantile' and to None when strategy='uniform' or strategy='kmeans'. subsample=None means that all the training samples are used when computing the quantiles that determine the binning thresholds. Since quantile computation relies on sorting each column of X and that sorting has an n log(n) time complexity, it is recommended to use subsampling on datasets with a very large number of samples.
  - 'warn' *or* None
  - *or* integer, >=0

Notes

constraint-1 : negated type *of* 'X/isSparse'

A sparse matrix was passed, but dense data is required. Use X.toarray() to convert to a dense numpy array.

fit(*X*, *y=None*, ***fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

- **X** (*Any*) – Data to be discretized.
- **y** (*Any*) –
- **sample_weight** (*union type, optional, default None*) – Contains weight values to be associated with each sample. Only possible when strategy is set to "quantile".
  - array *of* items : float
  - *or* None

transform(*X*, *y=None*)

Transform the data.

Note: The transform method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (*Any*) – Data to be discretized.

Returns

result – Data in the binned space.

Return type

Any

lale.lib.autogen.kernel_pca module

class lale.lib.autogen.kernel_pca.**KernelPCA**(***, *n_components=None*, *kernel='linear'*, *gamma=None*, *degree=3*, *coef0=1*, *kernel_params=None*, *alpha=1.0*, *fit_inverse_transform=False*, *eigen_solver='auto'*, *tol=0*, *max_iter=None*, *remove_zero_eig=False*, *random_state=None*, *copy_X=True*, *n_jobs=1*)

Bases: *PlannedIndividualOp*

Combined schema for expected data and hyperparameters.

This documentation is auto-generated from JSON schemas.

Parameters

- **n_components** (*union type, default None*) – Number of components
  - integer, >=2 for optimizer, <=256 for optimizer, uniform distribution
  - *or* None
- **kernel** ('linear', 'poly', 'rbf', 'sigmoid', 'cosine', *or* 'precomputed', default 'linear') – Kernel

  See also *constraint-1*.
- **gamma** (*None, not for optimizer, default None*) – Kernel coefficient for rbf, poly and sigmoid kernels
- **degree** (*union type, default 3*) – Degree for poly kernels
  - integer, >=2 for optimizer, <=3 for optimizer, uniform distribution
  - *or* float, not for optimizer
- **coef0** (*float, >=0.0 for optimizer, <=1.0 for optimizer, uniform distribution, default 1*) – Independent term in poly and sigmoid kernels
- **kernel_params** (*None, not for optimizer, default None*) – Parameters (keyword arguments) and values for kernel passed as callable object
- **alpha** (*union type, default 1.0*) – Hyperparameter of the ridge regression that learns the inverse transform (when fit_inverse_transform=True).
  - integer, not for optimizer
  - *or* float, >=1e-10 for optimizer, <=1.0 for optimizer, loguniform distribution
- **fit_inverse_transform** (*boolean, not for optimizer, default False*) – Learn the inverse transform for non-precomputed kernels

  See also *constraint-1*.
- **eigen_solver** ('auto', 'dense', *or* 'arpack', default 'auto') – Select eigensolver to use
- **tol** (*float, >=1e-08 for optimizer, <=0.01 for optimizer, default 0*) – Convergence tolerance for arpack
- **max_iter** (*union type, default None*) – Maximum number of iterations for arpack
  - integer, >=10 for optimizer, <=1000 for optimizer, uniform distribution
  - *or* None
- **remove_zero_eig** (*boolean, default False*) – If True, then all components with zero eigenvalues are removed, so that the number of components in the output may be < n_components (and sometimes even zero due to numerical instability)

- **random_state** (*union type, not for optimizer, default None*) – If int, random_state is the seed used by the random number generator; If RandomState instance, random_state is the random number generator; If None, the random number generator is the RandomState instance used by *np.random*
  - integer
  - *or* numpy.random.RandomState
  - *or* None
- **copy_X** (*boolean, default True*) – If True, input X is copied and stored by the model in the *X_fit_* attribute
- **n_jobs** (*union type, not for optimizer, default 1*) – The number of parallel jobs to run
  - integer
  - *or* None

Notes

constraint-1 : union type

Cannot fit_inverse_transform with a precomputed kernel.

- fit_inverse_transform : False
- *or* kernel : negated type *of* 'precomputed'

fit(*X*, *y=None*, ***fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

X (array *of* items : array *of* items : float) – Training vector, where n_samples in the number of samples and n_features is the number of features.

transform(*X*, *y=None*)

Transform the data.

Note: The transform method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (array *of* items : array *of* items : float) –

Returns

result – Transform X.

Return type

array *of* items : array *of* items : float

lale.lib.autogen.kernel_ridge module

class lale.lib.autogen.kernel_ridge.**KernelRidge**(**, alpha=1, kernel='linear', gamma=None, degree=3, coef0=1, kernel_params=None*)

Bases: *PlannedIndividualOp*

Combined schema for expected data and hyperparameters.

This documentation is auto-generated from JSON schemas.

Parameters

- **alpha** (*integer, >=1 for optimizer, <=2 for optimizer, uniform distribution, default 1*) – Small positive values of alpha improve the conditioning of the problem and reduce the variance of the estimates
- **kernel** (*union type, default 'linear'*) – Kernel mapping used internally
  - callable, not for optimizer
  - *or* 'linear', 'poly', 'precomputed', 'rbf', *or* 'sigmoid'
- **gamma** (*union type, not for optimizer, default None*) – Gamma parameter for the RBF, laplacian, polynomial, exponential chi2 and sigmoid kernels
  - float
  - *or* None
- **degree** (*union type, default 3*) – Degree of the polynomial kernel
  - integer, >=0 for optimizer, <=100 for optimizer, uniform distribution
  - *or* float, not for optimizer
- **coef0** (*float, >=0.0 for optimizer, <=1.0 for optimizer, uniform distribution, default 1*) – Zero coefficient for polynomial and sigmoid kernels
- **kernel_params** (*None, not for optimizer, default None*) – Additional parameters (keyword arguments) for kernel function passed as callable object.

`fit`(*X*, *y=None*, ***fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

- **X** (array *of* items : array *of* items : float) – Training data
- **y** (*union type*) – Target values
  - array *of* items : float
  - *or* array *of* items : array *of* items : float
- **sample_weight** (*union type, optional*) – Individual weights for each sample, ignored if None is passed.
  - float
  - *or* array *of* items : float

`predict`(*X*, ***predict_params*)

Make predictions.

Note: The predict method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (array *of* items : array *of* items : float) – Samples

Returns

result – Returns predicted values.

- array *of* items : float
- *or* array *of* items : array *of* items : float

Return type

union type

lale.lib.autogen.label_binarizer module

class lale.lib.autogen.label_binarizer.**LabelBinarizer**(**, neg_label=0, pos_label=1, sparse_output=False*)

Bases: *PlannedIndividualOp*

Combined schema for expected data and hyperparameters.

This documentation is auto-generated from JSON schemas.

Parameters

- **neg_label** (*integer, >=0 for optimizer, <=1 for optimizer, uniform distribution, default 0*) – Value with which negative labels must be encoded.
- **pos_label** (*integer, >=1 for optimizer, <=2 for optimizer, uniform distribution, default 1*) – Value with which positive labels must be encoded.
- **sparse_output** (*boolean, default False*) – True if the returned array from transform is desired to be in sparse CSR format.

fit(*X, y=None, **fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

y (*union type*) – Target values

- array *of* items : float
- *or* array *of* items : array *of* items : float

transform(*X, y=None*)

Transform the data.

Note: The transform method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

y (*union type*) – Target values

- array *of* items : Any
- *or* array *of* items : float
- *or* array *of* items : array *of* items : float

Returns

result – Shape will be [n_samples, 1] for binary problems.

Return type

Any

lale.lib.autogen.label_encoder module

class lale.lib.autogen.label_encoder.**LabelEncoder**

Bases: *PlannedIndividualOp*

Combined schema for expected data and hyperparameters.

This documentation is auto-generated from JSON schemas.

fit(*X, y=None, **fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters
y (array *of* items : float) – Target values.

transform(*X*, *y=None*)
Transform the data.

Note: The transform method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters
y (array *of* items : float) – Target values.

Returns
result – Transform labels to normalized encoding.

Return type
array *of* items : float

lale.lib.autogen.label_propagation module

class lale.lib.autogen.label_propagation.**LabelPropagation**(**, kernel='rbf', gamma=20, n_neighbors=7, max_iter=1000, tol=0.001, n_jobs=1*)

Bases: `PlannedIndividualOp`

Combined schema for expected data and hyperparameters.

This documentation is auto-generated from JSON schemas.

Parameters

- **kernel** (*union type, default 'rbf'*) – String identifier for kernel function to use or the kernel function itself
  - 'knn' *or* 'rbf'
  - *or* callable, not for optimizer
- **gamma** (*float, >=0 for optimizer, <=20 for optimizer, uniform distribution, default 20*) – Parameter for rbf kernel
- **n_neighbors** (*integer, >=5 for optimizer, <=20 for optimizer, uniform distribution, default 7*) – Parameter for knn kernel
- **max_iter** (*integer, >=10 for optimizer, <=1000 for optimizer, uniform distribution, default 1000*) – Change maximum number of iterations allowed
- **tol** (*float, >=1e-08 for optimizer, <=0.01 for optimizer, default 0.001*) – Convergence tolerance: threshold to consider the system at steady state
- **n_jobs** (*union type, not for optimizer, default 1*) – The number of parallel jobs to run
  - integer
  - *or* None

Notes

constraint-1 : any type

`fit`(*X*, *y=None*, ***fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

- **X** (array *of* items : array *of* items : float) – A {n_samples by n_samples} size matrix will be created from this
- **y** (array *of* items : float) – n_labeled_samples (unlabeled points are marked as -1) All unlabeled samples will be transductively assigned labels

`predict`(*X*, ***predict_params*)

Make predictions.

Note: The predict method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters
X (array *of* items : array *of* items : float) –

Returns
result – Predictions for input data

Return type
array *of* items : float

`predict_proba`(*X*)

Probability estimates for all classes.

Note: The predict_proba method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters
X (array *of* items : array *of* items : float) –

Returns
result – Normalized probability distributions across class labels

Return type
array *of* items : array *of* items : float

lale.lib.autogen.label_spreading module

class `lale.lib.autogen.label_spreading.LabelSpreading`(**, kernel='rbf', gamma=20, n_neighbors=7, alpha=0.2, max_iter=30, tol=0.001, n_jobs=1*)

Bases: `PlannedIndividualOp`

Combined schema for expected data and hyperparameters.

This documentation is auto-generated from JSON schemas.

Parameters

- **kernel** (*union type, default 'rbf'*) – String identifier for kernel function to use or the kernel function itself
  - 'knn' *or* 'rbf'
  - *or* callable, not for optimizer

- **gamma** (*float, >=0 for optimizer, <=20 for optimizer, uniform distribution, default 20*) – parameter for rbf kernel
- **n_neighbors** (*integer, >=5 for optimizer, <=20 for optimizer, uniform distribution, default 7*) – parameter for knn kernel
- **alpha** (*float, >=1e-10 for optimizer, <=1.0 for optimizer, loguniform distribution, default 0.2*) – Clamping factor
- **max_iter** (*integer, >=10 for optimizer, <=1000 for optimizer, uniform distribution, default 30*) – maximum number of iterations allowed
- **tol** (*float, >=1e-08 for optimizer, <=0.01 for optimizer, default 0.001*) – Convergence tolerance: threshold to consider the system at steady state
- **n_jobs** (*union type, not for optimizer, default 1*) – The number of parallel jobs to run
  - integer
  - *or* None

Notes

constraint-1 : any type

fit(*X*, *y=None*, ***fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

- **X** (array *of* items : array *of* items : float) – A {n_samples by n_samples} size matrix will be created from this
- **y** (array *of* items : float) – n_labeled_samples (unlabeled points are marked as -1) All unlabeled samples will be transductively assigned labels

predict(*X*, ***predict_params*)

Make predictions.

Note: The predict method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (array *of* items : array *of* items : float) –

Returns

result – Predictions for input data

Return type

array *of* items : float

predict_proba(*X*)

Probability estimates for all classes.

Note: The predict_proba method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (array *of* items : array *of* items : float) –

Returns

result – Normalized probability distributions across class labels

Return type

array *of* items : array *of* items : float

lale.lib.autogen.lars module

class lale.lib.autogen.lars.**Lars**(**, fit_intercept=True, verbose=False, precompute='auto', n_nonzero_coefs=500, eps=2.220446049250313e-16, copy_X=True, fit_path=True, jitter=None, random_state=None*)

Bases: `PlannedIndividualOp`

Combined schema for expected data and hyperparameters.

This documentation is auto-generated from JSON schemas.

Parameters

- **fit_intercept** (*boolean, default True*) – Whether to calculate the intercept for this model
- **verbose** (*union type, not for optimizer, default False*) – Sets the verbosity amount
  - boolean
  - *or* integer
- **precompute** (*union type, default 'auto'*) – Whether to use a precomputed Gram matrix to speed up calculations
  - array, not for optimizer *of* items : Any
  - *or* 'auto'
- **n_nonzero_coefs** (*integer, >=500 for optimizer, <=501 for optimizer, uniform distribution, default 500*) – Target number of non-zero coefficients
- **eps** (*float, >=0.001 for optimizer, <=0.1 for optimizer, loguniform distribution, default 2.220446049250313e-16*) – The machine-precision regularization in the computation of the Cholesky diagonal factors
- **copy_X** (*boolean, default True*) – If `True`, X will be copied; else, it may be overwritten.
- **fit_path** (*boolean, default True*) – If True the full path is stored in the `coef_path_` attribute
- **jitter** (*union type, not for optimizer, default None*) – Upper bound on a uniform noise parameter to be added to the y values, to satisfy the model's assumption of one-at-a-time computations
  - float
  - *or* None
- **random_state** (*union type, not for optimizer, default None*) – The seed of the pseudo random number generator to use when shuffling the data
  - integer
  - *or* numpy.random.RandomState
  - *or* None

fit(*X, y=None, **fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

- **X** (array *of* items : array *of* items : float) – Training data.
- **y** (*union type*) – Target values.
  - array *of* items : float
  - *or* array *of* items : array *of* items : float
- **Xy** (*Any, optional*) – Xy = np.dot(X.T, y) that can be precomputed

predict(*X, **predict_params*)

Make predictions.

Note: The predict method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (*union type*) – Samples.

- array *of* items : Any
- *or* array *of* items : array *of* items : float

Returns

result – Returns predicted values.

Return type

array *of* items : float

lale.lib.autogen.lars_cv module

class lale.lib.autogen.lars_cv.**LarsCV**(**, fit_intercept=True, verbose=False, max_iter=500, precompute='auto', cv=None, max_n_alphas=1000, n_jobs=1, eps=2.220446049250313e-16, copy_X=True*)

Bases: *PlannedIndividualOp*

Combined schema for expected data and hyperparameters.

This documentation is auto-generated from JSON schemas.

Parameters

- **fit_intercept** (*boolean, default True*) – whether to calculate the intercept for this model
- **verbose** (*union type, not for optimizer, default False*) – Sets the verbosity amount
  - boolean
  - *or* integer
- **max_iter** (*integer, >=10 for optimizer, <=1000 for optimizer, uniform distribution, default 500*) – Maximum number of iterations to perform.
- **precompute** (*union type, default 'auto'*) – Whether to use a precomputed Gram matrix to speed up calculations
  - array, not for optimizer *of* items : Any
  - *or* 'auto'
- **cv** (*union type, default None*) –

  Cross-validation as integer or as object that has a split function.

  The fit method performs cross validation on the input dataset for per trial, and uses the mean cross validation performance for optimization. This behavior is also impacted by handle_cv_failure flag. If integer: number of folds in sklearn.model_selection.StratifiedKFold. If object with split function: generator yielding (train, test) splits as arrays of indices. Can use any of the iterators from https://scikit-learn.org/stable/modules/cross_validation.html#cross-validation-iterators.

  - integer, >=1, >=3 for optimizer, <=4 for optimizer, uniform distribution, default 5
  - *or* Any, not for optimizer
  - *or* None
- **max_n_alphas** (*integer, >=1000 for optimizer, <=1001 for optimizer, uniform distribution, default 1000*) – The maximum number of points on the

path used to compute the residuals in the cross-validation

- **n_jobs** (*union type, not for optimizer, default 1*) – Number of CPUs to use during the cross validation
  - integer
  - *or* None
- **eps** (*float, >=0.001 for optimizer, <=0.1 for optimizer, loguniform distribution, default 2.220446049250313e-16*) – The machine-precision regularization in the computation of the Cholesky diagonal factors
- **copy_X** (*boolean, default True*) – If True, X will be copied; else, it may be overwritten.

Notes

constraint-1 : any type

fit(*X*, *y=None*, ***fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

- **X** (array *of* items : array *of* items : float) – Training data.
- **y** (array *of* items : float) – Target values.

predict(*X*, ***predict_params*)

Make predictions.

Note: The predict method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (*union type*) – Samples.

- array *of* items : Any
- *or* array *of* items : array *of* items : float

Returns

result – Returns predicted values.

Return type

array *of* items : float

lale.lib.autogen.lasso module

class lale.lib.autogen.lasso.**Lasso**(***, *alpha=1.0*, *fit_intercept=True*, *precompute=False*, *copy_X=True*, *max_iter=1000*, *tol=0.0001*, *warm_start=False*, *positive=False*, *random_state=None*, *selection='cyclic'*)

Bases: *PlannedIndividualOp*

Combined schema for expected data and hyperparameters.

This documentation is auto-generated from JSON schemas.

Parameters

- **alpha** (*float, >=1e-10 for optimizer, <=1.0 for optimizer, loguniform distribution, default 1.0*) – Constant that multiplies the L1 term
- **fit_intercept** (*boolean, default True*) – Whether to calculate the intercept for this model

- **precompute** (*union type, not for optimizer, default False*) – Whether to use a precomputed Gram matrix to speed up calculations
  - array *of* items : Any
  - *or* boolean
- **copy_X** (*boolean, default True*) – If `True`, X will be copied; else, it may be overwritten.
- **max_iter** (*integer, >=10 for optimizer, <=1000 for optimizer, uniform distribution, default 1000*) – The maximum number of iterations
- **tol** (*float, >=1e-08 for optimizer, <=0.01 for optimizer, default 0.0001*) – The tolerance for the optimization: if the updates are smaller than `tol`, the optimization code checks the dual gap for optimality and continues until it is smaller than `tol`.
- **warm_start** (*boolean, not for optimizer, default False*) – When set to True, reuse the solution of the previous call to fit as initialization, otherwise, just erase the previous solution
- **positive** (*boolean, default False*) – When set to `True`, forces the coefficients to be positive.
- **random_state** (*union type, not for optimizer, default None*) – The seed of the pseudo random number generator that selects a random feature to update
  - integer
  - *or* numpy.random.RandomState
  - *or* None
- **selection** ('random' *or* 'cyclic', default 'cyclic') – If set to 'random', a random coefficient is updated every iteration rather than looping over features sequentially by default

fit(*X*, *y=None*, ***fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

- **X** (*Any*) – Data
- **y** (*Any*) – Target
- **check_input** (*boolean, optional, default True*) – Allow to bypass several input checking

predict(*X*, ***predict_params*)

Make predictions.

Note: The predict method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (*union type*) – Samples.

- array *of* items : Any
- *or* array *of* items : array *of* items : float

Returns

result – Returns predicted values.

Return type

array *of* items : float

lale.lib.autogen.lasso_cv module

class lale.lib.autogen.lasso_cv.**LassoCV**(***, *eps=0.001*, *n_alphas=100*, *alphas=None*, *fit_intercept=True*, *precompute='auto'*, *max_iter=1000*, *tol=0.0001*, *copy_X=True*, *cv*, *verbose=False*, *n_jobs=1*, *positive=False*, *random_state=None*, *selection='cyclic'*)

Bases: `PlannedIndividualOp`

Combined schema for expected data and hyperparameters.

This documentation is auto-generated from JSON schemas.

Parameters

- **eps** (*float, >=0.001 for optimizer, <=0.1 for optimizer, loguniform distribution, default 0.001*) – Length of the path
- **n_alphas** (*integer, >=100 for optimizer, <=101 for optimizer, uniform distribution, default 100*) – Number of alphas along the regularization path
- **alphas** (*union type, not for optimizer, default None*) – List of alphas where to compute the models
  - array *of* items : Any
  - *or* None
- **fit_intercept** (*boolean, default True*) – whether to calculate the intercept for this model
- **precompute** (*union type, default 'auto'*) – Whether to use a precomputed Gram matrix to speed up calculations
  - array, not for optimizer *of* items : Any
  - *or* 'auto'
- **max_iter** (*integer, >=10 for optimizer, <=1000 for optimizer, uniform distribution, default 1000*) – The maximum number of iterations
- **tol** (*float, >=1e-08 for optimizer, <=0.01 for optimizer, default 0.0001*) – The tolerance for the optimization: if the updates are smaller than `tol`, the optimization code checks the dual gap for optimality and continues until it is smaller than `tol`.
- **copy_X** (*boolean, default True*) – If `True`, X will be copied; else, it may be overwritten.
- **cv** (*union type*) –

  Cross-validation as integer or as object that has a split function.

  The fit method performs cross validation on the input dataset for per trial, and uses the mean cross validation performance for optimization. This behavior is also impacted by handle_cv_failure flag. If integer: number of folds in sklearn.model_selection.StratifiedKFold. If object with split function: generator yielding (train, test) splits as arrays of indices. Can use any of the iterators from https://scikit-learn.org/stable/modules/cross_validation.html#cross-validation-iterators.

  - integer, >=1, >=3 for optimizer, <=4 for optimizer, uniform distribution, default 5
  - *or* Any, not for optimizer
- **verbose** (*union type, not for optimizer, default False*) – Amount of verbosity.
  - boolean
  - *or* integer
- **n_jobs** (*union type, not for optimizer, default 1*) – Number of CPUs to use during the cross validation

    - integer
    - *or* None
- **positive** (*boolean, default False*) – If positive, restrict regression coefficients to be positive
- **random_state** (*union type, not for optimizer, default None*) – The seed of the pseudo random number generator that selects a random feature to update
    - integer
    - *or* numpy.random.RandomState
    - *or* None
- **selection** ('random' *or* 'cyclic', default 'cyclic') – If set to 'random', a random coefficient is updated every iteration rather than looping over features sequentially by default

fit(*X*, *y=None*, ***fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters
- **X** (*Any*) – Training data
- **y** (*union type*) – Target values
    - array *of* items : float
    - *or* array *of* items : array *of* items : float

predict(*X*, ***predict_params*)

Make predictions.

Note: The predict method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (*union type*) – Samples.
- array *of* items : Any
- *or* array *of* items : array *of* items : float

Returns

result – Returns predicted values.

Return type

array *of* items : float

lale.lib.autogen.lasso_lars module

class lale.lib.autogen.lasso_lars.**LassoLars**(***, *alpha=1.0*, *fit_intercept=True*, *verbose=False*, *precompute='auto'*, *max_iter=500*, *eps=2.220446049250313e-16*, *copy_X=True*, *fit_path=True*, *positive=False*)

Bases: *PlannedIndividualOp*

Combined schema for expected data and hyperparameters.

This documentation is auto-generated from JSON schemas.

Parameters
- **alpha** (*float, >=1e-10 for optimizer, <=1.0 for optimizer, loguniform distribution, default 1.0*) – Constant that multiplies the penalty term
- **fit_intercept** (*boolean, default True*) – whether to calculate the intercept for this model

- **verbose** (*union type, not for optimizer, default False*) – Sets the verbosity amount
  - boolean
  - *or* integer
- **precompute** (*union type, default 'auto'*) – Whether to use a precomputed Gram matrix to speed up calculations
  - array, not for optimizer *of* items : Any
  - *or* 'auto'
- **max_iter** (*integer, >=10 for optimizer, <=1000 for optimizer, uniform distribution, default 500*) – Maximum number of iterations to perform.
- **eps** (*float, >=0.001 for optimizer, <=0.1 for optimizer, loguniform distribution, default 2.220446049250313e-16*) – The machine-precision regularization in the computation of the Cholesky diagonal factors
- **copy_X** (*boolean, default True*) – If True, X will be copied; else, it may be overwritten.
- **fit_path** (*boolean, not for optimizer, default True*) – If `True` the full path is stored in the `coef_path_` attribute
- **positive** (*boolean, default False*) – Restrict coefficients to be >= 0

Notes

constraint-1 : any type

`fit`(*X*, *y=None*, ***fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

- **X** (array *of* items : array *of* items : float) – Training data.
- **y** (*union type*) – Target values.
  - array *of* items : float
  - *or* array *of* items : array *of* items : float
- **Xy** (*Any, optional*) – Xy = np.dot(X.T, y) that can be precomputed

`predict`(*X*, ***predict_params*)

Make predictions.

Note: The predict method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (*union type*) – Samples.

- array *of* items : Any
- *or* array *of* items : array *of* items : float

Returns

result – Returns predicted values.

Return type

array *of* items : float

lale.lib.autogen.lasso_lars_cv module

class `lale.lib.autogen.lasso_lars_cv.`**`LassoLarsCV`**(**, fit_intercept=True, verbose=False, max_iter=500, precompute='auto', cv, max_n_alphas=1000, n_jobs=1, eps=2.220446049250313e-16, copy_X=True, positive=False*)

Bases: `PlannedIndividualOp`

Combined schema for expected data and hyperparameters.

This documentation is auto-generated from JSON schemas.

Parameters

- **fit_intercept** (*boolean, default True*) – whether to calculate the intercept for this model
- **verbose** (*union type, not for optimizer, default False*) – Sets the verbosity amount
  - boolean
  - *or* integer
- **max_iter** (*integer, >=10 for optimizer, <=1000 for optimizer, uniform distribution, default 500*) – Maximum number of iterations to perform.
- **precompute** (True, False, *or* 'auto', default 'auto') – Whether to use a precomputed Gram matrix to speed up calculations
- **cv** (*union type*) –

  Cross-validation as integer or as object that has a split function.

  The fit method performs cross validation on the input dataset for per trial, and uses the mean cross validation performance for optimization. This behavior is also impacted by handle_cv_failure flag. If integer: number of folds in sklearn.model_selection.StratifiedKFold. If object with split function: generator yielding (train, test) splits as arrays of indices. Can use any of the iterators from https://scikit-learn.org/stable/modules/cross_validation.html#cross-validation-iterators.

  - integer, >=1, >=3 for optimizer, <=4 for optimizer, uniform distribution, default 5
  - *or* Any, not for optimizer
- **max_n_alphas** (*integer, >=1000 for optimizer, <=1001 for optimizer, uniform distribution, default 1000*) – The maximum number of points on the path used to compute the residuals in the cross-validation
- **n_jobs** (*union type, not for optimizer, default 1*) – Number of CPUs to use during the cross validation
  - integer
  - *or* None
- **eps** (*float, >=0.001 for optimizer, <=0.1 for optimizer, loguniform distribution, default 2.220446049250313e-16*) – The machine-precision regularization in the computation of the Cholesky diagonal factors
- **copy_X** (*boolean, default True*) – If True, X will be copied; else, it may be overwritten.
- **positive** (*boolean, default False*) – Restrict coefficients to be >= 0

Notes

constraint-1 : any type

constraint-2 : any type

fit(*X*, *y=None*, ***fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

- **X** (array *of* items : array *of* items : float) – Training data.
- **y** (array *of* items : float) – Target values.

predict(*X*, ***predict_params*)

Make predictions.

Note: The predict method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (*union type*) – Samples.

- array *of* items : Any
- *or* array *of* items : array *of* items : float

Returns

result – Returns predicted values.

Return type

array *of* items : float

lale.lib.autogen.lasso_lars_ic module

class lale.lib.autogen.lasso_lars_ic.**LassoLarsIC**(***, *criterion='aic'*, *fit_intercept=True*, *verbose=False*, *precompute='auto'*, *max_iter=500*, *eps=2.220446049250313e-16*, *copy_X=True*, *positive=False*)

Bases: *PlannedIndividualOp*

Combined schema for expected data and hyperparameters.

This documentation is auto-generated from JSON schemas.

Parameters

- **criterion** ('bic' *or* 'aic', default 'aic') – The type of criterion to use.
- **fit_intercept** (*boolean, default True*) – whether to calculate the intercept for this model
- **verbose** (*union type, not for optimizer, default False*) – Sets the verbosity amount
  - boolean
  - *or* integer
- **precompute** (*union type, default 'auto'*) – Whether to use a precomputed Gram matrix to speed up calculations
  - array, not for optimizer *of* items : Any
  - *or* boolean
  - *or* 'auto'

  See also *constraint-2*, *constraint-3*.

- **max_iter** (*integer, >=10 for optimizer, <=1000 for optimizer, uniform distribution, default 500*) – Maximum number of iterations to perform
- **eps** (*float, >=0.001 for optimizer, <=0.1 for optimizer, loguniform distribution, default 2.220446049250313e-16*) – The machine-precision regularization in the computation of the Cholesky diagonal factors
- **copy_X** (*boolean, default True*) – If True, X will be copied; else, it may be overwritten.
- **positive** (*boolean, default False*) – Restrict coefficients to be >= 0

Notes

constraint-1 : any type

constraint-2 : union type
X cannot be None if Gram is not NoneUse lars_path_gram to avoid passing X and y.)
- any type
- *or* precompute : None

constraint-3 : union type
From /linear_model/_least_angle.py:None:_lars_path_solver, Exception: raise ValueError('X and Gram cannot both be unspecified.')
- precompute : negated type *of* None *or* False
- *or* any type

fit(*X*, *y=None*, ***fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters
- **X** (array *of* items : array *of* items : float) – training data.
- **y** (array *of* items : float) – target values
- **copy_X** (*boolean, optional, default True*) – If True, X will be copied; else, it may be overwritten.

predict(*X*, ***predict_params*)

Make predictions.

Note: The predict method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters
X (*union type*) – Samples.
- array *of* items : Any
- *or* array *of* items : array *of* items : float

Returns
result – Returns predicted values.

Return type
array *of* items : float

lale.lib.autogen.latent_dirichlet_allocation module

class lale.lib.autogen.latent_dirichlet_allocation.**LatentDirichletAllocation**(**, n_components=10, doc_topic_prior=None, topic_word_prior=None, learning_method='batch', learning_decay=0.7, learning_offset=10.0, max_iter=10, batch_size=128, evaluate_every=-1, total_samples=1000000.0, perp_tol=0.1, mean_change_tol=0.001, max_doc_update_iter=100, n_jobs=1, verbose=0, random_state=None*)

Bases: *PlannedIndividualOp*

Combined schema for expected data and hyperparameters.

This documentation is auto-generated from JSON schemas.

Parameters

- **n_components** (*integer, >=2 for optimizer, <='X/items/maxItems', <=256 for optimizer, uniform distribution, default 10*) – Number of topics.
- **doc_topic_prior** (*union type, not for optimizer, default None*) – Prior of document topic distribution *theta*
  - float
  - *or* None
- **topic_word_prior** (*union type, not for optimizer, default None*) – Prior of topic word distribution *beta*
  - float
  - *or* None
- **learning_method** ('batch' *or* 'online', default 'batch') – Method used to update *_component*
- **learning_decay** (*float, not for optimizer, default 0.7*) – It is a parameter that control learning rate in the online learning method
- **learning_offset** (*float, not for optimizer, default 10.0*) – A (positive) parameter that downweights early iterations in online learning
- **max_iter** (*integer, >=10 for optimizer, <=1000 for optimizer, uniform distribution, default 10*) – The maximum number of iterations.
- **batch_size** (*integer, >=3 for optimizer, <=128 for optimizer, uniform distribution, default 128*) – Number of documents to use in each EM iteration
- **evaluate_every** (*integer, >=-1 for optimizer, <=0 for optimizer, uniform distribution, default -1*) – How often to evaluate perplexity

- **total_samples** (*union type, default 1000000.0*) – Total number of documents
  - integer, not for optimizer
  - *or* float, >=0.0 for optimizer, <=1.0 for optimizer, uniform distribution
- **perp_tol** (*float, not for optimizer, default 0.1*) – Perplexity tolerance in batch learning
- **mean_change_tol** (*float, not for optimizer, default 0.001*) – Stopping tolerance for updating document topic distribution in E-step.
- **max_doc_update_iter** (*integer, >=100 for optimizer, <=101 for optimizer, uniform distribution, default 100*) – Max number of iterations for updating document topic distribution in the E-step.
- **n_jobs** (*union type, not for optimizer, default 1*) – The number of jobs to use in the E-step
  - integer
  - *or* None
- **verbose** (*integer, not for optimizer, default 0*) – Verbosity level.
- **random_state** (*union type, not for optimizer, default None*) – If int, random_state is the seed used by the random number generator; If RandomState instance, random_state is the random number generator; If None, the random number generator is the RandomState instance used by *np.random*.
  - integer
  - *or* numpy.random.RandomState
  - *or* None

Notes

constraint-1 : any type

fit(*X*, *y=None*, ***fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

- **X** (*union type*) – Document word matrix.
  - array *of* items : Any
  - *or* array *of* items : array *of* items : float
- **y** (*any type*) –

transform(*X*, *y=None*)

Transform the data.

Note: The transform method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (*union type*) – Document word matrix.

- array *of* items : Any
- *or* array *of* items : array *of* items : float

Returns

result – Document topic distribution for X.

Return type

Any

lale.lib.autogen.linear_discriminant_analysis module

class lale.lib.autogen.linear_discriminant_analysis.**LinearDiscriminantAnalysis**(*, *solver='svd'*, *shrinkage=None*, *priors=None*, *n_components=None*, *store_covariance=False*, *tol=0.0001*, *covariance_estimator=None*)

Bases: *PlannedIndividualOp*

Combined schema for expected data and hyperparameters.

This documentation is auto-generated from JSON schemas.

Parameters

- **solver** ('eigen', 'lsqr', *or* 'svd', default 'svd') – Solver to use, possible values: - 'svd': Singular value decomposition (default)

  See also *constraint-1*, *constraint-2*, *constraint-3*, *constraint-4*, *constraint-4*.
- **shrinkage** (*union type, default None*) – Shrinkage parameter, possible values: - None: no shrinkage (default)
  - 'auto'
  - *or* float, >0, >=0 for optimizer, <1, <=1 for optimizer, uniform distribution
  - *or* None

  See also *constraint-1*, *constraint-4*.
- **priors** (*None, not for optimizer, default None*) – Class priors.
- **n_components** (*union type, default None*) – Number of components (< n_classes - 1) for dimensionality reduction.
  - integer, >=2 for optimizer, <='X/items/maxItems', <=256 for optimizer, uniform distribution
  - *or* None
- **store_covariance** (*boolean, not for optimizer, default False*) – Additionally compute class covariance matrix (default False), used only in 'svd' solver

  See also *constraint-2*.
- **tol** (*float, >=1e-08 for optimizer, <=0.01 for optimizer, default 0.0001*) – Threshold used for rank estimation in SVD solver
- **covariance_estimator** (*union type, optional, not for optimizer, default None*) – type of (covariance estimator). Estimate the covariance matrices instead of relying on the empirical covariance estimator (with potential shrinkage)
  - string, not for optimizer
  - *or* None

  See also *constraint-3*, *constraint-4*.

Notes

constraint-1 : union type
  shrinkage, only with 'lsqr' and 'eigen' solvers
  - shrinkage : None
  - *or* solver : 'lsqr' *or* 'eigen'

constraint-2 : union type
  store_covariance, only in 'svd' solver
  - store_covariance : False
  - *or* solver : 'svd'

constraint-3 : union type
  covariance estimator is not supported with svd solver. Try another solver
  - solver : negated type *of* 'svd'
  - *or* covariance_estimator : None

constraint-4 : union type
  covariance_estimator and shrinkage parameters are not None. Only one of the two can be set.
  - solver : 'svd' *or* 'lsqr'
  - *or* solver : negated type *of* 'eigen'
  - *or* covariance_estimator : None
  - *or* shrinkage : None *or* 0

`decision_function`(*X*)

Confidence scores for all classes.

Note: The decision_function method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (`union type`) – Samples.
- array *of* items : Any
- *or* array *of* items : array *of* items : float

Returns

result – Confidence scores per (sample, class) combination

Return type

Any

`fit`(*X*, *y=None*, ***fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters
- **X** (array *of* items : array *of* items : float) – Training data.
- **y** (array *of* items : float) – Target values.

`predict`(*X*, ***predict_params*)

Make predictions.

Note: The predict method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (`union type`) – Samples.
- array *of* items : Any
- *or* array *of* items : array *of* items : float

Returns

result – Predicted class label per sample.

Return type
array *of* items : float

predict_proba(*X*)

Probability estimates for all classes.

Note: The predict_proba method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters
X (array *of* items : array *of* items : float) – Input data.

Returns
result – Estimated probabilities.

Return type
array *of* items : array *of* items : float

transform(*X*, *y=None*)

Transform the data.

Note: The transform method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters
X (array *of* items : array *of* items : float) – Input data.

Returns
result – Transformed data.

Return type
array *of* items : array *of* items : float

lale.lib.autogen.locally_linear_embedding module

class lale.lib.autogen.locally_linear_embedding.**LocallyLinearEmbedding**(*, *n_neighbors=5*, *n_components=2*, *reg=0.001*, *eigen_solver='auto'*, *tol=1e-06*, *max_iter=100*, *method='standard'*, *hessian_tol=0.0001*, *modified_tol=1e-12*, *neighbors_algorithm='auto'*, *random_state=None*, *n_jobs=1*)

Bases: *PlannedIndividualOp*

Combined schema for expected data and hyperparameters.

This documentation is auto-generated from JSON schemas.

Parameters

- **n_neighbors** (*integer, >=5 for optimizer, <=20 for optimizer, uniform distribution, default 5*) – number of neighbors to consider for each point.
- **n_components** (*integer, >=2 for optimizer, <='X/items/maxItems', <=256 for optimizer, uniform distribution, default 2*) – number of coordinates for the manifold

- **reg** (*float, not for optimizer, default 0.001*) – regularization constant, multiplies the trace of the local covariance matrix of the distances.
- **eigen_solver** ('auto', 'arpack', *or* 'dense', default 'auto') – auto : algorithm will attempt to choose the best method for input data arpack : use arnoldi iteration in shift-invert mode
- **tol** (*float, >=1e-08 for optimizer, <=0.01 for optimizer, default 1e-06*) – Tolerance for 'arpack' method Not used if eigen_solver=='dense'.
- **max_iter** (*integer, >=10 for optimizer, <=1000 for optimizer, uniform distribution, default 100*) – maximum number of iterations for the arpack solver
- **method** ('ltsa', 'modified', 'standard', *or* 'hessian', default 'standard') – standard : use the standard locally linear embedding algorithm

  See also *constraint-1*, *constraint-2*, *constraint-3*, *constraint-3*.
- **hessian_tol** (*float, not for optimizer, default 0.0001*) – Tolerance for Hessian eigenmapping method

  See also *constraint-1*.
- **modified_tol** (*float, not for optimizer, default 1e-12*) – Tolerance for modified LLE method

  See also *constraint-2*.
- **neighbors_algorithm** ('auto', 'brute', 'kd_tree', *or* 'ball_tree', default 'auto') – algorithm to use for nearest neighbors search, passed to neighbors.NearestNeighbors instance
- **random_state** (*union type, not for optimizer, default None*) – If int, random_state is the seed used by the random number generator; If RandomState instance, random_state is the random number generator; If None, the random number generator is the RandomState instance used by *np.random*
  - integer
  - *or* numpy.random.RandomState
  - *or* None
- **n_jobs** (*union type, not for optimizer, default 1*) – The number of parallel jobs to run
  - integer
  - *or* None

Notes

constraint-1 : union type
  hessian_tol, only used if method == 'hessian'
  - hessian_tol : 0.0001
  - *or* method : 'hessian'

constraint-2 : union type
  modified_tol, only used if method == 'modified'
  - modified_tol : 1e-12
  - *or* method : 'modified'

constraint-3 : union type
  for method='hessian', n_neighbors must be greater than [n_components * (n_components + 3) / 2]
  - method : 'standard'
  - *or* method : negated type *of* 'hessian'
  - *or* any type

constraint-4 : negated type *of* 'X/isSparse'
  A sparse matrix was passed, but dense data is required. Use X.toarray() to convert to a dense numpy array.)

fit(*X*, *y=None*, ***fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

- **X** (array *of* items : array *of* items : float) – training set.
- **y** (*any type*) –

transform(*X*, *y=None*)

Transform the data.

Note: The transform method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (array *of* items : array *of* items : float) –

Returns

result – Transform new points into embedding space.

Return type

array *of* items : array *of* items : float

lale.lib.autogen.logistic_regression_cv module

class lale.lib.autogen.logistic_regression_cv.**LogisticRegressionCV**(***, *Cs=10*, *fit_intercept=True*, *cv*, *dual=False*, *penalty='l2'*, *scoring=None*, *solver='lbfgs'*, *tol=0.0001*, *max_iter=100*, *class_weight='balanced'*, *n_jobs=1*, *verbose=0*, *refit=True*, *intercept_scaling=1.0*, *multi_class='ovr'*, *random_state=None*)

Bases: *PlannedIndividualOp*

Combined schema for expected data and hyperparameters.

This documentation is auto-generated from JSON schemas.

Parameters

- **Cs** (*integer, >=10 for optimizer, <=11 for optimizer, uniform distribution, default 10*) – Each of the values in Cs describes the inverse of regularization strength
- **fit_intercept** (*boolean, default True*) – Specifies if a constant (a.k.a
- **cv** (*union type*) –

  Cross-validation as integer or as object that has a split function.

  The fit method performs cross validation on the input dataset for per trial, and uses the mean cross validation performance for optimization. This behavior is also impacted by handle_cv_failure flag. If integer: number of folds in sklearn.model_selection.StratifiedKFold. If object with split function: generator yielding (train, test) splits as arrays of indices. Can use any of the iterators from https://scikit-learn.org/stable/modules/cross_validation.html#cross-validation-iterators.

  - integer, >=1, >=3 for optimizer, <=4 for optimizer, uniform distribution, default 5
  - *or* Any, not for optimizer
- **dual** (*boolean, default False*) – Dual or primal formulation
- **penalty** ('l1' *or* 'l2', default 'l2') – Used to specify the norm used in the penalization
- **scoring** (*union type, default None*) – A string (see model evaluation documentation) or a scorer callable object / function with signature `scorer(estimator, X, y)`
  - callable, not for optimizer
  - *or* 'accuracy' *or* None
- **solver** ('newton-cg', 'lbfgs', 'liblinear', 'sag', *or* 'saga', default 'lbfgs') – Algorithm to use in the optimization problem
- **tol** (*float, >=1e-08 for optimizer, <=0.01 for optimizer, default 0.0001*) – Tolerance for stopping criteria.
- **max_iter** (*integer, >=10 for optimizer, <=1000 for optimizer, uniform distribution, default 100*) – Maximum number of iterations of the optimization algorithm.
- **class_weight** (*'balanced', not for optimizer, default 'balanced'*) – Weights associated with classes in the form `{class_label: weight}`
- **n_jobs** (*union type, not for optimizer, default 1*) – Number of CPU cores used during the cross-validation loop
  - integer
  - *or* None
- **verbose** (*integer, not for optimizer, default 0*) – For the 'liblinear', 'sag' and 'lbfgs' solvers set verbose to any positive number for verbosity.
- **refit** (*boolean, not for optimizer, default True*) – If set to True, the scores are averaged across all folds, and the coefs and the C that corresponds to the best score is taken, and a final refit is done using these parameters
- **intercept_scaling** (*float, not for optimizer, default 1.0*) – Useful only when the solver 'liblinear' is used and self.fit_intercept is set to True
- **multi_class** ('ovr', 'multinomial', *or* 'auto', default 'ovr') – If the option chosen is 'ovr', then a binary problem is fit for each label
- **random_state** (*union type, not for optimizer, default None*) – If int, random_state is the seed used by the random number generator; If RandomState instance, random_state is the random number generator; If None, the random number generator is the RandomState instance used by *np.random*.
  - integer
  - *or* numpy.random.RandomState
  - *or* None

Notes

constraint-1 : any type

constraint-2 : any type

constraint-3 : any type

constraint-4 : any type

decision_function(*X*)

Confidence scores for all classes.

Note: The decision_function method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (*union type*) – Samples.

- array *of* items : Any
- *or* array *of* items : array *of* items : float

Returns

result – Confidence scores per (sample, class) combination

Return type

Any

fit(*X*, *y=None*, ***fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

- **X** (array *of* items : array *of* items : float) – Training vector, where n_samples is the number of samples and n_features is the number of features.
- **y** (array *of* items : float) – Target vector relative to X.
- **sample_weight** (*Any, optional*) – Array of weights that are assigned to individual samples

predict(*X*, ***predict_params*)

Make predictions.

Note: The predict method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (*union type*) – Samples.

- array *of* items : Any
- *or* array *of* items : array *of* items : float

Returns

result – Predicted class label per sample.

Return type

array *of* items : float

predict_proba(*X*)

Probability estimates for all classes.

Note: The predict_proba method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (array *of* items : array *of* items : float) –

Returns

result – Returns the probability of the sample for each class in the model, where classes are ordered as they are in `self.classes_`.

Return type

array *of* items : array *of* items : float

lale.lib.autogen.max_abs_scaler module

class `lale.lib.autogen.max_abs_scaler.MaxAbsScaler`(*, *copy=True*)

Bases: *PlannedIndividualOp*

Combined schema for expected data and hyperparameters.

This documentation is auto-generated from JSON schemas.

Parameters

copy (*boolean, default True*) – Set to False to perform inplace scaling and avoid a copy (if the input is already a numpy array).

fit(*X*, *y=None*, ***fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

X (array *of* items : array *of* items : float) – The data used to compute the per-feature minimum and maximum used for later scaling along the features axis.

transform(*X*, *y=None*)

Transform the data.

Note: The transform method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (array *of* items : Any) – The data that should be scaled.

Returns

result – Scale the data

Return type

Any

lale.lib.autogen.mini_batch_dictionary_learning module

class lale.lib.autogen.mini_batch_dictionary_learning.**MiniBatchDictionaryLearning**(*, *n_components=None*, *alpha=1*, *fit_algorithm='lars'*, *n_jobs=1*, *batch_size=3*, *shuffle=True*, *dict_init=None*, *transform_algorithm='omp'*, *transform_n_nonzero_coefs=None*, *transform_alpha=None*, *verbose=False*, *split_sign=False*, *random_state=None*, *positive_code=False*, *positive_dict=False*, *transform_max_iter=1000*, *max_iter=1000*)

Bases: *PlannedIndividualOp*

Combined schema for expected data and hyperparameters.

This documentation is auto-generated from JSON schemas.

Parameters

- **n_components** (*union type, default None*) – number of dictionary elements to extract
  - integer, >=2 for optimizer, <=256 for optimizer, uniform distribution
  - *or* None
- **alpha** (*float, >=1e-10 for optimizer, <=1.0 for optimizer, loguniform distribution, default 1*) – sparsity controlling parameter
- **fit_algorithm** ('lars' *or* 'cd', default 'lars') – lars: uses the least angle regression method to solve the lasso problem (linear_model.lars_path) cd: uses the coordinate descent method to compute the Lasso solution (linear_model.Lasso)

  See also *constraint-2*.
- **n_jobs** (*union type, not for optimizer, default 1*) – Number of parallel jobs to run
  - integer
  - *or* None
- **batch_size** (*integer, >=3 for optimizer, <=128 for optimizer, uniform distribution, default 3*) – number of samples in each mini-batch
- **shuffle** (*boolean, default True*) – whether to shuffle the samples before forming batches
- **dict_init** (*union type, not for optimizer, default None*) – initial value of the dictionary for warm restart scenarios
  - array *of* items : array *of* items : float

    - *or* None
- **transform_algorithm** ('lasso_lars', 'lasso_cd', 'lars', 'omp', *or* 'threshold', default 'omp') – Algorithm used to transform the data
- **transform_n_nonzero_coefs** (*None, not for optimizer, default None*) – Number of nonzero coefficients to target in each column of the solution
- **transform_alpha** (*union type, not for optimizer, default None*) – If *algorithm='lasso_lars'* or *algorithm='lasso_cd'*, *alpha* is the penalty applied to the L1 norm
    - float
    - *or* None
- **verbose** (*boolean, not for optimizer, default False*) – To control the verbosity of the procedure.
- **split_sign** (*boolean, not for optimizer, default False*) – Whether to split the sparse feature vector into the concatenation of its negative part and its positive part
- **random_state** (*union type, not for optimizer, default None*) – If int, random_state is the seed used by the random number generator; If RandomState instance, random_state is the random number generator; If None, the random number generator is the RandomState instance used by *np.random*.
    - integer
    - *or* numpy.random.RandomState
    - *or* None
- **positive_code** (*boolean, not for optimizer, default False*) – Whether to enforce positivity when finding the code

  See also *constraint-2*.
- **positive_dict** (*boolean, not for optimizer, default False*) – Whether to enforce positivity when finding the dictionary
- **transform_max_iter** (*integer, >=100 for optimizer, <=2000 for optimizer, uniform distribution, optional, not for optimizer, default 1000*) – Maximum number of iterations to perform if algorithm='lasso_cd' or 'lasso_lars'
- **max_iter** (*integer, >=5 for optimizer, <=1000 for optimizer, uniform distribution, optional, not for optimizer, default 1000*) – total number of iterations to perform

Notes

constraint-1 : any type

constraint-2 : union type

From /decomposition/_dict_learning.py:None:_check_positive_coding, Exception: raise ValueError("Positive constraint not supported for '{}' coding method." .format(method))

- positive_code : False
- *or* fit_algorithm : negated type *of* 'omp' *or* 'lars'

fit(*X*, *y=None*, ***fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

- **X** (array *of* items : array *of* items : float) – Training vector, where n_samples in the number of samples and n_features is the number of features.
- **y** (*any type*) –

transform(*X*, *y=None*)

Transform the data.

Note: The transform method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (array *of* items : array *of* items : float) – Test data to be transformed, must have the same number of features as the data used to train the model.

Returns

result – Transformed data

Return type

array *of* items : array *of* items : float

lale.lib.autogen.mini_batch_k_means module

class lale.lib.autogen.mini_batch_k_means.**MiniBatchKMeans**(**, n_clusters=8, init='k-means++', max_iter=100, batch_size=100, verbose=0, compute_labels=True, random_state=None, tol=0.0, max_no_improvement=10, init_size=None, n_init=3, reassignment_ratio=0.01*)

Bases: *PlannedIndividualOp*

Combined schema for expected data and hyperparameters.

This documentation is auto-generated from JSON schemas.

Parameters

- **n_clusters** (*integer, >=2 for optimizer, <=8 for optimizer, uniform distribution, default 8*) – The number of clusters to form as well as the number of centroids to generate.
- **init** (*union type, default 'k-means++'*) – Method for initialization, defaults to 'k-means++': 'k-means++' : selects initial cluster centers for k-mean clustering in a smart way to speed up convergence
  - 'k-means++' *or* 'random'
  - *or* callable, not for optimizer
- **max_iter** (*integer, >=10 for optimizer, <=1000 for optimizer, uniform distribution, default 100*) – Maximum number of iterations over the complete dataset before stopping independently of any early stopping criterion heuristics.
- **batch_size** (*integer, >=3 for optimizer, <=128 for optimizer, uniform distribution, default 100*) – Size of the mini batches.
- **verbose** (*union type, not for optimizer, default 0*) – Verbosity mode.
  - boolean
  - *or* integer
- **compute_labels** (*boolean, default True*) – Compute label assignment and inertia for the complete dataset once the minibatch optimization has converged in fit.
- **random_state** (*union type, not for optimizer, default None*) – Determines random number generation for centroid initialization and random reassignment
  - integer
  - *or* numpy.random.RandomState
  - *or* None

- **tol** (*float*, *>=1e-08 for optimizer*, *<=0.01 for optimizer*, *default 0.0*) – Control early stopping based on the relative center changes as measured by a smoothed, variance-normalized of the mean center squared position changes
- **max_no_improvement** (*integer*, *>=10 for optimizer*, *<=11 for optimizer*, *uniform distribution*, *default 10*) – Control early stopping based on the consecutive number of mini batches that does not yield an improvement on the smoothed inertia
- **init_size** (*None*, *not for optimizer*, *default None*) – Number of samples to randomly sample for speeding up the initialization (sometimes at the expense of accuracy): the only algorithm is initialized by running a batch KMeans on a random subset of the data
- **n_init** (*integer*, *>=3 for optimizer*, *<=10 for optimizer*, *uniform distribution*, *default 3*) – Number of random initializations that are tried
- **reassignment_ratio** (*float*, *not for optimizer*, *default 0.01*) – Control the fraction of the maximum number of counts for a center to be reassigned

Notes

constraint-1 : any type

constraint-2 : any type

`fit`(*X*, *y=None*, ***fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

- **X** (*union type*) – Training instances to cluster
  - array *of* items : Any
  - *or* array *of* items : array *of* items : float
- **y** (*any type*) – not used, present here for API consistency by convention.
- **sample_weight** (*union type*, *optional*, *default 'deprecated'*) – The parameter *sample_weight* is deprecated in version 1.3 and will be removed in 1.5.
  - array *of* items : float
  - *or* None *or* 'deprecated'

`predict`(*X*, ***predict_params*)

Make predictions.

Note: The predict method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

- **X** (array *of* items : array *of* items : float) – New data to predict.
- **sample_weight** (*union type*, *optional*, *default None*) – The weights for each observation in X
  - array *of* items : float
  - *or* None

Returns

result – Index of the cluster each sample belongs to.

Return type

array *of* items : float

transform(*X*, *y=None*)

Transform the data.

Note: The transform method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (array *of* items : array *of* items : float) – New data to transform.

Returns

result – X transformed in the new space.

Return type

array *of* items : array *of* items : float

lale.lib.autogen.mini_batch_sparse_pca module

class lale.lib.autogen.mini_batch_sparse_pca.**MiniBatchSparsePCA**(*, *n_components=None*, *alpha=1*, *ridge_alpha=0.01*, *callback=None*, *batch_size=3*, *verbose=False*, *shuffle=True*, *n_jobs=1*, *method='lars'*, *random_state=None*, *max_no_improvement=10*, *tol=0.001*, *max_iter=1000*)

Bases: *PlannedIndividualOp*

Combined schema for expected data and hyperparameters.

This documentation is auto-generated from JSON schemas.

Parameters

- **n_components** (*union type, default None*) – number of sparse atoms to extract
  - integer, >=2 for optimizer, <=256 for optimizer, uniform distribution
  - *or* None
- **alpha** (*integer, >=1 for optimizer, <=2 for optimizer, uniform distribution, default 1*) – Sparsity controlling parameter
- **ridge_alpha** (*float, not for optimizer, default 0.01*) – Amount of ridge shrinkage to apply in order to improve conditioning when calling the transform method.
- **callback** (*union type, not for optimizer, default None*) – callable that gets invoked every five iterations
  - callable
  - *or* None
- **batch_size** (*integer, >=3 for optimizer, <=128 for optimizer, uniform distribution, default 3*) – the number of features to take in each mini batch
- **verbose** (*union type, not for optimizer, default False*) – Controls the verbosity; the higher, the more messages
  - integer
  - *or* boolean
- **shuffle** (*boolean, default True*) – whether to shuffle the data before splitting it in batches
- **n_jobs** (*union type, not for optimizer, default 1*) – Number of parallel jobs to run
  - integer
  - *or* None

- **method** ('lars' *or* 'cd', default 'lars') – lars: uses the least angle regression method to solve the lasso problem (linear_model.lars_path) cd: uses the coordinate descent method to compute the Lasso solution (linear_model.Lasso)
- **random_state** (*union type, not for optimizer, default None*) – If int, random_state is the seed used by the random number generator; If RandomState instance, random_state is the random number generator; If None, the random number generator is the RandomState instance used by *np.random*.
  - integer
  - *or* numpy.random.RandomState
  - *or* None
- **max_no_improvement** (*union type, optional, not for optimizer, default 10*) – Control early stopping based on the consecutive number of mini batches that does not yield an improvement on the smoothed cost function.
  - integer, >=1
  - *or* None

    Disable convergence detection based on cost function.
- **tol** (*float, optional, not for optimizer, default 0.001*) – Control early stopping based on the norm of the differences in the dictionary between 2 steps.

  To disable early stopping based on changes in the dictionary, set tol to 0.0.
- **max_iter** (*union type, optional, not for optimizer, default 1000*) – Maximum number of iterations over the complete dataset before stopping independently of any early stopping criterion heuristics.
  - integer, >=5 for optimizer, <=1000 for optimizer, uniform distribution
  - *or* None

    deprecated

Notes

constraint-1 : negated type *of* 'X/isSparse'
  A sparse matrix was passed, but dense data is required. Use X.toarray() to convert to a dense numpy array.

fit(*X*, *y=None*, ***fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

- **X** (array *of* items : array *of* items : float) – Training vector, where n_samples in the number of samples and n_features is the number of features.
- **y** (*any type*) –

transform(*X*, *y=None*)

Transform the data.

Note: The transform method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

- **X** (array *of* items : array *of* items : float) – Test data to be transformed, must have the same number of features as the data used to train the model.
- **ridge_alpha** (*float, optional, default 0.01*) – Amount of ridge shrinkage to apply in order to improve conditioning

Returns

result – Transformed data.

Return type
Any

lale.lib.autogen.mlp_regressor module

class lale.lib.autogen.mlp_regressor.**MLPRegressor**(**, hidden_layer_sizes='(100,)', activation='relu', solver='adam', alpha=0.0001, batch_size='auto', learning_rate='constant', learning_rate_init=0.001, power_t=0.5, max_iter=200, shuffle=True, random_state=None, tol=0.0001, verbose=False, warm_start=False, momentum=0.9, nesterovs_momentum=True, early_stopping=False, validation_fraction=0.1, beta_1=0.9, beta_2=0.999, epsilon=1e-08, n_iter_no_change=10*)

Bases: *PlannedIndividualOp*

Combined schema for expected data and hyperparameters.

This documentation is auto-generated from JSON schemas.

Parameters

- **hidden_layer_sizes** (*tuple, not for optimizer, default (100,)*) – The ith element represents the number of neurons in the ith hidden layer.
- **activation** ('identity', 'logistic', 'tanh', *or* 'relu', default 'relu') – Activation function for the hidden layer
- **solver** ('lbfgs', 'sgd', *or* 'adam', default 'adam') – The solver for weight optimization

  See also *constraint-1*, *constraint-2*, *constraint-3*, *constraint-4*, *constraint-5*, *constraint-7*, *constraint-9*, *constraint-10*, *constraint-11*, *constraint-12*.
- **alpha** (*float, >=1e-10 for optimizer, <=1.0 for optimizer, loguniform distribution, default 0.0001*) – L2 penalty (regularization term) parameter.
- **batch_size** (*union type, default 'auto'*) – Size of minibatches for stochastic optimizers
  - integer, >=3 for optimizer, <=128 for optimizer, uniform distribution
  - *or* 'auto'
- **learning_rate** ('constant', 'invscaling', *or* 'adaptive', default 'constant') – Learning rate schedule for weight updates

  See also *constraint-1*.
- **learning_rate_init** (*float, not for optimizer, default 0.001*) – The initial learning rate used

  See also *constraint-2*.
- **power_t** (*float, not for optimizer, default 0.5*) – The exponent for inverse scaling learning rate

  See also *constraint-3*.
- **max_iter** (*integer, >=10 for optimizer, <=1000 for optimizer, uniform distribution, default 200*) – Maximum number of iterations
- **shuffle** (*boolean, default True*) – Whether to shuffle samples in each iteration

  See also *constraint-4*.
- **random_state** (*union type, not for optimizer, default None*) – If int, random_state is the seed used by the random number generator; If RandomState in-

stance, random_state is the random number generator; If None, the random number generator is the RandomState instance used by *np.random*.

  - integer
  - *or* numpy.random.RandomState
  - *or* None

- **tol** (*float, >=1e-08 for optimizer, <=0.01 for optimizer, default 0.0001*) – Tolerance for the optimization
- **verbose** (*boolean, not for optimizer, default False*) – Whether to print progress messages to stdout.
- **warm_start** (*boolean, not for optimizer, default False*) – When set to True, reuse the solution of the previous call to fit as initialization, otherwise, just erase the previous solution
- **momentum** (*float, not for optimizer, default 0.9*) – Momentum for gradient descent update

  See also *constraint-5*.
- **nesterovs_momentum** (*boolean, default True*) – Whether to use Nesterov's momentum
- **early_stopping** (*boolean, not for optimizer, default False*) – Whether to use early stopping to terminate training when validation score is not improving

  See also *constraint-7*, *constraint-8*.
- **validation_fraction** (*float, not for optimizer, default 0.1*) – The proportion of training data to set aside as validation set for early stopping

  See also *constraint-8*.
- **beta_1** (*float, not for optimizer, default 0.9*) – Exponential decay rate for estimates of first moment vector in adam, should be in [0, 1)

  See also *constraint-9*.
- **beta_2** (*float, not for optimizer, default 0.999*) – Exponential decay rate for estimates of second moment vector in adam, should be in [0, 1)

  See also *constraint-10*.
- **epsilon** (*float, >=1e-08 for optimizer, <=1.35 for optimizer, loguniform distribution, default 1e-08*) – Value for numerical stability in adam

  See also *constraint-11*.
- **n_iter_no_change** (*integer, not for optimizer, default 10*) – Maximum number of epochs to not meet `tol` improvement

  See also *constraint-12*.

Notes

constraint-1 : union type
  learning_rate, only used when solver='sgd'
  - learning_rate : 'constant'
  - *or* solver : 'sgd'

constraint-2 : union type
  learning_rate_init, only used when solver='sgd' or 'adam'
  - learning_rate_init : 0.001
  - *or* solver : 'sgd' *or* 'adam'

constraint-3 : union type
  power_t, only used when solver='sgd'
  - power_t : 0.5

  - *or* solver : 'sgd'

constraint-4 : union type
: shuffle, only used when solver='sgd' or 'adam'
  - shuffle : True
  - *or* solver : 'sgd' *or* 'adam'

constraint-5 : union type
: momentum, only used when solver='sgd'
  - momentum : 0.9
  - *or* solver : 'sgd'

constraint-6 : any type

constraint-7 : union type
: early_stopping, only effective when solver='sgd' or 'adam'
  - early_stopping : False
  - *or* solver : 'sgd' *or* 'adam'

constraint-8 : union type
: validation_fraction, only used if early_stopping is true
  - validation_fraction : 0.1
  - *or* early_stopping : True

constraint-9 : union type
: beta_1, only used when solver='adam'
  - beta_1 : 0.9
  - *or* solver : 'adam'

constraint-10 : union type
: beta_2, only used when solver='adam'
  - beta_2 : 0.999
  - *or* solver : 'adam'

constraint-11 : union type
: epsilon, only used when solver='adam'
  - epsilon : 1e-08
  - *or* solver : 'adam'

constraint-12 : union type
: n_iter_no_change, only effective when solver='sgd' or 'adam'
  - n_iter_no_change : 10
  - *or* solver : 'sgd' *or* 'adam'

`fit`(*X*, *y=None*, ***fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

- **X** (*union type*) – The input data.
  - array *of* items : Any
  - *or* array *of* items : array *of* items : float
- **y** (*union type*) – The target values (class labels in classification, real numbers in regression).
  - array *of* items : float
  - *or* array *of* items : array *of* items : float

`predict`(*X*, ***predict_params*)

Make predictions.

Note: The predict method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters
: **X** (array *of* items : array *of* items : float) – The input data.

Returns
: **result** – The predicted values.

Return type
: array *of* items : array *of* items : float

lale.lib.autogen.multi_label_binarizer module

class `lale.lib.autogen.multi_label_binarizer.`**`MultiLabelBinarizer`**(*, *classes=None*, *sparse_output=False*)

Bases: `PlannedIndividualOp`

Combined schema for expected data and hyperparameters.

This documentation is auto-generated from JSON schemas.

Parameters

- **classes** (*None, not for optimizer, default None*) – Indicates an ordering for the class labels
- **sparse_output** (*boolean, default False*) – Set to true if output binary array is desired in CSR sparse format

`fit`(*X*, *y=None*, ***fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters
: **y** (*Any*) – A set of labels (any orderable and hashable object) for each sample

`transform`(*X*, *y=None*)

Transform the data.

Note: The transform method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters
: **y** (*Any*) – A set of labels (any orderable and hashable object) for each sample

Returns
: **result** – A matrix such that *y_indicator[i, j] = 1* iff *classes_[j]* is in *y[i]*, and 0 otherwise.

Return type
: Any

lale.lib.autogen.multi_task_elastic_net module

class `lale.lib.autogen.multi_task_elastic_net.`**`MultiTaskElasticNet`**(*, *alpha=1.0*, *l1_ratio=0.5*, *fit_intercept=True*, *copy_X=True*, *max_iter=1000*, *tol=0.0001*, *warm_start=False*, *random_state=None*, *selection='cyclic'*)

Bases: `PlannedIndividualOp`

Combined schema for expected data and hyperparameters.

This documentation is auto-generated from JSON schemas.

Parameters

- **alpha** (*float, >=1e-10 for optimizer, <=1.0 for optimizer, loguniform distribution, default 1.0*) – Constant that multiplies the L1/L2 term
- **l1_ratio** (*float, not for optimizer, default 0.5*) – The ElasticNet mixing parameter, with 0 < l1_ratio <= 1
- **fit_intercept** (*boolean, default True*) – whether to calculate the intercept for this model
- **copy_X** (*boolean, default True*) – If `True`, X will be copied; else, it may be overwritten.
- **max_iter** (*integer, >=10 for optimizer, <=1000 for optimizer, uniform distribution, default 1000*) – The maximum number of iterations
- **tol** (*float, >=1e-08 for optimizer, <=0.01 for optimizer, default 0.0001*) – The tolerance for the optimization: if the updates are smaller than `tol`, the optimization code checks the dual gap for optimality and continues until it is smaller than `tol`.
- **warm_start** (*boolean, not for optimizer, default False*) – When set to `True`, reuse the solution of the previous call to fit as initialization, otherwise, just erase the previous solution
- **random_state** (*union type, not for optimizer, default None*) – The seed of the pseudo random number generator that selects a random feature to update
  - integer
  - *or* numpy.random.RandomState
  - *or* None
- **selection** ('cyclic' *or* 'random', not for optimizer, default 'cyclic') – If set to 'random', a random coefficient is updated every iteration rather than looping over features sequentially by default

fit(*X*, *y=None*, ***fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

- **X** (*Any*) – Data
- **y** (*Any*) – Target

predict(*X*, ***predict_params*)

Make predictions.

Note: The predict method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (*union type*) – Samples.

- array *of* items : Any
- *or* array *of* items : array *of* items : float

Returns

result – Returns predicted values.

Return type

array *of* items : float

lale.lib.autogen.multi_task_elastic_net_cv module

class lale.lib.autogen.multi_task_elastic_net_cv.**MultiTaskElasticNetCV**(*, *l1_ratio=0.5*, *eps=0.001*, *n_alphas=100*, *alphas=None*, *fit_intercept=True*, *max_iter=1000*, *tol=0.0001*, *cv*, *copy_X=True*, *verbose=0*, *n_jobs=1*, *random_state=None*, *selection='cyclic'*)

Bases: *PlannedIndividualOp*

Combined schema for expected data and hyperparameters.

This documentation is auto-generated from JSON schemas.

Parameters

- **l1_ratio** (*float, not for optimizer, default 0.5*) – The ElasticNet mixing parameter, with 0 < l1_ratio <= 1
- **eps** (*float, >=0.001 for optimizer, <=0.1 for optimizer, loguniform distribution, default 0.001*) – Length of the path
- **n_alphas** (*integer, >=100 for optimizer, <=101 for optimizer, uniform distribution, default 100*) – Number of alphas along the regularization path
- **alphas** (*union type, not for optimizer, default None*) – List of alphas where to compute the models
  - array *of* items : Any
  - *or* None
- **fit_intercept** (*boolean, default True*) – whether to calculate the intercept for this model
- **max_iter** (*integer, >=10 for optimizer, <=1000 for optimizer, uniform distribution, default 1000*) – The maximum number of iterations
- **tol** (*float, >=1e-08 for optimizer, <=0.01 for optimizer, default 0.0001*) – The tolerance for the optimization: if the updates are smaller than `tol`, the optimization code checks the dual gap for optimality and continues until it is smaller than `tol`.
- **cv** (*union type*) –

  Cross-validation as integer or as object that has a split function.

  The fit method performs cross validation on the input dataset for per trial, and uses the mean cross validation performance for optimization. This behavior is also impacted by handle_cv_failure flag. If integer: number of folds in sklearn.model_selection.StratifiedKFold. If object with split function: generator yielding (train, test) splits as arrays of indices. Can use any of the iterators from https://scikit-learn.org/stable/modules/cross_validation.html#cross-validation-iterators.

  - integer, >=1, >=3 for optimizer, <=4 for optimizer, uniform distribution, default 5
  - *or* Any, not for optimizer
- **copy_X** (*boolean, default True*) – If `True`, X will be copied; else, it may be overwritten.

- **verbose** (*union type, not for optimizer, default 0*) – Amount of verbosity.
  - boolean
  - *or* integer
- **n_jobs** (*union type, not for optimizer, default 1*) – Number of CPUs to use during the cross validation
  - integer
  - *or* None
- **random_state** (*union type, not for optimizer, default None*) – The seed of the pseudo random number generator that selects a random feature to update
  - integer
  - *or* numpy.random.RandomState
  - *or* None
- **selection** (*string, not for optimizer, default 'cyclic'*) – If set to 'random', a random coefficient is updated every iteration rather than looping over features sequentially by default

Notes

constraint-1 : any type

fit(*X*, *y=None*, ***fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

- **X** (*Any*) – Training data
- **y** (*union type*) – Target values
  - array *of* items : float
  - *or* array *of* items : array *of* items : float

predict(*X*, ***predict_params*)

Make predictions.

Note: The predict method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (*union type*) – Samples.

- array *of* items : Any
- *or* array *of* items : array *of* items : float

Returns

result – Returns predicted values.

Return type

array *of* items : float

lale.lib.autogen.multi_task_lasso module

class lale.lib.autogen.multi_task_lasso.**MultiTaskLasso**(*, *alpha=1.0*, *fit_intercept=True*, *copy_X=True*, *max_iter=1000*, *tol=0.0001*, *warm_start=False*, *random_state=None*, *selection='cyclic'*)

Bases: *PlannedIndividualOp*

Combined schema for expected data and hyperparameters.

This documentation is auto-generated from JSON schemas.

Parameters

- **alpha** (*float*, *>=1e-10 for optimizer*, *<=1.0 for optimizer*, *loguniform distribution*, *default 1.0*) – Constant that multiplies the L1/L2 term
- **fit_intercept** (*boolean*, *default True*) – whether to calculate the intercept for this model
- **copy_X** (*boolean*, *default True*) – If True, X will be copied; else, it may be overwritten.
- **max_iter** (*integer*, *>=10 for optimizer*, *<=1000 for optimizer*, *uniform distribution*, *default 1000*) – The maximum number of iterations
- **tol** (*float*, *>=1e-08 for optimizer*, *<=0.01 for optimizer*, *default 0.0001*) – The tolerance for the optimization: if the updates are smaller than tol, the optimization code checks the dual gap for optimality and continues until it is smaller than tol.
- **warm_start** (*boolean*, *not for optimizer*, *default False*) – When set to True, reuse the solution of the previous call to fit as initialization, otherwise, just erase the previous solution
- **random_state** (*union type*, *not for optimizer*, *default None*) – The seed of the pseudo random number generator that selects a random feature to update
  - integer
  - *or* numpy.random.RandomState
  - *or* None
- **selection** (*string*, *not for optimizer*, *default 'cyclic'*) – If set to 'random', a random coefficient is updated every iteration rather than looping over features sequentially by default

fit(*X*, *y=None*, ***fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

- **X** (*Any*) – Data
- **y** (*Any*) – Target

predict(*X*, ***predict_params*)

Make predictions.

Note: The predict method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (*union type*) – Samples.

- array *of* items : Any

- *or* array *of* items : array *of* items : float

Returns
: **result** – Returns predicted values.

Return type
: array *of* items : float

lale.lib.autogen.multi_task_lasso_cv module

class lale.lib.autogen.multi_task_lasso_cv.**MultiTaskLassoCV**(***, *eps=0.001*, *n_alphas=100*, *alphas=None*, *fit_intercept=True*, *max_iter=1000*, *tol=0.0001*, *copy_X=True*, *cv*, *verbose=False*, *n_jobs=1*, *random_state=None*, *selection='cyclic'*)

Bases: *PlannedIndividualOp*

Combined schema for expected data and hyperparameters.

This documentation is auto-generated from JSON schemas.

Parameters

- **eps** (*float, >=0.001 for optimizer, <=0.1 for optimizer, loguniform distribution, default 0.001*) – Length of the path
- **n_alphas** (*integer, >=100 for optimizer, <=101 for optimizer, uniform distribution, default 100*) – Number of alphas along the regularization path
- **alphas** (*union type, not for optimizer, default None*) – List of alphas where to compute the models
  - array *of* items : Any
  - *or* None
- **fit_intercept** (*boolean, default True*) – whether to calculate the intercept for this model
- **max_iter** (*integer, >=10 for optimizer, <=1000 for optimizer, uniform distribution, default 1000*) – The maximum number of iterations.
- **tol** (*float, >=1e-08 for optimizer, <=0.01 for optimizer, default 0.0001*) – The tolerance for the optimization: if the updates are smaller than `tol`, the optimization code checks the dual gap for optimality and continues until it is smaller than `tol`.
- **copy_X** (*boolean, default True*) – If `True`, X will be copied; else, it may be overwritten.
- **cv** (*union type*) –

  Cross-validation as integer or as object that has a split function.
  : The fit method performs cross validation on the input dataset for per trial, and uses the mean cross validation performance for optimization. This behavior is also impacted by handle_cv_failure flag. If integer: number of folds in sklearn.model_selection.StratifiedKFold. If object with split function: generator yielding (train, test) splits as arrays of indices. Can use any of the iterators from https://scikit-learn.org/stable/modules/cross_validation.html#cross-validation-iterators.

  - integer, >=1, >=3 for optimizer, <=4 for optimizer, uniform distribution, default 5
  - *or* Any, not for optimizer

- **verbose** (*union type, not for optimizer, default False*) – Amount of verbosity.
  - boolean
  - *or* integer
- **n_jobs** (*union type, not for optimizer, default 1*) – Number of CPUs to use during the cross validation
  - integer
  - *or* None
- **random_state** (*union type, not for optimizer, default None*) – The seed of the pseudo random number generator that selects a random feature to update
  - integer
  - *or* numpy.random.RandomState
  - *or* None
- **selection** (*string, not for optimizer, default 'cyclic'*) – If set to 'random', a random coefficient is updated every iteration rather than looping over features sequentially by default

Notes

constraint-1 : any type

fit(*X*, *y=None*, ***fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

- **X** (*Any*) – Training data
- **y** (*union type*) – Target values
  - array *of* items : float
  - *or* array *of* items : array *of* items : float

predict(*X*, ***predict_params*)

Make predictions.

Note: The predict method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (*union type*) – Samples.

- array *of* items : Any
- *or* array *of* items : array *of* items : float

Returns

result – Returns predicted values.

Return type

array *of* items : float

lale.lib.autogen.nearest_centroid module

class lale.lib.autogen.nearest_centroid.**NearestCentroid**(**, metric='euclidean', shrink_threshold=None*)

Bases: *PlannedIndividualOp*

Combined schema for expected data and hyperparameters.

This documentation is auto-generated from JSON schemas.

Parameters

- **metric** (*union type, default 'euclidean'*) – The metric to use when calculating distance between instances in a feature array
  - callable, not for optimizer
  - *or* 'cityblock', 'cosine', 'euclidean', 'l1', 'l2', 'manhattan', 'braycurtis', 'canberra', 'chebyshev', 'correlation', 'dice', 'hamming', 'jaccard', 'kulsinski', 'mahalanobis', 'minkowski', 'rogerstanimoto', 'russellrao', 'seuclidean', 'sokalmichener', 'sokalsneath', 'sqeuclidean', *or* 'yule'
- **shrink_threshold** (*union type, default None*) – Threshold for shrinking centroids to remove features.
  - float, >=0.0 for optimizer, <=1.0 for optimizer, uniform distribution
  - *or* None

  See also *constraint-1*.

Notes

constraint-1 : union type

threshold shrinking not supported for sparse input

- negated type *of* 'X/isSparse'
- *or* shrink_threshold : None

fit(*X, y=None, **fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

- **X** (array *of* items : array *of* items : float) – Training vector, where n_samples is the number of samples and n_features is the number of features
- **y** (array *of* items : float) – Target values (integers)

predict(*X, **predict_params*)

Make predictions.

Note: The predict method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (array *of* items : array *of* items : float) –

Returns

result – Perform classification on an array of test vectors X.

Return type

array *of* items : float

lale.lib.autogen.nu_svc module

class lale.lib.autogen.nu_svc.**NuSVC**(**, nu=0.5, kernel='rbf', degree=3, gamma='scale', coef0=0.0, shrinking=True, probability=False, tol=0.001, cache_size=200, class_weight='balanced', verbose=False, max_iter=-1, decision_function_shape='ovr', break_ties=False, random_state=None*)

Bases: `PlannedIndividualOp`

Combined schema for expected data and hyperparameters.

This documentation is auto-generated from JSON schemas.

Parameters

- **nu** (*float, not for optimizer, default 0.5*) – An upper bound on the fraction of training errors and a lower bound of the fraction of support vectors
- **kernel** ('linear', 'poly', 'precomputed', 'sigmoid', *or* 'rbf', default 'rbf') – Specifies the kernel type to be used in the algorithm
- **degree** (*integer, >=2 for optimizer, <=3 for optimizer, uniform distribution, default 3*) – Degree of the polynomial kernel function ('poly')
- **gamma** (*union type, default 'scale'*) – Kernel coefficient for 'rbf', 'poly' and 'sigmoid'
  - float, not for optimizer
  - *or* 'scale' *or* 'auto'
- **coef0** (*float, not for optimizer, default 0.0*) – Independent term in kernel function
- **shrinking** (*boolean, default True*) – Whether to use the shrinking heuristic.
- **probability** (*boolean, default False*) – Whether to enable probability estimates
- **tol** (*float, >=1e-08 for optimizer, <=0.01 for optimizer, default 0.001*) – Tolerance for stopping criterion.
- **cache_size** (*float, >=0.0 for optimizer, <=1.0 for optimizer, uniform distribution, default 200*) – Specify the size of the kernel cache (in MB).
- **class_weight** ('dict' *or* 'balanced', not for optimizer, default 'balanced') – Set the parameter C of class i to class_weight[i]*C for SVC
- **verbose** (*boolean, not for optimizer, default False*) – Enable verbose output
- **max_iter** (*integer, >=10 for optimizer, <=1000 for optimizer, uniform distribution, default -1*) – Hard limit on iterations within solver, or -1 for no limit.
- **decision_function_shape** ('ovr' *or* 'ovo', default 'ovr') – Whether to return a one-vs-rest ('ovr') decision function of shape (n_samples, n_classes) as all other classifiers, or the original one-vs-one ('ovo') decision function of libsvm which has shape (n_samples, n_classes * (n_classes - 1) / 2)
- **break_ties** (*boolean, not for optimizer, default False*) – If true, decision_function_shape='ovr', and number of classes > 2, predict will break ties according to the confidence values of decision_function; otherwise the first class among the tied classes is returned.
- **random_state** (*union type, not for optimizer, default None*) – The seed of the pseudo random number generator used when shuffling the data for probability estimates
  - integer
  - *or* numpy.random.RandomState
  - *or* None

Notes

constraint-1 : any type

`decision_function`(*X*)

Confidence scores for all classes.

Note: The decision_function method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters
X (array *of* items : array *of* items : float) –

Returns
result – Returns the decision function of the sample for each class in the model

Return type
Any

`fit`(*X*, *y=None*, ***fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters
- **X** (array *of* items : array *of* items : float) – Training vectors, where n_samples is the number of samples and n_features is the number of features
- **y** (array *of* items : float) – Target values (class labels in classification, real numbers in regression)
- **sample_weight** (array, optional *of* items : float) – Per-sample weights

`predict`(*X*, ***predict_params*)

Make predictions.

Note: The predict method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters
X (array *of* items : array *of* items : float) – For kernel="precomputed", the expected shape of X is [n_samples_test, n_samples_train]

Returns
result – Class labels for samples in X.

Return type
array *of* items : float

`predict_proba`(*X*)

Probability estimates for all classes.

Note: The predict_proba method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters
X (array *of* items : array *of* items : float) – For kernel="precomputed", the expected shape of X is [n_samples_test, n_samples_train]

Returns
result – Returns the probability of the sample for each class in the model

Return type
array *of* items : array *of* items : float

lale.lib.autogen.nu_svr module

class lale.lib.autogen.nu_svr.**NuSVR**(**, nu=0.5, C=1.0, kernel='rbf', degree=3, gamma='scale', coef0=0.0, shrinking=True, tol=0.001, cache_size=200, verbose=False, max_iter=-1*)

Bases: *PlannedIndividualOp*

Combined schema for expected data and hyperparameters.

This documentation is auto-generated from JSON schemas.

Parameters

- **nu** (*float, not for optimizer, default 0.5*) – An upper bound on the fraction of training errors and a lower bound of the fraction of support vectors
- **C** (*float, not for optimizer, default 1.0*) – Penalty parameter C of the error term.
- **kernel** ('linear', 'poly', 'precomputed', 'sigmoid', *or* 'rbf', default 'rbf') – Specifies the kernel type to be used in the algorithm
- **degree** (*integer, >=2 for optimizer, <=3 for optimizer, uniform distribution, default 3*) – Degree of the polynomial kernel function ('poly')
- **gamma** (*union type, default 'scale'*) – Kernel coefficient for 'rbf', 'poly' and 'sigmoid'
  - float, not for optimizer
  - *or* 'scale' *or* 'auto'
- **coef0** (*float, not for optimizer, default 0.0*) – Independent term in kernel function
- **shrinking** (*boolean, default True*) – Whether to use the shrinking heuristic.
- **tol** (*float, >=1e-08 for optimizer, <=0.01 for optimizer, default 0.001*) – Tolerance for stopping criterion.
- **cache_size** (*float, >=0.0 for optimizer, <=1.0 for optimizer, uniform distribution, default 200*) – Specify the size of the kernel cache (in MB).
- **verbose** (*boolean, not for optimizer, default False*) – Enable verbose output
- **max_iter** (*integer, >=10 for optimizer, <=1000 for optimizer, uniform distribution, default -1*) – Hard limit on iterations within solver, or -1 for no limit.

Notes

constraint-1 : any type

fit(*X, y=None, **fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

- **X** (array *of* items : array *of* items : float) – Training vectors, where n_samples is the number of samples and n_features is the number of features
- **y** (array *of* items : float) – Target values (class labels in classification, real numbers in regression)
- **sample_weight** (array, optional *of* items : float) – Per-sample weights

predict(*X*, ***predict_params*)

Make predictions.

Note: The predict method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (array *of* items : array *of* items : float) – For kernel="precomputed", the expected shape of X is (n_samples_test, n_samples_train).

Returns

result – Perform regression on samples in X.

Return type

array *of* items : float

lale.lib.autogen.orthogonal_matching_pursuit module

class lale.lib.autogen.orthogonal_matching_pursuit.**OrthogonalMatchingPursuit**(*, *n_nonzero_coefs=None*, *tol=None*, *fit_intercept=True*, *precompute='auto'*)

Bases: *PlannedIndividualOp*

Combined schema for expected data and hyperparameters.

This documentation is auto-generated from JSON schemas.

Parameters

- **n_nonzero_coefs** (*union type, default None*) – Desired number of non-zero entries in the solution
  - integer, >=500 for optimizer, <=501 for optimizer, uniform distribution
  - *or* None
- **tol** (*union type, default None*) – Maximum norm of the residual
  - float, >=1e-08 for optimizer, <=0.01 for optimizer
  - *or* None
- **fit_intercept** (*boolean, default True*) – whether to calculate the intercept for this model
- **precompute** (True, False, *or* 'auto', default 'auto') – Whether to use a precomputed Gram and Xy matrix to speed up calculations

fit(*X*, *y=None*, ***fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

- **X** (array *of* items : array *of* items : float) – Training data.
- **y** (*union type*) – Target values
  - array *of* items : float
  - *or* array *of* items : array *of* items : float

predict(*X*, ***predict_params*)

Make predictions.

Note: The predict method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters
X (*union type*) – Samples.
- array *of* items : Any
- *or* array *of* items : array *of* items : float

Returns
result – Returns predicted values.

Return type
array *of* items : float

lale.lib.autogen.orthogonal_matching_pursuit_cv module

`class lale.lib.autogen.orthogonal_matching_pursuit_cv.OrthogonalMatchingPursuitCV(*, copy=True, fit_intercept=True, max_iter=None, cv, n_jobs=1, verbose=False)`

Bases: *PlannedIndividualOp*

Combined schema for expected data and hyperparameters.

This documentation is auto-generated from JSON schemas.

Parameters
- **copy** (*boolean, default True*) – Whether the design matrix X must be copied by the algorithm
- **fit_intercept** (*boolean, default True*) – whether to calculate the intercept for this model
- **max_iter** (*union type, default None*) – Maximum numbers of iterations to perform, therefore maximum features to include
  - integer, >=10 for optimizer, <=1000 for optimizer, uniform distribution
  - *or* None
- **cv** (*union type*) –
  Cross-validation as integer or as object that has a split function.
  The fit method performs cross validation on the input dataset for per trial, and uses the mean cross validation performance for optimization. This behavior is also impacted by handle_cv_failure flag. If integer: number of folds in sklearn.model_selection.StratifiedKFold. If object with split function: generator yielding (train, test) splits as arrays of indices. Can use any of the iterators from https://scikit-learn.org/stable/modules/cross_validation.html#cross-validation-iterators.
  - integer, >=1, >=3 for optimizer, <=4 for optimizer, uniform distribution, default 5
  - *or* Any, not for optimizer
- **n_jobs** (*union type, not for optimizer, default 1*) – Number of CPUs to use during the cross validation
  - integer
  - *or* None
- **verbose** (*union type, not for optimizer, default False*) – Sets the verbosity amount
  - boolean
  - *or* integer

Notes

constraint-1 : any type

`fit`(*X*, *y=None*, ***fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

- **X** (array *of* items : array *of* items : float) – Training data.
- **y** (array *of* items : float) – Target values

`predict`(*X*, ***predict_params*)

Make predictions.

Note: The predict method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (*union type*) – Samples.

- array *of* items : Any
- *or* array *of* items : array *of* items : float

Returns

result – Returns predicted values.

Return type

array *of* items : float

lale.lib.autogen.passive_aggressive_regressor module

`class lale.lib.autogen.passive_aggressive_regressor.PassiveAggressiveRegressor`(**, C=1.0, fit_intercept=True, max_iter=1000, tol=0.001, early_stopping=False, validation_fraction=0.1, n_iter_no_change=5, shuffle=True, verbose=0, loss='epsilon_insensitive', epsilon=0.1, random_state=None, warm_start=False, average=False*)

Bases: *PlannedIndividualOp*

Combined schema for expected data and hyperparameters.

This documentation is auto-generated from JSON schemas.

Parameters

- **C** (*float, not for optimizer, default 1.0*) – Maximum step size (regularization)

- **fit_intercept** (*boolean, default True*) – Whether the intercept should be estimated or not
- **max_iter** (*integer, >=10 for optimizer, <=1000 for optimizer, uniform distribution, default 1000*) – The maximum number of passes over the training data (aka epochs)
- **tol** (*union type, default 0.001*) – The stopping criterion
  - float, >=1e-08 for optimizer, <=0.01 for optimizer
  - *or* None
- **early_stopping** (*boolean, not for optimizer, default False*) – Whether to use early stopping to terminate training when validation

  See also *constraint-2*.
- **validation_fraction** (*float, not for optimizer, default 0.1*) – The proportion of training data to set aside as validation set for early stopping

  See also *constraint-2*.
- **n_iter_no_change** (*integer, not for optimizer, default 5*) – Number of iterations with no improvement to wait before early stopping
- **shuffle** (*boolean, default True*) – Whether or not the training data should be shuffled after each epoch.
- **verbose** (*integer, not for optimizer, default 0*) – The verbosity level
- **loss** ('huber', 'squared_epsilon_insensitive', 'squared_loss', *or* 'epsilon_insensitive', default 'epsilon_insensitive') – The loss function to be used: epsilon_insensitive: equivalent to PA-I in the reference paper
- **epsilon** (*float, >=1e-08 for optimizer, <=1.35 for optimizer, loguniform distribution, default 0.1*) – If the difference between the current prediction and the correct label is below this threshold, the model is not updated.
- **random_state** (*union type, not for optimizer, default None*) – The seed of the pseudo random number generator to use when shuffling the data
  - integer
  - *or* numpy.random.RandomState
  - *or* None
- **warm_start** (*boolean, not for optimizer, default False*) – When set to True, reuse the solution of the previous call to fit as initialization, otherwise, just erase the previous solution
- **average** (*union type, not for optimizer, default False*) – When set to True, computes the averaged SGD weights and stores the result in the `coef_` attribute
  - boolean
  - *or* integer

Notes

constraint-1 : any type

constraint-2 : union type

validation_fraction, only used if early_stopping is true

- validation_fraction : 0.1
- *or* early_stopping : True

fit(*X*, *y=None*, ***fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

- **X** (array *of* items : array *of* items : float) – Training data
- **y** (array *of* items : float) – Target values
- **coef_init** (array, optional *of* items : float) – The initial coefficients to warm-start the optimization.
- **intercept_init** (array, optional *of* items : float) – The initial intercept to warm-start the optimization.

predict(*X*, ***predict_params*)

Make predictions.

Note: The predict method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (array *of* items : array *of* items : float) –

Returns

result – Predicted target values per element in X.

Return type

array *of* items : float

lale.lib.autogen.perceptron module

class lale.lib.autogen.perceptron.**Perceptron**(**, penalty=None, alpha=0.0001, fit_intercept=True, max_iter=None, tol=None, shuffle=True, verbose=0, eta0=1.0, n_jobs=1, random_state=None, early_stopping=False, validation_fraction=0.1, n_iter_no_change=5, class_weight='balanced', warm_start=False*)

Bases: *PlannedIndividualOp*

Combined schema for expected data and hyperparameters.

This documentation is auto-generated from JSON schemas.

Parameters

- **penalty** (*None, not for optimizer, default None*) – The penalty (aka regularization term) to be used
- **alpha** (*float, >=1e-10 for optimizer, <=1.0 for optimizer, loguniform distribution, default 0.0001*) – Constant that multiplies the regularization term if regularization is used
- **fit_intercept** (*boolean, default True*) – Whether the intercept should be estimated or not
- **max_iter** (*union type, default None*) – The maximum number of passes over the training data (aka epochs)
  - integer, >=10 for optimizer, <=1000 for optimizer, uniform distribution
  - *or* None
- **tol** (*union type, default None*) – The stopping criterion
  - float, >=1e-08 for optimizer, <=0.01 for optimizer
  - *or* None
- **shuffle** (*boolean, default True*) – Whether or not the training data should be shuffled after each epoch.
- **verbose** (*integer, not for optimizer, default 0*) – The verbosity level
- **eta0** (*float, >=0.01 for optimizer, <=1.0 for optimizer, loguniform distribution, default 1.0*) – Constant by which the updates are multiplied
- **n_jobs** (*union type, not for optimizer, default 1*) – The number of CPUs to use to do the OVA (One Versus All, for multi-class problems) computation

  - integer
  - *or* None
- **random_state** (*union type, not for optimizer, default None*) – The seed of the pseudo random number generator to use when shuffling the data
  - integer
  - *or* numpy.random.RandomState
  - *or* None
- **early_stopping** (*boolean, not for optimizer, default False*) – Whether to use early stopping to terminate training when validation

  See also *constraint-2*.
- **validation_fraction** (*float, not for optimizer, default 0.1*) – The proportion of training data to set aside as validation set for early stopping

  See also *constraint-2*.
- **n_iter_no_change** (*integer, not for optimizer, default 5*) – Number of iterations with no improvement to wait before early stopping
- **class_weight** (*'balanced', not for optimizer, default 'balanced'*) – Preset for the class_weight fit parameter
- **warm_start** (*boolean, not for optimizer, default False*) – When set to True, reuse the solution of the previous call to fit as initialization, otherwise, just erase the previous solution

Notes

constraint-1 : any type

constraint-2 : union type

validation_fraction, only used if early_stopping is true

- validation_fraction : 0.1
- *or* early_stopping : True

decision_function(*X*)

Confidence scores for all classes.

Note: The decision_function method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (*union type*) – Samples.
- array *of* items : Any
- *or* array *of* items : array *of* items : float

Returns

result – Confidence scores per (sample, class) combination

Return type

Any

fit(*X*, *y=None*, ***fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters
- **X** (array *of* items : array *of* items : float) – Training data
- **y** (array *of* items : float) – Target values
- **coef_init** (array, optional *of* items : array *of* items : float) – The initial coefficients to warm-start the optimization.

- **intercept_init** (array, optional *of* items : float) – The initial intercept to warm-start the optimization.
- **sample_weight** (*union type, optional, default None*) – Weights applied to individual samples
  - array *of* items : float
  - *or* None

predict(*X*, ***predict_params*)

Make predictions.

Note: The predict method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (*union type*) – Samples.
- array *of* items : Any
- *or* array *of* items : array *of* items : float

Returns

result – Predicted class label per sample.

Return type

array *of* items : float

lale.lib.autogen.pls_canonical module

class lale.lib.autogen.pls_canonical.**PLSCanonical**(*, *n_components=2*, *scale=True*, *algorithm='nipals'*, *max_iter=500*, *tol=1e-06*, *copy=True*)

Bases: *PlannedIndividualOp*

Combined schema for expected data and hyperparameters.

This documentation is auto-generated from JSON schemas.

Parameters

- **n_components** (*integer, >=2 for optimizer, <=256 for optimizer, uniform distribution, default 2*) – Number of components to keep
- **scale** (*boolean, default True*) – Option to scale data
- **algorithm** ('nipals' *or* 'svd', default 'nipals') – The algorithm used to estimate the weights
- **max_iter** (*integer, >=10 for optimizer, <=1000 for optimizer, uniform distribution, default 500*) – the maximum number of iterations of the NIPALS inner loop (used only if algorithm="nipals")
- **tol** (*float, >=1e-08 for optimizer, <=0.01 for optimizer, default 1e-06*) – the tolerance used in the iterative algorithm
- **copy** (*boolean, default True*) – Whether the deflation should be done on a copy

Notes

constraint-1 : any type

`fit`(*X*, *y=None*, ***fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

- **X** (array *of* items : array *of* items : float) – Training vectors, where n_samples is the number of samples and n_features is the number of predictors.
- **Y** (array, optional *of* items : array *of* items : float) – Target vectors, where n_samples is the number of samples and n_targets is the number of response variables.

`predict`(*X*, ***predict_params*)

Make predictions.

Note: The predict method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

- **X** (array *of* items : array *of* items : float) – Training vectors, where n_samples is the number of samples and n_features is the number of predictors.
- **copy** (*boolean, optional, default True*) – Whether to copy X and Y, or perform in-place normalization.

Returns

result – Apply the dimension reduction learned on the train data.

Return type

Any

`transform`(*X*, *y=None*)

Transform the data.

Note: The transform method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

- **X** (array *of* items : array *of* items : float) – Training vectors, where n_samples is the number of samples and n_features is the number of predictors.
- **Y** (array, optional *of* items : array *of* items : float) – Target vectors, where n_samples is the number of samples and n_targets is the number of response variables.
- **copy** (*boolean, optional, default True*) – Whether to copy X and Y, or perform in-place normalization.

Returns

result – Apply the dimension reduction learned on the train data.

Return type

Any

lale.lib.autogen.pls_regression module

class lale.lib.autogen.pls_regression.**PLSRegression**(**, n_components=2, scale=True, max_iter=500, tol=1e-06, copy=True*)

Bases: *PlannedIndividualOp*

Combined schema for expected data and hyperparameters.

This documentation is auto-generated from JSON schemas.

Parameters

- **n_components** (*integer, >=2 for optimizer, <=256 for optimizer, uniform distribution, default 2*) – Number of components to keep.
- **scale** (*boolean, default True*) – whether to scale the data
- **max_iter** (*integer, >=10 for optimizer, <=1000 for optimizer, uniform distribution, default 500*) – the maximum number of iterations of the NIPALS inner loop (used only if algorithm="nipals")
- **tol** (*float, >=1e-08 for optimizer, <=0.01 for optimizer, default 1e-06*) – Tolerance used in the iterative algorithm default 1e-06.
- **copy** (*boolean, default True*) – Whether the deflation should be done on a copy

Notes

constraint-1 : any type

fit(*X, y=None, **fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

- **X** (array *of* items : array *of* items : float) – Training vectors, where n_samples is the number of samples and n_features is the number of predictors.
- **Y** (array, optional *of* items : array *of* items : float) – Target vectors, where n_samples is the number of samples and n_targets is the number of response variables.

predict(*X, **predict_params*)

Make predictions.

Note: The predict method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

- **X** (array *of* items : array *of* items : float) – Training vectors, where n_samples is the number of samples and n_features is the number of predictors.
- **copy** (*boolean, optional, default True*) – Whether to copy X and Y, or perform in-place normalization.

Returns

result – Apply the dimension reduction learned on the train data.

Return type

Any

transform(*X, y=None*)

Transform the data.

Note: The transform method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

- **X** (array *of* items : array *of* items : float) – Training vectors, where n_samples is the number of samples and n_features is the number of predictors.
- **Y** (array, optional *of* items : array *of* items : float) – Target vectors, where n_samples is the number of samples and n_targets is the number of response variables.
- **copy** (*boolean, optional, default True*) – Whether to copy X and Y, or perform in-place normalization.

Returns

result – Apply the dimension reduction learned on the train data.

Return type

Any

lale.lib.autogen.plssvd module

class `lale.lib.autogen.plssvd.PLSSVD`(**, n_components=2, scale=True, copy=True*)

Bases: `PlannedIndividualOp`

Combined schema for expected data and hyperparameters.

This documentation is auto-generated from JSON schemas.

Parameters

- **n_components** (*integer, >=2 for optimizer, <=256 for optimizer, uniform distribution, default 2*) – Number of components to keep.
- **scale** (*boolean, default True*) – Whether to scale X and Y.
- **copy** (*boolean, default True*) – Whether to copy X and Y, or perform in-place computations.

fit(*X, y=None, **fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

- **X** (array *of* items : array *of* items : float) – Training vectors, where n_samples is the number of samples and n_features is the number of predictors.
- **Y** (array, optional *of* items : array *of* items : float) – Target vectors, where n_samples is the number of samples and n_targets is the number of response variables.

transform(*X, y=None*)

Transform the data.

Note: The transform method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

- **X** (array *of* items : array *of* items : float) – Training vectors, where n_samples is the number of samples and n_features is the number of predictors.
- **Y** (array, optional *of* items : array *of* items : float) – Target vectors, where n_samples is the number of samples and n_targets is the number of response variables.

Returns

result – Apply the dimension reduction learned on the train data.

Return type
Any

lale.lib.autogen.power_transformer module

class lale.lib.autogen.power_transformer.**PowerTransformer**(**, method='yeo-johnson', standardize=True, copy=True*)

Bases: `PlannedIndividualOp`

Combined schema for expected data and hyperparameters.

This documentation is auto-generated from JSON schemas.

Parameters

- **method** ('yeo-johnson' *or* 'box-cox', default 'yeo-johnson') – The power transform method
- **standardize** (*boolean, default True*) – Set to True to apply zero-mean, unit-variance normalization to the transformed output.
- **copy** (*boolean, not for optimizer, default True*) – Set to False to perform inplace computation during transformation.

Notes

constraint-1 : any type

constraint-2 : negated type *of* 'X/isSparse'
FA sparse matrix was passed, but dense data is required. Use X.toarray() to convert to a dense numpy array.

fit(*X, y=None, **fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

- **X** (array *of* items : array *of* items : float) – The data used to estimate the optimal transformation parameters.
- **y** (*any type*) –

transform(*X, y=None*)

Transform the data.

Note: The transform method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters
X (array *of* items : array *of* items : float) – The data to be transformed using a power transformation.

Returns
result – The transformed data.

Return type
array *of* items : array *of* items : float

lale.lib.autogen.radius_neighbors_classifier module

class lale.lib.autogen.radius_neighbors_classifier.**RadiusNeighborsClassifier**(**, radius=1.0, weights='uniform', algorithm='auto', leaf_size=30, p=2, metric='minkowski', outlier_label=None, metric_params=None, n_jobs=None*)

Bases: *PlannedIndividualOp*

Combined schema for expected data and hyperparameters.

This documentation is auto-generated from JSON schemas.

Parameters

- **radius** (*float, not for optimizer, default 1.0*) – Range of parameter space to use by default for `radius_neighbors()` queries.
- **weights** (*union type, default 'uniform'*) – weight function used in prediction
  - callable, not for optimizer
  - *or* 'distance' *or* 'uniform'
- **algorithm** ('auto', 'ball_tree', 'kd_tree', *or* 'brute', default 'auto') – Algorithm used to compute the nearest neighbors: - 'ball_tree' will use `BallTree` - 'kd_tree' will use `KDTree` - 'brute' will use a brute-force search
- **leaf_size** (*integer, >=30 for optimizer, <=31 for optimizer, uniform distribution, default 30*) – Leaf size passed to BallTree or KDTree
- **p** (*integer, >=1 for optimizer, <=3 for optimizer, uniform distribution, default 2*) – Power parameter for the Minkowski metric
- **metric** (*union type, default 'minkowski'*) – the distance metric to use for the tree
  - callable, not for optimizer
  - *or* 'euclidean', 'manhattan', 'minkowski', *or* 'precomputed'
- **outlier_label** (*union type, default None*) – Label, which is given for outlier samples (samples with no neighbors on given radius)
  - integer, not for optimizer
  - *or* 'most_frequent'
  - *or* None
- **metric_params** (*union type, not for optimizer, default None*) – Additional keyword arguments for the metric function.
  - dict
  - *or* None
- **n_jobs** (*union type, not for optimizer, default None*) – The number of parallel jobs to run for neighbors search
  - integer
  - *or* None

fit(*X, y=None, **fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters
- **X** (array *of* items : Any) – Training data
- **y** (array *of* items : Any) – Target values of shape = [n_samples] or [n_samples, n_outputs]

`predict`(*X*, ***predict_params*)

Make predictions.

Note: The predict method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters
X (*Any*) – Test samples.

Returns
result – Class labels for each data sample.
- array *of* items : float
- *or* array *of* items : array *of* items : float

Return type
union type

lale.lib.autogen.radius_neighbors_regressor module

class `lale.lib.autogen.radius_neighbors_regressor.RadiusNeighborsRegressor`(***, *radius=1.0*, *weights='uniform'*, *algorithm='auto'*, *leaf_size=30*, *p=2*, *metric='minkowski'*, *metric_params=None*, *n_jobs=None*)

Bases: *PlannedIndividualOp*

Combined schema for expected data and hyperparameters.

This documentation is auto-generated from JSON schemas.

Parameters
- **radius** (*float, not for optimizer, default 1.0*) – Range of parameter space to use by default for `radius_neighbors()` queries.
- **weights** (*union type, default 'uniform'*) – weight function used in prediction
  - callable, not for optimizer
  - *or* 'distance' *or* 'uniform'
- **algorithm** ('auto', 'ball_tree', 'kd_tree', *or* 'brute', default 'auto') – Algorithm used to compute the nearest neighbors: - 'ball_tree' will use `BallTree` - 'kd_tree' will use `KDTree` - 'brute' will use a brute-force search
- **leaf_size** (*integer, >=30 for optimizer, <=31 for optimizer, uniform distribution, default 30*) – Leaf size passed to BallTree or KDTree
- **p** (*integer, >=1 for optimizer, <=3 for optimizer, uniform distribution, default 2*) – Power parameter for the Minkowski metric
- **metric** (*union type, default 'minkowski'*) – the distance metric to use for the tree
  - callable, not for optimizer
  - *or* 'euclidean', 'manhattan', 'minkowski', *or* 'precomputed'
- **metric_params** (*union type, not for optimizer, default None*) – Additional keyword arguments for the metric function.

  - dict
  - *or* None
- **n_jobs** (*union type, not for optimizer, default None*) – The number of parallel jobs to run for neighbors search
  - integer
  - *or* None

fit(*X*, *y=None*, ***fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

- **X** (array *of* items : Any) – Training data
- **y** (array *of* items : Any) – Target values, array of float values, shape = [n_samples] or [n_samples, n_outputs]

predict(*X*, ***predict_params*)

Make predictions.

Note: The predict method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (*Any*) – Test samples.

Returns

result – Target values

Return type

Any

lale.lib.autogen.random_trees_embedding module

class lale.lib.autogen.random_trees_embedding.**RandomTreesEmbedding**(**, n_estimators=10, max_depth=5, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_leaf_nodes=None, min_impurity_decrease=0.0, sparse_output=True, n_jobs=1, random_state=None, verbose=0, warm_start=False*)

Bases: *PlannedIndividualOp*

Combined schema for expected data and hyperparameters.

This documentation is auto-generated from JSON schemas.

Parameters

- **n_estimators** (*integer, >=10 for optimizer, <=100 for optimizer, uniform distribution, default 10*) – Number of trees in the forest
- **max_depth** (*integer, >=3 for optimizer, <=5 for optimizer, uniform distribution, default 5*) – The maximum depth of each tree

- **min_samples_split** (*union type, default 2*) – The minimum number of samples required to split an internal node: - If int, then consider *min_samples_split* as the minimum number
  - integer, not for optimizer
  - *or* float, >=0.01 for optimizer, <=0.5 for optimizer, uniform distribution
- **min_samples_leaf** (*union type, default 1*) – The minimum number of samples required to be at a leaf node
  - integer, not for optimizer
  - *or* float, >=0.01 for optimizer, <=0.5 for optimizer, uniform distribution
- **min_weight_fraction_leaf** (*float, not for optimizer, default 0.0*) – The minimum weighted fraction of the sum total of weights (of all the input samples) required to be at a leaf node
- **max_leaf_nodes** (*union type, not for optimizer, default None*) – Grow trees with `max_leaf_nodes` in best-first fashion
  - integer
  - *or* None
- **min_impurity_decrease** (*float, not for optimizer, default 0.0*) – A node will be split if this split induces a decrease of the impurity greater than or equal to this value
- **sparse_output** (*boolean, default True*) – Whether or not to return a sparse CSR matrix, as default behavior, or to return a dense array compatible with dense pipeline operators.
- **n_jobs** (*union type, not for optimizer, default 1*) – The number of jobs to run in parallel for both *fit* and *predict*
  - integer
  - *or* None
- **random_state** (*union type, not for optimizer, default None*) – If int, random_state is the seed used by the random number generator; If RandomState instance, random_state is the random number generator; If None, the random number generator is the RandomState instance used by *np.random*.
  - integer
  - *or* numpy.random.RandomState
  - *or* None
- **verbose** (*integer, not for optimizer, default 0*) – Controls the verbosity when fitting and predicting.
- **warm_start** (*boolean, not for optimizer, default False*) – When set to `True`, reuse the solution of the previous call to fit and add more estimators to the ensemble, otherwise, just fit a whole new forest

Notes

constraint-1 : any type

`fit`(*X*, *y=None*, ***fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

- **X** (*union type*) – The input samples
  - array *of* items : Any
  - *or* array *of* items : array *of* items : float
- **sample_weight** (*union type, optional*) – Sample weights
  - array *of* items : float

  - *or* None

transform(*X*, *y=None*)

Transform the data.

Note: The transform method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (*union type*) – Input data to be transformed
- array *of* items : Any
- *or* array *of* items : array *of* items : float

Returns

result – Transformed dataset.

Return type

array *of* items : array *of* items : float

lale.lib.autogen.ransac_regressor module

class lale.lib.autogen.ransac_regressor.**RANSACRegressor**(**, min_samples=None, residual_threshold=None, is_data_valid=None, is_model_valid=None, max_trials=100, max_skips=inf, stop_n_inliers=inf, stop_score=inf, stop_probability=0.99, loss='absolute_error', random_state=None, estimator=None*)

Bases: *PlannedIndividualOp*

Combined schema for expected data and hyperparameters.

This documentation is auto-generated from JSON schemas.

Parameters

- **min_samples** (*union type, default None*) – Minimum number of samples chosen randomly from original data
  - float, >=0.0 for optimizer, <=1.0 for optimizer, uniform distribution
  - *or* None
- **residual_threshold** (*union type, not for optimizer, default None*) – Maximum residual for a data sample to be classified as an inlier
  - float
  - *or* None
- **is_data_valid** (*union type, not for optimizer, default None*) – This function is called with the randomly selected data before the model is fitted to it: *is_data_valid(X, y)*
  - callable
  - *or* None
- **is_model_valid** (*union type, not for optimizer, default None*) – This function is called with the estimated model and the randomly selected data: *is_model_valid(model, X, y)*
  - callable
  - *or* None
- **max_trials** (*integer, >=100 for optimizer, <=101 for optimizer, uniform distribution, default 100*) – Maximum number of iterations for random sample selection.

- **max_skips** (*union type, default inf*) – Maximum number of iterations that can be skipped due to finding zero inliers or invalid data defined by `is_data_valid` or invalid models defined by `is_model_valid`
  - integer, not for optimizer
  - *or* inf
- **stop_n_inliers** (*union type, default inf*) – Stop iteration if at least this number of inliers are found.
  - integer, not for optimizer
  - *or* inf
- **stop_score** (*float, not for optimizer, default inf*) – Stop iteration if score is greater equal than this threshold.
- **stop_probability** (*float, not for optimizer, default 0.99*) – RANSAC iteration stops if at least one outlier-free set of the training data is sampled in RANSAC
- **loss** (*union type, default 'absolute_error'*) – String inputs, "absolute_error" and "squared_error" are supported which find the absolute error and squared error per sample respectively
  - callable, not for optimizer
  - *or* 'absolute_error' *or* 'squared_error'
- **random_state** (*union type, not for optimizer, default None*) – The generator used to initialize the centers
  - integer
  - *or* numpy.random.RandomState
  - *or* None
- **estimator** (*union type, optional, not for optimizer, default None*) – Base estimator object which implements the following methods: * *fit(X, y)*: Fit model to given training data and target values
  - dict
  - *or* None

Notes

constraint-1 : any type

constraint-2 : any type

fit(*X*, *y=None*, ***fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

- **X** (*union type*) – Training data.
  - array *of* items : Any
  - *or* array *of* items : array *of* items : float
- **y** (*union type*) – Target values.
  - array *of* items : float
  - *or* array *of* items : array *of* items : float
- **sample_weight** (array, optional *of* items : float) – Individual weights for each sample raises error if sample_weight is passed and base_estimator fit method does not support it.

predict(*X*, ***predict_params*)

Make predictions.

Note: The predict method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (array *of* items : array *of* items : float) –

Returns

result – Returns predicted values.

- array *of* items : float
- *or* array *of* items : array *of* items : float

Return type

union type

lale.lib.autogen.rbf_sampler module

class lale.lib.autogen.rbf_sampler.**RBFSampler**(*, *gamma=1.0*, *n_components=100*, *random_state=None*)

Bases: *PlannedIndividualOp*

Combined schema for expected data and hyperparameters.

This documentation is auto-generated from JSON schemas.

Parameters

- **gamma** (*float, not for optimizer, default 1.0*) – Parameter of RBF kernel: exp(-gamma * x^2)
- **n_components** (*integer, >=2 for optimizer, <=256 for optimizer, uniform distribution, default 100*) – Number of Monte Carlo samples per original feature
- **random_state** (*union type, not for optimizer, default None*) – If int, random_state is the seed used by the random number generator; If RandomState instance, random_state is the random number generator; If None, the random number generator is the RandomState instance used by *np.random*.
  - integer
  - *or* numpy.random.RandomState
  - *or* None

fit(*X*, *y=None*, ***fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

X (array *of* items : array *of* items : float) – Training data, where n_samples in the number of samples and n_features is the number of features.

transform(*X*, *y=None*)

Transform the data.

Note: The transform method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (array *of* items : array *of* items : float) – New data, where n_samples in the number of samples and n_features is the number of features.

Returns

result – Apply the approximate feature map to X.

Return type

array *of* items : array *of* items : float

lale.lib.autogen.ridge_classifier_cv module

class lale.lib.autogen.ridge_classifier_cv.**RidgeClassifierCV**(***, *alphas=['0.1', '1.0', '10.0']*, *fit_intercept=True*, *scoring=None*, *cv=None*, *class_weight=None*, *store_cv_values=False*)

Bases: *PlannedIndividualOp*

Combined schema for expected data and hyperparameters.

This documentation is auto-generated from JSON schemas.

Parameters

- **alphas** (array, not for optimizer, default [0.1, 1.0, 10.0] *of* items : float) – Array of alpha values to try
- **fit_intercept** (*boolean, default True*) – Whether to calculate the intercept for this model
- **scoring** (*union type, default None*) – A string (see model evaluation documentation) or a scorer callable object / function with signature `scorer(estimator, X, y)`.
  - callable, not for optimizer
  - *or* 'accuracy' *or* None
- **cv** (*union type, default None*) –

  Cross-validation as integer or as object that has a split function.

  The fit method performs cross validation on the input dataset for per trial, and uses the mean cross validation performance for optimization. This behavior is also impacted by handle_cv_failure flag. If integer: number of folds in sklearn.model_selection.StratifiedKFold. If object with split function: generator yielding (train, test) splits as arrays of indices. Can use any of the iterators from https://scikit-learn.org/stable/modules/cross_validation.html#cross-validation-iterators.

  - integer, >=1, >=3 for optimizer, <=4 for optimizer, uniform distribution, default 5
  - *or* Any, not for optimizer
  - *or* None

  See also *constraint-2*.
- **class_weight** (*union type, default None*) – Weights associated with classes in the form `{class_label: weight}`
  - 'balanced'
  - *or* None
- **store_cv_values** (*boolean, default False*) – Flag indicating if the cross-validation values corresponding to each alpha should be stored in the `cv_values_` attribute (see below)

  See also *constraint-2*.

Notes

constraint-1 : any type

constraint-2 : union type

cv!=None and store_cv_values=True are incompatible

- cv : None
- *or* store_cv_values : False

decision_function(*X*)

Confidence scores for all classes.

Note: The decision_function method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (*union type*) – Samples.

- array *of* items : Any
- *or* array *of* items : array *of* items : float

Returns

result – Confidence scores per (sample, class) combination

Return type

Any

fit(*X*, *y=None*, ***fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

- **X** (array *of* items : array *of* items : float) – Training vectors, where n_samples is the number of samples and n_features is the number of features.
- **y** (array *of* items : float) – Target values
- **sample_weight** (*union type, optional*) – Sample weight.
  - float
  - *or* array *of* items : float

predict(*X*, ***predict_params*)

Make predictions.

Note: The predict method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (*union type*) – Samples.

- array *of* items : Any
- *or* array *of* items : array *of* items : float

Returns

result – Predicted class label per sample.

Return type

array *of* items : float

lale.lib.autogen.ridge_cv module

class lale.lib.autogen.ridge_cv.**RidgeCV**(**, alphas=['0.1', '1.0', '10.0'], fit_intercept=True, scoring=None, cv, gcv_mode=None, store_cv_values=False*)

Bases: `PlannedIndividualOp`

Combined schema for expected data and hyperparameters.

This documentation is auto-generated from JSON schemas.

Parameters

- **alphas** (array, not for optimizer, default [0.1, 1.0, 10.0] *of* items : float) – Array of alpha values to try
- **fit_intercept** (*boolean, default True*) – Whether to calculate the intercept for this model
- **scoring** (*union type, default None*) – A string (see model evaluation documentation) or a scorer callable object / function with signature `scorer(estimator, X, y)`.
  - callable, not for optimizer
  - *or* 'accuracy' *or* None
- **cv** (*union type*) –

  Cross-validation as integer or as object that has a split function.

  The fit method performs cross validation on the input dataset for per trial, and uses the mean cross validation performance for optimization. This behavior is also impacted by handle_cv_failure flag. If integer: number of folds in sklearn.model_selection.StratifiedKFold. If object with split function: generator yielding (train, test) splits as arrays of indices. Can use any of the iterators from https://scikit-learn.org/stable/modules/cross_validation.html#cross-validation-iterators.

  - integer, >=1, >=3 for optimizer, <=4 for optimizer, uniform distribution, default 5
  - *or* Any, not for optimizer
- **gcv_mode** (None, 'auto', 'svd', *or* 'eigen', default None) – Flag indicating which strategy to use when performing Generalized Cross-Validation
- **store_cv_values** (*boolean, default False*) – Flag indicating if the cross-validation values corresponding to each alpha should be stored in the `cv_values_` attribute (see below)

Notes

constraint-1 : any type

fit(*X, y=None, **fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

- **X** (array *of* items : array *of* items : float) – Training data
- **y** (*union type*) – Target values
  - array *of* items : float
  - *or* array *of* items : array *of* items : float
- **sample_weight** (*union type, optional*) – Sample weight
  - float

– *or* array *of* items : float

`predict`(*X*, ***predict_params*)

Make predictions.

Note: The predict method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (*union type*) – Samples.

- array *of* items : Any
- *or* array *of* items : array *of* items : float

Returns

result – Returns predicted values.

Return type

array *of* items : float

lale.lib.autogen.skewed_chi2_sampler module

class lale.lib.autogen.skewed_chi2_sampler.**SkewedChi2Sampler**(*, *skewedness=1.0*, *n_components=100*, *random_state=None*)

Bases: *PlannedIndividualOp*

Combined schema for expected data and hyperparameters.

This documentation is auto-generated from JSON schemas.

Parameters

- **skewedness** (*float, not for optimizer, default 1.0*) – "skewedness" parameter of the kernel
- **n_components** (*integer, >=2 for optimizer, <=256 for optimizer, uniform distribution, default 100*) – number of Monte Carlo samples per original feature
- **random_state** (*union type, not for optimizer, default None*) – If int, random_state is the seed used by the random number generator; If RandomState instance, random_state is the random number generator; If None, the random number generator is the RandomState instance used by *np.random*.
  - integer
  - *or* numpy.random.RandomState
  - *or* None

Notes

constraint-1 : negated type *of* 'X/isSparse'

A sparse matrix was passed, but dense data is required. Use X.toarray() to convert to a dense numpy array.

`fit`(*X*, *y=None*, ***fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

X (array *of* items : array *of* items : float) – Training data, where n_samples in the number of samples and n_features is the number of features.

transform(*X*, *y=None*)

Transform the data.

Note: The transform method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (array *of* items : array *of* items : float) – New data, where n_samples in the number of samples and n_features is the number of features

Returns

result – Apply the approximate feature map to X.

Return type

array *of* items : array *of* items : float

lale.lib.autogen.sparse_pca module

class lale.lib.autogen.sparse_pca.**SparsePCA**(*, *n_components=None*, *alpha=1*, *ridge_alpha=0.01*, *max_iter=1000*, *tol=1e-08*, *method='lars'*, *n_jobs=1*, *U_init=None*, *V_init=None*, *verbose=False*, *random_state=None*)

Bases: *PlannedIndividualOp*

Combined schema for expected data and hyperparameters.

This documentation is auto-generated from JSON schemas.

Parameters

- **n_components** (*union type, default None*) – Number of sparse atoms to extract.
  - integer, >=2 for optimizer, <=256 for optimizer, uniform distribution
  - *or* None
- **alpha** (*float, >=1e-10 for optimizer, <=1.0 for optimizer, loguniform distribution, default 1*) – Sparsity controlling parameter
- **ridge_alpha** (*float, not for optimizer, default 0.01*) – Amount of ridge shrinkage to apply in order to improve conditioning when calling the transform method.
- **max_iter** (*integer, >=10 for optimizer, <=1000 for optimizer, uniform distribution, default 1000*) – Maximum number of iterations to perform.
- **tol** (*float, >=1e-08 for optimizer, <=0.01 for optimizer, default 1e-08*) – Tolerance for the stopping condition.
- **method** ('lars' *or* 'cd', default 'lars') – lars: uses the least angle regression method to solve the lasso problem (linear_model.lars_path) cd: uses the coordinate descent method to compute the Lasso solution (linear_model.Lasso)
- **n_jobs** (*union type, not for optimizer, default 1*) – Number of parallel jobs to run
  - integer
  - *or* None
- **U_init** (*union type, not for optimizer, default None*) – Initial values for the loadings for warm restart scenarios.
  - array *of* items : array *of* items : float
  - *or* None
- **V_init** (*union type, not for optimizer, default None*) – Initial values for the components for warm restart scenarios.
  - array *of* items : array *of* items : float
  - *or* None
- **verbose** (*union type, not for optimizer, default False*) – Controls the verbosity; the higher, the more messages

  - integer
  - *or* boolean
- **random_state** (*union type, not for optimizer, default None*) – If int, random_state is the seed used by the random number generator; If RandomState instance, random_state is the random number generator; If None, the random number generator is the RandomState instance used by *np.random*.
  - integer
  - *or* numpy.random.RandomState
  - *or* None

Notes

constraint-1 : negated type *of* 'X/isSparse'
A sparse matrix was passed, but dense data is required. Use X.toarray() to convert to a dense numpy array.

fit(*X*, *y=None*, ***fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

- **X** (array *of* items : array *of* items : float) – Training vector, where n_samples in the number of samples and n_features is the number of features.
- **y** (*any type*) –

transform(*X*, *y=None*)

Transform the data.

Note: The transform method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

- **X** (array *of* items : array *of* items : float) – Test data to be transformed, must have the same number of features as the data used to train the model.
- **ridge_alpha** (*float, optional, default 0.01*) – Amount of ridge shrinkage to apply in order to improve conditioning

Returns
result – Transformed data.

Return type
Any

lale.lib.autogen.sparse_random_projection module

`class lale.lib.autogen.sparse_random_projection.`**SparseRandomProjection**(*, *n_components='auto'*, *density='auto'*, *eps=0.1*, *dense_output=False*, *random_state=None*)

Bases: *PlannedIndividualOp*

Combined schema for expected data and hyperparameters.

This documentation is auto-generated from JSON schemas.

Parameters

- **n_components** (*union type, default 'auto'*) – Dimensionality of the target projection space
  - integer, >=2 for optimizer, <=256 for optimizer, uniform distribution
  - *or* 'auto'
- **density** (*union type, default 'auto'*) – Ratio of non-zero component in the random projection matrix
  - 'auto'
  - *or* float, >=0 for optimizer, <=1 for optimizer, uniform distribution
- **eps** (*float, >=0.001 for optimizer, <=0.1 for optimizer, loguniform distribution, default 0.1*) – Parameter to control the quality of the embedding according to the Johnson-Lindenstrauss lemma when n_components is set to 'auto'
- **dense_output** (*boolean, default False*) – If True, ensure that the output of the random projection is a dense numpy array even if the input and random projection matrix are both sparse
- **random_state** (*union type, not for optimizer, default None*) – Control the pseudo random number generator used to generate the matrix at fit time
  - integer
  - *or* numpy.random.RandomState
  - *or* None

fit(*X, y=None, **fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

- **X** (*union type*) – Training set: only the shape is used to find optimal random matrix dimensions based on the theory referenced in the afore mentioned papers.
  - array *of* items : Any
  - *or* array *of* items : array *of* items : float
- **y** (*Any*) – Ignored

transform(*X, y=None*)

Transform the data.

Note: The transform method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (*union type*) – The input data to project into a smaller dimensional space.

- array *of* items : Any
- *or* array *of* items : array *of* items : float

Returns

result – Projected array.

- array *of* items : Any
- *or* array *of* items : array *of* items : float

Return type

union type

lale.lib.autogen.theil_sen_regressor module

class lale.lib.autogen.theil_sen_regressor.**TheilSenRegressor**(*, *fit_intercept=True*, *copy_X=True*, *max_subpopulation=10000*, *n_subsamples=None*, *max_iter=300*, *tol=0.001*, *random_state=None*, *n_jobs=1*, *verbose=False*)

Bases: *PlannedIndividualOp*

Combined schema for expected data and hyperparameters.

This documentation is auto-generated from JSON schemas.

Parameters

- **fit_intercept** (*boolean, default True*) – Whether to calculate the intercept for this model
- **copy_X** (*boolean, default True*) – If True, X will be copied; else, it may be overwritten.
- **max_subpopulation** (*integer, >=10000 for optimizer, <=10001 for optimizer, uniform distribution, default 10000*) – Instead of computing with a set of cardinality 'n choose k', where n is the number of samples and k is the number of subsamples (at least number of features), consider only a stochastic subpopulation of a given maximal size if 'n choose k' is larger than max_subpopulation
- **n_subsamples** (*union type, not for optimizer, default None*) – Number of samples to calculate the parameters
  - integer
  - *or* None
- **max_iter** (*integer, >=10 for optimizer, <=1000 for optimizer, uniform distribution, default 300*) – Maximum number of iterations for the calculation of spatial median.
- **tol** (*float, >=1e-08 for optimizer, <=0.01 for optimizer, default 0.001*) – Tolerance when calculating spatial median.
- **random_state** (*union type, not for optimizer, default None*) – A random number generator instance to define the state of the random permutations generator
  - integer
  - *or* numpy.random.RandomState
  - *or* None
- **n_jobs** (*union type, not for optimizer, default 1*) – Number of CPUs to use during the cross validation
  - integer
  - *or* None
- **verbose** (*boolean, not for optimizer, default False*) – Verbose mode when fitting the model.

Notes

constraint-1 : any type

fit(*X*, *y=None*, ***fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

- **X** (array *of* items : array *of* items : float) – Training data

- **y** (array *of* items : float) – Target values

predict(*X*, ***predict_params*)

Make predictions.

Note: The predict method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (*union type*) – Samples.

- array *of* items : Any
- *or* array *of* items : array *of* items : float

Returns

result – Returns predicted values.

Return type

array *of* items : float

lale.lib.autogen.transformed_target_regressor module

class lale.lib.autogen.transformed_target_regressor.**TransformedTargetRegressor**(*, *regressor=None*, *transformer=None*, *func=None*, *inverse_func=None*, *check_inverse=True*)

Bases: *PlannedIndividualOp*

Combined schema for expected data and hyperparameters.

This documentation is auto-generated from JSON schemas.

Parameters

- **regressor** (*None, not for optimizer, default None*) – Regressor object such as derived from `RegressorMixin`
- **transformer** (*union type, not for optimizer, default None*) – Estimator object such as derived from `TransformerMixin`
  - dict
  - *or* None

  See also *constraint-1*, *constraint-2*.
- **func** (*None, not for optimizer, default None*) – Function to apply to `y` before passing to `fit`

  See also *constraint-2*.
- **inverse_func** (*None, not for optimizer, default None*) – Function to apply to the prediction of the regressor

  See also *constraint-2*.
- **check_inverse** (*boolean, not for optimizer, default True*) – Whether to check that `transform` followed by `inverse_transform` or `func` followed by `inverse_func` leads to the original targets.

Notes

constraint-1 : union type
transformer' and functions 'func'/'inverse_func' cannot both be set.
- transformer : None
- *or* intersection type
  - dict *of* func : None
  - *and* dict *of* inverse_func : None

constraint-2 : union type
When 'func' is provided, 'inverse_func' must also be provided
- intersection type
  - dict *of* transformer : negated type *of* None
  - *and* union type
    - dict *of* func : negated type *of* None
    - *or* dict *of* inverse_func : negated type *of* None
- *or* transformer : negated type *of* None
- *or* func : None
- *or* inverse_func : negated type *of* None

fit(*X*, *y=None*, ***fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters
- **X** (array *of* items : array *of* items : float) – Training vector, where n_samples is the number of samples and n_features is the number of features.
- **y** (array *of* items : float) – Target values.
- **sample_weight** (*Any, optional*) – Array of weights that are assigned to individual samples

predict(*X*, ***predict_params*)

Make predictions.

Note: The predict method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters
X (array *of* items : array *of* items : float) – Samples.

Returns
result – Predicted values.

Return type
array *of* items : float

lale.lib.autogen.truncated_svd module

class lale.lib.autogen.truncated_svd.**TruncatedSVD**(***, *n_components=2*, *algorithm='randomized'*, *n_iter=5*, *random_state=None*, *tol=0.0*)

Bases: *PlannedIndividualOp*

Combined schema for expected data and hyperparameters.

This documentation is auto-generated from JSON schemas.

Parameters

- **n_components** (*integer, >=2 for optimizer, <='X/items/maxItems', <=256 for optimizer, uniform distribution, default 2*) – Desired dimensionality of output data
- **algorithm** ('arpack' *or* 'randomized', default 'randomized') – SVD solver to use
- **n_iter** (*integer, >=5 for optimizer, <=1000 for optimizer, uniform distribution, default 5*) – Number of iterations for randomized SVD solver
- **random_state** (*union type, not for optimizer, default None*) – If int, random_state is the seed used by the random number generator; If RandomState instance, random_state is the random number generator; If None, the random number generator is the RandomState instance used by *np.random*.
  - integer
  - *or* numpy.random.RandomState
  - *or* None
- **tol** (*float, >=1e-08 for optimizer, <=0.01 for optimizer, default 0.0*) – Tolerance for ARPACK

fit(*X*, *y=None*, ***fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

- **X** (array *of* items : array *of* items : float) – Training data.
- **y** (*any type*) –

transform(*X*, *y=None*)

Transform the data.

Note: The transform method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (array *of* items : array *of* items : float) – New data.

Returns

result – Reduced version of X

Return type

array *of* items : array *of* items : float

Module contents

Lale autogen schemas

The JSON schemas of the operators defined in this module were automatically generated from the source code of 115 scikit-learn operators. The resulting schemas are all valid and usable to build Lale pipelines.

The following paper describes the schema extractor:

```
@InProceedings{baudart_et_al_2020,
  title = "Mining Documentation to Extract Hyperparameter Schemas",
  author = "Baudart, Guillaume and Kirchner, Peter and Hirzel, Martin and Kate, Kiran",
  booktitle = "ICML Workshop on Automated Machine Learning (AutoML@ICML)",
  year = 2020,
  url = "https://arxiv.org/abs/2006.16984" }
```

lale.lib.category_encoders package

Submodules

lale.lib.category_encoders.hashing_encoder module

class lale.lib.category_encoders.hashing_encoder.**HashingEncoder**(**, n_components=8, cols=None, hash_method='md5'*)

Bases: `PlannedIndividualOp`

Hashing encoder transformer from scikit-learn contrib that encodes categorical features as numbers.

This documentation is auto-generated from JSON schemas.

Parameters

- **n_components** (*integer, not for optimizer, default 8*) – how many bits to use to represent the feature.
- **cols** (*union type, not for optimizer, default None*) – a list of columns to encode, if None, all string columns will be encoded.
  - None
  - *or* array *of* items : string
- **hash_method** ('blake2s', 'sha3_224', 'shake_256', 'sha384', 'sha224', 'sha1', 'sha256', 'sha3_256', 'md5-sha1', 'shake_128', 'md5', 'sha512', 'sha512_256', 'sha3_384', 'sm3', 'blake2b', 'sha512_224', *or* 'sha3_512', not for optimizer, default 'md5') – which hashing method to use.

fit(*X, y=None, **fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

- **X** (*array*) – Features; the outer array is over samples.
  - items : array
    - items : union type
      - float
      - *or* string
- **y** (*any type, optional*) – Target class labels; the array is over samples.

transform(*X, y=None*)

Transform the data.

Note: The transform method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (*array*) – Features; the outer array is over samples.

- items : array
  - items : union type
    - float
    - *or* string

Returns

result – Hash codes.

Return type

array *of* items : array *of* items : float

lale.lib.category_encoders.target_encoder module

class lale.lib.category_encoders.target_encoder.**TargetEncoder**(**, verbose=0, cols=None, drop_invariant=False, return_df=True, handle_missing='value', handle_unknown='value', min_samples_leaf=1, smoothing=1.0*)

Bases: `PlannedIndividualOp`

Target encoder transformer from scikit-learn contrib that encodes categorical features as numbers.

This documentation is auto-generated from JSON schemas.

Parameters

- **verbose** (*integer, not for optimizer, default 0*) – Verbosity of the output, 0 for none.
- **cols** (*union type, not for optimizer, default None*) – Columns to encode.
  - None
    All string columns will be encoded.
  - *or* array *of* items : string
- **drop_invariant** (*boolean, not for optimizer, default False*) – Whether to drop columns with 0 variance.
- **return_df** (*boolean, not for optimizer, default True*) – Whether to return a pandas DataFrame from transform (otherwise it will be a numpy array).
- **handle_missing** ('error', 'return_nan', *or* 'value', not for optimizer, default 'value') – Given 'value', return the target mean.
- **handle_unknown** ('error', 'return_nan', *or* 'value', not for optimizer, default 'value') – Given 'value', return the target mean.
- **min_samples_leaf** (*integer, >=1, <=10 for optimizer, not for optimizer, default 1*) – For regularization the weighted average between category mean and global mean is taken. The weight is an S-shaped curve between 0 and 1 with the number of samples for a category on the x-axis. The curve reaches 0.5 at min_samples_leaf. (parameter k in the original paper)
- **smoothing** (*float, >0.0, <=10.0 for optimizer, not for optimizer, default 1.0*) – Smoothing effect to balance categorical average vs prior. Higher value means stronger regularization. The value must be strictly bigger than 0. Higher values mean a flatter S-curve (see min_samples_leaf).

fit(*X, y=None, **fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

- **X** (*array*) – Features; the outer array is over samples.
  - items : array
    - items : union type
      - float
      - *or* string
- **y** (*union type*) – Target class labels; the array is over samples.
  - array *of* items : float
  - *or* array *of* items : string

transform(*X, y=None*)

Transform the data.

Note: The transform method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (*array*) – Features; the outer array is over samples.

- items : array
  - items : union type
    - float
    - *or* string

Returns

result

Return type

array *of* items : array *of* items : float

Module contents

Schema-enhanced versions of some of the operators from category_encoders to enable hyperparameter tuning.

Operators

- lale.lib.category_encoders. HashingEncoder
- lale.lib.category_encoders. TargetEncoder

lale.lib.imblearn package

Submodules

lale.lib.imblearn.adasyn module

`class lale.lib.imblearn.adasyn.ADASYN(*, operator, sampling_strategy='auto', random_state=None, n_neighbors=5, n_jobs=1)`

Bases: `PlannedIndividualOp`

Perform over-sampling using Adaptive Synthetic (ADASYN) sampling approach for imbalanced datasets.

This documentation is auto-generated from JSON schemas.

Parameters

- **operator** (*operator, optional*) – Trainable Lale pipeline that is trained using the data obtained from the current imbalance corrector.

  Predict, transform, predict_proba or decision_function would just be forwarded to the trained pipeline. If operator is a Planned pipeline, the current imbalance corrector can't be trained without using an optimizer to choose a trainable operator first. Please refer to lale/examples for more examples.
- **sampling_strategy** (*union type, optional, not for optimizer, default 'auto'*) – Sampling information to resample the data set.
  - float, not for optimizer

Desired ratio of the number of samples in the minority class over the number of samples in the majority class after resampling. Therefore, the ratio is expressed as $\alpha_{os} = N_{rm}/N_M$ where N_{rm} is the number of samples in the minority class after resampling and N_M is the number of samples in the majority class.

> **Warning:** Only available for **binary** classification. An error is raised for multi-class classification.

- *or* 'minority', 'not minority', 'not majority', 'all', *or* 'auto'
  The class targeted by the resampling. The number of samples in the different classes will be equalized. Possible choices are:
  * `'minority'`: resample only the minority class;
  * `'not minority'`: resample all classes but the minority class;
  * `'not majority'`: resample all classes but the majority class;
  * `'all'`: resample all classes;
  * `'auto'`: equivalent to `'not majority'`.
- *or* dict, not for optimizer
  Keys correspond to the targeted classes and values correspond to the desired number of samples for each targeted class.
- *or* callable, not for optimizer
  Function taking `y` and returns a `dict`. The keys correspond to the targeted classes and the values correspond to the desired number of samples for each class.

- **random_state** (*union type, optional, not for optimizer, default None*) – Control the randomization of the algorithm.
  - None
    RandomState used by np.random
  - *or* integer
    The seed used by the random number generator
  - *or* numpy.random.RandomState
    Random number generator instance.
- **n_neighbors** (*union type, optional, not for optimizer, default 5*) – Number of neighbors.
  - integer
    Number of nearest neighbours to use to construct synthetic samples.
  - *or* Any
    An estimator that inherits from `sklearn.neighbors.base.KNeighborsMixin` that will be used to find the *n_neighbors*.
- **n_jobs** (*integer, optional, not for optimizer, default 1*) – The number of threads to open if possible.

decision_function(*X*)

Confidence scores for all classes.

Note: The decision_function method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters
  X (array *of* items : array *of* items : float) – Features; the outer array is over samples.

Returns
  result – Output data schema for predictions.

Return type
  Any

`fit`(*X*, *y=None*, ***fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

- **X** (array *of* items : array *of* items : float) – Features; the outer array is over samples.
- **y** (*union type*) – Target class labels; the array is over samples.
  - array *of* items : float
  - *or* array *of* items : string

`predict`(*X*, ***predict_params*)

Make predictions.

Note: The predict method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (array *of* items : array *of* items : float) – Features; the outer array is over samples.

Returns

result – Output data schema for predictions.

Return type

Any

`predict_proba`(*X*)

Probability estimates for all classes.

Note: The predict_proba method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (array *of* items : array *of* items : float) – Features; the outer array is over samples.

Returns

result – Probability of the sample for each class in the model.

Return type

array *of* items : array *of* items : float

`transform`(*X*, *y=None*)

Transform the data.

Note: The transform method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

- **X** (array *of* items : array *of* items : float) – Features; the outer array is over samples.
- **y** (*union type*) – Target class labels; the array is over samples.
  - array *of* items : float
  - *or* array *of* items : string
  - *or* None

Returns

result – Output data schema for transformed data.

Return type

Any

lale.lib.imblearn.all_knn module

class lale.lib.imblearn.all_knn.**AllKNN**(***, *operator*, *sampling_strategy='auto'*, *n_neighbors=3*, *kind_sel='all'*, *allow_minority=False*, *n_jobs=1*)

Bases: `PlannedIndividualOp`

Class to perform under-sampling based on the AllKNN method.

This documentation is auto-generated from JSON schemas.

Parameters

- **operator** (*operator, optional*) – Trainable Lale pipeline that is trained using the data obtained from the current imbalance corrector.

  Predict, transform, predict_proba or decision_function would just be forwarded to the trained pipeline. If operator is a Planned pipeline, the current imbalance corrector can't be trained without using an optimizer to choose a trainable operator first. Please refer to lale/examples for more examples.
- **sampling_strategy** (*union type, optional, not for optimizer, default 'auto'*) – Sampling information to resample the data set.
  - 'minority', 'not minority', 'not majority', 'all', *or* 'auto'

    The class targeted by the resampling. The number of samples in the different classes will be equalized. Possible choices are:
    - `'minority'`: resample only the minority class;
    - `'not minority'`: resample all classes but the minority class;
    - `'not majority'`: resample all classes but the majority class;
    - `'all'`: resample all classes;
    - `'auto'`: equivalent to `'not majority'`.
  - *or* union type, not for optimizer

    Classes targeted by the resampling.
    - array *of* items : float
    - *or* array *of* items : string
  - *or* callable, not for optimizer

    Function taking `y` and returns a `dict`. The keys correspond to the targeted classes and the values correspond to the desired number of samples for each class.
- **n_neighbors** (*union type, optional, not for optimizer, default 3*) – Number of neighbors.
  - integer

    Number of nearest neighbours to use to construct synthetic samples.
  - *or* Any

    An estimator that inherits from `sklearn.neighbors.base.KNeighborsMixin` that will be used to find the *n_neighbors*.
- **kind_sel** ('all' *or* 'mode', optional, not for optimizer, default 'all') – Strategy to use in order to exclude samples. If `all`, all neighbours will have to agree with the samples of interest to not be excluded. If `mode`, the majority vote of the neighbours will be used in order to exclude a sample.
- **allow_minority** (*boolean, optional, not for optimizer, default False*) – If True, it allows the majority classes to become the minority class without early stopping.
- **n_jobs** (*integer, optional, not for optimizer, default 1*) – The number of threads to open if possible.

decision_function(*X*)

Confidence scores for all classes.

Note: The decision_function method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

`X` (array *of* items : array *of* items : float) – Features; the outer array is over samples.

Returns

result – Output data schema for predictions.

Return type

Any

`fit`(*X*, *y=None*, ***fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

- `X` (array *of* items : array *of* items : float) – Features; the outer array is over samples.
- `y` (*union type*) – Target class labels; the array is over samples.
  - array *of* items : float
  - *or* array *of* items : string

`predict`(*X*, ***predict_params*)

Make predictions.

Note: The predict method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

`X` (array *of* items : array *of* items : float) – Features; the outer array is over samples.

Returns

result – Output data schema for predictions.

Return type

Any

`predict_proba`(*X*)

Probability estimates for all classes.

Note: The predict_proba method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

`X` (array *of* items : array *of* items : float) – Features; the outer array is over samples.

Returns

result – Probability of the sample for each class in the model.

Return type

array *of* items : array *of* items : float

`transform`(*X*, *y=None*)

Transform the data.

Note: The transform method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

- `X` (array *of* items : array *of* items : float) – Features; the outer array is over samples.
- `y` (*union type*) – Target class labels; the array is over samples.

- array *of* items : float
- *or* array *of* items : string
- *or* None

Returns
  result – Output data schema for transformed data.

Return type
  Any

lale.lib.imblearn.base_resampler module

lale.lib.imblearn.borderline_smote module

class `lale.lib.imblearn.borderline_smote.BorderlineSMOTE`(*, *operator*, *sampling_strategy='auto'*, *random_state=None*, *k_neighbors=5*, *n_jobs=1*, *m_neighbors=10*, *kind='borderline-1'*)

Bases: `PlannedIndividualOp`

Over-sampling using Borderline SMOTE, which is a variant of the original SMOTE algorithm.

This documentation is auto-generated from JSON schemas.

Borderline samples will be detected and used to generate new synthetic samples.

Parameters

- **operator** (*operator, optional*) – Trainable Lale pipeline that is trained using the data obtained from the current imbalance corrector.

  Predict, transform, predict_proba or decision_function would just be forwarded to the trained pipeline. If operator is a Planned pipeline, the current imbalance corrector can't be trained without using an optimizer to choose a trainable operator first. Please refer to lale/examples for more examples.
- **sampling_strategy** (*union type, optional, not for optimizer, default 'auto'*) – Sampling information to resample the data set.
  - float, not for optimizer

    Desired ratio of the number of samples in the minority class over the number of samples in the majority class after resampling. Therefore, the ratio is expressed as $\alpha_{os} = N_{rm}/N_M$ where N_{rm} is the number of samples in the minority class after resampling and N_M is the number of samples in the majority class.

    > **Warning:** Only available for **binary** classification. An error is raised for multi-class classification.
  - *or* 'minority', 'not minority', 'not majority', 'all', *or* 'auto'

    The class targeted by the resampling. The number of samples in the different classes will be equalized. Possible choices are:
    * `'minority'`: resample only the minority class;
    * `'not minority'`: resample all classes but the minority class;
    * `'not majority'`: resample all classes but the majority class;
    * `'all'`: resample all classes;
    * `'auto'`: equivalent to `'not majority'`.
  - *or* dict, not for optimizer

Keys correspond to the targeted classes and values correspond to the desired number of samples for each targeted class.

  - *or* callable, not for optimizer

    Function taking `y` and returns a `dict`. The keys correspond to the targeted classes and the values correspond to the desired number of samples for each class.

- **random_state** (*union type, optional, not for optimizer, default None*) – Control the randomization of the algorithm.
  - None

    RandomState used by np.random
  - *or* integer

    The seed used by the random number generator
  - *or* numpy.random.RandomState

    Random number generator instance.
- **k_neighbors** (*union type, optional, not for optimizer, default 5*) – Number of nearest neighbours to use to construct synthetic samples.
  - integer

    Number of nearest neighbours to use to construct synthetic samples.
  - *or* Any

    An estimator that inherits from `sklearn.neighbors.base.KNeighborsMixin` that will be used to find the *n_neighbors*.
- **n_jobs** (*integer, optional, not for optimizer, default 1*) – The number of threads to open if possible.
- **m_neighbors** (*union type, optional, not for optimizer, default 10*) – Number of nearest neighbours to use to determine if a minority sample is in danger.
  - integer

    Number of nearest neighbours to use to construct synthetic samples.
  - *or* Any

    An estimator that inherits from `sklearn.neighbors.base.KNeighborsMixin` that will be used to find the *n_neighbors*.
- **kind** ('borderline-1' *or* 'borderline-2', optional, not for optimizer, default 'borderline-1') – The type of SMOTE algorithm to use.

decision_function(*X*)

Confidence scores for all classes.

Note: The decision_function method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (array *of* items : array *of* items : float) – Features; the outer array is over samples.

Returns

result – Output data schema for predictions.

Return type

Any

fit(*X, y=None, **fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

- **X** (array *of* items : array *of* items : float) – Features; the outer array is over samples.
- **y** (*union type*) – Target class labels; the array is over samples.
  - array *of* items : float

– *or* array *of* items : string

`predict`(*X*, ***predict_params*)

Make predictions.

Note: The predict method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters
X (array *of* items : array *of* items : float) – Features; the outer array is over samples.

Returns
result – Output data schema for predictions.

Return type
Any

`predict_proba`(*X*)

Probability estimates for all classes.

Note: The predict_proba method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters
X (array *of* items : array *of* items : float) – Features; the outer array is over samples.

Returns
result – Probability of the sample for each class in the model.

Return type
array *of* items : array *of* items : float

`transform`(*X*, *y=None*)

Transform the data.

Note: The transform method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters
- **X** (array *of* items : array *of* items : float) – Features; the outer array is over samples.
- **y** (*union type*) – Target class labels; the array is over samples.
  - array *of* items : float
  - *or* array *of* items : string
  - *or* None

Returns
result – Output data schema for transformed data.

Return type
Any

lale.lib.imblearn.condensed_nearest_neighbour module

`class lale.lib.imblearn.condensed_nearest_neighbour.CondensedNearestNeighbour`(*, *operator*, *sampling_strategy='auto'*, *random_state=None*, *n_neighbors=None*, *n_seeds_S=1*, *n_jobs=1*)

Bases: `PlannedIndividualOp`

Class to perform under-sampling based on the condensed nearest neighbour method.

This documentation is auto-generated from JSON schemas.

Parameters

- **operator** (*operator, optional*) – Trainable Lale pipeline that is trained using the data obtained from the current imbalance corrector.

  Predict, transform, predict_proba or decision_function would just be forwarded to the trained pipeline. If operator is a Planned pipeline, the current imbalance corrector can't be trained without using an optimizer to choose a trainable operator first. Please refer to lale/examples for more examples.
- **sampling_strategy** (*union type, optional, not for optimizer, default 'auto'*) – Sampling information to resample the data set.
  - 'minority', 'not minority', 'not majority', 'all', *or* 'auto'
    The class targeted by the resampling. The number of samples in the different classes will be equalized. Possible choices are:
    * `'minority'`: resample only the minority class;
    * `'not minority'`: resample all classes but the minority class;
    * `'not majority'`: resample all classes but the majority class;
    * `'all'`: resample all classes;
    * `'auto'`: equivalent to `'not majority'`.
  - *or* union type, not for optimizer
    Classes targeted by the resampling.
    * array *of* items : float
    * *or* array *of* items : string
  - *or* callable, not for optimizer
    Function taking `y` and returns a `dict`. The keys correspond to the targeted classes and the values correspond to the desired number of samples for each class.
- **random_state** (*union type, optional, not for optimizer, default None*) – Control the randomization of the algorithm.
  - None
    RandomState used by np.random
  - *or* integer
    The seed used by the random number generator
  - *or* numpy.random.RandomState
    Random number generator instance.
- **n_neighbors** (*union type, optional, not for optimizer, default None*) – Number of neighbors.
  - integer
    Number of nearest neighbours to use to construct synthetic samples.
  - *or* Any
    An estimator that inherits from `sklearn.neighbors.base.KNeighborsMixin` that will be used to find the *n_neighbors*.
  - *or* None
    KNeighborsClassifier(n_neighbors=1)
- **n_seeds_S** (*integer, optional, not for optimizer, default 1*) – Number of samples to extract in order to build the set S.
- **n_jobs** (*integer, optional, not for optimizer, default 1*) – The number of threads to open if possible.

decision_function(*X*)

Confidence scores for all classes.

Note: The decision_function method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters
: **X** (array *of* items : array *of* items : float) – Features; the outer array is over samples.

Returns
: **result** – Output data schema for predictions.

Return type
: Any

`fit`(*X*, *y=None*, ***fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters
- **X** (array *of* items : array *of* items : float) – Features; the outer array is over samples.
- **y** (*union type*) – Target class labels; the array is over samples.
  - array *of* items : float
  - *or* array *of* items : string

`predict`(*X*, ***predict_params*)

Make predictions.

Note: The predict method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters
: **X** (array *of* items : array *of* items : float) – Features; the outer array is over samples.

Returns
: **result** – Output data schema for predictions.

Return type
: Any

`predict_proba`(*X*)

Probability estimates for all classes.

Note: The predict_proba method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters
: **X** (array *of* items : array *of* items : float) – Features; the outer array is over samples.

Returns
: **result** – Probability of the sample for each class in the model.

Return type
: array *of* items : array *of* items : float

`transform`(*X*, *y=None*)

Transform the data.

Note: The transform method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters
- **X** (array *of* items : array *of* items : float) – Features; the outer array is over samples.

- **y** (*union type*) – Target class labels; the array is over samples.
  - array *of* items : float
  - *or* array *of* items : string
  - *or* None

Returns
result – Output data schema for transformed data.

Return type
Any

lale.lib.imblearn.edited_nearest_neighbours module

class lale.lib.imblearn.edited_nearest_neighbours.**EditedNearestNeighbours**(*, *operator*, *sampling_strategy='auto'*, *n_neighbors=3*, *kind_sel='all'*, *n_jobs=1*)

Bases: *PlannedIndividualOp*

Class to perform under-sampling based on the edited nearest neighbour method.

This documentation is auto-generated from JSON schemas.

Parameters

- **operator** (*operator, optional*) – Trainable Lale pipeline that is trained using the data obtained from the current imbalance corrector.

  Predict, transform, predict_proba or decision_function would just be forwarded to the trained pipeline. If operator is a Planned pipeline, the current imbalance corrector can't be trained without using an optimizer to choose a trainable operator first. Please refer to lale/examples for more examples.
- **sampling_strategy** (*union type, optional, not for optimizer, default 'auto'*) – Sampling information to resample the data set.
  - 'minority', 'not minority', 'not majority', 'all', *or* 'auto'
    The class targeted by the resampling. The number of samples in the different classes will be equalized. Possible choices are:
    - `'minority'`: resample only the minority class;
    - `'not minority'`: resample all classes but the minority class;
    - `'not majority'`: resample all classes but the majority class;
    - `'all'`: resample all classes;
    - `'auto'`: equivalent to `'not majority'`.
  - *or* union type, not for optimizer
    Classes targeted by the resampling.
    - array *of* items : float
    - *or* array *of* items : string
  - *or* callable, not for optimizer
    Function taking `y` and returns a `dict`. The keys correspond to the targeted classes and the values correspond to the desired number of samples for each class.
- **n_neighbors** (*union type, optional, not for optimizer, default 3*) – Number of neighbors.
  - integer
    Number of nearest neighbours to use to construct synthetic samples.
  - *or* Any

An estimator that inherits from `sklearn.neighbors.base.KNeighborsMixin` that will be used to find the *n_neighbors*.

- **kind_sel** ('all' *or* 'mode', optional, not for optimizer, default 'all') – Strategy to use in order to exclude samples. If `all`, all neighbours will have to agree with the samples of interest to not be excluded. If `mode`, the majority vote of the neighbours will be used in order to exclude a sample.
- **n_jobs** (*integer, optional, not for optimizer, default 1*) – The number of threads to open if possible.

decision_function(*X*)

Confidence scores for all classes.

Note: The decision_function method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters
X (array *of* items : array *of* items : float) – Features; the outer array is over samples.

Returns
result – Output data schema for predictions.

Return type
Any

fit(*X*, *y=None*, ***fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters
- **X** (array *of* items : array *of* items : float) – Features; the outer array is over samples.
- **y** (*union type*) – Target class labels; the array is over samples.
  - array *of* items : float
  - *or* array *of* items : string

predict(*X*, ***predict_params*)

Make predictions.

Note: The predict method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters
X (array *of* items : array *of* items : float) – Features; the outer array is over samples.

Returns
result – Output data schema for predictions.

Return type
Any

predict_proba(*X*)

Probability estimates for all classes.

Note: The predict_proba method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters
X (array *of* items : array *of* items : float) – Features; the outer array is over samples.

Returns
result – Probability of the sample for each class in the model.

Return type
array *of* items : array *of* items : float

transform(*X*, *y=None*)

Transform the data.

Note: The transform method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

- **X** (array *of* items : array *of* items : float) – Features; the outer array is over samples.
- **y** (*union type*) – Target class labels; the array is over samples.
  - array *of* items : float
  - *or* array *of* items : string
  - *or* None

Returns

result – Output data schema for transformed data.

Return type

Any

lale.lib.imblearn.instance_hardness_threshold module

class lale.lib.imblearn.instance_hardness_threshold.**InstanceHardnessThreshold**(**, operator, estimator=None, sampling_strategy='auto', random_state=None, cv=5, n_jobs=1*)

Bases: *PlannedIndividualOp*

Class to perform under-sampling based on the instance hardness threshold.

This documentation is auto-generated from JSON schemas.

Parameters

- **operator** (*operator, optional*) – Trainable Lale pipeline that is trained using the data obtained from the current imbalance corrector.

  Predict, transform, predict_proba or decision_function would just be forwarded to the trained pipeline. If operator is a Planned pipeline, the current imbalance corrector can't be trained without using an optimizer to choose a trainable operator first. Please refer to lale/examples for more examples.
- **estimator** (*union type, optional, not for optimizer, default None*) – Classifier to be used to estimate instance hardness of the samples. By default a `sklearn.ensemble.RandomForestClassifer` will be used. If `str`, the choices using a string are the following: `'knn'`, `'decision-tree'`, `'random-forest'`, `'adaboost'`, `'gradient-boosting'` and `'linear-svm'`. If object, an estimator inherited from `sklearn.base.ClassifierMixin` and having an attribute *predict_proba()*.
  - Any
  - *or* 'knn', 'decision-tree', 'random-forest', 'adaboost', 'gradient-boosting', *or* 'linear-svm'
  - *or* None
- **sampling_strategy** (*union type, optional, not for optimizer, default 'auto'*) – Sampling information to resample the data set.

- float, not for optimizer

  Desired ratio of the number of samples in the minority class over the number of samples in the majority class after resampling. Therefore, the ratio is expressed as $\alpha_{os} = N_{rm}/N_M$ where N_{rm} is the number of samples in the minority class after resampling and N_M is the number of samples in the majority class.

  > **Warning:** Only available for **binary** classification. An error is raised for multi-class classification.

- *or* 'majority', 'not minority', 'not majority', 'all', *or* 'auto'

  The class targeted by the resampling. The number of samples in the different classes will be equalized. Possible choices are:

  * `'majority'`: resample only the majority class;
  * `'not minority'`: resample all classes but the minority class;
  * `'not majority'`: resample all classes but the majority class;
  * `'all'`: resample all classes;
  * `'auto'`: equivalent to `'not minority'`.

- *or* dict, not for optimizer

  Keys correspond to the targeted classes and values correspond to the desired number of samples for each targeted class.

- *or* callable, not for optimizer

  Function taking `y` and returns a `dict`. The keys correspond to the targeted classes and the values correspond to the desired number of samples for each class.

- **random_state** (*union type, optional, not for optimizer, default None*) – Control the randomization of the algorithm.
  - None

    RandomState used by np.random
  - *or* integer

    The seed used by the random number generator
  - *or* numpy.random.RandomState

    Random number generator instance.
- **cv** (*integer, >=1, optional, not for optimizer, default 5*) – Number of folds to be used when estimating samples' instance hardness.
- **n_jobs** (*integer, optional, not for optimizer, default 1*) – The number of threads to open if possible.

decision_function(*X*)

Confidence scores for all classes.

Note: The decision_function method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (array *of* items : array *of* items : float) – Features; the outer array is over samples.

Returns

result – Output data schema for predictions.

Return type

Any

fit(*X*, *y=None*, ***fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

- **X** (array *of* items : array *of* items : float) – Features; the outer array is over samples.
- **y** (*union type*) – Target class labels; the array is over samples.
  - array *of* items : float
  - *or* array *of* items : string

`predict(X, **predict_params)`

Make predictions.

Note: The predict method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (array *of* items : array *of* items : float) – Features; the outer array is over samples.

Returns

result – Output data schema for predictions.

Return type

Any

`predict_proba(X)`

Probability estimates for all classes.

Note: The predict_proba method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (array *of* items : array *of* items : float) – Features; the outer array is over samples.

Returns

result – Probability of the sample for each class in the model.

Return type

array *of* items : array *of* items : float

`transform(X, y=None)`

Transform the data.

Note: The transform method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

- **X** (array *of* items : array *of* items : float) – Features; the outer array is over samples.
- **y** (*union type*) – Target class labels; the array is over samples.
  - array *of* items : float
  - *or* array *of* items : string
  - *or* None

Returns

result – Output data schema for transformed data.

Return type

Any

lale.lib.imblearn.random_over_sampler module

class lale.lib.imblearn.random_over_sampler.**RandomOverSampler**(*, *operator*, *sampling_strategy='auto'*, *random_state=None*)

Bases: *PlannedIndividualOp*

Class to perform random over-sampling, i.e. over-sample the minority class(es) by picking samples at random with replacement.

This documentation is auto-generated from JSON schemas.

Parameters

- **operator** (*operator*) – Trainable Lale pipeline that is trained using the data obtained from the current imbalance corrector.

  Predict, transform, predict_proba or decision_function would just be forwarded to the trained pipeline. If operator is a Planned pipeline, the current imbalance corrector can't be trained without using an optimizer to choose a trainable operator first. Please refer to lale/examples for more examples.
- **sampling_strategy** (*union type, optional, not for optimizer, default 'auto'*) – Sampling information to resample the data set.
  - float, not for optimizer

    Desired ratio of the number of samples in the minority class over the number of samples in the majority class after resampling. Therefore, the ratio is expressed as $\alpha_{os} = N_{rm}/N_M$ where N_{rm} is the number of samples in the minority class after resampling and N_M is the number of samples in the majority class.

    > **Warning:** Only available for **binary** classification. An error is raised for multi-class classification.
  - *or* 'minority', 'not minority', 'not majority', 'all', *or* 'auto'

    The class targeted by the resampling. The number of samples in the different classes will be equalized. Possible choices are:
    - 'minority': resample only the minority class;
    - 'not minority': resample all classes but the minority class;
    - 'not majority': resample all classes but the majority class;
    - 'all': resample all classes;
    - 'auto': equivalent to 'not majority'.
  - *or* dict, not for optimizer

    Keys correspond to the targeted classes and values correspond to the desired number of samples for each targeted class.
  - *or* callable, not for optimizer

    Function taking y and returns a dict. The keys correspond to the targeted classes and the values correspond to the desired number of samples for each class.
- **random_state** (*union type, optional, not for optimizer, default None*) – Control the randomization of the algorithm.
  - None

    RandomState used by np.random
  - *or* integer

    The seed used by the random number generator
  - *or* numpy.random.RandomState

Random number generator instance.

decision_function(*X*)

Confidence scores for all classes.

Note: The decision_function method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (array *of* items : array *of* items : float) – Features; the outer array is over samples.

Returns

result – Output data schema for predictions.

Return type

Any

fit(*X*, *y=None*, ***fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

- **X** (array *of* items : array *of* items : float) – Features; the outer array is over samples.
- **y** (*union type*) – Target class labels; the array is over samples.
  - array *of* items : float
  - *or* array *of* items : string

predict(*X*, ***predict_params*)

Make predictions.

Note: The predict method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (array *of* items : array *of* items : float) – Features; the outer array is over samples.

Returns

result – Output data schema for predictions.

Return type

Any

predict_proba(*X*)

Probability estimates for all classes.

Note: The predict_proba method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (array *of* items : array *of* items : float) – Features; the outer array is over samples.

Returns

result – Probability of the sample for each class in the model.

Return type

array *of* items : array *of* items : float

transform(*X*, *y=None*)

Transform the data.

Note: The transform method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

- **X** (array *of* items : array *of* items : float) – Features; the outer array is over samples.
- **y** (*union type*) – Target class labels; the array is over samples.
  - array *of* items : float
  - *or* array *of* items : string
  - *or* None

Returns
result – Output data schema for transformed data.

Return type
Any

lale.lib.imblearn.random_under_sampler module

class lale.lib.imblearn.random_under_sampler.**RandomUnderSampler**(*, *operator*, *sampling_strategy='auto'*, *random_state=None*, *replacement=False*)

Bases: *PlannedIndividualOp*

Class to perform random under-sampling, i.e. under-sample the minority class(es) by picking samples at random with replacement.

This documentation is auto-generated from JSON schemas.

Parameters

- **operator** (*operator*) – Trainable Lale pipeline that is trained using the data obtained from the current imbalance corrector.

  Predict, transform, predict_proba or decision_function would just be forwarded to the trained pipeline. If operator is a Planned pipeline, the current imbalance corrector can't be trained without using an optimizer to choose a trainable operator first. Please refer to lale/examples for more examples.
- **sampling_strategy** (*union type, optional, not for optimizer, default 'auto'*) – Sampling information to resample the data set.
  - float, not for optimizer

    Desired ratio of the number of samples in the minority class over the number of samples in the majority class after resampling. Therefore, the ratio is expressed as $\alpha_{os} = N_{rm}/N_M$ where N_{rm} is the number of samples in the minority class after resampling and N_M is the number of samples in the majority class.

    Warning: Only available for **binary** classification. An error is raised for multi-class classification.
  - *or* 'majority', 'not minority', 'not majority', 'all', *or* 'auto'

    The class targeted by the resampling. The number of samples in the different classes will be equalized. Possible choices are:
    * 'majority': resample only the majority class;
    * 'not minority': resample all classes but the minority class;
    * 'not majority': resample all classes but the majority class;
    * 'all': resample all classes;
    * 'auto': equivalent to 'not minority'.
  - *or* dict, not for optimizer

Keys correspond to the targeted classes and values correspond to the desired number of samples for each targeted class.

  - *or* callable, not for optimizer

    Function taking `y` and returns a `dict`. The keys correspond to the targeted classes and the values correspond to the desired number of samples for each class.

- **random_state** (*union type, optional, not for optimizer, default None*) – Control the randomization of the algorithm.
  - None

    RandomState used by np.random
  - *or* integer

    The seed used by the random number generator
  - *or* numpy.random.RandomState

    Random number generator instance.
- **replacement** (*boolean, optional, not for optimizer, default False*) – Whether the sample is with or without replacement.

decision_function(*X*)

Confidence scores for all classes.

Note: The decision_function method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (*array*) – Features; the outer array is over samples.
- items : array
  - items : union type
    - float
    - *or* string

Returns

result – Output data schema for predictions.

Return type

Any

fit(*X*, *y=None*, ***fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

- **X** (*array*) – Features; the outer array is over samples.
  - items : array
    - items : union type
      - float
      - *or* string
- **y** (*union type*) – Target class labels; the array is over samples.
  - array *of* items : float
  - *or* array *of* items : string

predict(*X*, ***predict_params*)

Make predictions.

Note: The predict method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (*array*) – Features; the outer array is over samples.

- items : array
  - items : union type
    - float
    - *or* string

Returns
    result – Output data schema for predictions.

Return type
    Any

`predict_proba`(*X*)

Probability estimates for all classes.

Note: The predict_proba method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters
    X (*array*) – Features; the outer array is over samples.

- items : array
  - items : union type
    - float
    - *or* string

Returns
    result – Probability of the sample for each class in the model.

Return type
    array *of* items : array *of* items : float

`transform`(*X*, *y=None*)

Transform the data.

Note: The transform method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

- **X** (*array*) – Features; the outer array is over samples.
  - items : array
    - items : union type
      - float
      - *or* string
- **y** (*union type*) – Target class labels; the array is over samples.
  - array *of* items : float
  - *or* array *of* items : string
  - *or* None

Returns
    result – Output data schema for transformed data.

Return type
    Any

lale.lib.imblearn.repeated_edited_nearest_neighbours module

class lale.lib.imblearn.repeated_edited_nearest_neighbours.**RepeatedEditedNearestNeighbours**(**, operator, sampling_strategy='auto', n_neighbors=3, max_iter=100, kind_sel='all', n_jobs=1*)

Bases: *PlannedIndividualOp*

Class to perform under-sampling based on the repeated edited nearest neighbour method.

This documentation is auto-generated from JSON schemas.

Parameters

- **operator** (*operator, optional*) – Trainable Lale pipeline that is trained using the data obtained from the current imbalance corrector.

  Predict, transform, predict_proba or decision_function would just be forwarded to the trained pipeline. If operator is a Planned pipeline, the current imbalance corrector can't be trained without using an optimizer to choose a trainable operator first. Please refer to lale/examples for more examples.
- **sampling_strategy** (*union type, optional, not for optimizer, default 'auto'*) – Sampling information to resample the data set.
  - 'minority', 'not minority', 'not majority', 'all', *or* 'auto'

    The class targeted by the resampling. The number of samples in the different classes will be equalized. Possible choices are:
    - `'minority'`: resample only the minority class;
    - `'not minority'`: resample all classes but the minority class;
    - `'not majority'`: resample all classes but the majority class;
    - `'all'`: resample all classes;
    - `'auto'`: equivalent to `'not majority'`.
  - *or* union type, not for optimizer

    Classes targeted by the resampling.
    - array *of* items : float
    - *or* array *of* items : string
  - *or* callable, not for optimizer

    Function taking `y` and returns a `dict`. The keys correspond to the targeted classes and the values correspond to the desired number of samples for each class.
- **n_neighbors** (*union type, optional, not for optimizer, default 3*) – Number of neighbors.
  - integer

    Number of nearest neighbours to use to construct synthetic samples.
  - *or* Any

    An estimator that inherits from `sklearn.neighbors.base.KNeighborsMixin` that will be used to find the *n_neighbors*.
- **max_iter** (*integer, optional, not for optimizer, default 100*) – Maximum number of iterations of the edited nearest neighbours algorithm for a single run.
- **kind_sel** ('all' *or* 'mode', optional, not for optimizer, default 'all') – Strategy to use

in order to exclude samples. If `all`, all neighbours will have to agree with the samples of interest to not be excluded. If `mode`, the majority vote of the neighbours will be used in order to exclude a sample.

- **n_jobs** (*integer, optional, not for optimizer, default 1*) – The number of threads to open if possible.

decision_function(*X*)

Confidence scores for all classes.

Note: The decision_function method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (array *of* items : array *of* items : float) – Features; the outer array is over samples.

Returns

result – Output data schema for predictions.

Return type

Any

fit(*X*, *y=None*, ***fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

- **X** (array *of* items : array *of* items : float) – Features; the outer array is over samples.
- **y** (*union type*) – Target class labels; the array is over samples.
  - array *of* items : float
  - *or* array *of* items : string

predict(*X*, ***predict_params*)

Make predictions.

Note: The predict method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (array *of* items : array *of* items : float) – Features; the outer array is over samples.

Returns

result – Output data schema for predictions.

Return type

Any

predict_proba(*X*)

Probability estimates for all classes.

Note: The predict_proba method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (array *of* items : array *of* items : float) – Features; the outer array is over samples.

Returns

result – Probability of the sample for each class in the model.

Return type

array *of* items : array *of* items : float

transform(*X*, *y=None*)

Transform the data.

Note: The transform method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

- **X** (array *of* items : array *of* items : float) – Features; the outer array is over samples.
- **y** (*union type*) – Target class labels; the array is over samples.
  - array *of* items : float
  - *or* array *of* items : string
  - *or* None

Returns

result – Output data schema for transformed data.

Return type

Any

lale.lib.imblearn.smote module

class lale.lib.imblearn.smote.**SMOTE**(**, operator, sampling_strategy='auto', random_state=None, k_neighbors=5, n_jobs=1*)

Bases: *PlannedIndividualOp*

Class to perform over-sampling using Synthetic Minority Over-sampling Technique (SMOTE).

This documentation is auto-generated from JSON schemas.

Parameters

- **operator** (*operator*) – Trainable Lale pipeline that is trained using the data obtained from the current imbalance corrector.

  Predict, transform, predict_proba or decision_function would just be forwarded to the trained pipeline. If operator is a Planned pipeline, the current imbalance corrector can't be trained without using an optimizer to choose a trainable operator first. Please refer to lale/examples for more examples.
- **sampling_strategy** (*union type, optional, default 'auto'*) – Sampling information to resample the data set.
  - float, not for optimizer

    Desired ratio of the number of samples in the minority class over the number of samples in the majority class after resampling. Therefore, the ratio is expressed as $\alpha_{os} = N_{rm}/N_M$ where N_{rm} is the number of samples in the minority class after resampling and N_M is the number of samples in the majority class.

    > **Warning:** Only available for **binary** classification. An error is raised for multi-class classification.

  - *or* 'minority', 'not minority', 'not majority', 'all', *or* 'auto'

    The class targeted by the resampling. The number of samples in the different classes will be equalized. Possible choices are:
    - 'minority': resample only the minority class;
    - 'not minority': resample all classes but the minority class;
    - 'not majority': resample all classes but the majority class;

      * `'all'`: resample all classes;
      * `'auto'`: equivalent to `'not majority'`.
  - *or* dict, not for optimizer
    Keys correspond to the targeted classes and values correspond to the desired number of samples for each targeted class.
  - *or* callable, not for optimizer
    Function taking `y` and returns a `dict`. The keys correspond to the targeted classes and the values correspond to the desired number of samples for each class.
- **random_state** (*union type, optional, not for optimizer, default None*) – Control the randomization of the algorithm.
  - None
    RandomState used by np.random
  - *or* integer
    The seed used by the random number generator
  - *or* numpy.random.RandomState
    Random number generator instance.
- **k_neighbors** (*union type, optional, not for optimizer, default 5*) – Number of nearest neighbours to use to construct synthetic samples.
  - integer
    Number of nearest neighbours to use to construct synthetic samples.
  - *or* Any
    An estimator that inherits from `sklearn.neighbors.base.KNeighborsMixin` that will be used to find the *n_neighbors*.
- **n_jobs** (*integer, optional, not for optimizer, default 1*) – The number of threads to open if possible.

decision_function(*X*)

Confidence scores for all classes.

Note: The decision_function method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters
  X (array *of* items : array *of* items : float) – Features; the outer array is over samples.

Returns
  result – Output data schema for predictions.

Return type
  Any

fit(*X*, *y=None*, ***fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters
- **X** (array *of* items : array *of* items : float) – Features; the outer array is over samples.
- **y** (*union type*) – Target class labels; the array is over samples.
  - array *of* items : float
  - *or* array *of* items : string

predict(*X*, ***predict_params*)

Make predictions.

Note: The predict method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters
: **X** (array *of* items : array *of* items : float) – Features; the outer array is over samples.

Returns
: **result** – Output data schema for predictions.

Return type
: Any

`predict_proba`(*X*)

Probability estimates for all classes.

Note: The predict_proba method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters
: **X** (array *of* items : array *of* items : float) – Features; the outer array is over samples.

Returns
: **result** – Probability of the sample for each class in the model.

Return type
: array *of* items : array *of* items : float

`transform`(*X*, *y=None*)

Transform the data.

Note: The transform method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters
- **X** (array *of* items : array *of* items : float) – Features; the outer array is over samples.
- **y** (*union type*) – Target class labels; the array is over samples.
  - array *of* items : float
  - *or* array *of* items : string
  - *or* None

Returns
: **result** – Output data schema for transformed data.

Return type
: Any

lale.lib.imblearn.smoteenn module

class `lale.lib.imblearn.smoteenn.SMOTEENN`(*, *operator*, *sampling_strategy='auto'*, *random_state=None*, *smote=None*, *enn=None*)

Bases: *PlannedIndividualOp*

Class to perform over-sampling using SMOTE and cleaning using ENN.

This documentation is auto-generated from JSON schemas.

Combine over- and under-sampling using SMOTE and Edited Nearest Neighbours.

Parameters
- **operator** (*operator, optional*) – Trainable Lale pipeline that is trained using the data obtained from the current imbalance corrector.

  Predict, transform, predict_proba or decision_function would just be forwarded to the trained pipeline. If operator is a Planned pipeline, the current imbalance corrector can't be trained without using an optimizer to choose a trainable operator first. Please refer to lale/examples for more examples.

- **sampling_strategy** (*union type, optional, not for optimizer, default 'auto'*) – Sampling information to resample the data set.
  - float, not for optimizer

    Desired ratio of the number of samples in the minority class over the number of samples in the majority class after resampling. Therefore, the ratio is expressed as $\alpha_{os} = N_{rm}/N_M$ where N_{rm} is the number of samples in the minority class after resampling and N_M is the number of samples in the majority class.

    Warning: Only available for **binary** classification. An error is raised for multi-class classification.

  - *or* 'minority', 'not minority', 'not majority', 'all', *or* 'auto'

    The class targeted by the resampling. The number of samples in the different classes will be equalized. Possible choices are:
    * `'minority'`: resample only the minority class;
    * `'not minority'`: resample all classes but the minority class;
    * `'not majority'`: resample all classes but the majority class;
    * `'all'`: resample all classes;
    * `'auto'`: equivalent to `'not majority'`.
  - *or* dict, not for optimizer

    Keys correspond to the targeted classes and values correspond to the desired number of samples for each targeted class.
  - *or* callable, not for optimizer

    Function taking `y` and returns a `dict`. The keys correspond to the targeted classes and the values correspond to the desired number of samples for each class.
- **random_state** (*union type, optional, not for optimizer, default None*) – Control the randomization of the algorithm.
  - None

    RandomState used by np.random
  - *or* integer

    The seed used by the random number generator
  - *or* numpy.random.RandomState

    Random number generator instance.
- **smote** (*union type, optional, not for optimizer, default None*) – The imblearn.over_sampling.SMOTE object to use. If not given, a imblearn.over_sampling.SMOTE object with default parameters will be given.
  - Any
  - *or* None
- **enn** (*union type, optional, not for optimizer, default None*) – The imblearn.under_sampling.EditedNearestNeighbours object to use. If not given, a imblearn.under_sampling.EditedNearestNeighbours object with sampling strategy='all' will be given.
  - Any
  - *or* None

decision_function(*X*)

Confidence scores for all classes.

Note: The decision_function method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (array *of* items : array *of* items : float) – Features; the outer array is over samples.

Returns

result – Output data schema for predictions.

Return type

Any

fit(*X*, *y=None*, ***fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

- **X** (array *of* items : array *of* items : float) – Features; the outer array is over samples.
- **y** (*union type*) – Target class labels; the array is over samples.
  - array *of* items : float
  - *or* array *of* items : string

predict(*X*, ***predict_params*)

Make predictions.

Note: The predict method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (array *of* items : array *of* items : float) – Features; the outer array is over samples.

Returns

result – Output data schema for predictions.

Return type

Any

predict_proba(*X*)

Probability estimates for all classes.

Note: The predict_proba method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (array *of* items : array *of* items : float) – Features; the outer array is over samples.

Returns

result – Probability of the sample for each class in the model.

Return type

array *of* items : array *of* items : float

transform(*X*, *y=None*)

Transform the data.

Note: The transform method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

- **X** (array *of* items : array *of* items : float) – Features; the outer array is over samples.
- **y** (*union type*) – Target class labels; the array is over samples.
  - array *of* items : float
  - *or* array *of* items : string

- *or* None

Returns
result – Output data schema for transformed data.

Return type
Any

lale.lib.imblearn.smoten module

class lale.lib.imblearn.smoten.**SMOTEN**(**, operator, sampling_strategy='auto', random_state=None, k_neighbors=5, n_jobs=1*)

Bases: *PlannedIndividualOp*

Synthetic Minority Over-sampling Technique for Nominal (SMOTEN).

This documentation is auto-generated from JSON schemas.

Expects that the data to resample are only made of categorical features.

Parameters

- **operator** (*operator*) – Trainable Lale pipeline that is trained using the data obtained from the current imbalance corrector.

  Predict, transform, predict_proba or decision_function would just be forwarded to the trained pipeline. If operator is a Planned pipeline, the current imbalance corrector can't be trained without using an optimizer to choose a trainable operator first. Please refer to lale/examples for more examples.

- **sampling_strategy** (*union type, optional, not for optimizer, default 'auto'*) – Sampling information to resample the data set.
  - float, not for optimizer
    Desired ratio of the number of samples in the minority class over the number of samples in the majority class after resampling. Therefore, the ratio is expressed as $\alpha_{os} = N_{rm}/N_M$ where N_{rm} is the number of samples in the minority class after resampling and N_M is the number of samples in the majority class.

    > **Warning:** Only available for **binary** classification. An error is raised for multi-class classification.

  - *or* 'minority', 'not minority', 'not majority', 'all', *or* 'auto'
    The class targeted by the resampling. The number of samples in the different classes will be equalized. Possible choices are:
    - `'minority'`: resample only the minority class;
    - `'not minority'`: resample all classes but the minority class;
    - `'not majority'`: resample all classes but the majority class;
    - `'all'`: resample all classes;
    - `'auto'`: equivalent to `'not majority'`.
  - *or* dict, not for optimizer
    Keys correspond to the targeted classes and values correspond to the desired number of samples for each targeted class.
  - *or* callable, not for optimizer
    Function taking `y` and returns a `dict`. The keys correspond to the targeted classes and the values correspond to the desired number of samples for each class.

- **random_state** (*union type, optional, not for optimizer, default None*) – Control the randomization of the algorithm.
  - None

    RandomState used by np.random
  - *or* integer

    The seed used by the random number generator
  - *or* numpy.random.RandomState

    Random number generator instance.
- **k_neighbors** (*union type, optional, not for optimizer, default 5*) – Number of nearest neighbours to use to construct synthetic samples.
  - integer

    Number of nearest neighbours to use to construct synthetic samples.
  - *or* Any

    An estimator that inherits from `sklearn.neighbors.base.KNeighborsMixin` that will be used to find the *n_neighbors*.
- **n_jobs** (*integer, optional, not for optimizer, default 1*) – The number of threads to open if possible.

`decision_function(X)`

Confidence scores for all classes.

Note: The decision_function method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (*array*) – Features; the outer array is over samples.

- items : array
  - items : union type
    - float
    - *or* string

Returns

result – Output data schema for predictions.

Return type

Any

`fit(X, y=None, **fit_params)`

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

- **X** (*array*) – Features; the outer array is over samples.
  - items : array
    - items : union type
      - float
      - *or* string
- **y** (*union type*) – Target class labels; the array is over samples.
  - array *of* items : float
  - *or* array *of* items : string

`predict(X, **predict_params)`

Make predictions.

Note: The predict method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters
: **X** (*array*) – Features; the outer array is over samples.
  - items : array
    - items : union type
      - float
      - *or* string

Returns
: **result** – Output data schema for predictions.

Return type
: Any

`predict_proba`(*X*)

Probability estimates for all classes.

Note: The predict_proba method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters
: **X** (*array*) – Features; the outer array is over samples.
  - items : array
    - items : union type
      - float
      - *or* string

Returns
: **result** – Probability of the sample for each class in the model.

Return type
: array *of* items : array *of* items : float

`transform`(*X*, *y=None*)

Transform the data.

Note: The transform method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters
- **X** (*array*) – Features; the outer array is over samples.
  - items : array
    - items : union type
      - float
      - *or* string
- **y** (*union type*) – Target class labels; the array is over samples.
  - array *of* items : float
  - *or* array *of* items : string
  - *or* None

Returns
: **result** – Output data schema for transformed data.

Return type
: Any

lale.lib.imblearn.smotenc module

class `lale.lib.imblearn.smotenc.SMOTENC`(**, operator, categorical_features=None, sampling_strategy='auto', random_state=None, k_neighbors=5, n_jobs=1*)

Bases: *PlannedIndividualOp*

Synthetic Minority Over-sampling Technique for Nominal and Continuous (SMOTENC).

This documentation is auto-generated from JSON schemas.

Can handle some nominal features, but not designed to work with only nominal features.

Parameters

- **operator** (*operator*) – Trainable Lale pipeline that is trained using the data obtained from the current imbalance corrector.

  Predict, transform, predict_proba or decision_function would just be forwarded to the trained pipeline. If operator is a Planned pipeline, the current imbalance corrector can't be trained without using an optimizer to choose a trainable operator first. Please refer to lale/examples for more examples.
- **categorical_features** (*union type, optional, not for optimizer, default None*) – Specifies which features are categorical.
  - None

    Treat all features with non-numeric dtype as categorical.
  - *or* array *of* items : integer

    Indices specifying the categorical features.
  - *or* array *of* items : boolean

    Mask array of shape *(n_features,)* where True indicates the categorical features.
- **sampling_strategy** (*union type, optional, not for optimizer, default 'auto'*) – Sampling information to resample the data set.
  - float, not for optimizer

    Desired ratio of the number of samples in the minority class over the number of samples in the majority class after resampling. Therefore, the ratio is expressed as $\alpha_{os} = N_{rm}/N_M$ where N_{rm} is the number of samples in the minority class after resampling and N_M is the number of samples in the majority class.

    > **Warning:** Only available for **binary** classification. An error is raised for multi-class classification.
  - *or* 'minority', 'not minority', 'not majority', 'all', *or* 'auto'

    The class targeted by the resampling. The number of samples in the different classes will be equalized. Possible choices are:
    - `'minority'`: resample only the minority class;
    - `'not minority'`: resample all classes but the minority class;
    - `'not majority'`: resample all classes but the majority class;
    - `'all'`: resample all classes;
    - `'auto'`: equivalent to `'not majority'`.
  - *or* dict, not for optimizer

    Keys correspond to the targeted classes and values correspond to the desired number of samples for each targeted class.
  - *or* callable, not for optimizer

Function taking `y` and returns a `dict`. The keys correspond to the targeted classes and the values correspond to the desired number of samples for each class.

- **random_state** (*union type, optional, not for optimizer, default None*) – Control the randomization of the algorithm.
  - None
    RandomState used by np.random
  - *or* integer
    The seed used by the random number generator
  - *or* numpy.random.RandomState
    Random number generator instance.
- **k_neighbors** (*union type, optional, not for optimizer, default 5*) – Number of nearest neighbours to use to construct synthetic samples.
  - integer
    Number of nearest neighbours to use to construct synthetic samples.
  - *or* Any
    An estimator that inherits from `sklearn.neighbors.base.KNeighborsMixin` that will be used to find the *n_neighbors*.
- **n_jobs** (*integer, optional, not for optimizer, default 1*) – The number of threads to open if possible.

decision_function(*X*)

Confidence scores for all classes.

Note: The decision_function method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (*array*) – Features; the outer array is over samples.
- items : array
  - items : union type
    - float
    - *or* string

Returns

result – Output data schema for predictions.

Return type

Any

fit(*X*, *y=None*, ***fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

- **X** (*array*) – Features; the outer array is over samples.
  - items : array
    - items : union type
      - float
      - *or* string
- **y** (*union type*) – Target class labels; the array is over samples.
  - array *of* items : float
  - *or* array *of* items : string

predict(*X*, ***predict_params*)

Make predictions.

Note: The predict method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (*array*) – Features; the outer array is over samples.

- items : array
  - items : union type
    - float
    - *or* string

Returns

result – Output data schema for predictions.

Return type

Any

`predict_proba`(*X*)

Probability estimates for all classes.

Note: The predict_proba method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (*array*) – Features; the outer array is over samples.

- items : array
  - items : union type
    - float
    - *or* string

Returns

result – Probability of the sample for each class in the model.

Return type

array *of* items : array *of* items : float

`transform`(*X*, *y=None*)

Transform the data.

Note: The transform method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

- **X** (*array*) – Features; the outer array is over samples.
  - items : array
    - items : union type
      - float
      - *or* string
- **y** (*union type*) – Target class labels; the array is over samples.
  - array *of* items : float
  - *or* array *of* items : string
  - *or* None

Returns

result – Output data schema for transformed data.

Return type

Any

lale.lib.imblearn.svm_smote module

class lale.lib.imblearn.svm_smote.**SVMSMOTE**(***, *operator*, *sampling_strategy='auto'*, *random_state=None*, *k_neighbors=5*, *n_jobs=1*, *m_neighbors=10*, *svm_estimator=None*, *out_step=0.5*)

Bases: `PlannedIndividualOp`

Over-sampling using SVM-SMOTE,

This documentation is auto-generated from JSON schemas.

Variant of SMOTE algorithm which use an SVM algorithm to detect sample to use for generating new synthetic samples.

Parameters

- **operator** (*operator*) – Trainable Lale pipeline that is trained using the data obtained from the current imbalance corrector.

  Predict, transform, predict_proba or decision_function would just be forwarded to the trained pipeline. If operator is a Planned pipeline, the current imbalance corrector can't be trained without using an optimizer to choose a trainable operator first. Please refer to lale/examples for more examples.

- **sampling_strategy** (*union type, optional, not for optimizer, default 'auto'*) – Sampling information to resample the data set.
  - float, not for optimizer

    Desired ratio of the number of samples in the minority class over the number of samples in the majority class after resampling. Therefore, the ratio is expressed as $\alpha_{os} = N_{rm}/N_M$ where N_{rm} is the number of samples in the minority class after resampling and N_M is the number of samples in the majority class.

    > **Warning:** Only available for **binary** classification. An error is raised for multi-class classification.

  - *or* 'minority', 'not minority', 'not majority', 'all', *or* 'auto'

    The class targeted by the resampling. The number of samples in the different classes will be equalized. Possible choices are:
    - `'minority'`: resample only the minority class;
    - `'not minority'`: resample all classes but the minority class;
    - `'not majority'`: resample all classes but the majority class;
    - `'all'`: resample all classes;
    - `'auto'`: equivalent to `'not majority'`.

  - *or* dict, not for optimizer

    Keys correspond to the targeted classes and values correspond to the desired number of samples for each targeted class.

  - *or* callable, not for optimizer

    Function taking `y` and returns a `dict`. The keys correspond to the targeted classes and the values correspond to the desired number of samples for each class.

- **random_state** (*union type, optional, not for optimizer, default None*) – Control the randomization of the algorithm.
  - None

    RandomState used by np.random

  - *or* integer

    The seed used by the random number generator

  - *or* numpy.random.RandomState
    Random number generator instance.
- `k_neighbors` (*union type, optional, not for optimizer, default 5*) – Number of nearest neighbours to use to construct synthetic samples.
  - integer
    Number of nearest neighbours to use to construct synthetic samples.
  - *or* Any
    An estimator that inherits from `sklearn.neighbors.base.KNeighborsMixin` that will be used to find the *n_neighbors*.
- `n_jobs` (*integer, optional, not for optimizer, default 1*) – The number of threads to open if possible.
- `m_neighbors` (*union type, optional, not for optimizer, default 10*) – Number of nearest neighbours to use to determine if a minority sample is in danger.
  - integer
    Number of nearest neighbours to use to construct synthetic samples.
  - *or* Any
    An estimator that inherits from `sklearn.neighbors.base.KNeighborsMixin` that will be used to find the *n_neighbors*.
- `svm_estimator` (*union type, optional, not for optimizer, default None*) – A parametrized sklearn.svm.SVC classifier can be passed.
  - Any
  - *or* None
- `out_step` (*float, optional, not for optimizer, default 0.5*) – Step size when extrapolating.

`decision_function`(*X*)

Confidence scores for all classes.

Note: The decision_function method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters
`X` (array *of* items : array *of* items : float) – Features; the outer array is over samples.

Returns
result – Output data schema for predictions.

Return type
Any

`fit`(*X*, *y=None*, ***fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters
- `X` (array *of* items : array *of* items : float) – Features; the outer array is over samples.
- `y` (*union type*) – Target class labels; the array is over samples.
  - array *of* items : float
  - *or* array *of* items : string

`predict`(*X*, ***predict_params*)

Make predictions.

Note: The predict method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters
X (array *of* items : array *of* items : float) – Features; the outer array is over samples.

Returns
result – Output data schema for predictions.

Return type
Any

`predict_proba`(*X*)

Probability estimates for all classes.

Note: The predict_proba method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters
X (array *of* items : array *of* items : float) – Features; the outer array is over samples.

Returns
result – Probability of the sample for each class in the model.

Return type
array *of* items : array *of* items : float

`transform`(*X*, *y=None*)

Transform the data.

Note: The transform method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

- **X** (array *of* items : array *of* items : float) – Features; the outer array is over samples.
- **y** (*union type*) – Target class labels; the array is over samples.
  - array *of* items : float
  - *or* array *of* items : string
  - *or* None

Returns
result – Output data schema for transformed data.

Return type
Any

Module contents

Scikit-learn compatible wrappers for a subset of the operators from imbalanced-learn along with schemas to enable hyperparameter tuning.

Operators:

- CondensedNearestNeighbour
- EditedNearestNeighbours
- RepeatedEditedNearestNeighbours
- AllKNN
- InstanceHardnessThreshold
- ADASYN
- BorderlineSMOTE

- RandomOverSampler
- RandomUnderSampler
- SMOTE
- SMOTEN
- SMOTENC
- SVMSMOTE
- SMOTEENN

lale.lib.lale package

Submodules

lale.lib.lale.auto_pipeline module

class lale.lib.lale.auto_pipeline.**AutoPipeline**(*, *prediction_type='classification'*, *scoring*, *best_score=0.0*, *verbose=False*, *max_evals=100*, *max_opt_time=600.0*, *max_eval_time=120.0*, *cv=5*)

Bases: `PlannedIndividualOp`

Automatically find a pipeline for a dataset.

This documentation is auto-generated from JSON schemas.

This is a high-level entry point to get an initial trained pipeline without having to specify your own planned pipeline first. It is designed to be simple at the expense of not offering much control. For an example, see demo_auto_pipeline.ipynb.

Parameters

- **prediction_type** ('binary', 'multiclass', 'classification', *or* 'regression', not for optimizer, default 'classification') – The kind of learning problem.
- **scoring** (*union type, not for optimizer*) – Scorer object, or known scorer named by string.
  - None

    When not specified, use *accuracy* for classification tasks and *r2* for regression.
  - *or* union type

    Scorer object, or known scorer named by string.
    - callable

      Callable with signature `scoring(estimator, X, y)` as documented in sklearn scoring.

      The callable has to return a scalar value, such that a higher score is better. This may be created from one of the sklearn metrics using make_scorer. Or it can be one of the scoring callables returned by the factory functions in lale.lib.aif360 metrics, for example, `symmetric_disparate_impact(**fairness_info)`. Or it can be a completely custom user-written Python callable.
    - *or* 'accuracy', 'explained_variance', 'max_error', 'roc_auc', 'roc_auc_ovr', 'roc_auc_ovo', 'roc_auc_ovr_weighted',

'roc_auc_ovo_weighted', 'balanced_accuracy', 'average_precision', 'neg_log_loss', *or* 'neg_brier_score'
Known scorer for classification task.
* *or* 'r2', 'neg_mean_squared_error', 'neg_mean_absolute_error', 'neg_root_mean_squared_error', 'neg_mean_squared_log_error', *or* 'neg_median_absolute_error'
Known scorer for regression task.

- **best_score** (*float, optional, not for optimizer, default 0.0*) – The best score for the specified scorer.

  Given that higher scores are better, passing (best_score - score) as a loss to the minimizing optimizer will maximize the score. By specifying best_score, the loss can be >=0, where 0 is the best loss.
- **verbose** (*boolean, optional, not for optimizer, default False*) – Whether to print errors from each of the trials if any. This is also logged using logger.warning in Hyperopt.
- **max_evals** (*integer, >=1, not for optimizer, default 100*) – Number of trials of Hyperopt search.
- **max_opt_time** (*union type, not for optimizer, default 600.0*) – Maximum amount of time in seconds for the optimization.
  - float, >=0.0
  - *or* None
    No runtime bound.
- **max_eval_time** (*union type, not for optimizer, default 120.0*) – Maximum time in seconds for each evaluation.
  - float, >0.0
  - *or* None
    No runtime bound.
- **cv** (*union type, not for optimizer, default 5*) – Cross-validation as integer or as object that has a split function.

  The fit method performs cross validation on the input dataset for per trial, and uses the mean cross validation performance for optimization. This behavior is also impacted by the handle_cv_failure flag.
  - union type
    * integer, >=2, >=3 for optimizer, <=4 for optimizer, uniform distribution, default 5
      Number of folds for cross-validation.
    * *or* None, not for optimizer
      to use the default 5-fold cross validation
  - *or* CrossvalGenerator, not for optimizer
    Object with split function: generator yielding (train, test) splits as arrays of indices. Can use any of the iterators from https://scikit-learn.org/stable/modules/cross_validation.html#cross-validation-iterators

fit(*X*, *y=None*, ***fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

- **X** (array *of* items : array *of* items : Any) –
- **y** (*union type*) –
  - array *of* items : float

- *or* array *of* items : string
- *or* array *of* items : boolean

predict(*X*, ***predict_params*)

Make predictions.

Note: The predict method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (array *of* items : array *of* items : Any) –

Returns

result –

- array *of* items : float
- *or* array *of* items : string
- *or* array *of* items : boolean

Return type

union type

lale.lib.lale.auto_pipeline.**auto_gbt**(*prediction_type*)

lale.lib.lale.auto_pipeline.**auto_prep**(*X*)

lale.lib.lale.both module

class lale.lib.lale.both.**Both**(**, order='forward', op1, op2*)

Bases: `PlannedIndividualOp`

Combined schema for expected data and hyperparameters.

This documentation is auto-generated from JSON schemas.

Parameters

- **order** ('forward' *or* 'backward', optional, default 'forward') –
- **op1** (`operator, optional`) –
- **op2** (`operator, optional`) –

fit(*X*, *y=None*, ***fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

X (`any type`) –

predict(*X*, ***predict_params*)

Make predictions.

Note: The predict method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

- **X** (`any type`) –
- **y** (`any type`) –

Returns

result – Output data schema for transformations using both.

Return type

Any

predict_proba(*X*)

Probability estimates for all classes.

Note: The predict_proba method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

- **X** (*any type*) –
- **y** (*any type*) –

Returns

result – Output data schema for transformations using both.

Return type

Any

transform(*X*, *y=None*)

Transform the data.

Note: The transform method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

- **X** (*any type*) –
- **y** (*any type*) –

Returns

result – Output data schema for transformations using both.

Return type

Any

lale.lib.lale.concat_features module

lale.lib.lale.grid_search_cv module

class lale.lib.lale.grid_search_cv.**GridSearchCV**(***, *estimator=None*, *scoring*, *cv=5*, *verbose=0*, *n_jobs=None*, *lale_num_samples=None*, *lale_num_grids=None*, *param_grid=None*, *pgo=None*, *observer=None*, *max_opt_time=None*)

Bases: `PlannedIndividualOp`

GridSearchCV performs an exhaustive search over a discretized space.

This documentation is auto-generated from JSON schemas.

Parameters

- **estimator** (*union type, default None*) – Planned Lale individual operator or pipeline.
  - operator
  - *or* None

    lale.lib.sklearn.LogisticRegression
- **scoring** (*union type, not for optimizer*) – Scorer object, or known scorer named by string.
  - None

    When not specified, use *accuracy* for classification tasks and *r2* for regression.
  - *or* union type

    Scorer object, or known scorer named by string.
    - callable

Callable with signature `scoring(estimator, X, y)` as documented in sklearn scoring.

The callable has to return a scalar value, such that a higher score is better. This may be created from one of the sklearn metrics using make_scorer. Or it can be one of the scoring callables returned by the factory functions in lale.lib.aif360 metrics, for example, `symmetric_disparate_impact(**fairness_info)`. Or it can be a completely custom user-written Python callable.

    * *or* 'accuracy', 'explained_variance', 'max_error', 'roc_auc', 'roc_auc_ovr', 'roc_auc_ovo', 'roc_auc_ovr_weighted', 'roc_auc_ovo_weighted', 'balanced_accuracy', 'average_precision', 'neg_log_loss', *or* 'neg_brier_score'

      Known scorer for classification task.

    * *or* 'r2', 'neg_mean_squared_error', 'neg_mean_absolute_error', 'neg_root_mean_squared_error', 'neg_mean_squared_log_error', *or* 'neg_median_absolute_error'

      Known scorer for regression task.

- **cv** (*union type, not for optimizer, default 5*) – Cross-validation as integer or as object that has a split function.

  The fit method performs cross validation on the input dataset for per trial, and uses the mean cross validation performance for optimization. This behavior is also impacted by the handle_cv_failure flag.

  - union type
    * integer, >=2, >=3 for optimizer, <=4 for optimizer, uniform distribution, default 5

      Number of folds for cross-validation.

    * *or* None, not for optimizer

      to use the default 5-fold cross validation

  - *or* CrossvalGenerator, not for optimizer

    Object with split function: generator yielding (train, test) splits as arrays of indices. Can use any of the iterators from https://scikit-learn.org/stable/modules/cross_validation.html#cross-validation-iterators

- **verbose** (*integer, not for optimizer, default 0*) – Controls the verbosity: the higher, the more messages.
- **n_jobs** (*union type, not for optimizer, default None*) – Number of jobs to run in parallel.
  - None

    1 unless in joblib.parallel_backend context.

  - *or* -1

    Use all processors.

  - *or* integer, >=1

    Number of jobs to run in parallel.

- **lale_num_samples** (*union type, not for optimizer, default None*) – How many samples to draw when discretizing a continuous hyperparameter.
  - integer, >=1
  - *or* None

    lale.search.lale_grid_search_cv.DEFAULT_SAMPLES_PER_DISTRIBUTION

- **lale_num_grids** (*union type, not for optimizer, default None*) – How

many top-level disjuncts to explore.
  - None
    If not set, keep all grids.
  - *or* float, >0.0, <1.0
    Fraction of grids to keep.
  - *or* integer, >=1
    Number of grids to keep.
- **param_grid** (*union type, optional, not for optimizer, default None*) –
  - None
    Generated automatically.
  - *or* any type
    Dictionary of hyperparameter ranges in the grid.
- **pgo** (*union type, not for optimizer, default None*) –
  - any type
    lale.search.PGO
  - *or* None
- **observer** (*Any, optional, not for optimizer, default None*) – a class or object with callbacks for observing the state of the optimization
- **max_opt_time** (*union type, not for optimizer, default None*) – Maximum amount of time in seconds for the optimization.
  - float, >=0.0
  - *or* None
    No runtime bound.

fit(*X*, *y=None*, ***fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters
- **X** (*any type*) –
- **y** (*any type*) –

predict(*X*, ***predict_params*)

Make predictions.

Note: The predict method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters
X (*any type*) –

Returns
result

Return type
any type

lale.lib.lale.halving_grid_search_cv module

class lale.lib.lale.halving_grid_search_cv.**HalvingGridSearchCV**(**, estimator=None, scoring, cv=5, verbose=0, factor=3, resource='n_samples', max_resources='auto', min_resources='exhaust', aggressive_elimination=False, refit=True, error_score=nan, return_train_score=False, random_state=None, n_jobs=None, lale_num_samples=None, lale_num_grids=None, param_grid=None, pgo=None, observer=None, max_opt_time=None*)

Bases: `PlannedIndividualOp`

GridSearchCV performs an exhaustive search over a discretized space.

This documentation is auto-generated from JSON schemas.

Parameters

- **estimator** (*union type, default None*) – Planned Lale individual operator or pipeline.
  - operator
  - *or* None

    lale.lib.sklearn.LogisticRegression
- **scoring** (*union type, not for optimizer*) – Scorer object, or known scorer named by string.
  - None

    When not specified, use *accuracy* for classification tasks and *r2* for regression.
  - *or* union type

    Scorer object, or known scorer named by string.
    - callable

      Callable with signature `scoring(estimator, X, y)` as documented in sklearn scoring.

      The callable has to return a scalar value, such that a higher score is better. This may be created from one of the sklearn metrics using make_scorer. Or it can be one of the scoring callables returned by the factory functions in lale.lib.aif360 metrics, for example, `symmetric_disparate_impact(**fairness_info)`. Or it can be a completely custom user-written Python callable.
    - *or* 'accuracy', 'explained_variance', 'max_error', 'roc_auc', 'roc_auc_ovr', 'roc_auc_ovo', 'roc_auc_ovr_weighted', 'roc_auc_ovo_weighted', 'balanced_accuracy', 'average_precision', 'neg_log_loss', *or* 'neg_brier_score'

      Known scorer for classification task.
    - *or* 'r2', 'neg_mean_squared_error', 'neg_mean_absolute_error', 'neg_root_mean_squared_error',

'neg_mean_squared_log_error', *or* 'neg_median_absolute_error'
Known scorer for regression task.

- **cv** (*union type, not for optimizer, default 5*) – Cross-validation as integer or as object that has a split function.

  The fit method performs cross validation on the input dataset for per trial, and uses the mean cross validation performance for optimization. This behavior is also impacted by the handle_cv_failure flag.
  - union type
    - integer, >=2, >=3 for optimizer, <=4 for optimizer, uniform distribution, default 5
      Number of folds for cross-validation.
    - *or* None, not for optimizer
      to use the default 5-fold cross validation
  - *or* CrossvalGenerator, not for optimizer
    Object with split function: generator yielding (train, test) splits as arrays of indices. Can use any of the iterators from https://scikit-learn.org/stable/modules/cross_validation.html#cross-validation-iterators
- **verbose** (*integer, >=0, optional, not for optimizer, default 0*) – Controls the verbosity: the higher, the more messages.
- **factor** (*float, >1, >=2 for optimizer, <=5 for optimizer, optional, not for optimizer, default 3*) – The *halving* parameter, which determines the proportion of candidates that are selected for each subsequent iteration. For example, factor=3 means that only one third of the candidates are selected.
- **resource** (*string, optional, not for optimizer, default 'n_samples'*) – Defines the resource that increases with each iteration. By default, the resource is the number of samples. It can also be set to any parameter of the base estimator that accepts positive integer values, e.g. 'n_iterations' or 'n_estimators' for a gradient boosting estimator.
- **max_resources** (*union type, optional, not for optimizer, default 'auto'*) – The maximum amount of resource that any candidate is allowed to use for a given iteration.
  - 'auto'
  - *or* integer, >=1, not for optimizer
- **min_resources** (*union type, optional, not for optimizer, default 'exhaust'*) – The minimum amount of resource that any candidate is allowed to use for a given iteration
  - 'smallest'
    A heuristic that sets r0 to a small value
  - *or* 'exhaust'
    Sets r0 such that the last iteration uses as much resources as possible
  - *or* integer, >=1, not for optimizer
- **aggressive_elimination** (*boolean, optional, not for optimizer, default False*) – Enable aggresive elimination when there aren't enough resources to reduce the remaining candidates to at most factor after the last iteration
- **refit** (*boolean, optional, not for optimizer, default True*) – Refit an estimator using the best found parameters on the whole dataset.
- **error_score** (*union type, optional, not for optimizer, default nan*) – Value to assign to the score if an error occurs in estimator fitting.
  - 'raise'
    Raise the error
  - *or* nan

- *or* float, not for optimizer
- **return_train_score** (*boolean, optional, not for optimizer, default False*) – Include training scores
- **random_state** (*union type, optional, not for optimizer, default None*) – Pseudo random number generator state used for subsampling the dataset when resources != 'n_samples'. Ignored otherwise.
  - None
    RandomState used by np.random
  - *or* numpy.random.RandomState
    Use the provided random state, only affecting other users of that same random state instance.
  - *or* integer
    Explicit seed.
- **n_jobs** (*union type, not for optimizer, default None*) – Number of jobs to run in parallel.
  - None
    1 unless in joblib.parallel_backend context.
  - *or* -1
    Use all processors.
  - *or* integer, >=1
    Number of jobs to run in parallel.
- **lale_num_samples** (*union type, not for optimizer, default None*) – How many samples to draw when discretizing a continuous hyperparameter.
  - integer, >=1
  - *or* None
    lale.search.lale_grid_search_cv.DEFAULT_SAMPLES_PER_DISTRIBUTION
- **lale_num_grids** (*union type, not for optimizer, default None*) – How many top-level disjuncts to explore.
  - None
    If not set, keep all grids.
  - *or* float, >0.0, <1.0
    Fraction of grids to keep.
  - *or* integer, >=1
    Number of grids to keep.
- **param_grid** (*union type, optional, not for optimizer, default None*) –
  - None
    Generated automatically.
  - *or* any type
    Dictionary of hyperparameter ranges in the grid.
- **pgo** (*union type, not for optimizer, default None*) –
  - any type
    lale.search.PGO
  - *or* None
- **observer** (*Any, optional, not for optimizer, default None*) – a class or object with callbacks for observing the state of the optimization
- **max_opt_time** (*union type, not for optimizer, default None*) – Maximum amount of time in seconds for the optimization.
  - float, >=0.0
  - *or* None
    No runtime bound.

Notes

constraint-1 : any type
max_resources is set to 'auto' if and only if resource is set to 'n_samples'penalty with the liblinear solver.

fit(*X*, *y=None*, ***fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

- **X** (*any type*) –
- **y** (*any type*) –

predict(*X*, ***predict_params*)

Make predictions.

Note: The predict method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (*any type*) –

Returns

result

Return type

any type

lale.lib.lale.hyperopt module

class lale.lib.lale.hyperopt.**Hyperopt**(**, estimator=None, scoring, best_score=0.0, args_to_scorer=None, cv=5, handle_cv_failure=False, verbose=False, show_progressbar=True, algo='tpe', max_evals=50, frac_evals_with_defaults=0, max_opt_time=None, max_eval_time=None, pgo=None*)

Bases: *PlannedIndividualOp*

Hyperopt is a popular open-source Bayesian optimizer.

This documentation is auto-generated from JSON schemas.

Examples

```
>>> from lale.lib.sklearn import LogisticRegression as LR
>>> clf = Hyperopt(estimator=LR, cv=3, max_evals=5)
>>> from sklearn import datasets
>>> diabetes = datasets.load_diabetes()
>>> X = diabetes.data[:150]
>>> y = diabetes.target[:150]
>>> trained = clf.fit(X, y)
>>> predictions = trained.predict(X)
```

Other scoring metrics:

```
>>> from sklearn.metrics import make_scorer, f1_score
>>> clf = Hyperopt(estimator=LR,
...     scoring=make_scorer(f1_score, average='macro'), cv=3, max_evals=5)
```

Parameters

- **estimator** (*union type, default None*) – Planned Lale individual operator or pipeline.
  - operator
  - *or* None

    lale.lib.sklearn.LogisticRegression
- **scoring** (*union type, optional, not for optimizer*) – Scorer object, or known scorer named by string.
  - None

    When not specified, use *accuracy* for classification tasks and *r2* for regression.
  - *or* union type

    Scorer object, or known scorer named by string.
    - callable

      Callable with signature `scoring(estimator, X, y)` as documented in sklearn scoring.

      The callable has to return a scalar value, such that a higher score is better. This may be created from one of the sklearn metrics using make_scorer. Or it can be one of the scoring callables returned by the factory functions in lale.lib.aif360 metrics, for example, `symmetric_disparate_impact(**fairness_info)`. Or it can be a completely custom user-written Python callable.
    - *or* 'accuracy', 'explained_variance', 'max_error', 'roc_auc', 'roc_auc_ovr', 'roc_auc_ovo', 'roc_auc_ovr_weighted', 'roc_auc_ovo_weighted', 'balanced_accuracy', 'average_precision', 'neg_log_loss', *or* 'neg_brier_score'

      Known scorer for classification task.
    - *or* 'r2', 'neg_mean_squared_error', 'neg_mean_absolute_error', 'neg_root_mean_squared_error', 'neg_mean_squared_log_error', *or* 'neg_median_absolute_error'

      Known scorer for regression task.
- **best_score** (*float, optional, not for optimizer, default 0.0*) – The best score for the specified scorer.

  Given that higher scores are better, passing `(best_score - score)` as a loss to the minimizing optimizer will maximize the score. By specifying best_score, the loss can be >=0, where 0 is the best loss.
- **args_to_scorer** (*union type, optional, not for optimizer, default None*) – A dictionary of additional keyword arguments to pass to the scorer. Used for cases where the scorer has a signature such as `scorer(estimator, X, y, **kwargs)`.
  - dict
  - *or* None
- **cv** (*union type, default 5*) – Cross-validation as integer or as object that has a split function.

The fit method performs cross validation on the input dataset for per trial, and uses the mean cross validation performance for optimization. This behavior is also impacted by the handle_cv_failure flag.

- union type
  - integer, >=2, >=3 for optimizer, <=4 for optimizer, uniform distribution, default 5

    Number of folds for cross-validation.
  - *or* None, not for optimizer

    to use the default 5-fold cross validation
- *or* CrossvalGenerator, not for optimizer

  Object with split function: generator yielding (train, test) splits as arrays of indices. Can use any of the iterators from https://scikit-learn.org/stable/modules/cross_validation.html#cross-validation-iterators

- **handle_cv_failure** (*boolean, not for optimizer, default False*) – How to deal with cross validation failure for a trial.

  If True, continue the trial by doing a 80-20 percent train-validation split of the dataset input to fit and report the score on the validation part. If False, terminate the trial with FAIL status.
- **verbose** (*boolean, optional, not for optimizer, default False*) – Whether to print errors from each of the trials if any. This is also logged using logger.warning.
- **show_progressbar** (*boolean, not for optimizer, default True*) – Display progress bar during optimization.
- **algo** (*union type, optional, not for optimizer, default 'tpe'*) – Algorithm for searching the space.
  - 'tpe'

    tree-structured Parzen estimator: https://proceedings.neurips.cc/paper/2011/hash/86e8f7ab32cfd12577bc2619bc635690-Abstract.html
  - *or* 'atpe'

    adaptive TPE
  - *or* 'rand'

    random search
  - *or* 'anneal'

    variant on random search that takes some advantage of a smooth response surface
- **max_evals** (*integer, >=1, default 50*) – Number of trials of Hyperopt search.
- **frac_evals_with_defaults** (*float, >=0.0, optional, not for optimizer, default 0*) – Sometimes, using default values of hyperparameters works quite well. This value would allow a fraction of the trials to use default values. Hyperopt searches the entire search space for (1-frac_evals_with_defaults) fraction of max_evals.
- **max_opt_time** (*union type, not for optimizer, default None*) – Maximum amount of time in seconds for the optimization.
  - float, >=0.0
  - *or* None

    No runtime bound.
- **max_eval_time** (*union type, optional, not for optimizer, default None*) – Maximum amout of time in seconds for each evaluation.
  - float, >=0.0
  - *or* None

    No runtime bound.

- **pgo** (*union type, not for optimizer, default None*) –
  - any type
    lale.search.PGO
  - *or* None

fit(*X*, *y=None*, ***fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

- **X** (*any type*) –
- **y** (*any type*) –

predict(*X*, ***predict_params*)

Make predictions.

Note: The predict method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (*any type*) –

Returns

result

Return type

any type

lale.lib.lale.identity_wrapper module

class lale.lib.lale.identity_wrapper.**IdentityWrapper**(***, *op*)

Bases: *PlannedIndividualOp*

Combined schema for expected data and hyperparameters.

This documentation is auto-generated from JSON schemas.

Parameters

op (*operator, optional*) –

fit(*X*, *y=None*, ***fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

X (*any type*) –

predict(*X*, ***predict_params*)

Make predictions.

Note: The predict method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

- **X** (*any type*) –
- **y** (*any type*) –

Returns

result – Output data schema for transformations using identity.

Return type
Any

`predict_proba(X)`

Probability estimates for all classes.

Note: The predict_proba method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters
- **X** (*any type*) –
- **y** (*any type*) –

Returns
result – Output data schema for transformations using identity.

Return type
Any

`transform(X, y=None)`

Transform the data.

Note: The transform method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters
- **X** (*any type*) –
- **y** (*any type*) –

Returns
result – Output data schema for transformations using identity.

Return type
Any

lale.lib.lale.no_op module

`class lale.lib.lale.no_op.NoOp(*args, _lale_trained=False, _lale_impl=None, **kwargs)`

Bases: *TrainedIndividualOp*

Passes the data through unchanged.

This documentation is auto-generated from JSON schemas.

`transform(X, y=None)`

Transform the data.

Parameters
X (*Any*) – Features; no restrictions on data type.

Returns
result – Features; no restrictions on data type.

Return type
Any

lale.lib.lale.observing module

class lale.lib.lale.observing.**LoggingObserver**

Bases: object

An observer that logs everything. This is also useful for debugging, since you can set breakpoints here

class lale.lib.lale.observing.**Observing**(**, op, observer*)

Bases: *PlannedIndividualOp*

This should functionally be identical to the identity wrapper, except that it calls methods on the observer (if they exist) before and after calls to the underlying wrapper. This is similar to aspect-oriented programming. See also Tee, which provides a simpler method for observing/logging data.

This documentation is auto-generated from JSON schemas.

Parameters

- **op** (*operator, optional*) –
- **observer** (*Any, optional, not for optimizer*) –

fit(*X, y=None, **fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

X (*any type*) –

predict(*X, **predict_params*)

Make predictions.

Note: The predict method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

- **X** (*any type*) –
- **y** (*any type*) –

Returns

result – Output data schema for transformations using identity.

Return type

Any

predict_proba(*X*)

Probability estimates for all classes.

Note: The predict_proba method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

- **X** (*any type*) –
- **y** (*any type*) –

Returns

result – Output data schema for transformations using identity.

Return type

Any

transform(*X, y=None*)

Transform the data.

Note: The transform method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

- **X** (*any type*) –
- **y** (*any type*) –

Returns

result – Output data schema for transformations using identity.

Return type

Any

`lale.lib.lale.observing.observe(f)`

lale.lib.lale.optimize_last module

`class lale.lib.lale.optimize_last.OptimizeLast(*, estimator=None, last_optimizer=None, optimizer_args=None)`

Bases: *PlannedIndividualOp*

OptimizeLast is a wrapper around other optimizers, which runs the given optimizer

This documentation is auto-generated from JSON schemas.

against the suffix, after transforming the data according to the prefix, and then stitches the result together into a single trained pipeline.

Examples

Parameters

- **estimator** (*union type, not for optimizer, default None*) – Planned Lale individual operator or pipeline.
  - operator
  - *or* None

    lale.lib.sklearn.LogisticRegression
- **last_optimizer** (*union type, not for optimizer, default None*) – Lale optimizer. If (default) None is specified, Hyperopt is used.
  - operator *of* None
  - *or* None
- **optimizer_args** (*union type, optional, not for optimizer, default None*) – Parameters to be passed to the optimizer
  - dict
  - *or* None

`fit(X, y=None, **fit_params)`

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

- **X** (*any type*) –
- **y** (*any type*) –

`predict(X, **predict_params)`

Make predictions.

Note: The predict method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (*any type*) –

Returns

result

Return type

any type

lale.lib.lale.optimize_suffix module

class lale.lib.lale.optimize_suffix.**OptimizeSuffix**(**, prefix=None, suffix=None, optimizer=None, optimizer_args=None*)

Bases: *PlannedIndividualOp*

OptimizeSuffix is a wrapper around other optimizers, which runs the given optimizer

This documentation is auto-generated from JSON schemas.

against the suffix, after transforming the data according to the prefix, and then stitches the result together into a single trained pipeline.

Examples

Parameters

- **prefix** (*union type, not for optimizer, default None*) – Trained Lale operator or pipeline, by default None.
  - operator *of* None
  - *or* None
- **suffix** (*union type, not for optimizer, default None*) – Lale operator or pipeline, which is to be optimized. If (default) None is specified, LogisticRegression is used.
  - operator *of* None
  - *or* None
- **optimizer** (*union type, not for optimizer, default None*) – Lale optimizer. If (default) None is specified, Hyperopt is used.
  - operator *of* None
  - *or* None
- **optimizer_args** (*union type, optional, not for optimizer, default None*) – Parameters to be passed to the optimizer
  - dict
  - *or* None

fit(*X, y=None, **fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

- **X** (*any type*) –
- **y** (*any type*) –

predict(*X, **predict_params*)

Make predictions.

Note: The predict method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (*any type*) –

Returns

result

Return type

any type

lale.lib.lale.sample_based_voting module

class lale.lib.lale.sample_based_voting.**SampleBasedVoting**(**args*, *_lale_trained=False*, *_lale_impl=None*, ***kwargs*)

Bases: *TrainedIndividualOp*

Treat the input as labels and use the end_index_list to produce labels using voting. Note that here, X contains the label and no y is accepted.

This documentation is auto-generated from JSON schemas.

transform(*X*, *y=None*)

Transform the data.

Parameters

- **X** (array *of* items : Any) – Labels from the previous component in a pipeline.
- **end_index_list** (*Any, optional*) – For each output label to be produced, end_index_list is supposed to contain the index of the last element corresponding to the original input.

Returns

result

Return type

array *of* items : Any

lale.lib.lale.smac module

class lale.lib.lale.smac.**SMAC**(***, *estimator=None*, *scoring*, *best_score=0.0*, *cv=5*, *handle_cv_failure=False*, *max_evals=50*, *max_opt_time=None*, *lale_num_grids=None*)

Bases: *PlannedIndividualOp*

SMAC, the optimizer used inside auto-weka and auto-sklearn.

This documentation is auto-generated from JSON schemas.

Examples

```
>>> from sklearn.metrics import make_scorer, f1_score, accuracy_score
>>> lr = LogisticRegression()
>>> clf = SMAC(estimator=lr, scoring='accuracy', cv=5)
>>> from sklearn import datasets
>>> diabetes = datasets.load_diabetes()
>>> X = diabetes.data[:150]
>>> y = diabetes.target[:150]
>>> trained = clf.fit(X, y)
>>> predictions = trained.predict(X)
```

Other scoring metrics:

```
>>> clf = SMAC(estimator=lr, scoring=make_scorer(f1_score, average='macro'), cv=3,␣
↪max_evals=2)
```

Parameters

- `estimator` (*union type, default None*) – Planned Lale individual operator or pipeline.
  - operator
  - *or* None

    lale.lib.sklearn.LogisticRegression
- `scoring` (*union type, optional, not for optimizer*) – Scorer object, or known scorer named by string.
  - None

    When not specified, use *accuracy* for classification tasks and *r2* for regression.
  - *or* union type

    Scorer object, or known scorer named by string.
    - callable

      Callable with signature `scoring(estimator, X, y)` as documented in sklearn scoring.

      The callable has to return a scalar value, such that a higher score is better. This may be created from one of the sklearn metrics using make_scorer. Or it can be one of the scoring callables returned by the factory functions in lale.lib.aif360 metrics, for example, `symmetric_disparate_impact(**fairness_info)`. Or it can be a completely custom user-written Python callable.
    - *or* 'accuracy', 'explained_variance', 'max_error', 'roc_auc', 'roc_auc_ovr', 'roc_auc_ovo', 'roc_auc_ovr_weighted', 'roc_auc_ovo_weighted', 'balanced_accuracy', 'average_precision', 'neg_log_loss', *or* 'neg_brier_score'

      Known scorer for classification task.
    - *or* 'r2', 'neg_mean_squared_error', 'neg_mean_absolute_error', 'neg_root_mean_squared_error', 'neg_mean_squared_log_error', *or* 'neg_median_absolute_error'

      Known scorer for regression task.
- `best_score` (*float, optional, not for optimizer, default 0.0*) – The best score for the specified scorer.

  Given that higher scores are better, passing `(best_score - score)` as a loss to the minimizing optimizer will maximize the score. By specifying best_score, the loss can be >=0, where 0 is the best loss.
- `cv` (*union type, not for optimizer, default 5*) – Cross-validation as integer or as object that has a split function.

  The fit method performs cross validation on the input dataset for per trial, and uses the mean cross validation performance for optimization. This behavior is also impacted by the handle_cv_failure flag.
  - union type
    - integer, >=2, >=3 for optimizer, <=4 for optimizer, uniform distribution, default 5

Number of folds for cross-validation.

    * *or* None, not for optimizer

      to use the default 5-fold cross validation

  - *or* CrossvalGenerator, not for optimizer

    Object with split function: generator yielding (train, test) splits as arrays of indices. Can use any of the iterators from https://scikit-learn.org/stable/modules/cross_validation.html#cross-validation-iterators

- **handle_cv_failure** (*boolean, not for optimizer, default False*) – How to deal with cross validation failure for a trial.

  If True, continue the trial by doing a 80-20 percent train-validation split of the dataset input to fit and report the score on the validation part. If False, terminate the trial with FAIL status.

- **max_evals** (*integer, >=1, not for optimizer, default 50*) – Number of trials of SMAC search i.e. runcount_limit of SMAC.
- **max_opt_time** (*union type, not for optimizer, default None*) – Maximum amount of time in seconds for the optimization.
  - float, >=0.0
  - *or* None

    No runtime bound.

- **lale_num_grids** (*union type, not for optimizer, default None*) –
  - None

    If not set, keep all grids.

  - *or* float, >0.0, <1.0

    Fraction of grids to keep.

  - *or* integer, >=1

    Number of grids to keep.

fit(*X*, *y=None*, ***fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

- **X** (*array*) –
  - items : union type
    * array *of* items : number *or* string
    * *or* string
- **y** (array *of* items : float) –

predict(*X*, ***predict_params*)

Make predictions.

Note: The predict method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (*array, optional*) –

- items : union type
  - array *of* items : number *or* string
  - *or* string

Returns

result

Return type

array *of* items : float

lale.lib.lale.tee module

class lale.lib.lale.tee.**Tee**(**args*, *_lale_trained=False*, *_lale_impl=None*, ***kwargs*)

Bases: `TrainedIndividualOp`

Passes the data through unchanged (like NoOp), first giving it to an listener. Useful for debugging and logging.Similar to Observing, which provides a higher order operator with more comprehensive abilities.

This documentation is auto-generated from JSON schemas.

Parameters

listener (*union type, optional, not for optimizer*) –

- callable

  A callable (lambda, method, class that implements __call__, . . .)that accepts to arguments: X and y (which may be None). When transformis called on this operator, the callable will be passed the givenX and y values

- *or* None

  No listener. Causes this operator to behave like NoOp.

transform(*X*, *y=None*)

Transform the data.

Parameters

X (*Any*) – Features; no restrictions on data type.

Returns

result – Features; no restrictions on data type.

Return type

Any

lale.lib.lale.time_series_transformer module

class lale.lib.lale.time_series_transformer.**CorrelationMatrix**

Bases: `object`

Calculate correlation coefficients matrix across all EEG channels.

apply(*data*)

get_name()

class lale.lib.lale.time_series_transformer.**Eigenvalues**

Bases: `object`

Take eigenvalues of a matrix, and sort them by magnitude in order to make them useful as features (as they have no inherent order).

apply(*data*)

get_name()

class lale.lib.lale.time_series_transformer.**FFT**

Bases: `object`

Apply Fast Fourier Transform to the last axis.

apply(*data*)

get_name()

class lale.lib.lale.time_series_transformer.**FFTWithTimeFreqCorrelation**(*start*, *end*, *max_hz*, *scale_option*)

Bases: `object`

Combines FFT with time and frequency correlation, taking both correlation coefficients and eigenvalues.

apply(*data*)

get_name()

class lale.lib.lale.time_series_transformer.**FreqCorrelation**(*start*, *end*, *scale_option*, *with_fft=False*, *with_corr=True*, *with_eigen=True*)

Bases: `object`

Correlation in the frequency domain. First take FFT with (start, end) slice options, then calculate correlation co-efficients on the FFT output, followed by calculating eigenvalues on the correlation co-efficients matrix. The output features are (fft, upper_right_diagonal(correlation_coefficients), eigenvalues) Features can be selected/omitted using the constructor arguments.

apply(*data*)

get_name()

class lale.lib.lale.time_series_transformer.**Log10**

Bases: `object`

Apply Log10

apply(*data*)

get_name()

class lale.lib.lale.time_series_transformer.**Magnitude**

Bases: `object`

Job: Take magnitudes of Complex data

apply(*data*)

get_name()

class lale.lib.lale.time_series_transformer.**Pipeline**(*pipeline*)

Bases: `object`

A Pipeline is an object representing the data transformations to make on the input data, finally outputting extracted features. pipeline: List of transforms to apply one by one to the input data

apply(*data*)

get_name()

class lale.lib.lale.time_series_transformer.**Resample**(*sample_rate*)

Bases: `object`

Resample time-series data.

apply(*data*)

get_name()

class lale.lib.lale.time_series_transformer.**Slice**(*start*, *stop*)

Bases: `object`

Job: Take a slice of the data on the last axis. Note: Slice(x, y) works like a normal python slice, that is x to (y-1) will be taken.

apply(*data*)

get_name()

class lale.lib.lale.time_series_transformer.**StandardizeFirst**

Bases: `object`

Scale across the first axis.

apply(*data*)

get_name()

class lale.lib.lale.time_series_transformer.**StandardizeLast**

Bases: `object`

Scale across the last axis.

apply(*data*)

get_name()

class lale.lib.lale.time_series_transformer.**TimeCorrelation**(*max_hz*, *scale_option*, *with_corr=True*, *with_eigen=True*)

Bases: `object`

Correlation in the time domain. First downsample the data, then calculate correlation co-efficients followed by calculating eigenvalues on the correlation co-efficients matrix. The output features are (upper_right_diagonal(correlation_coefficients), eigenvalues) Features can be selected/omitted using the constructor arguments.

apply(*data*)

get_name()

class lale.lib.lale.time_series_transformer.**TimeFreqEigenVectors**(**args*, *_lale_trained=False*, *_lale_impl=None*, ***kwargs*)

Bases: *TrainedIndividualOp*

Combined schema for expected data and hyperparameters.

This documentation is auto-generated from JSON schemas.

Parameters

- **window_length** (*float, >=0.25 for optimizer, <=2 for optimizer, uniform distribution, default 1*) – TODO
- **window_step** (*float, >=0.25 for optimizer, <=1 for optimizer, uniform distribution, default 0.5*) – TODO
- **fft_min_freq** (*integer, not for optimizer, default 1*) – TODO
- **fft_max_freq** (*integer, >=2 for optimizer, <=30 for optimizer, uniform distribution, default 24*) – TODO
- **sampling_frequency** (*integer, not for optimizer, default 250*) – TODO

transform(*X*, *y=None*)

Transform the data.

Parameters

- **X** (array *of* items : array *of* items : array *of* items : float) – The input data to complete.
- **y** (*array*) –
  - items : union type
    - integer
    - *or* string

Returns

result – The input data to complete.

Return type

array *of* items : array

class lale.lib.lale.time_series_transformer.**seizure_type_data**(*seizure_type*, *data*)

Bases: `tuple`

Create new instance of seizure_type_data(seizure_type, data)

data

Alias for field number 1

seizure_type

Alias for field number 0

lale.lib.lale.time_series_transformer.**upper_right_triangle**(*matrix*)

lale.lib.lale.topk_voting_classifier module

class lale.lib.lale.topk_voting_classifier.**TopKVotingClassifier**(*, *estimator=None*, *optimizer=None*, *args_to_optimizer=None*, *k=10*)

Bases: *PlannedIndividualOp*

This operator creates a voting ensemble from top k performing pipelines from the given planned pipeline.

This documentation is auto-generated from JSON schemas.

Parameters

- **estimator** (*union type, not for optimizer, default None*) – Planned Lale individual operator or pipeline.
  - operator *of* None
  - *or* None
- **optimizer** (*union type, optional, not for optimizer, default None*) –
  Optimizer class to be used during the two stages of optimization.
  Default of None uses Hyperopt internally. Currently, only Hyperopt is supported as an optimizer.
  - operator *of* None
  - *or* None
- **args_to_optimizer** (*union type, optional, not for optimizer, default None*) –
  Dictionary of keyword arguments required to be used for the given optimizer
  as applicable for the given task. For example, max_evals, cv, scoring etc. for Hyperopt. If None, default values for the optimizer would be used.
  - dict

– *or* None
- **k** (*integer, >=1, optional, not for optimizer, default 10*) –
Number of top pipelines to be used for the voting ensemble. If the number of
successful trials of the optimizer are less than k, the ensemble will use only successful trials.

fit(*X*, *y=None*, ***fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

- **X** (*any type*) –
- **y** (*any type*) –

predict(*X*, ***predict_params*)

Make predictions.

Note: The predict method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters
X (*any type*) –

Returns
result

Return type
any type

Module contents

Lale operators with schemas.

Operators

Estimators:

- lale.lib.lale. AutoPipeline
- lale.lib.lale. GridSearchCV
- lale.lib.lale. HalvingGridSearchCV
- lale.lib.lale. Hyperopt
- lale.lib.lale. OptimizeLast
- lale.lib.lale. OptimizeSuffix
- lale.lib.lale. SMAC
- lale.lib.lale. TopKVotingClassifier

Transformers:

- lale.lib.rasl. Aggregate
- lale.lib.rasl. Alias
- lale.lib.rasl. Batching

- lale.lib.rasl. ConcatFeatures
- lale.lib.rasl. Filter
- lale.lib.rasl. GroupBy
- lale.lib.rasl. Join
- lale.lib.rasl. Map
- lale.lib.lale. NoOp
- lale.lib.rasl. OrderBy
- lale.lib.rasl. Project
- lale.lib.rasl. Relational
- lale.lib.lale. SampleBasedVoting
- lale.lib.rasl. Scan
- lale.lib.rasl. SplitXy
- lale.lib.lale. Tee

Estimators and transformers:

- lale.lib.lale. Both
- lale.lib.lale. IdentityWrapper
- lale.lib.lale. Observing

Functions:

- lale.lib.lale. categorical
- lale.lib.lale. date_time
- SparkExplainer. spark_explainer

lale.lib.lightgbm package

Submodules

lale.lib.lightgbm.lgbm_classifier module

class lale.lib.lightgbm.lgbm_classifier.**LGBMClassifier**(**, boosting_type='gbdt', num_leaves=31, max_depth=-1, learning_rate=0.1, n_estimators=200, subsample_for_bin=200000, objective=None, class_weight=None, min_split_gain=0.0, min_child_weight=0.001, min_child_samples=20, subsample=1.0, subsample_freq=0, colsample_bytree=1.0, reg_alpha=0.0, reg_lambda=0.0, random_state=None, n_jobs=-1, importance_type='split', n_job=None*)

Bases: `PlannedIndividualOp`

Combined schema for expected data and hyperparameters.

This documentation is auto-generated from JSON schemas.

Parameters

- **boosting_type** (*union type, default 'gbdt'*) –
  - 'gbdt'
    Traditional Gradient Boosting Decision Tree.
  - *or* 'dart'
    Dropouts meet Multiple Additive Regression Trees.
  - *or* 'goss', not for optimizer
    Gradient-based One-Side Sampling.
  - *or* 'rf', not for optimizer
    Random Forest.

  See also *constraint-1*, *constraint-2*.
- **num_leaves** (*union type, optional, default 31*) – Maximum tree leaves for base learners
  - integer, not for optimizer
  - *or* 2, 4, 8, 32, 64, 128, *or* 16
- **max_depth** (union type, not for optimizer, default -1 *of* integer, >=3 for optimizer, <=5 for optimizer) – Maximum tree depth for base learners, <=0 means no limit
- **learning_rate** (*float, >=0.02 for optimizer, <=1.0 for optimizer, loguniform distribution, default 0.1*) – Boosting learning rate.
- **n_estimators** (*integer, >=50 for optimizer, <=1000 for optimizer, uniform distribution, default 200*) – Number of boosted trees to fit.
- **subsample_for_bin** (*integer, optional, not for optimizer, default 200000*) – Number of samples for constructing bins.
- **objective** (*union type, optional, not for optimizer, default None*) – Specify the learning task and the corresponding learning objective or a custom objective function to be used
  - dict
  - *or* 'binary', 'multiclass', *or* None
- **class_weight** (*union type, optional, not for optimizer, default None*) – Weights associated with classes
  - dict
  - *or* 'balanced' *or* None
- **min_split_gain** (*float, optional, not for optimizer, default 0.0*) – Minimum loss reduction required to make a further partition on a leaf node of the tree.
- **min_child_weight** (*float, >=0.0001 for optimizer, <=0.01 for optimizer, optional, default 0.001*) – Minimum sum of instance weight (hessian) needed in a child (leaf).
- **min_child_samples** (*integer, >=5 for optimizer, <=30 for optimizer, uniform distribution, default 20*) – Minimum number of data needed in a child (leaf).
- **subsample** (*float, >0.0, >=0.01 for optimizer, <=1.0, <=1.0 for optimizer, uniform distribution, default 1.0*) – Subsample ratio of the training instance.

  See also *constraint-2*.
- **subsample_freq** (*integer, >=0 for optimizer, <=5 for optimizer, uniform distribution, default 0*) – Frequence of subsample, <=0 means no enable.

  See also *constraint-2*.

- **colsample_bytree** (*float, >=0.01 for optimizer, <=1.0 for optimizer, optional, default 1.0*) – Subsample ratio of columns when constructing each tree.
- **reg_alpha** (*float, >=0.0 for optimizer, <=1.0 for optimizer, optional, default 0.0*) – L1 regularization term on weights.
- **reg_lambda** (*float, >=0.0 for optimizer, <=1.0 for optimizer, optional, default 0.0*) – L2 regularization term on weights.
- **random_state** (*union type, optional, not for optimizer, default None*) – Random number seed. If None, default seeds in C++ code will be used.
  - integer
  - *or* numpy.random.RandomState
  - *or* None
- **n_jobs** (*integer, optional, not for optimizer, default -1*) – Number of parallel threads.
- **importance_type** ('split' *or* 'gain', optional, not for optimizer, default 'split') – The type of feature importance to be filled into *feature_importances_*.
- **n_job** (*union type, optional, not for optimizer, default None*) – Number of parallel threads to use for training (can be changed at prediction time by passing it as an extra keyword argument). For better performance, it is recommended to set this to the number of physical cores in the CPU. Negative integers are interpreted as following joblib's formula (n_cpus + 1 + n_jobs), just like scikit-learn (so e.g. -1 means using all threads). A value of zero corresponds the default number of threads configured for OpenMP in the system.
  - integer

    Number of parallel threads.
  - *or* None

    Use the number of physical cores in the system (its correct detection requires either the joblib or the psutil util libraries to be installed).

Notes

constraint-1 : union type

boosting_type *rf* needs bagging (which means subsample_freq > 0 and subsample < 1.0)

- boosting_type : negated type *of* 'rf'
- *or* intersection type
  - dict *of* subsample_freq : negated type *of* 0
  - *and* dict *of* subsample : negated type *of* 1.0

constraint-2 : union type

boosting_type *goss* cannot use bagging (which means subsample_freq = 0 and subsample = 1.0)

- boosting_type : negated type *of* 'goss'
- *or* subsample_freq : 0
- *or* subsample : 1.0

fit(*X*, *y=None*, ***fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

- **X** (array *of* items : array *of* items : float) – The input samples. Internally, it will be converted to
- **y** (*union type*) – Labels
  - array *of* items : float
  - *or* array *of* items : string

  - *or* array *of* items : boolean
- **sample_weight** (*union type, optional, default None*) – Weights of training data.
  - array *of* items : float
  - *or* None
- **init_score** (*union type, optional, default None*) – Init score of training data.
  - array *of* items : float
  - *or* None
- **group** (*any type, optional, default None*) – Group data of training data.
- **eval_set** (*any type, optional, default None*) – A list of (X, y) tuple pairs to use as validation sets.
- **eval_names** (*any type, optional, default None*) – Names of eval_set.
- **eval_sample_weight** (*any type, optional, default None*) – Weights of eval data.
- **eval_class_weight** (*union type, optional, default None*) – Class weights of eval data.
  - array *of* items : float
  - *or* None
- **eval_init_score** (*any type, optional, default None*) – Init score of eval data.
- **eval_group** (*any type, optional, default None*) – Group data of eval data.
- **eval_metric** (*union type, optional, default None*) – string, list of strings, callable or None, optional (default=None).
  - array *of* items : string
  - *or* 'logloss' *or* None
  - *or* callable
- **early_stopping_rounds** (*union type, optional, default None*) – Activates early stopping. The model will train until the validation score stops improving.
  - integer
  - *or* None
- **verbose** (*union type, optional, default True*) – Requires at least one evaluation data.
  - boolean
  - *or* integer
- **feature_name** (*union type, optional, default 'auto'*) – Feature names. If 'auto' and data is pandas DataFrame, data columns names are used.
  - array *of* items : string
  - *or* 'auto'
- **categorical_feature** (*union type, optional, default 'auto'*) – Categorical features. If list of int, interpreted as indices. If list of strings, interpreted as feature names.
  - array
    - items : union type
      - string
      - *or* integer
  - *or* 'auto'
- **callbacks** (*union type, optional, default None*) – List of callback functions that are applied at each iteration.
  - array *of* items : dict
  - *or* None

partial_fit(*X*, *y=None*, ***fit_params*)

Incremental fit to train train the operator on a batch of samples.

Note: The partial_fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

- **X** (array *of* items : array *of* items : float) – The input samples. Internally, it will be converted to
- **y** (*union type*) – Labels
  - array *of* items : float
  - *or* array *of* items : string
  - *or* array *of* items : boolean
- **sample_weight** (*union type, optional, default None*) – Weights of training data.
  - array *of* items : float
  - *or* None
- **init_score** (*union type, optional, default None*) – Init score of training data.
  - array *of* items : float
  - *or* None
- **group** (*any type, optional, default None*) – Group data of training data.
- **eval_set** (*any type, optional, default None*) – A list of (X, y) tuple pairs to use as validation sets.
- **eval_names** (*any type, optional, default None*) – Names of eval_set.
- **eval_sample_weight** (*any type, optional, default None*) – Weights of eval data.
- **eval_class_weight** (*union type, optional, default None*) – Class weights of eval data.
  - array *of* items : float
  - *or* None
- **eval_init_score** (*any type, optional, default None*) – Init score of eval data.
- **eval_group** (*any type, optional, default None*) – Group data of eval data.
- **eval_metric** (*union type, optional, default None*) – string, list of strings, callable or None, optional (default=None).
  - array *of* items : string
  - *or* 'logloss' *or* None
  - *or* callable
- **early_stopping_rounds** (*union type, optional, default None*) – Activates early stopping. The model will train until the validation score stops improving.
  - integer
  - *or* None
- **verbose** (*union type, optional, default True*) – Requires at least one evaluation data.
  - boolean
  - *or* integer
- **feature_name** (*union type, optional, default 'auto'*) – Feature names. If 'auto' and data is pandas DataFrame, data columns names are used.
  - array *of* items : string
  - *or* 'auto'
- **categorical_feature** (*union type, optional, default 'auto'*) – Categorical features. If list of int, interpreted as indices. If list of strings, interpreted

as feature names.

- array
  - items : union type
    - string
    - *or* integer
- *or* 'auto'

- **callbacks** (*union type, optional, default None*) – List of callback functions that are applied at each iteration.
  - array *of* items : dict
  - *or* None

predict(*X*, ***predict_params*)

Make predictions.

Note: The predict method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

- **X** (array, optional *of* items : array *of* items : float) – Input features matrix.
- **raw_score** (*boolean, optional, default False*) – Whether to predict raw scores.
- **num_iteration** (*union type, optional, default None*) – Limit number of iterations in the prediction.
  - integer
  - *or* None
- **pred_leaf** (*boolean, optional, default False*) – Whether to predict leaf index.
- **pred_contrib** (*boolean, optional, default False*) – Whether to predict feature contributions.

Returns

result – Return the predicted value for each sample.

- array *of* items : float
- *or* array *of* items : string
- *or* array *of* items : boolean

Return type

union type

predict_proba(*X*)

Probability estimates for all classes.

Note: The predict_proba method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

- **X** (array, optional *of* items : array *of* items : float) – Input features matrix.
- **raw_score** (*boolean, optional, default False*) – Whether to predict raw scores.
- **num_iteration** (*union type, optional, default None*) – Limit number of iterations in the prediction.
  - integer
  - *or* None
- **pred_leaf** (*boolean, optional, default False*) – Whether to predict leaf index.
- **pred_contrib** (*boolean, optional, default False*) – Whether to predict feature contributions.

Returns
    result – Return the predicted probability for each class for each sample.

Return type
    array *of* items : array *of* items : float

lale.lib.lightgbm.lgbm_regressor module

class lale.lib.lightgbm.lgbm_regressor.**LGBMRegressor**(*, *boosting_type='gbdt'*, *num_leaves=31*, *max_depth=-1*, *learning_rate=0.1*, *n_estimators=200*, *subsample_for_bin=200000*, *objective=None*, *class_weight=None*, *min_split_gain=0.0*, *min_child_weight=0.001*, *min_child_samples=20*, *subsample=1.0*, *subsample_freq=0*, *colsample_bytree=1.0*, *reg_alpha=0.0*, *reg_lambda=0.0*, *random_state=None*, *n_jobs=-1*, *silent='warn'*, *importance_type='split'*, *n_job=None*)

Bases: *PlannedIndividualOp*

Combined schema for expected data and hyperparameters.

This documentation is auto-generated from JSON schemas.

Parameters

- **boosting_type** (*union type, optional, default 'gbdt'*) –
  - 'gbdt'
    Traditional Gradient Boosting Decision Tree.
  - *or* 'dart'
    Dropouts meet Multiple Additive Regression Trees.
  - *or* 'goss', not for optimizer
    Gradient-based One-Side Sampling.
  - *or* 'rf', not for optimizer
    Random Forest.

  See also *constraint-1*, *constraint-2*.
- **num_leaves** (*union type, optional, default 31*) – Maximum tree leaves for base learners
  - integer, not for optimizer
  - *or* 2, 4, 8, 32, 64, 128, *or* 16
- **max_depth** (union type, optional, not for optimizer, default -1 *of* integer, >=3 for optimizer, <=5 for optimizer) – Maximum tree depth for base learners, <=0 means no limit
- **learning_rate** (*float, >=0.02 for optimizer, <=1.0 for optimizer, loguniform distribution, optional, default 0.1*) – Boosting learning rate.
- **n_estimators** (*integer, >=50 for optimizer, <=1000 for optimizer, uniform distribution, optional, default 200*) – Number of boosted trees to fit.
- **subsample_for_bin** (*integer, optional, not for optimizer, default 200000*) – Number of samples for constructing bins.
- **objective** (*union type, optional, not for optimizer, default None*) – Specify the learning task and the corresponding learning objective or a custom objective function to be used
  - dict
  - *or* 'regression' *or* None

- **class_weight** (*union type, optional, not for optimizer, default None*) – Weights associated with classes
  - dict
  - *or* 'balanced' *or* None
- **min_split_gain** (*float, optional, not for optimizer, default 0.0*) – Minimum loss reduction required to make a further partition on a leaf node of the tree.
- **min_child_weight** (*float, >=0.0001 for optimizer, <=0.01 for optimizer, optional, default 0.001*) – Minimum sum of instance weight (hessian) needed in a child (leaf).
- **min_child_samples** (*integer, >=5 for optimizer, <=30 for optimizer, uniform distribution, optional, default 20*) – Minimum number of data needed in a child (leaf).
- **subsample** (*float, >=0.01 for optimizer, <=1.0 for optimizer, uniform distribution, optional, default 1.0*) – Subsample ratio of the training instance.

  See also *constraint-2*.
- **subsample_freq** (*integer, >=0 for optimizer, <=5 for optimizer, uniform distribution, optional, default 0*) – Frequence of subsample, <=0 means no enable.

  See also *constraint-2*.
- **colsample_bytree** (*float, >=0.01 for optimizer, <=1.0 for optimizer, optional, default 1.0*) – Subsample ratio of columns when constructing each tree.
- **reg_alpha** (*float, >=0.0 for optimizer, <=1.0 for optimizer, optional, default 0.0*) – L1 regularization term on weights.
- **reg_lambda** (*float, >=0.0 for optimizer, <=1.0 for optimizer, optional, default 0.0*) – L2 regularization term on weights.
- **random_state** (*union type, optional, not for optimizer, default None*) – Random number seed. If None, default seeds in C++ code will be used.
  - integer
  - *or* numpy.random.RandomState
  - *or* None
- **n_jobs** (*integer, optional, not for optimizer, default -1*) – Number of parallel threads.
- **silent** (*union type, optional, not for optimizer, default 'warn'*) – Whether to print messages while running boosting.
  - 'warn'
  - *or* boolean
- **importance_type** ('split' *or* 'gain', optional, not for optimizer, default 'split') – The type of feature importance to be filled into *feature_importances_*.
- **n_job** (*union type, optional, not for optimizer, default None*) – Number of parallel threads to use for training (can be changed at prediction time by passing it as an extra keyword argument). For better performance, it is recommended to set this to the number of physical cores in the CPU. Negative integers are interpreted as following joblib's formula (n_cpus + 1 + n_jobs), just like scikit-learn (so e.g. -1 means using all threads). A value of zero corresponds the default number of threads configured for OpenMP in the system.
  - integer

    Number of parallel threads.
  - *or* None

    Use the number of physical cores in the system (its correct detection requires either the joblib or the psutil util libraries to be installed).

Notes

constraint-1 : union type
: boosting_type *rf* needs bagging (which means subsample_freq > 0 and subsample < 1.0)
  - boosting_type : negated type *of* 'rf'
  - *or* intersection type
    - dict *of* subsample_freq : negated type *of* 0
    - *and* dict *of* subsample : negated type *of* 1.0

constraint-2 : union type
: boosting_type *goss* cannot use bagging (which means subsample_freq = 0 and subsample = 1.0)
  - boosting_type : negated type *of* 'goss'
  - *or* subsample_freq : 0
  - *or* subsample : 1.0

fit(*X*, *y=None*, ***fit_params*)
: Train the operator.

  Note: The fit method is not available until this operator is trainable.

  Once this method is available, it will have the following signature:

  Parameters
  - **X** (array *of* items : array *of* items : float) – The input samples. Internally, it will be converted to
  - **y** (array *of* items : float) – Target values real numbers
  - **sample_weight** (*union type, optional, default None*) – Weights of training data.
    - array *of* items : float
    - *or* None
  - **init_score** (*union type, optional, default None*) – Init score of training data.
    - array *of* items : float
    - *or* None
  - **group** (*any type, optional, default None*) – Group data of training data.
  - **eval_set** (*any type, optional, default None*) – A list of (X, y) tuple pairs to use as validation sets.
  - **eval_names** (*any type, optional, default None*) – Names of eval_set.
  - **eval_sample_weight** (*any type, optional, default None*) – Weights of eval data.
  - **eval_class_weight** (*union type, optional, default None*) – Class weights of eval data.
    - array *of* items : float
    - *or* None
  - **eval_init_score** (*any type, optional, default None*) – Init score of eval data.
  - **eval_group** (*any type, optional, default None*) – Group data of eval data.
  - **eval_metric** (*union type, optional, default None*) – string, list of strings, callable or None, optional (default=None).
    - array *of* items : string
    - *or* 'l2' *or* None
    - *or* callable
  - **early_stopping_rounds** (*union type, optional, default None*) – Activates early stopping. The model will train until the validation score stops improving.
    - integer
    - *or* None

- **verbose** (*union type, optional, default True*) – Requires at least one evaluation data.
  - boolean
  - *or* integer
- **feature_name** (*union type, optional, default 'auto'*) – Feature names. If 'auto' and data is pandas DataFrame, data columns names are used.
  - array *of* items : string
  - *or* 'auto'
- **categorical_feature** (*union type, optional, default 'auto'*) – Categorical features. If list of int, interpreted as indices. If list of strings, interpreted as feature names.
  - array
    - items : union type
      - string
      - *or* integer
  - *or* 'auto'
- **callbacks** (*union type, optional, default None*) – List of callback functions that are applied at each iteration.
  - array *of* items : dict
  - *or* None

partial_fit(*X*, *y=None*, ***fit_params*)

Incremental fit to train train the operator on a batch of samples.

Note: The partial_fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

- **X** (array *of* items : array *of* items : float) – The input samples. Internally, it will be converted to
- **y** (array *of* items : float) – Target values real numbers
- **sample_weight** (*union type, optional, default None*) – Weights of training data.
  - array *of* items : float
  - *or* None
- **init_score** (*union type, optional, default None*) – Init score of training data.
  - array *of* items : float
  - *or* None
- **group** (*any type, optional, default None*) – Group data of training data.
- **eval_set** (*any type, optional, default None*) – A list of (X, y) tuple pairs to use as validation sets.
- **eval_names** (*any type, optional, default None*) – Names of eval_set.
- **eval_sample_weight** (*any type, optional, default None*) – Weights of eval data.
- **eval_class_weight** (*union type, optional, default None*) – Class weights of eval data.
  - array *of* items : float
  - *or* None
- **eval_init_score** (*any type, optional, default None*) – Init score of eval data.
- **eval_group** (*any type, optional, default None*) – Group data of eval data.
- **eval_metric** (*union type, optional, default None*) – string, list of strings, callable or None, optional (default=None).

    - array *of* items : string
    - *or* 'l2' *or* None
    - *or* callable
- **early_stopping_rounds** (*union type, optional, default None*) – Activates early stopping. The model will train until the validation score stops improving.
    - integer
    - *or* None
- **verbose** (*union type, optional, default True*) – Requires at least one evaluation data.
    - boolean
    - *or* integer
- **feature_name** (*union type, optional, default 'auto'*) – Feature names. If 'auto' and data is pandas DataFrame, data columns names are used.
    - array *of* items : string
    - *or* 'auto'
- **categorical_feature** (*union type, optional, default 'auto'*) – Categorical features. If list of int, interpreted as indices. If list of strings, interpreted as feature names.
    - array
        * items : union type
            · string
            · *or* integer
    - *or* 'auto'
- **callbacks** (*union type, optional, default None*) – List of callback functions that are applied at each iteration.
    - array *of* items : dict
    - *or* None

predict(*X*, ***predict_params*)

Make predictions.

Note: The predict method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

- **X** (array, optional *of* items : array *of* items : float) – Input features matrix.
- **raw_score** (*boolean, optional, default False*) – Whether to predict raw scores.
- **num_iteration** (*union type, optional, default None*) – Limit number of iterations in the prediction.
    - integer
    - *or* None
- **pred_leaf** (*boolean, optional, default False*) – Whether to predict leaf index.
- **pred_contrib** (*boolean, optional, default False*) – Whether to predict feature contributions.

Returns

result – Return the predicted value for each sample.

Return type

array *of* items : float

Module contents

Scikit-learn compatible wrappers for LightGBM along with schemas to enable hyperparameter tuning.

Operators:

- LGBMClassifier
- LGBMRegressor

lale.lib.rasl package

Submodules

lale.lib.rasl.aggregate module

class lale.lib.rasl.aggregate.**Aggregate**(**args*, *_lale_trained=False*, *_lale_impl=None*, ***kwargs*)

Bases: `TrainedIndividualOp`

Relational algebra aggregate operator.

This documentation is auto-generated from JSON schemas.

Parameters

- **columns** (*union type, optional, not for optimizer, default []*) – Aggregations for producing output columns.
  - dict
    Dictionary of output column names and aggregation expressions.
  - *or* array *of* items : expression
    List of aggregation expressions. The output column name is determined by a heuristic based on the input column name and the transformation function.
- **group_by** (*union type, optional, not for optimizer, default []*) – Group by columns for aggregates.
  - expression
    Expressions for columns name if there is a single column.
  - *or* array *of* items : expression
    List of expressions for columns.
- **exclude_value** (*Any, optional, not for optimizer, default None*) – Exclude this value in computation of aggregates. Useful for missing value imputation.

transform(*X*, *y=None*)

Transform the data.

Parameters

X (array, >=1 items *of* items : array *of* items : Any) – Output of the group by operator - Pandas / Pyspark grouped dataframe

Returns

result – The outer array is over rows.
- items : array *of* items : Any
  The inner array is over columns.

Return type

array

lale.lib.rasl.alias module

class lale.lib.rasl.alias.**Alias**(**args*, *_lale_trained=False*, *_lale_impl=None*, ***kwargs*)

Bases: `TrainedIndividualOp`

Relational algebra alias operator.

This documentation is auto-generated from JSON schemas.

Parameters

name (*string, not for optimizer*) – The table name to be given to the output dataframe.

transform(*X*, *y=None*)

Transform the data.

Parameters

X (array, >=1 items *of* items : array *of* items : Any) – Input table or dataframe

Returns

result – Features; no restrictions on data type.

Return type

Any

lale.lib.rasl.batched_bagging_classifier module

class lale.lib.rasl.batched_bagging_classifier.**BatchedBaggingClassifier**(***, *base_estimator=None*)

Bases: `PlannedIndividualOp`

Implementation of a homomorphic bagging classifier.

This documentation is auto-generated from JSON schemas.

As proposed in https://izbicki.me/public/papers/icml2013-algebraic-classifiers.pdf

Parameters

base_estimator (*union type, not for optimizer, default None*) – Planned Lale individual operator or pipeline.

- operator
- *or* None

  lale.lib.sklearn.LogisticRegression

fit(*X*, *y=None*, ***fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

- **X** (array *of* items : array *of* items : float) – The training input samples. Sparse matrices are accepted only if
- **y** (*union type*) – The target values (class labels).
  - array *of* items : float
  - *or* array *of* items : string
  - *or* array *of* items : boolean
- **sample_weight** (*union type, optional*) – Sample weights. If None, then samples are equally weighted.
  - array *of* items : float
  - *or* None

partial_fit(*X*, *y=None*, ***fit_params*)

Incremental fit to train train the operator on a batch of samples.

Note: The partial_fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

predict(*X*, ***predict_params*)

Make predictions.

Note: The predict method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (array *of* items : array *of* items : float) –

Returns

result –

- array *of* items : string
- *or* array *of* items : float
- *or* array *of* items : boolean

Return type

union type

lale.lib.rasl.batching module

class lale.lib.rasl.batching.**Batching**(***, *operator*, *batch_size=64*, *shuffle=False*, *num_workers=0*, *inmemory=False*, *num_epochs=None*, *max_resident=None*, *scoring=None*, *progress_callback=None*, *partial_transform=False*, *priority='resource_aware'*, *verbose=0*)

Bases: `PlannedIndividualOp`

Batching trains the given pipeline using batches.

This documentation is auto-generated from JSON schemas.

The batch_size is used across all steps of the pipeline, serializing the intermediate outputs if specified.

Parameters

- **operator** (*operator, optional, not for optimizer*) – A lale pipeline object to be used inside of batching
- **batch_size** (*integer, >=1, >=32 for optimizer, <=128 for optimizer, uniform distribution, optional, default 64*) – Batch size used for transform.
- **shuffle** (*boolean, optional, not for optimizer, default False*) – Shuffle dataset before batching or not.
- **num_workers** (*integer, optional, not for optimizer, default 0*) – Number of workers for pytorch dataloader.
- **inmemory** (*boolean, optional, not for optimizer, default False*) –

  Whether all the computations are done in memory

  or intermediate outputs are serialized. Only applies to transform/predict. For fit, use the *max_resident* argument.
- **num_epochs** (*union type, optional, not for optimizer, default None*) – Number of epochs. If the operator has *num_epochs* as a parameter, that takes precedence.
  - integer
  - *or* None

- **max_resident** (*union type, optional, not for optimizer, default None*) – Amount of memory to be used in bytes.
  - integer
  - *or* None
- **scoring** (*union type, optional, not for optimizer, default None*) – Batch-wise scoring metrics from *lale.lib.rasl*.
  - callable
  - *or* None
- **progress_callback** (*union type, optional, not for optimizer, default None*) – Callback function to get performance metrics per batch.
  - callable
  - *or* None
- **partial_transform** (*boolean, optional, not for optimizer, default False*) – Whether to allow partially-trained upstream operators to transform data for training downstream operators even before the upstream operator has been fully trained.
- **priority** ('batch', 'step', *or* 'resource_aware', optional, not for optimizer, default 'resource_aware') – Scheduling priority in task graphs. "batch" will execute tasks from earlier batches first. "step" will execute tasks from earlier steps first, like nested-loop algorithm. And "resource_aware" will execute tasks with less non-resident data first.
- **verbose** (*integer, optional, not for optimizer, default 0*) – Verbosity level, higher values mean more information.

fit(*X*, *y=None*, ***fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

- **X** (*union type*) – Features; the outer array is over samples.
  - array
    - items : union type
      - float
      - *or* string
      - *or* boolean
  - *or* array
    - items : array
      - items : union type
      - float
      - *or* string
      - *or* boolean
  - *or* dict
- **y** (*union type*) –
  - array
    - items : union type
      - integer
      - *or* float
      - *or* string
  - *or* None
- **classes** (*union type, optional*) – The total number of classes in the entire training dataset.
  - array
    - items : union type
      - float
      - *or* string

            · *or* boolean
      – *or* None

predict(*X*, ***predict_params*)

Make predictions.

Note: The predict method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

- **X** (*union type*) – Features; the outer array is over samples.
  - array
    - items : union type
      - float
      - *or* string
      - *or* boolean
  - *or* array
    - items : array
      - items : union type
      - float
      - *or* string
      - *or* boolean
  - *or* any type
- **y** (*array, optional*) –
  - items : union type
    - integer
    - *or* float

Returns

result – Output data schema for transformed data.

Return type

Any

transform(*X*, *y=None*)

Transform the data.

Note: The transform method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

- **X** (*union type*) – Features; the outer array is over samples.
  - array
    - items : union type
      - float
      - *or* string
      - *or* boolean
  - *or* array
    - items : array
      - items : union type
      - float
      - *or* string
      - *or* boolean
  - *or* any type
- **y** (*array, optional*) –
  - items : union type
    - integer
    - *or* float

Returns
result – Output data schema for transformed data.

Return type
Any

lale.lib.rasl.concat_features module

class lale.lib.rasl.concat_features.**ConcatFeatures**(**args*, *_lale_trained=False*, *_lale_impl=None*, ***kwargs*)

Bases: *TrainedIndividualOp*

Horizontal stacking concatenates features (aka columns) of input datasets.

This documentation is auto-generated from JSON schemas.

Examples

```
>>> A = [ [11, 12, 13],
...       [21, 22, 23],
...       [31, 32, 33] ]
>>> B = [ [14, 15],
...       [24, 25],
...       [34, 35] ]
>>> ConcatFeatures.transform([A, B])
NDArrayWithSchema([[11, 12, 13, 14, 15],
                   [21, 22, 23, 24, 25],
                   [31, 32, 33, 34, 35]])
```

transform(*X*, *y=None*)

Transform the data.

Parameters
X (*array*) – Outermost array dimension is over datasets.

- items : array
  Middle array dimension is over samples (aka rows).
  - items : union type
    Innermost array dimension is over features (aka columns).
    - array *of* items : float
    - *or* float

Returns
result – Features; the outer array is over samples.

- items : array
  Outer array dimension is over samples (aka rows).
  - items : Any
    Inner array dimension is over features (aka columns).

Return type
array

lale.lib.rasl.convert module

class lale.lib.rasl.convert.**Convert**(**args*, *_lale_trained=False*, *_lale_impl=None*, ***kwargs*)

Bases: *TrainedIndividualOp*

Convert data to different representation if necessary.

This documentation is auto-generated from JSON schemas.

Parameters

astype ('pandas' *or* 'spark', not for optimizer, default 'pandas') – Type to convert to.

transform(*X*, *y=None*)

Transform the data.

Parameters

X (array *of* items : array *of* items : Any) – Input features as numpy, pandas, or PySpark.

Returns

result

Return type

array *of* items : array *of* items : Any

transform_X_y(*X*, *y*)

Transform the data and target.

Parameters

- **X** (array *of* items : array *of* items : Any) – Input features as numpy, pandas, or PySpark.
- **y** (*union type*) –
  - None
  - *or* array *of* items : Any

    Input labels as numpy, pandas, or PySpark.

Returns

result –

- item 0 : array *of* items : array *of* items : Any

  X
- item 1 : union type
  - None
  - *or* array *of* items : Any

    Input labels as numpy, pandas, or PySpark.

Return type

tuple

lale.lib.rasl.datasets module

lale.lib.rasl.datasets.**arff_data_loader**(*file_name: str*, *label_name: str*, *rows_per_batch: int*) → Iterable[Tuple[DataFrame, Series]]

Incrementally load an ARFF file and yield it one (X, y) batch at a time.

lale.lib.rasl.datasets.**csv_data_loader**(*file_name: str*, *label_name: str*, *rows_per_batch: int*) → Iterable[Tuple[DataFrame, Series]]

Incrementally load an CSV file and yield it one (X, y) batch at a time.

lale.lib.rasl.datasets.**mockup_data_loader**(*X: DataFrame*, *y: Series*, *n_batches: int*, *astype: Literal['pandas']*, *shuffle: bool = False*) → Iterable[Tuple[DataFrame, Series]] lale.lib.rasl.datasets.**mockup_data_loader**(*X: DataFrame*, *y: Series*, *n_batches: int*, *astype: Literal['pandas', 'spark']*, *shuffle: bool = False*) → Iterable[Tuple[DataFrame, Series]]

Split (X, y) into batches to emulate loading them incrementally.

Only intended for testing purposes, because if X and y are already materialized in-memory, there is little reason to batch them.

lale.lib.rasl.datasets.**openml_data_loader**(*dataset_name: str*, *batch_size: int*) → Iterable[Tuple[DataFrame, Series]]

Download the OpenML dataset, incrementally load it, and yield it one (X,y) batch at a time.

lale.lib.rasl.filter module

class lale.lib.rasl.filter.**Filter**(**args*, *_lale_trained=False*, *_lale_impl=None*, ***kwargs*)

Bases: *TrainedIndividualOp*

Relational algebra filter operator.

This documentation is auto-generated from JSON schemas.

Parameters

pred (*Any, not for optimizer*) – Filter predicate. Given as Python AST expression.

transform(*X*, *y=None*)

Transform the data.

Parameters

X (array, >=1 items *of* items : array *of* items : Any) – Input table or dataframe

Returns

result – Features; no restrictions on data type.

Return type

Any

lale.lib.rasl.functions module

class lale.lib.rasl.functions.**ColumnMonoidFactory**(*col_maker: Callable[[Union[str, int]], MonoidFactory[Any, bool, _D]]*)

Bases: *ColumnSelector*[*DictMonoid*[_D]]

Given a MonoidFactory for deciding if a given column is valid, This returns the list of valid columns

from_monoid(*monoid: DictMonoid[_D]*) → List[Union[str, int]]

Given the monoid instance, return the appropriate type of output. This method may also modify self based on the monoid instance.

to_monoid(*batch*)

Create a monoid instance representing the input data

class lale.lib.rasl.functions.**ColumnSelector**(**args*, ***kwargs*)

Bases: *MonoidFactory*[Any, List[Union[str, int]], _D], Protocol

class lale.lib.rasl.functions.**DictMonoid**(*m: Dict[Any, _D]*)

Bases: Generic[_D], *Monoid*

Given a monoid, this class lifts it to a dictionary pointwise

combine(*other: DictMonoid[_D]*)

Combines this monoid instance with another, producing a result. This operation must be observationally associative, satisfying `x.from_monoid(a.combine(b.combine(c))) == x.from_monoid(a.combine(b).combine(c)))` where *x* is the instance of :class:MonoidFactory that created these instances.

property is_absorbing

A monoid value *x* is absorbing if for all *y*, *x.combine(y)* == *x*. This can help stop training early for monoids with learned coefficients.

class lale.lib.rasl.functions.**categorical**(*max_values: int = 5*)

Bases: `ColumnMonoidFactory`

Creates a MonoidFactory (and callable) for projecting categorical columns with sklearn's ColumnTransformer or Lale's Project operator.

Parameters

max_values (`int`) – Maximum number of unique values in a column for it to be considered categorical.

Returns

Function that, given a dataset X, returns a list of columns, containing either string column names or integer column indices.

Return type

callable

class lale.lib.rasl.functions.**categorical_column**(*col: Union[str, int], threshold: int = 5*)

Bases: `MonoidFactory[Any, bool, _column_distinct_count_data]`

Determines if a column should be considered categorical, by seeing if there are more than threshold distinct values in it

from_monoid(*monoid: _column_distinct_count_data*) → bool

Given the monoid instance, return the appropriate type of output. This method may also modify self based on the monoid instance.

to_monoid(*batch*) → _column_distinct_count_data

Create a monoid instance representing the input data

class lale.lib.rasl.functions.**count_distinct_column**(*col: Union[str, int], limit: Optional[int] = None*)

Bases: `MonoidFactory[Any, int, _column_distinct_count_data]`

Counts the number of distinct elements in a given column. If a limit is specified, then, once the limit is reached, the count may no longer be accurate (but will always remain over the limit).

from_monoid(*monoid: _column_distinct_count_data*) → int

Given the monoid instance, return the appropriate type of output. This method may also modify self based on the monoid instance.

to_monoid(*batch*) → _column_distinct_count_data

Create a monoid instance representing the input data

class lale.lib.rasl.functions.**date_time**(*fmt*)

Bases: `object`

Creates a callable for projecting date/time columns with sklearn's ColumnTransformer or Lale's Project operator.

Parameters

fmt (`str`) – Format string for *strptime()*, see https://docs.python.org/3/library/datetime.html#strftime-strptime-behavior

Returns
Function that, given a dataset X, returns a list of columns, containing either string column names or integer column indices.

Return type
callable

`lale.lib.rasl.functions.`**`filter_isnan`**`(`*df: Any*, *column_name: str*`)`

`lale.lib.rasl.functions.`**`filter_isnotnan`**`(`*df: Any*, *column_name: str*`)`

`lale.lib.rasl.functions.`**`filter_isnotnull`**`(`*df: Any*, *column_name: str*`)`

`lale.lib.rasl.functions.`**`filter_isnull`**`(`*df: Any*, *column_name: str*`)`

`class lale.lib.rasl.functions.`**`make_categorical_column`**`(`*threshold=5*`)`

Bases: `object`

lale.lib.rasl.group_by module

`class lale.lib.rasl.group_by.`**`GroupBy`**`(`**args*, *_lale_trained=False*, *_lale_impl=None*, ***kwargs*`)`

Bases: *`TrainedIndividualOp`*

Relational algebra group_by operator.

This documentation is auto-generated from JSON schemas.

Parameters
by (array, not for optimizer *of* items : expression) – GroupBy key(s).

`transform``(`*X*, *y=None*`)`

Transform the data.

Parameters
X (array, >=1 items *of* items : array *of* items : Any) – List of tables.

Returns
result – Features; no restrictions on data type.

Return type
Any

lale.lib.rasl.hashing_encoder module

`class lale.lib.rasl.hashing_encoder.`**`HashingEncoder`**`(`***, *n_components=8*, *cols=None*, *hash_method='md5'*`)`

Bases: *`PlannedIndividualOp`*

Relational algebra reimplementation of scikit-learn contrib's HashingEncoder transformer.

This documentation is auto-generated from JSON schemas.

Works on both pandas and Spark dataframes by using Map for *transform*, which in turn use the appropriate backend.

Parameters
- **n_components** (*integer, not for optimizer, default 8*) – how many bits to use to represent the feature.
- **cols** (*union type, not for optimizer, default None*) – a list of columns to encode, if None, all string columns will be encoded.
  - None
  - *or* array *of* items : string

- **hash_method** ('blake2s', 'sha3_224', 'shake_256', 'sha384', 'sha224', 'sha1', 'sha256', 'sha3_256', 'md5-sha1', 'shake_128', 'md5', 'sha512', 'sha512_256', 'sha3_384', 'sm3', 'blake2b', 'sha512_224', *or* 'sha3_512', not for optimizer, default 'md5') – which hashing method to use.

fit(*X*, *y=None*, ***fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

- **X** (*array*) – Features; the outer array is over samples.
  - items : array
    - items : union type
      - float
      - *or* string
- **y** (*any type, optional*) – Target class labels; the array is over samples.

partial_fit(*X*, *y=None*, ***fit_params*)

Incremental fit to train train the operator on a batch of samples.

Note: The partial_fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

transform(*X*, *y=None*)

Transform the data.

Note: The transform method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (*array*) – Features; the outer array is over samples.

- items : array
  - items : union type
    - float
    - *or* string

Returns

result – Hash codes.

Return type

array *of* items : array *of* items : float

lale.lib.rasl.join module

class lale.lib.rasl.join.**Join**(**args*, *_lale_trained=False*, *_lale_impl=None*, ***kwargs*)

Bases: *TrainedIndividualOp*

Relational algebra join operator.

This documentation is auto-generated from JSON schemas.

Parameters

- **pred** (*Any, not for optimizer*) – Join predicate. Given as Python AST expression.
- **join_limit** (*union type, not for optimizer, default None*) – Not yet implemented! For join paths that are one-to-many, join_limit is use to sample the joined results. When the right hand side of the join has a timestamp column, the join_limit is applied to select the most recent rows. When the right hand side does not have a

timestamp, it randomly samples join_limit number of rows. Sampling is applied after each pair of tables are joined.
  - float
  - *or* None
- **sliding_window_length** (*union type, not for optimizer, default None*) – Not yet implemented! sliding_window_length is also used for sampling the joined results, only rows in a recent window of length sliding_window_length seconds is used in addition to join_limit.
  - float
  - *or* None
- **join_type** (‘inner’, ‘left’, *or* ‘right’, not for optimizer, default ‘inner’) – There are various types of SQL joins available and join_type gives the user the option to choose which type of join the user wants to implement.
- **name** (*union type, not for optimizer, default None*) – The table name to be given to the output dataframe.
  - string
    String (cannot be all spaces).
  - *or* None
    No table name.

transform(*X*, *y=None*)

Transform the data.

Parameters
X (array, >=1 items *of* items : array *of* items : Any) – List of tables.

Returns
result – Features; no restrictions on data type.

Return type
Any

lale.lib.rasl.map module

class lale.lib.rasl.map.**Map**(*, *columns=[]*, *remainder='drop'*)

Bases: *PlannedIndividualOp*

Relational algebra map operator.

This documentation is auto-generated from JSON schemas.

Parameters
- **columns** (*union type, optional, not for optimizer, default []*) – Mappings for producing output columns.
  - dict
    Dictionary of output column names and mapping expressions.
  - *or* array *of* items : expression
    List of mapping expressions. The output column name is determined by a heuristic based on the input column name and the transformation function.
  - *or* callable, not for optimizer
    A callable which, when given the input data, returns either a list or dictionary of mapping expressions, as above.
- **remainder** (*union type, optional, not for optimizer, default 'drop'*) – Transformation for the remaining columns.
  - ’passthrough’ *or* ‘drop’
  - *or* operator
    Mapping expression.

fit(*X*, *y=None*, ***fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

X (*union type*) – The outer array is over rows.

- Any
- *or* array
  - items : array *of* items : Any

    The inner array is over columns.

transform(*X*, *y=None*)

Transform the data.

Note: The transform method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (*union type*) – The outer array is over rows.

- Any
- *or* array
  - items : array *of* items : Any

    The inner array is over columns.

Returns

result – The outer array is over rows.

- array
  - items : array *of* items : Any

    The inner array is over columns.
- *or* Any

Return type

union type

lale.lib.rasl.metrics module

class lale.lib.rasl.metrics.**MetricMonoidFactory**(**args*, ***kwargs*)

Bases: *MonoidFactory*[Tuple[Union[Series, ndarray], Union[Series, ndarray], DataFrame], float, _M], Protocol

Abstract base class for factories that create metrics with an associative monoid interface.

abstract score_data(*y_true: Series*, *y_pred: Series*, *X: Optional[DataFrame] = None*) → float

score_data_batched(*batches: Iterable[Tuple[Union[Series, ndarray], Union[Series, ndarray], DataFrame]]*) → float

abstract score_estimator(*estimator:* TrainedOperator, *X: DataFrame*, *y: Series*) → float

score_estimator_batched(*estimator:* TrainedOperator, *batches: Iterable[Tuple[DataFrame, Series]]*) → float

abstract to_monoid(*batch: Tuple[Union[Series, ndarray], Union[Series, ndarray], DataFrame]*) → _M

Create a monoid instance representing the input data lale.lib.rasl.metrics.**accuracy_score**(*y_true: Series*, *y_pred: Series*) → float

Replacement for sklearn's accuracy_score function.

lale.lib.rasl.metrics.**balanced_accuracy_score**(*y_true: Series*, *y_pred: Series*) → float

Replacement for sklearn's balanced_accuracy_score function.

lale.lib.rasl.metrics.**f1_score**(*y_true: Series*, *y_pred: Series*, *pos_label: Union[int, float, str] = 1*) → float

Replacement for sklearn's f1_score function.

lale.lib.rasl.metrics.**get_scorer**(*scoring: str*, ***kwargs*) → *MetricMonoidFactory*

Replacement for sklearn's get_scorer function.

lale.lib.rasl.metrics.**r2_score**(*y_true: Series*, *y_pred: Series*) → float

Replacement for sklearn's r2_score function.

lale.lib.rasl.min_max_scaler module

class lale.lib.rasl.min_max_scaler.**MinMaxScaler**(***, *feature_range='(0, 1)'*, *copy=True*, *clip=False*)

Bases: `PlannedIndividualOp`

Relational algebra implementation of MinMaxScaler.

This documentation is auto-generated from JSON schemas.

Parameters

- **feature_range** (*tuple, >=2 items, <=2 items, not for optimizer, default (0, 1)*) – Desired range of transformed data.
  - item 0 : float, >=-1 for optimizer, <=0 for optimizer
  - item 1 : float, >=0.001 for optimizer, <=1 for optimizer
- **copy** (*True, not for optimizer, default True*) – *copy=True* is the only value currently supported by this implementation
- **clip** (*boolean, optional, not for optimizer, default False*) – Set to True to clip transformed values of held-out data to provided feature range.

Notes

constraint-1 : negated type *of* 'X/isSparse'

MinMaxScaler does not support sparse input. Consider using MaxAbsScaler instead.

fit(*X*, *y=None*, ***fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

- **X** (array *of* items : array *of* items : float) – Features; the outer array is over samples.
- **y** (*any type, optional*) –

partial_fit(*X*, *y=None*, ***fit_params*)

Incremental fit to train train the operator on a batch of samples.

Note: The partial_fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

transform(*X*, *y=None*)

Transform the data.

Note: The transform method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (array *of* items : array *of* items : float) – Features; the outer array is over samples.

Returns

result – Output data schema for transformed data.

Return type

array *of* items : array *of* items : float

lale.lib.rasl.monoid module

class lale.lib.rasl.monoid.**Monoid**

Bases: `ABC`

Data that can be combined in an associative way. See :class:MonoidFactory for ways to create/unpack a given monoid.

abstract combine(*other: _SelfType*) → _SelfType

Combines this monoid instance with another, producing a result. This operation must be observationally associative, satisfying `x.from_monoid(a.combine(b.combine(c))) == x.from_monoid(a.combine(b).combine(c)))` where *x* is the instance of :class:MonoidFactory that created these instances.

property is_absorbing

A monoid value *x* is absorbing if for all *y*, *x.combine(y)* == *x*. This can help stop training early for monoids with learned coefficients.

class lale.lib.rasl.monoid.**MonoidFactory**(**args*, ***kwargs*)

Bases: `Generic`[_InputType_contra, _OutputType_co, _M], `Protocol`

This protocol determines if a class supports creating a monoid and using it to support associative computation. Due to the `runtime_checkable` decorator, `isinstance(obj, MonoidFactory)` will succeed if the object has the requisite methods, even if it does not have this protocol as a base class.

abstract from_monoid(*monoid: _M*) → _OutputType_co

Given the monoid instance, return the appropriate type of output. This method may also modify self based on the monoid instance.

abstract to_monoid(*batch: _InputType_contra*) → _M

Create a monoid instance representing the input data

class lale.lib.rasl.monoid.**MonoidableOperator**(**args*, ***kwargs*)

Bases: `MonoidFactory`[`Any`, `None`, _M], `Protocol`

This is a useful base class for operator implementations that support associative (monoid-based) fit. Given the implementation supplied :class:MonoidFactory methods, this class provides default :method:partial_fit and :method:fit implementations.

fit(*X*, *y=None*)

partial_fit(*X*, *y=None*)

lale.lib.rasl.one_hot_encoder module

class lale.lib.rasl.one_hot_encoder.**OneHotEncoder**(**, categories='auto', sparse=False, dtype='float64', handle_unknown='ignore', drop=None*)

Bases: *PlannedIndividualOp*

Relational algebra reimplementation of scikit-learn's OneHotEncoder transformer that encodes categorical features as numbers.

This documentation is auto-generated from JSON schemas.

Works on both pandas and Spark dataframes by using Aggregate for *fit* and Map for *transform*, which in turn use the appropriate backend.

Parameters

- **categories** (*union type, not for optimizer, default 'auto'*) –
  - 'auto' *or* None
    Determine categories automatically from training data.
  - *or* array
    The ith list element holds the categories expected in the ith column.
    - items : union type
      - array *of* items : string
      - *or* array *of* items : float
        Should be sorted.
- **sparse** (*False, optional, not for optimizer, default False*) – This implementation only supports *sparse=False*.
- **dtype** (*'float64', not for optimizer, default 'float64'*) – This implementation only supports *dtype='float64'*.
- **handle_unknown** (*'ignore', not for optimizer, default 'ignore'*) – This implementation only supports *handle_unknown='ignore'*.
- **drop** (*None, optional, not for optimizer, default None*) – This implementation only supports *drop=None*.

fit(*X, y=None, **fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

- **X** (*array*) – Features; the outer array is over samples.
  - items : array
    - items : union type
      - float
      - *or* string
- **y** (*any type, optional*) – Target class labels; the array is over samples.

partial_fit(*X, y=None, **fit_params*)

Incremental fit to train train the operator on a batch of samples.

Note: The partial_fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

transform(*X, y=None*)

Transform the data.

Note: The transform method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (*array*) – Features; the outer array is over samples.

- items : array
  - items : union type
    - float
    - *or* string

Returns

result – One-hot codes.

Return type

array *of* items : array *of* items : float

lale.lib.rasl.orderby module

class lale.lib.rasl.orderby.**OrderBy**(**args*, *_lale_trained=False*, *_lale_impl=None*, ***kwargs*)

Bases: *TrainedIndividualOp*

Relational algebra OrderBy (sort) operator.

This documentation is auto-generated from JSON schemas.

Parameters

by (*Any, not for optimizer*) – OrderBy key(s).

transform(*X*, *y=None*)

Transform the data.

Parameters

X (*union type*) – The outer array is over rows.

- Any
- *or* array
  - items : array *of* items : Any

    The inner array is over columns.

Returns

result – The outer array is over rows.

- array
  - items : array *of* items : Any

    The inner array is over columns.
- *or* Any

Return type

union type

lale.lib.rasl.ordinal_encoder module

class lale.lib.rasl.ordinal_encoder.**OrdinalEncoder**(***, *categories='auto'*, *dtype='float64'*, *handle_unknown='use_encoded_value'*, *unknown_value*)

Bases: *PlannedIndividualOp*

Relational algebra reimplementation of scikit-learn's OrdinalEncoder transformer that encodes categorical features as numbers.

This documentation is auto-generated from JSON schemas.

Works on both pandas and Spark dataframes by using Aggregate for *fit* and Map for *transform*, which in turn use the appropriate backend.

Parameters

- **categories** (*union type, not for optimizer, default 'auto'*) –
  - 'auto' *or* None
    Determine categories automatically from training data.
  - *or* array
    The ith list element holds the categories expected in the ith column.
    * items : union type
      · array *of* items : string
      · *or* array *of* items : float
        Should be sorted.
- **dtype** (*'float64', not for optimizer, default 'float64'*) – This implementation only supports *dtype='float64'*.
- **handle_unknown** (*'use_encoded_value', optional, not for optimizer, default 'use_encoded_value'*) – This implementation only supports *handle_unknown='use_encoded_value'*.
- **unknown_value** (*union type, optional, not for optimizer*) – The encoded value of unknown categories to use when *handle_unknown='use_encoded_value'*. It has to be distinct from the values used to encode any of the categories in fit. If set to np.nan, the dtype hyperparameter must be a float dtype.
  - integer
  - *or* nan *or* None

fit(*X*, *y=None*, ***fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

- **X** (*array*) – Features; the outer array is over samples.
  - items : union type
    * array *of* items : float
    * *or* array *of* items : string
- **y** (*any type, optional*) – Target class labels; the array is over samples.

partial_fit(*X*, *y=None*, ***fit_params*)

Incremental fit to train train the operator on a batch of samples.

Note: The partial_fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

transform(*X*, *y=None*)

Transform the data.

Note: The transform method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (*array*) – Features; the outer array is over samples.

- items : union type
  - array *of* items : float
  - *or* array *of* items : string

Returns

result – Ordinal codes.

Return type

array *of* items : array *of* items : float

lale.lib.rasl.project module

class lale.lib.rasl.project.**Project**(*, *columns=None*, *drop_columns=None*)

Bases: *PlannedIndividualOp*

Projection keeps a subset of the columns, like in relational algebra.

This documentation is auto-generated from JSON schemas.

Examples

```
>>> df = pd.DataFrame(data={'A': [1,2], 'B': ['x','y'], 'C': [3,4]})
>>> keep_numbers = Project(columns={'type': 'number'})
>>> keep_numbers.fit(df).transform(df)
NDArrayWithSchema([[1, 3],
                   [2, 4]])
```

Parameters

- **columns** (*union type, not for optimizer, default None*) – The subset of columns to retain.

  The supported column specification formats include some of the ones from scikit-learn's ColumnTransformer, and in addition, filtering by using a JSON subschema check.

  – None

    If not specified, keep all columns.

  – *or* array *of* items : integer

    Multiple columns by index.

  – *or* array *of* items : string

    Multiple Dataframe columns by names.

  – *or* callable

    Callable that is passed the input data X and can return a list of column names or indices.

  – *or* dict

    Keep columns whose schema is a subschema of this JSON schema.

- **drop_columns** (*union type, not for optimizer, default None*) – The subset of columns to remove.

  The *drop_columns* argument supports the same formats as *columns*. If both are specified, keep everything from *columns* that is not also in *drop_columns*.

  – None

    If not specified, drop no further columns.

  – *or* array *of* items : integer

    Multiple columns by index.

  – *or* array *of* items : string

    Multiple Dataframe columns by names.

  – *or* callable

    Callable that is passed the input data X and can return a list of column names or indices.

  – *or* dict

    Remove columns whose schema is a subschema of this JSON schema.

fit(*X*, *y=None*, ***fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

- **X** (*array*) – Features; the outer array is over samples.
  - items : array
    - items : union type
      - float
      - *or* string
- **y** (*any type, optional*) – Target for supervised learning (ignored).

partial_fit(*X*, *y=None*, ***fit_params*)

Incremental fit to train train the operator on a batch of samples.

Note: The partial_fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

transform(*X*, *y=None*)

Transform the data.

Note: The transform method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (*array*) – Features; the outer array is over samples.

- items : array
  - items : union type
    - float
    - *or* string

Returns

result – Features; the outer array is over samples.

- items : array
  - items : union type
    - float
    - *or* string

Return type

array

lale.lib.rasl.project.**get_column_factory**(*columns*, *kind*) → *MonoidFactory*

lale.lib.rasl.relational module

class lale.lib.rasl.relational.**Relational**(**, *operator*)

Bases: *PlannedIndividualOp*

Higher order operator that contains a nested data join pipeline that has

This documentation is auto-generated from JSON schemas.

multiple table joins and aggregates on those joins.

Parameters

operator (*operator, optional, not for optimizer*) – A lale pipeline object to be used inside of relational that captures the data join and aggregate operations.

fit(*X*, *y=None*, ***fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

- **X** (*any*) – Features; the outer array is over samples.
- **y** (*union type*) –
  - array *of* items : float
  - *or* array *of* items : string
  - *or* array *of* items : boolean

transform(*X*, *y=None*)

Transform the data.

Note: The transform method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters
X (*any*) – The input data for transform.

Returns
result – Output data schema for transform.

Return type
array *of* items : array *of* items : float

lale.lib.rasl.scan module

class lale.lib.rasl.scan.**Scan**(**args*, *_lale_trained=False*, *_lale_impl=None*, ***kwargs*)

Bases: *TrainedIndividualOp*

Scans a database table.

This documentation is auto-generated from JSON schemas.

Parameters
table (*expression, not for optimizer*) – Which table to scan.

transform(*X*, *y=None*)

Transform the data.

Parameters
X (*array, >=1 items*) – Outermost array dimension is over datasets that have table names.

- items : array
  Middle array dimension is over samples (aka rows).
  - items : array *of* items : Any
    Innermost array dimension is over features (aka columns).

Returns
result

Return type
array *of* items : array *of* items : Any

lale.lib.rasl.scores module

class lale.lib.rasl.scores.**FClassif**(**args*, ***kwargs*)

Bases: *ScoreMonoidFactory*[*FOnewayData*]

Compute the ANOVA F-value for the provided sample.

from_monoid(*monoid: FOnewayData*)

Given the monoid instance, return the appropriate type of output. This method may also modify self based on the monoid instance.

to_monoid(*batch: Tuple[Any, Any]*) → *FOnewayData*

Create a monoid instance representing the input data

class lale.lib.rasl.scores.**FOnewayData**(***, *classes*, *n_samples_per_class*, *n_samples*, *ss_alldata*, *sums_samples*, *sums_alldata*)

Bases: *Monoid*

Parameters

- **classes** (*list*) – The list of classes.
- **n_samples_per_class** (*dictionary*) – The number of samples in each class.
- **n_samples** (*number*) – The total number of samples.
- **ss_alldata** (*array*) – The sum of square of each feature.
- **sums_samples** (*dictionary*) – The sum of each feaure per class.
- **sums_alldata** (*array*) – The sum of each feaure.

combine(*other: FOnewayData*)

Combines this monoid instance with another, producing a result. This operation must be observationally associative, satisfying `x.from_monoid(a.combine(b.combine(c))) == x.from_monoid(a.combine(b).combine(c)))` where *x* is the instance of :class:MonoidFactory that created these instances.

class lale.lib.rasl.scores.**ScoreMonoidFactory**(**args*, ***kwargs*)

Bases: *MonoidFactory*[`Tuple`[`Any`, `Any`], `Tuple`[`float`, `float`], `_M`], `Protocol`

score(*X*, *y*) → Tuple[float, float]

lale.lib.rasl.select_k_best module

class lale.lib.rasl.select_k_best.**SelectKBest**(***, *score_func='<function f_classif>'*, *k=10*)

Bases: *PlannedIndividualOp*

Relational algebra implementation of SelectKBest.

This documentation is auto-generated from JSON schemas.

Parameters

- **score_func** (*callable, not for optimizer, default <function f_classif at 0x7f3272b1f160>*) – Function taking two arrays X and y, and returning a pair of arrays (scores, pvalues) or a single array with scores.
- **k** (*union type, default 10*) – Number of top features to select
  - integer, >=1, >=2 for optimizer, <='X/items/maxItems', <=15 for optimizer
  - *or* 'all'

fit(*X*, *y=None*, ***fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

- **X** (array *of* items : array *of* items : float) – Training input samples.
- **y** (*union type*) – Target values (class labels in classification, real numbers in regression).
  - array *of* items : float
  - *or* array *of* items : string
  - *or* array *of* items : boolean

partial_fit(*X*, *y=None*, ***fit_params*)

Incremental fit to train train the operator on a batch of samples.

Note: The partial_fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

transform(*X*, *y=None*)

Transform the data.

Note: The transform method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (array *of* items : array *of* items : float) – The input samples

Returns

result – The input samples with only the selected features.

Return type

array *of* items : array *of* items : float

lale.lib.rasl.simple_imputer module

class lale.lib.rasl.simple_imputer.**SimpleImputer**(***, *missing_values=nan*, *strategy='mean'*, *fill_value=None*, *verbose=0*, *copy=True*, *add_indicator=False*)

Bases: *PlannedIndividualOp*

Relational algebra reimplementation of scikit-learn's SimpleImputer.

This documentation is auto-generated from JSON schemas.

Works on both pandas and Spark dataframes by using Aggregate for *fit* and Map for *transform*, which in turn use the appropriate backend.

Parameters

- **missing_values** (*union type, not for optimizer, default nan*) – The placeholder for the missing values.
  - float
  - *or* string
  - *or* nan
  - *or* None
- **strategy** (*union type, default 'mean'*) – The imputation strategy.
  - 'constant', not for optimizer
  - *or* 'mean', 'median', *or* 'most_frequent'
- **fill_value** (*union type, not for optimizer, default None*) – When strategy == "constant", fill_value is used to replace all occurrences of missing_values
  - float
  - *or* string
  - *or* None

- **verbose** (*integer, not for optimizer, default 0*) – Controls the verbosity of the imputer.
- **copy** (*True, not for optimizer, default True*) – *copy=True* is the only value currently supported by this implementation
- **add_indicator** (*False, not for optimizer, default False*) – *add_indicator=False* is the only value currently supported by this implementation

fit(*X*, *y=None*, ***fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

- **X** (*array*) – Input data, where `n_samples` is the number of samples and `n_features` is the number of features.
  - items : array
    - items : union type
      - float
      - *or* string
- **y** (*any type, optional*) –

partial_fit(*X*, *y=None*, ***fit_params*)

Incremental fit to train train the operator on a batch of samples.

Note: The partial_fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

transform(*X*, *y=None*)

Transform the data.

Note: The transform method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (*array*) – The input data to complete.

- items : array
  - items : union type
    - float
    - *or* string

Returns

result – The input data to complete.

- items : array
  - items : union type
    - float
    - *or* string

Return type

array

lale.lib.rasl.sort_index module

class lale.lib.rasl.sort_index.**SortIndex**(**args*, *_lale_trained=False*, *_lale_impl=None*, ***kwargs*)

Bases: *TrainedIndividualOp*

SortIndex operator.

This documentation is auto-generated from JSON schemas.

Parameters

ascending (*boolean, not for optimizer, default True*) – Sort by index of the dataframe.

transform(*X*, *y=None*)

Transform the data.

Parameters

X (*union type*) – The outer array is over rows.

- Any
- *or* array
  - items : array *of* items : Any
    The inner array is over columns.

Returns

result – The outer array is over rows.

- array
  - items : array *of* items : Any
    The inner array is over columns.
- *or* Any

Return type

union type

transform_X_y(*X*, *y*)

Transform the data and target.

Parameters

- **X** (array *of* items : array *of* items : Any) – Input features as numpy, pandas, or PySpark.
- **y** (*union type*) –
  - None
  - *or* array *of* items : Any
    Input labels as numpy, pandas, or PySpark.

Returns

result –

- item 0 : array *of* items : array *of* items : Any
  X
- item 1 : union type
  - None
  - *or* array *of* items : Any
    Input labels as numpy, pandas, or PySpark.

Return type

tuple

lale.lib.rasl.spark_explainer module

class `lale.lib.rasl.spark_explainer.`**SparkExplainer**(*extended: Union[bool, str] = False*, *mode: Optional[str] = None*)

Bases: `object`

lale.lib.rasl.split_xy module

class `lale.lib.rasl.split_xy.`**SplitXy**(**args*, *_lale_trained=False*, *_lale_impl=None*, ***kwargs*)

Bases: `TrainedIndividualOp`

Relational algebra SplitXy operator.

This documentation is auto-generated from JSON schemas.

Parameters

label_name (*string, optional, not for optimizer, default 'y'*) – The name of the label column in the input dataframe X.

transform(*X*, *y=None*)

Transform the data.

Parameters

X (array *of* items : array *of* items : Any) – Features; the outer array is over samples.

Returns

result – Output data schema for transformed data.

Return type

array *of* items : array *of* items : Any

transform_X_y(*X*, *y*)

Transform the data and target.

Parameters

- **X** (array *of* items : array *of* items : Any) – Input features; the outer array is over samples.
- **y** (*Any*) – Input labels; ignored.

Returns

result –

- item 0 : array *of* items : array *of* items : Any

  X
- item 1 : array *of* items : Any

  y

Return type

tuple

lale.lib.rasl.standard_scaler module

class `lale.lib.rasl.standard_scaler.`**StandardScaler**(***, *copy=True*, *with_mean=True*, *with_std=True*)

Bases: `PlannedIndividualOp`

Relational algebra reimplementation of scikit-learn's StandardScaler transformer that standardizes features by removing the mean and scaling to unit variance.

This documentation is auto-generated from JSON schemas.

Works on both pandas and Spark dataframes by using Aggregate for *fit* and Map for *transform*, which in turn use the appropriate backend.

Parameters

- **copy** (*True, not for optimizer, default True*) – This implementation only supports *copy=True*.
- **with_mean** (*boolean, default True*) – If True, center the data before scaling.

  See also *constraint-1*.
- **with_std** (*boolean, default True*) – If True, scale the data to unit variance (or equivalently, unit standard deviation).

Notes

constraint-1 : union type

Setting *with_mean* to True does not work on sparse matrices, because centering them entails building a dense matrix which in common use cases is likely to be too large to fit in memory.

- with_mean : False
- *or* negated type *of* 'X/isSparse'

fit(*X*, *y=None*, ***fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

- **X** (array *of* items : array *of* items : float) – The data used to compute the mean and standard deviation
- **y** (*any type, optional*) – Ignored

partial_fit(*X*, *y=None*, ***fit_params*)

Incremental fit to train train the operator on a batch of samples.

Note: The partial_fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

transform(*X*, *y=None*)

Transform the data.

Note: The transform method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

- **X** (array *of* items : array *of* items : float) – The data used to scale along the features axis.
- **copy** (*union type, optional, default None*) – Copy the input X or not.
  - boolean
  - *or* None

Returns

result – Perform standardization by centering and scaling

Return type

array *of* items : array *of* items : float

lale.lib.rasl.standard_scaler.**scale**(*X*, ***kwargs*)

lale.lib.rasl.target_encoder module

class lale.lib.rasl.target_encoder.**TargetEncoder**(**, verbose=0, cols=None, drop_invariant=False, return_df=True, handle_missing='value', handle_unknown='value', min_samples_leaf=1, smoothing=1.0, classes=None*)

Bases: `PlannedIndividualOp`

Relational algebra reimplementation of scikit-learn contrib's TargetEncoder transformer.

This documentation is auto-generated from JSON schemas.

Works on both pandas and Spark dataframes by using Aggregate for *fit* and Map for *transform*, which in turn use the appropriate backend.

Parameters

- **verbose** (*integer, not for optimizer, default 0*) – Verbosity of the output, 0 for none.
- **cols** (*union type, not for optimizer, default None*) – Columns to encode.
  - None
    All string columns will be encoded.
  - *or* array *of* items : string
- **drop_invariant** (*False, not for optimizer, default False*) – This implementation only supports *drop_invariant=False*.
- **return_df** (*True, not for optimizer, default True*) – This implementation returns a pandas or spark dataframe if the input is a pandas or spark dataframe, respectively.
- **handle_missing** (*'value', not for optimizer, default 'value'*) – This implementation only supports *handle_missing='value'*.
- **handle_unknown** (*'value', not for optimizer, default 'value'*) – This implementation only supports *handle_unknown='value'*.
- **min_samples_leaf** (*integer, >=1, <=10 for optimizer, not for optimizer, default 1*) – For regularization the weighted average between category mean and global mean is taken. The weight is an S-shaped curve between 0 and 1 with the number of samples for a category on the x-axis. The curve reaches 0.5 at min_samples_leaf. (parameter k in the original paper)
- **smoothing** (*float, >0.0, <=10.0 for optimizer, not for optimizer, default 1.0*) – Smoothing effect to balance categorical average vs prior. Higher value means stronger regularization. The value must be strictly bigger than 0. Higher values mean a flatter S-curve (see min_samples_leaf).
- **classes** (*union type, optional, not for optimizer, default None*) –
  - None
    Regression task.
  - *or* array, >=2 items *of* items : float
    Classification task with numeric labels.
  - *or* array, >=2 items *of* items : string
    Classification task with string labels.
  - *or* array, >=2 items *of* items : boolean
    Classification task with Boolean labels.

fit(*X, y=None, **fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

- **X** (*array*) – Features; the outer array is over samples.

- items : array
  - items : union type
    - float
    - *or* string
- **y** (*union type*) – Target class labels; the array is over samples.
  - array *of* items : float
  - *or* array *of* items : string

partial_fit(*X*, *y=None*, ***fit_params*)

Incremental fit to train train the operator on a batch of samples.

Note: The partial_fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

transform(*X*, *y=None*)

Transform the data.

Note: The transform method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (*array*) – Features; the outer array is over samples.

- items : array
  - items : union type
    - float
    - *or* string

Returns

result

Return type

array *of* items : array *of* items : float

lale.lib.rasl.task_graphs module

class lale.lib.rasl.task_graphs.**Prio**

Bases: ABC

Abstract base class for scheduling priority in task graphs.

arity: int

batch_priority(*batch: _Batch*) → Any

bottom() → Any

abstract task_priority(*task: _Task*) → Any

class lale.lib.rasl.task_graphs.**PrioBatch**

Bases: *Prio*

Execute tasks from earlier batches first.

arity: int = 6

task_priority(*task: _Task*) → Any

class lale.lib.rasl.task_graphs.**PrioResourceAware**

Bases: *Prio*

Execute tasks with less non-resident data first.

arity: int = 5

task_priority(*task: _Task*) → Any

class lale.lib.rasl.task_graphs.**PrioStep**

Bases: *Prio*

Execute tasks from earlier steps first, like nested-loop algorithm.

arity: int = 6

task_priority(*task: _Task*) → Any

lale.lib.rasl.task_graphs.**cross_val_score**(*pipeline: TrainablePipeline[TrainableIndividualOp], batches: Iterable[Tuple[Any, Any]], scoring: MetricMonoidFactory, cv, unique_class_labels: List[Union[str, int, float]], max_resident: Optional[int], prio: Prio, same_fold: bool, verbose: int*) → List[float]

Replacement for sklearn's cross_val_score function (early interface, subject to change).

lale.lib.rasl.task_graphs.**cross_validate**(*pipeline: TrainablePipeline[TrainableIndividualOp], batches: Iterable[Tuple[Any, Any]], scoring: MetricMonoidFactory, cv, unique_class_labels: List[Union[str, int, float]], max_resident: Optional[int], prio: Prio, same_fold: bool, return_estimator: bool, verbose: int*) → Dict[str, Union[List[float], List[*TrainedPipeline*]]]

Replacement for sklearn's cross_validate function (early interface, subject to change).

lale.lib.rasl.task_graphs.**fit_with_batches**(*pipeline: TrainablePipeline[TrainableIndividualOp], batches_train: Iterable[Tuple[Any, Any]], batches_valid: Optional[List[Tuple[Any, Any]]], scoring: Optional[MetricMonoidFactory], unique_class_labels: List[Union[str, int, float]], max_resident: Optional[int], prio: Prio, partial_transform: Union[bool, str], verbose: int, progress_callback: Optional[Callable[[float, float, int, bool], None]]*) → *TrainedPipeline*[*TrainedIndividualOp*]

Replacement for the *fit* method on a pipeline (early interface, subject to change).

lale.lib.rasl.task_graphs.**is_associative**(*op: TrainableIndividualOp*) → bool

Is the operator pre-trained or does it implement MonoidFactory?

lale.lib.rasl.task_graphs.**is_incremental**(*op: TrainableIndividualOp*) → bool

Does the operator have a partial_fit method or is it pre-trained?

lale.lib.rasl.task_graphs.**is_pretrained**(*op: TrainableIndividualOp*) → bool

Is the operator frozen-trained or does it lack a fit method?

Module contents

RASL operators and functions (experimental).

Relational Algebra Operators

- lale.lib.rasl. Aggregate
- lale.lib.rasl. Alias
- lale.lib.rasl. Filter
- lale.lib.rasl. GroupBy
- lale.lib.rasl. Join
- lale.lib.rasl. Map
- lale.lib.rasl. OrderBy
- lale.lib.rasl. Project
- lale.lib.rasl. Relational

Transformers

- lale.lib.rasl. Batching
- lale.lib.rasl. ConcatFeatures
- lale.lib.rasl. Convert
- lale.lib.rasl. Scan
- lale.lib.rasl. SortIndex
- lale.lib.rasl. SplitXy

Scikit-learn Operators

- lale.lib.rasl. MinMaxScaler
- lale.lib.rasl. OneHotEncoder
- lale.lib.rasl. OrdinalEncoder
- lale.lib.rasl. HashingEncoder
- lale.lib.rasl. SelectKBest
- lale.lib.rasl. SimpleImputer
- lale.lib.rasl. StandardScaler
- lale.lib.rasl. TargetEncoder

Estimators

- lale.lib.rasl. BatchedBaggingClassifier

Functions

- lale.lib.rasl. categorical
- lale.lib.rasl. date_time
- lale.lib.rasl. SparkExplainer

Data Loaders

- lale.lib.rasl. csv_data_loader
- lale.lib.rasl. mockup_data_loader
- lale.lib.rasl. openml_data_loader

Metrics

- lale.lib.rasl. accuracy_score
- lale.lib.rasl. balanced_accuracy_score
- lale.lib.rasl. f1_score
- lale.lib.rasl. get_scorer
- lale.lib.rasl. r2_score

Other Facilities

- lale.lib.rasl. Prio
- lale.lib.rasl. PrioBatch
- lale.lib.rasl. PrioResourceAware
- lale.lib.rasl. PrioStep
- lale.lib.rasl. cross_val_score
- lale.lib.rasl. cross_validate
- lale.lib.rasl. fit_with_batches
- lale.lib.rasl. is_associative
- lale.lib.rasl. is_incremental

lale.lib.sklearn package

Submodules

lale.lib.sklearn.ada_boost_classifier module

class lale.lib.sklearn.ada_boost_classifier.**AdaBoostClassifier**(**, n_estimators=50, learning_rate=1.0, algorithm='SAMME.R', random_state=None, estimator=None*)

Bases: `PlannedIndividualOp`

AdaBoost classifier from scikit-learn for boosting ensemble.

This documentation is auto-generated from JSON schemas.

Parameters

- **n_estimators** (*integer, >=50 for optimizer, <=500 for optimizer, uniform distribution, default 50*) – The maximum number of estimators at which boosting is terminated.
- **learning_rate** (*float, >=0.01 for optimizer, <=1.0 for optimizer, loguniform distribution, default 1.0*) – Learning rate shrinks the contribution of each classifier by
- **algorithm** (*union type, default 'SAMME.R'*) – The boosting algorithm to use
  - 'SAMME'
    Use the SAMME discrete boosting algorithm.
  - *or* 'SAMME.R'
    deprecated
- **random_state** (*union type, not for optimizer, default None*) – If int, random_state is the seed used by the random number generator;
  - integer
  - *or* numpy.random.RandomState
  - *or* None
- **estimator** (*union type, optional, not for optimizer, default None*) – The base estimator to fit on random subsets of the dataset.
  - operator
  - *or* None
    DecisionTreeClassifier

  See also *constraint-1*.

Notes

constraint-1 : union type

Only *estimator* or *base_estimator* should be specified. As *base_estimator* is deprecated, use *estimator*.

- base_estimator : False *or* 'deprecated'
- *or* estimator : None

decision_function(*X*)

Confidence scores for all classes.

Note: The decision_function method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters
`X` (array *of* items : array *of* items : float) – Features; the outer array is over samples.

Returns
result – Confidence scores for samples for each class in the model.

- array *of* items : array *of* items : float
  In the multi-way case, score per (sample, class) combination.
- *or* array *of* items : float
  In the binary case, score for *self._classes[1]*.

Return type
union type

`fit`(*X*, *y=None*, ***fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

- `X` (array *of* items : array *of* items : float) – The training input samples. Sparse matrix can be CSC, CSR, COO,
- `y` (*union type*) – The target values (class labels).
  - array *of* items : float
  - *or* array *of* items : string
  - *or* array *of* items : boolean
- `sample_weight` (*union type, optional, default None*) – Sample weights. If None, the sample weights are initialized to
  - array *of* items : float
  - *or* None

`predict`(*X*, ***predict_params*)

Make predictions.

Note: The predict method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters
`X` (array, optional *of* items : array *of* items : float) – The training input samples. Sparse matrix can be CSC, CSR, COO,

Returns
result – The predicted classes.

- array *of* items : float
- *or* array *of* items : string
- *or* array *of* items : boolean

Return type
union type

`predict_proba`(*X*)

Probability estimates for all classes.

Note: The predict_proba method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters
`X` (array, optional *of* items : array *of* items : float) – The training input samples. Sparse matrix can be CSC, CSR, COO,

Returns
result – The class probabilities of the input samples. The order of

Return type
array *of* items : array *of* items : float

lale.lib.sklearn.ada_boost_regressor module

class lale.lib.sklearn.ada_boost_regressor.**AdaBoostRegressor**(**, n_estimators=50, learning_rate=1.0, loss='linear', random_state=None, estimator=None*)

Bases: *PlannedIndividualOp*

AdaBoost regressor from scikit-learn for boosting ensemble.

This documentation is auto-generated from JSON schemas.

Parameters

- **n_estimators** (*integer, >=50 for optimizer, <=500 for optimizer, uniform distribution, default 50*) – The maximum number of estimators at which boosting is terminated.
- **learning_rate** (*float, >=0.01 for optimizer, <=1.0 for optimizer, loguniform distribution, default 1.0*) – Learning rate shrinks the contribution of each regressor by
- **loss** ('linear', 'square', *or* 'exponential', default 'linear') – The loss function to use when updating the weights after each
- **random_state** (*union type, not for optimizer, default None*) – If int, random_state is the seed used by the random number generator;
  - integer
  - *or* numpy.random.RandomState
  - *or* None
- **estimator** (*union type, optional, not for optimizer, default None*) – The base estimator to fit on random subsets of the dataset.
  - operator
  - *or* None

  See also *constraint-1*.

Notes

constraint-1 : union type
Only *estimator* or *base_estimator* should be specified. As *base_estimator* is deprecated, use *estimator*.

- base_estimator : False *or* 'deprecated'
- *or* estimator : None

fit(*X, y=None, **fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

- **X** (array *of* items : array *of* items : float) – The training input samples. Sparse matrix can be CSC, CSR, COO,
- **y** (array *of* items : float) – The target values (real numbers).
- **sample_weight** (*union type, optional, default None*) – Sample weights. If None, the sample weights are initialized to
  - array *of* items : float

– *or* None

predict(*X*, ***predict_params*)

Make predictions.

Note: The predict method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (array, optional *of* items : array *of* items : float) – The training input samples. Sparse matrix can be CSC, CSR, COO,

Returns

result – The predicted regression values.

Return type

array *of* items : float

lale.lib.sklearn.bagging_classifier module

class lale.lib.sklearn.bagging_classifier.**BaggingClassifier**(**, n_estimators=10, max_samples=1.0, max_features=1.0, bootstrap=True, bootstrap_features=False, oob_score=False, warm_start=False, n_jobs=None, random_state=None, verbose=0, estimator=None*)

Bases: `PlannedIndividualOp`

Bagging classifier from scikit-learn for bagging ensemble.

This documentation is auto-generated from JSON schemas.

Parameters

- **n_estimators** (*integer, >=10 for optimizer, <=100 for optimizer, uniform distribution, default 10*) – The number of base estimators in the ensemble.
- **max_samples** (*union type, not for optimizer, default 1.0*) – The number of samples to draw from X to train each base estimator.
  - integer, >=2, <='X/maxItems', not for optimizer
    Draw max_samples samples.
  - *or* float, >0.0, <=1.0
    Draw max_samples * X.shape[0] samples.
- **max_features** (*union type, not for optimizer, default 1.0*) – The number of features to draw from X to train each base estimator.
  - integer, >=2, <='X/items/maxItems', not for optimizer
    Draw max_features features.
  - *or* float, >0.0, <=1.0
    Draw max_samples * X.shape[1] features.
- **bootstrap** (*boolean, default True*) – Whether samples are drawn with (True) or without (False) replacement.

  See also *constraint-1*.
- **bootstrap_features** (*boolean, not for optimizer, default False*) – Whether features are drawn with (True) or wrhout (False) replacement.
- **oob_score** (*boolean, not for optimizer, default False*) – Whether to use out-of-bag samples to estimate the generalization error.

  See also *constraint-1*, *constraint-2*.

- **warm_start** (*boolean, not for optimizer, default False*) – When set to True, reuse the solution of the previous call to fit and add more estimators to the ensemble, otherwise, just fit a whole new ensemble.

  See also *constraint-2*.
- **n_jobs** (*union type, not for optimizer, default None*) – The number of jobs to run in parallel for both *fit* and *predict*.
  - None
    1 unless in joblib.parallel_backend context.
  - *or* -1
    Use all processors.
  - *or* integer, >=1
    Number of CPU cores.
- **random_state** (*union type, not for optimizer, default None*) – If int, random_state is the seed used by the random number generator;
  - integer
  - *or* numpy.random.RandomState
  - *or* None
- **verbose** (*integer, not for optimizer, default 0*) – Controls the verbosity when fitting and predicting.
- **estimator** (*union type, optional, not for optimizer, default None*) – The base estimator to fit on random subsets of the dataset.
  - operator
  - *or* None
    DecisionTreeClassifier

  See also *constraint-3*.

Notes

constraint-1 : union type
  Out of bag estimation only available if bootstrap=True
  - bootstrap : True
  - *or* oob_score : False

constraint-2 : union type
  Out of bag estimate only available if warm_start=False
  - warm_start : False
  - *or* oob_score : False

constraint-3 : union type
  Only *estimator* or *base_estimator* should be specified. As *base_estimator* is deprecated, use *estimator*.
  - base_estimator : False *or* 'deprecated'
  - *or* estimator : None

decision_function(*X*)

Confidence scores for all classes.

Note: The decision_function method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters
  X (array *of* items : array *of* items : float) –

Returns
  result –
  - array *of* items : array *of* items : float
    In the multi-way case, score per (sample, class) combination.

- *or* array *of* items : float
  In the binary case, score for *self._classes[1]*.

Return type
union type

`fit`(*X*, *y=None*, ***fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

- **X** (array *of* items : array *of* items : float) – The training input samples. Sparse matrices are accepted only if
- **y** (*union type*) – The target values (class labels).
  - array *of* items : float
  - *or* array *of* items : string
  - *or* array *of* items : boolean
- **sample_weight** (*union type, optional*) – Sample weights. If None, then samples are equally weighted.
  - array *of* items : float
  - *or* None

`predict`(*X*, ***predict_params*)

Make predictions.

Note: The predict method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters
X (array *of* items : array *of* items : float) –

Returns

result –
- array *of* items : string
- *or* array *of* items : float
- *or* array *of* items : boolean

Return type
union type

`predict_proba`(*X*)

Probability estimates for all classes.

Note: The predict_proba method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters
X (array *of* items : array *of* items : float) –

Returns
result

Return type
array *of* items : array *of* items : float

lale.lib.sklearn.bagging_regressor module

class lale.lib.sklearn.bagging_regressor.**BaggingRegressor**(**, n_estimators=10, max_samples=1.0, max_features=1.0, bootstrap=True, bootstrap_features=False, oob_score=False, warm_start=False, n_jobs=None, random_state=None, verbose=0, estimator=None*)

Bases: `PlannedIndividualOp`

Bagging classifier from scikit-learn for bagging ensemble.

This documentation is auto-generated from JSON schemas.

Parameters

- **n_estimators** (*integer, >=10 for optimizer, <=100 for optimizer, uniform distribution, default 10*) – The number of base estimators in the ensemble.
- **max_samples** (*union type, not for optimizer, default 1.0*) – The number of samples to draw from X to train each base estimator.
  - integer, >=2, <='X/maxItems', not for optimizer
    Draw max_samples samples.
  - *or* float, >0.0, <=1.0
    Draw max_samples * X.shape[0] samples.
- **max_features** (*union type, not for optimizer, default 1.0*) – The number of features to draw from X to train each base estimator.
  - integer, >=2, <='X/items/maxItems', not for optimizer
    Draw max_features features.
  - *or* float, >0.0, <=1.0
    Draw max_samples * X.shape[1] features.
- **bootstrap** (*boolean, default True*) – Whether samples are drawn with (True) or without (False) replacement.

  See also *constraint-1*.
- **bootstrap_features** (*boolean, not for optimizer, default False*) – Whether features are drawn with (True) or wrhout (False) replacement.
- **oob_score** (*boolean, not for optimizer, default False*) – Whether to use out-of-bag samples to estimate the generalization error.

  See also *constraint-1*, *constraint-2*.
- **warm_start** (*boolean, not for optimizer, default False*) – When set to True, reuse the solution of the previous call to fit and add more estimators to the ensemble, otherwise, just fit a whole new ensemble.

  See also *constraint-2*.
- **n_jobs** (*union type, not for optimizer, default None*) – The number of jobs to run in parallel for both *fit* and *predict*.
  - None
    1 unless in joblib.parallel_backend context.
  - *or* -1
    Use all processors.
  - *or* integer, >=1
    Number of CPU cores.
- **random_state** (*union type, not for optimizer, default None*) – If int, random_state is the seed used by the random number generator;
  - integer
  - *or* numpy.random.RandomState

  - *or* None
- **verbose** (*integer, not for optimizer, default 0*) – Controls the verbosity when fitting and predicting.
- **estimator** (*union type, optional, not for optimizer, default None*) – The base estimator to fit on random subsets of the dataset.
  - operator
  - *or* None

    DecisionTreeClassifier

See also *constraint-3*.

Notes

constraint-1 : union type

Out of bag estimation only available if bootstrap=True

- bootstrap : True
- *or* oob_score : False

constraint-2 : union type

Out of bag estimate only available if warm_start=False

- warm_start : False
- *or* oob_score : False

constraint-3 : union type

Only *estimator* or *base_estimator* should be specified. As *base_estimator* is deprecated, use *estimator*.

- base_estimator : False *or* 'deprecated'
- *or* estimator : None

fit(*X*, *y=None*, ***fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

- **X** (array *of* items : array *of* items : float) – The training input samples. Sparse matrices are accepted only if they are supported by the base estimator.
- **y** (array *of* items : float) – The target values (class labels in classification, real numbers in regression)
- **sample_weight** (*union type, optional*) – Sample weights. Supported only if the base estimator supports sample weighting.
  - array *of* items : float
  - *or* None

    Samples are equally weighted.

predict(*X*, ***predict_params*)

Make predictions.

Note: The predict method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (array *of* items : array *of* items : float) –

Returns

result

Return type

array *of* items : float

lale.lib.sklearn.column_transformer module

class lale.lib.sklearn.column_transformer.**ColumnTransformer**(**, transformers, remainder='drop', sparse_threshold=0.3, n_jobs=None, transformer_weights=None, verbose=False, verbose_feature_names_out=True*)

Bases: *PlannedIndividualOp*

ColumnTransformer from scikit-learn applies transformers to columns of an array or pandas DataFrame.

This documentation is auto-generated from JSON schemas.

Parameters

- **transformers** (*array, not for optimizer*) – Operators or pipelines to be applied to subsets of the data.
  - items : tuple, >=3 items, <=3 items
    Tuple of (name, transformer, column(s)).
    - item 0 : string
      Name.
    - item 1 : union type
      Transformer.
      - operator
        Transformer supporting fit and transform.
      - *or* 'passthrough' *or* 'drop'
    - item 2 : union type
      Column(s).
      - integer
        One column by index.
      - *or* array *of* items : integer
        Multiple columns by index.
      - *or* string
        One Dataframe column by name.
      - *or* array *of* items : string
        Multiple Dataframe columns by names.
      - *or* array *of* items : boolean
        Boolean mask.
      - *or* callable *of* integer *or* array *or* string
        Callable that is passed the input data X and can return any of the above.
- **remainder** (*union type, optional, not for optimizer, default 'drop'*) – Transformation for columns that were not specified in transformers.
  - operator
    Transformer supporting fit and transform.
  - *or* 'passthrough' *or* 'drop'
- **sparse_threshold** (*float, >=0.0, <=1.0, optional, not for optimizer, default 0.3*) – If the output of the different transfromers contains sparse matrices, these will be stacked as a sparse matrix if the overall density is lower than this value. Use sparse_threshold=0 to always return dense.
- **n_jobs** (*union type, optional, not for optimizer, default None*) – Number of jobs to run in parallel

  - None
    1 unless in joblib.parallel_backend context.
  - *or* -1
    Use all processors.
  - *or* integer, >=1
    Number of CPU cores.
- **transformer_weights** (*union type, optional, not for optimizer, default None*) – Multiplicative weights for features per transformer. The output of the transformer is multiplied by these weights.
  - dict
    Keys are transformer names, values the weights.
  - *or* None
- **verbose** (*boolean, optional, not for optimizer, default False*) – If True, the time elapsed while fitting each transformer will be printed as it is completed.
- **verbose_feature_names_out** (*boolean, optional, not for optimizer, default True*) – If True, get_feature_names_out will prefix all feature names with the name of the transformer that generated that feature. If False, get_feature_names_out will not prefix any feature names and will error if feature names are not unique.

Notes

constraint-1 : negated type *of* 'X/isSparse'
A sparse matrix was passed, but dense data is required. Use X.toarray() to convert to a dense numpy array.

fit(*X*, *y=None*, ***fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

- **X** (*array*) – Features; the outer array is over samples.
  - items : array
    - items : union type
      - float
      - *or* string
- **y** (*any type, optional*) – Target for supervised learning (ignored).

transform(*X*, *y=None*)

Transform the data.

Note: The transform method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (*array*) – Features; the outer array is over samples.
- items : array
  - items : union type
    - float
    - *or* string

Returns

result – Features; the outer array is over samples.

Return type

array *of* items : array *of* items : float

lale.lib.sklearn.decision_tree_classifier module

class lale.lib.sklearn.decision_tree_classifier.**DecisionTreeClassifier**(**, criterion='gini', splitter='best', max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features=None, random_state=None, max_leaf_nodes=None, min_impurity_decrease=0.0, class_weight, ccp_alpha=0.0, monotonic_cst=None*)

Bases: *PlannedIndividualOp*

Decision tree classifier from scikit-learn.

This documentation is auto-generated from JSON schemas.

Parameters

- **criterion** ('gini' *or* 'entropy', optional, default 'gini') – The function to measure the quality of a split.
- **splitter** ('best' *or* 'random', optional, default 'best') – The strategy used to choose the split at each node.
- **max_depth** (*union type, optional, default None*) – The maximum depth of the tree.
  - integer, >=1, >=3 for optimizer, <=5 for optimizer
  - *or* None

    Nodes are expanded until all leaves are pure or until all leaves contain less than min_samples_split samples.
- **min_samples_split** (*union type, optional, default 2*) – The minimum number of samples required to split an internal node.
  - integer, >=2, <='X/maxItems', not for optimizer

    Consider min_samples_split as the minimum number.
  - *or* float, >0.0, >=0.01 for optimizer, <=1.0, <=0.5 for optimizer

    min_samples_split is a fraction and ceil(min_samples_split * n_samples) are the minimum number of samples for each split.
- **min_samples_leaf** (*union type, optional, default 1*) – The minimum number of samples required to be at a leaf node.
  - integer, >=1, <='X/maxItems', not for optimizer

    Consider min_samples_leaf as the minimum number.
  - *or* float, >0.0, <=0.5, default 0.05

    min_samples_leaf is a fraction and ceil(min_samples_leaf * n_samples) are the minimum number of samples for each node.
- **min_weight_fraction_leaf** (*float, >=0.0, <=0.5, optional, not for optimizer, default 0.0*) – The minimum weighted fraction of the sum total of weights (of all the input samples) required to be at a leaf node. Samples have equal weight when sample_weight is not provided.
- **max_features** (*union type, optional, default None*) – The number of features to consider when looking for the best split.
  - integer, >=2, <='X/items/maxItems', not for optimizer

    Consider max_features features at each split.
  - *or* float, >0.0, >=0.01 for optimizer, <=1.0, uniform distribution, default 0.5

max_features is a fraction and int(max_features * n_features) features are considered at each split.

- *or* 'sqrt', 'log2', *or* None

- **random_state** (*union type, optional, not for optimizer, default None*) – Seed of pseudo-random number generator.
  - numpy.random.RandomState
  - *or* None

    RandomState used by np.random
  - *or* integer

    Explicit seed.
- **max_leaf_nodes** (*union type, optional, not for optimizer, default None*) – Grow a tree with `max_leaf_nodes` in best-first fashion.
  - integer, >=1, >=3 for optimizer, <=1000 for optimizer
  - *or* None

    Unlimited number of leaf nodes.
- **min_impurity_decrease** (*float, >=0.0, <=10.0 for optimizer, optional, not for optimizer, default 0.0*) – A node will be split if this split induces a decrease of the impurity greater than or equal to this value.
- **class_weight** (*union type, not for optimizer*) – Weights associated with classes in the form `{class_label: weight}`.
  - dict
  - *or* array *of* items : dict
  - *or* 'balanced' *or* None
- **ccp_alpha** (*float, >=0.0, <=0.1 for optimizer, optional, not for optimizer, default 0.0*) – Complexity parameter used for Minimal Cost-Complexity Pruning. The subtree with the largest cost complexity that is smaller than ccp_alpha will be chosen. By default, no pruning is performed.
- **monotonic_cst** (*union type, optional, not for optimizer, default None*) – Indicates the monotonicity constraint to enforce on each feature. Monotonicity constraints are not supported for: multioutput regressions (i.e. when n_outputs > 1),

  regressions trained on data with missing values.
  - array *of* items : -1, 0, *or* 1

    array-like of int of shape (n_features)
  - *or* None

    No constraints are applied.

fit(*X*, *y=None*, ***fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

- **X** (*array*) – The outer array is over samples aka rows.
  - items : array *of* items : float

    The inner array is over features aka columns.
- **y** (*union type*) – The predicted classes.
  - array *of* items : float
  - *or* array *of* items : string
  - *or* array *of* items : boolean
- **sample_weight** (*union type, optional*) – Sample weights.
  - array *of* items : float
  - *or* None

    Samples are equally weighted.
- **check_input** (*boolean, optional, default True*) – Allow to bypass sev-

eral input checking.

- **X_idx_sorted** (*union type, optional, default None*) – The indexes of the sorted training input samples. If many tree
  - array *of* items : array *of* items : float
  - *or* None

predict(*X*, ***predict_params*)

Make predictions.

Note: The predict method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

- **X** (*array, optional*) – The outer array is over samples aka rows.
  - items : array *of* items : float

    The inner array is over features aka columns.
- **check_input** (*boolean, optional, default True*) – Allow to bypass several input checking.

Returns

result – The predicted classes.

- array *of* items : float
- *or* array *of* items : string
- *or* array *of* items : boolean

Return type

union type

predict_proba(*X*)

Probability estimates for all classes.

Note: The predict_proba method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

- **X** (*array, optional*) – The outer array is over samples aka rows.
  - items : array *of* items : float

    The inner array is over features aka columns.
- **check_input** (*boolean, optional*) – Run check_array on X.

Returns

result – The outer array is over samples aka rows.

- items : array *of* items : float

  The inner array has items corresponding to each class.

Return type

array

lale.lib.sklearn.decision_tree_regressor module

class lale.lib.sklearn.decision_tree_regressor.**DecisionTreeRegressor**(*, *criterion='squared_error'*, *splitter='best'*, *max_depth=None*, *min_samples_split=2*, *min_samples_leaf=1*, *min_weight_fraction_leaf=0.0*, *max_features=None*, *random_state=None*, *max_leaf_nodes=None*, *min_impurity_decrease=0.0*, *ccp_alpha=0.0*, *monotonic_cst=None*)

Bases: *PlannedIndividualOp*

Decision tree regressor from scikit-learn.

This documentation is auto-generated from JSON schemas.

Parameters

- **criterion** (*union type, default 'squared_error'*) – Function to measure the quality of a split.
  - 'squared_error', 'friedman_mse', 'absolute_error', *or* 'poisson'
  - *or* 'mae' *or* 'mse', not for optimizer
- **splitter** ('best' *or* 'random', default 'best') – Strategy to choose the split at each node.
- **max_depth** (*union type, default None*) – Maximum depth of the tree.
  - integer, >=1, >=3 for optimizer, <=5 for optimizer
  - *or* None

    If None, then nodes are expanded until all leaves are pure, or until all leaves contain less than min_samples_split samples.
- **min_samples_split** (*union type, default 2*) – The minimum number of samples required to split an internal node.
  - integer, >=2, <='X/maxItems', not for optimizer

    Consider min_samples_split as the minimum number.
  - *or* float, >0.0, >=0.01 for optimizer, <=1.0, <=0.5 for optimizer, default 0.05

    min_samples_split is a fraction and ceil(min_samples_split * n_samples) are the minimum number of samples for each split.
- **min_samples_leaf** (*union type, default 1*) – The minimum number of samples required to be at a leaf node.
  - integer, >=1, <='X/maxItems', not for optimizer

    Consider min_samples_leaf as the minimum number.
  - *or* float, >0.0, >=0.01 for optimizer, <=0.5, default 0.05

    min_samples_leaf is a fraction and ceil(min_samples_leaf * n_samples) are the minimum number of samples for each node.
- **min_weight_fraction_leaf** (*float, >=0.0, <=0.5, optional, not for optimizer, default 0.0*) – Minimum weighted fraction of the sum total of weights (of all the input samples) required to be at a leaf node.
- **max_features** (*union type, default None*) – The number of features to consider when looking for the best split.
  - integer, >=2, <='X/items/maxItems', not for optimizer

    Consider max_features features at each split.
  - *or* float, >0.0, >=0.01 for optimizer, <=1.0, uniform distribution, default 0.5

    max_features is a fraction and int(max_features * n_features) features are considered at each split.
  - *or* 'sqrt', 'log2', *or* None
- **random_state** (*union type, optional, not for optimizer, default*

None) – Seed of pseudo-random number generator.
  - numpy.random.RandomState
  - *or* None
    RandomState used by np.random
  - *or* integer
    Explicit seed.
- **max_leaf_nodes** (*union type, optional, not for optimizer, default None*) – Grow a tree with `max_leaf_nodes` in best-first fashion.
  - integer, >=1, >=3 for optimizer, <=1000 for optimizer
  - *or* None
    Unlimited number of leaf nodes.
- **min_impurity_decrease** (*float, >=0.0, <=10.0 for optimizer, optional, not for optimizer, default 0.0*) – A node will be split if this split induces a decrease of the impurity greater than or equal to this value.
- **ccp_alpha** (*float, >=0.0, <=0.1 for optimizer, optional, not for optimizer, default 0.0*) – Complexity parameter used for Minimal Cost-Complexity Pruning. The subtree with the largest cost complexity that is smaller than ccp_alpha will be chosen. By default, no pruning is performed.
- **monotonic_cst** (*union type, optional, not for optimizer, default None*) – Indicates the monotonicity constraint to enforce on each feature. Monotonicity constraints are not supported for: multioutput regressions (i.e. when n_outputs > 1),

  regressions trained on data with missing values.
  - array *of* items : -1, 0, *or* 1
    array-like of int of shape (n_features)
  - *or* None
    No constraints are applied.

fit(*X*, *y=None*, ***fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

- **X** (*array*) – The outer array is over samples aka rows.
  - items : array *of* items : float
    The inner array is over features aka columns.
- **y** (array *of* items : float) – The target values (real numbers).
- **sample_weight** (*union type, optional*) – Sample weights.
  - array *of* items : float
  - *or* None
    Samples are equally weighted.
- **check_input** (*boolean, optional, default True*) – Allow to bypass several input checking.
- **X_idx_sorted** (*union type, optional, default None*) – The indexes of the sorted training input samples. If many tree
  - array *of* items : array *of* items : float
  - *or* None

predict(*X*, ***predict_params*)

Make predictions.

Note: The predict method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

- **X** (*array, optional*) – The outer array is over samples aka rows.
  - items : array *of* items : float

    The inner array is over features aka columns.
- **check_input** (*boolean, optional, default True*) – Allow to bypass several input checking.

Returns

result – The predicted classes, or the predict values.

Return type

array *of* items : float

lale.lib.sklearn.dummy_classifier module

class lale.lib.sklearn.dummy_classifier.**DummyClassifier**(**, strategy='prior', random_state=None, constant=None*)

Bases: *PlannedIndividualOp*

Dummy classifier classifier that makes predictions using simple rules.

This documentation is auto-generated from JSON schemas.

Parameters

- **strategy** (*union type, not for optimizer, default 'prior'*) – Strategy to use to generate predictions.
  - 'stratified'

    Generates predictions by respecting the training set's class distribution.
  - *or* 'most_frequent'

    Always predicts the most frequent label in the training set.
  - *or* 'prior'

    Always predicts the class that maximizes the class prior (like 'most_frequent') and predict_proba returns the class prior.
  - *or* 'uniform'

    Generates predictions uniformly at random.
  - *or* 'constant', not for optimizer

    Always predicts a constant label that is provided by the user. This is useful for metrics that evaluate a non-majority class

  See also *constraint-1*.
- **random_state** (*union type, not for optimizer, default None*) – Seed of pseudo-random number generator for shuffling data when solver == 'sag', 'saga' or 'liblinear'.
  - None

    RandomState used by np.random
  - *or* numpy.random.RandomState

    Use the provided random state, only affecting other users of that same random state instance.
  - *or* integer

    Explicit seed.
- **constant** (*union type, optional, not for optimizer, default None*) – The explicit constant as predicted by the "constant" strategy. This parameter is useful only for the "constant" strategy.
  - string *or* number *or* boolean
  - *or* None

  See also *constraint-1*.

Notes

constraint-1 : union type

The constant strategy requires a non-None value for the constant hyperparameter.

- strategy : negated type *of* 'constant'
- *or* constant : negated type *of* None

fit(*X*, *y=None*, ***fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

- **X** (array *of* items : array *of* items : Any) – Features; the outer array is over samples.
- **y** (*union type*) – Target class labels.
  - array *of* items : string
  - *or* array *of* items : float
  - *or* array *of* items : boolean

predict(*X*, ***predict_params*)

Make predictions.

Note: The predict method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (array, optional *of* items : array *of* items : Any) – Features; the outer array is over samples.

Returns

result –

- array *of* items : string
- *or* array *of* items : float
- *or* array *of* items : boolean

Return type

union type

predict_proba(*X*)

Probability estimates for all classes.

Note: The predict_proba method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (array *of* items : array *of* items : Any) – Features; the outer array is over samples.

Returns

result

Return type

array *of* items : array *of* items : float

lale.lib.sklearn.dummy_regressor module

class lale.lib.sklearn.dummy_regressor.**DummyRegressor**(**, strategy='mean', constant=None, quantile=None*)

Bases: *PlannedIndividualOp*

Dummy regressor regressor that makes predictions using simple rules.

This documentation is auto-generated from JSON schemas.

Parameters

- **strategy** (*union type, not for optimizer, default 'mean'*) – Strategy to use to generate predictions.
  - 'mean'
    Always predicts the mean of the training set.
  - *or* 'median'
    Always predicts the median of the training set.
  - *or* 'quantile', not for optimizer
    Always predicts a specified quantile of the training set, provided with the quantile parameter.
  - *or* 'constant', not for optimizer
    Always predicts a constant label that is provided by the user. This is useful for metrics that evaluate a non-majority class

  See also *constraint-1*, *constraint-2*.
- **constant** (*union type, optional, not for optimizer, default None*) – The explicit constant as predicted by the "constant" strategy. This parameter is useful only for the "constant" strategy.
  - float
  - *or* None

  See also *constraint-1*.
- **quantile** (*union type, not for optimizer, default None*) – The quantile to predict using the "quantile" strategy. A quantile of 0.5 corresponds to the median, while 0.0 to the minimum and 1.0 to the maximum.
  - None
  - *or* float, >=0.0, <=1.0

  See also *constraint-2*.

Notes

constraint-1 : union type
The constant strategy requires a non-None value for the constant hyperparameter.
- strategy : negated type *of* 'constant'
- *or* constant : negated type *of* None

constraint-2 : union type
The quantile strategy requires a non-None value for the quantile hyperparameter.
- strategy : negated type *of* 'quantile'
- *or* quantile : negated type *of* None

fit(*X, y=None, **fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters
- **X** (array *of* items : array) – Features; the outer array is over samples.

- **y** (array *of* items : float) – Target values.

`predict`(*X*, ***predict_params*)

Make predictions.

Note: The predict method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (array, optional *of* items : array *of* items : Any) – Features; the outer array is over samples.

Returns

result – Predicted values per sample.

Return type

array *of* items : float

lale.lib.sklearn.extra_trees_classifier module

class lale.lib.sklearn.extra_trees_classifier.**ExtraTreesClassifier**(**, n_estimators=100, criterion='gini', max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features=None, max_leaf_nodes=None, min_impurity_decrease=0.0, bootstrap=False, oob_score=False, n_jobs=None, random_state=None, verbose=0, warm_start=False, class_weight=None, ccp_alpha=0.0, max_samples=None, monotonic_cst=None*)

Bases: *PlannedIndividualOp*

Extra trees classifier random forest from scikit-learn.

This documentation is auto-generated from JSON schemas.

Parameters

- **n_estimators** (*integer, >=1, >=10 for optimizer, <=100 for optimizer, optional, default 100*) – The number of trees in the forest.
- **criterion** ('gini' *or* 'entropy', optional, default 'gini') – The function to measure the quality of a split.
- **max_depth** (*union type, optional, default None*) – The maximum depth of the tree. If None, then nodes are expanded until
  - integer, >=1, >=3 for optimizer, <=5 for optimizer
  - *or* None
- **min_samples_split** (*union type, optional, default 2*) – The minimum number of samples required to split an internal node.
  - integer, >=2, <='X/maxItems', not for optimizer

    Consider min_samples_split as the minimum number.

  - *or* float, >0.0, >=0.01 for optimizer, <=1.0, <=0.5 for optimizer, default 0.05

    min_samples_split is a fraction and ceil(min_samples_split * n_samples) are the minimum number of samples for each split.

- **min_samples_leaf** (*union type, optional, default 1*) – The minimum number of samples required to be at a leaf node.
  - integer, >=1, <='X/maxItems', not for optimizer

    Consider min_samples_leaf as the minimum number.
  - *or* float, >0.0, <=0.5, default 0.05

    min_samples_leaf is a fraction and ceil(min_samples_leaf * n_samples) are the minimum number of samples for each node.
- **min_weight_fraction_leaf** (*float, >=0.0, <=0.5, optional, not for optimizer, default 0.0*) – The minimum weighted fraction of the sum total of weights (of all the input samples) required to be at a leaf node. Samples have equal weight when sample_weight is not provided.
- **max_features** (*union type, optional, default None*) – The number of features to consider when looking for the best split.
  - integer, >=2, <='X/items/maxItems', not for optimizer

    Consider max_features features at each split.
  - *or* float, >0.0, >=0.01 for optimizer, <=1.0, uniform distribution, default 0.5

    max_features is a fraction and int(max_features * n_features) features are considered at each split.
  - *or* 'sqrt', 'log2', *or* None
- **max_leaf_nodes** (*union type, optional, not for optimizer, default None*) – Grow trees with `max_leaf_nodes` in best-first fashion.
  - integer, >=1, >=3 for optimizer, <=1000 for optimizer
  - *or* None

    Unlimited number of leaf nodes.
- **min_impurity_decrease** (*float, >=0.0, <=10.0 for optimizer, optional, not for optimizer, default 0.0*) – A node will be split if this split induces a decrease of the impurity greater than or equal to this value.
- **bootstrap** (*boolean, optional, default False*) – Whether bootstrap samples are used when building trees. If False, the whole dataset is used to build each tree.

  See also *constraint-2*.
- **oob_score** (*union type, optional, not for optimizer, default False*) – Whether to use out-of-bag samples to estimate the generalization accuracy.
  - callable, not for optimizer

    A callable with signature metric(y_true, y_pred).
  - *or* boolean

  See also *constraint-2*.
- **n_jobs** (*union type, optional, not for optimizer, default None*) – The number of jobs to run in parallel for both *fit* and *predict*.
  - integer
  - *or* None
- **random_state** (*union type, optional, not for optimizer, default None*) – If int, random_state is the seed used by the random number generator;
  - integer
  - *or* numpy.random.RandomState
  - *or* None
- **verbose** (*integer, optional, not for optimizer, default 0*) – Controls the verbosity when fitting and predicting.
- **warm_start** (*boolean, optional, not for optimizer, default False*) – When set to True, reuse the solution of the previous call to fit and add more estimators to the ensemble, otherwise, just erase the previous solution.

- **class_weight** (*union type, not for optimizer, default None*) – Weights associated with classes in the form {class_label: weight}.
  - dict
  - *or* 'balanced', 'balanced_subsample', *or* None
- **ccp_alpha** (*float, >=0.0, <=0.1 for optimizer, optional, not for optimizer, default 0.0*) – Complexity parameter used for Minimal Cost-Complexity Pruning. The subtree with the largest cost complexity that is smaller than ccp_alpha will be chosen. By default, no pruning is performed.
- **max_samples** (*union type, optional, not for optimizer, default None*) – If bootstrap is True, the number of samples to draw from X to train each base estimator.
  - None

    Draw X.shape[0] samples.
  - *or* integer, >=1

    Draw max_samples samples.
  - *or* float, >0.0, <1.0

    Draw max_samples * X.shape[0] samples.
- **monotonic_cst** (*union type, optional, not for optimizer, default None*) – Indicates the monotonicity constraint to enforce on each feature. Monotonicity constraints are not supported for: multioutput regressions (i.e. when n_outputs > 1),

  regressions trained on data with missing values.
  - array *of* items : -1, 0, *or* 1

    array-like of int of shape (n_features)
  - *or* None

    No constraints are applied.

Notes

constraint-1 : negated type *of* 'y/isSparse'

This classifier does not support sparse labels.

constraint-2 : union type

Out of bag estimation only available if bootstrap=True

- bootstrap : True
- *or* oob_score : False

fit(*X*, *y=None*, ***fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

- **X** (union type *of* array *of* items : array *of* items : float) – The training input samples. Internally, its dtype will be converted
- **y** (*union type*) – The target values (class labels in classification, real numbers in
  - array *of* items : float
  - *or* array *of* items : string
  - *or* array *of* items : boolean
- **sample_weight** (*union type, optional*) – Sample weights. If None, then samples are equally weighted. Splits
  - array *of* items : float
  - *or* None

predict(*X*, ***predict_params*)

Make predictions.

Note: The predict method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (array, optional *of* items : array *of* items : float) – The input samples. Internally, its dtype will be converted to

Returns

result – The predicted classes.

- array *of* items : float
- *or* array *of* items : string
- *or* array *of* items : boolean

Return type

union type

predict_proba(*X*)

Probability estimates for all classes.

Note: The predict_proba method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (array, optional *of* items : array *of* items : float) – The input samples. Internally, its dtype will be converted to

Returns

result – such arrays if n_outputs > 1.

Return type

array *of* items : array *of* items : float

lale.lib.sklearn.extra_trees_regressor module

class lale.lib.sklearn.extra_trees_regressor.**ExtraTreesRegressor**(**, n_estimators=100, criterion='squared_error', max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features=None, max_leaf_nodes=None, min_impurity_decrease=0.0, bootstrap=False, oob_score=False, n_jobs=None, random_state=None, verbose=0, warm_start=False, ccp_alpha=0.0, max_samples=None, monotonic_cst=None*)

Bases: *PlannedIndividualOp*

Extra trees regressor random forest from scikit-learn.

This documentation is auto-generated from JSON schemas.

Parameters

- **n_estimators** (*integer, >=1, >=10 for optimizer, <=100 for optimizer, default 100*) – The number of trees in the forest.
- **criterion** (*union type, default 'squared_error'*) – The function to measure the quality of a split. Supported criteria are "squared_error" for the mean squared error, which is equal to variance reduction as feature selection criterion, and "absolute_error" for the mean absolute error.
  - 'squared_error' *or* 'absolute_error'
  - *or* 'mae' *or* 'mse', not for optimizer
- **max_depth** (*union type, default None*) – The maximum depth of the tree. If None, then nodes are expanded until
  - integer, >=3 for optimizer, <=5 for optimizer
  - *or* None
- **min_samples_split** (*union type, default 2*) – The minimum number of samples required to split an internal node:
  - integer, >=2, <='X/maxItems'
  - *or* float, >0.0, >=0.01 for optimizer, <=1.0, <=0.5 for optimizer, default 0.05
- **min_samples_leaf** (*union type, default 1*) – The minimum number of samples required to be at a leaf node.
  - integer, >=1, <='X/maxItems', not for optimizer
  - *or* float, >0.0, <=0.5, default 0.05
- **min_weight_fraction_leaf** (*float, >=0.0, <=0.5, optional, not for optimizer, default 0.0*) – The minimum weighted fraction of the sum total of weights (of all the input samples) required to be at a leaf node. Samples have equal weight when sample_weight is not provided.
- **max_features** (*union type, default None*) – The number of features to consider when looking for the best split.
  - integer, not for optimizer
  - *or* float, >0.0, >=0.01 for optimizer, <=1.0 for optimizer, uniform distribution, default 0.5
  - *or* 'sqrt', 'log2', *or* None
- **max_leaf_nodes** (*union type, optional, not for optimizer, default None*) – Grow trees with `max_leaf_nodes` in best-first fashion.
  - integer, >=1, >=3 for optimizer, <=1000 for optimizer
  - *or* None

    Unlimited number of leaf nodes.
- **min_impurity_decrease** (*float, >=0.0, <=10.0 for optimizer, optional, not for optimizer, default 0.0*) – A node will be split if this split induces a decrease of the impurity greater than or equal to this value.
- **bootstrap** (*boolean, default False*) – Whether bootstrap samples are used when building trees. If False, the

  See also *constraint-2*.
- **oob_score** (*union type, optional, not for optimizer, default False*) – Whether to use out-of-bag samples to estimate the generalization accuracy.
  - callable, not for optimizer

    A callable with signature metric(y_true, y_pred).
  - *or* boolean

  See also *constraint-2*.
- **n_jobs** (*union type, optional, not for optimizer, default None*) – The number of jobs to run in parallel for both *fit* and *predict*.
  - integer
  - *or* None
- **random_state** (*union type, optional, not for optimizer, default*

None) – If int, random_state is the seed used by the random number generator;
  - integer
  - *or* numpy.random.RandomState
  - *or* None
- **verbose** (*integer, optional, not for optimizer, default 0*) – Controls the verbosity when fitting and predicting.
- **warm_start** (*boolean, optional, not for optimizer, default False*) – When set to `True`, reuse the solution of the previous call to fit
- **ccp_alpha** (*float, >=0.0, <=0.1 for optimizer, optional, not for optimizer, default 0.0*) – Complexity parameter used for Minimal Cost-Complexity Pruning. The subtree with the largest cost complexity that is smaller than ccp_alpha will be chosen. By default, no pruning is performed.
- **max_samples** (*union type, optional, not for optimizer, default None*) – If bootstrap is True, the number of samples to draw from X to train each base estimator.
  - None
    Draw X.shape[0] samples.
  - *or* integer, >=1
    Draw max_samples samples.
  - *or* float, >0.0, <1.0
    Draw max_samples * X.shape[0] samples.
- **monotonic_cst** (*union type, optional, not for optimizer, default None*) – Indicates the monotonicity constraint to enforce on each feature. Monotonicity constraints are not supported for: multioutput regressions (i.e. when n_outputs > 1),

  regressions trained on data with missing values.
  - array *of* items : -1, 0, *or* 1
    array-like of int of shape (n_features)
  - *or* None
    No constraints are applied.

Notes

constraint-1 : negated type *of* 'y/isSparse'
    This classifier does not support sparse labels.
constraint-2 : union type
    Out of bag estimation only available if bootstrap=True
    - bootstrap : True
    - *or* oob_score : False

fit(*X*, *y=None*, ***fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

- **X** (array *of* items : array *of* items : float) – The training input samples. Internally, its dtype will be converted
- **y** (array *of* items : float) – The target values (class labels in classification, real numbers in
- **sample_weight** (*union type, optional*) – Sample weights. If None, then samples are equally weighted. Splits
  - array *of* items : float
  - *or* None

predict(*X*, ***predict_params*)

Make predictions.

Note: The predict method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (array, optional *of* items : array *of* items : float) – The input samples. Internally, its dtype will be converted to

Returns

result – The predicted values.

Return type

array *of* items : float

lale.lib.sklearn.feature_agglomeration module

class lale.lib.sklearn.feature_agglomeration.**FeatureAgglomeration**(*, *n_clusters=2*, *memory=None*, *connectivity=None*, *compute_full_tree='auto'*, *linkage='ward'*, *pooling_func='<function mean>'*, *distance_threshold=None*, *compute_distances=False*, *metric='euclidean'*)

Bases: *PlannedIndividualOp*

Feature agglomeration transformer from scikit-learn.

This documentation is auto-generated from JSON schemas.

Parameters

- **n_clusters** (*union type, optional, not for optimizer, default 2*) – The number of clusters to find.
  - integer, >=2 for optimizer, <='X/maxItems', <=8 for optimizer
  - *or* None, not for optimizer

  See also *constraint-3*.
- **memory** (*union type, not for optimizer, default None*) – Used to cache the output of the computation of the tree.
  - string

    Path to the caching directory.
  - *or* dict, not for optimizer

    Object with the joblib.Memory interface
  - *or* None

    No caching.
- **connectivity** (*union type, optional, not for optimizer, default None*) – Connectivity matrix. Defines for each feature the neighboring features following a given structure of the data.
  - array *of* items : array *of* items : float
  - *or* callable, not for optimizer

    A callable that transforms the data into a connectivity matrix, such as derived from kneighbors_graph.
  - *or* None
- **compute_full_tree** (*union type, default 'auto'*) – Stop early the construction of the tree at n_clusters.

  - boolean
  - *or* 'auto'

  See also *constraint-4*.

- **linkage** ('ward', 'complete', 'average', *or* 'single', optional, default 'ward') – Which linkage criterion to use. The linkage criterion determines which distance to use between sets of features.

  See also *constraint-1*.

- **pooling_func** (*callable, not for optimizer, default <function mean at 0x7f328b109cf0>*) – This combines the values of agglomerated features into a single value, and should accept an array of shape [M, N] and the keyword argument axis=1, and reduce it to an array of size [M].
- **distance_threshold** (*union type, optional, not for optimizer, default None*) – The linkage distance threshold above which, clusters will not be merged.
  - float
  - *or* None

  See also *constraint-3*, *constraint-4*.

- **compute_distances** (*boolean, optional, not for optimizer, default False*) – Computes distances between clusters even if distance_threshold is not used. This can be used to make dendrogram visualization, but introduces a computational and memory overhead.
- **metric** (*union type, optional, not for optimizer, default 'euclidean'*) – Metric used to compute the linkage. The default is *euclidean*
  - 'euclidean', 'l1', 'l2', 'manhattan', 'cosine', *or* 'precomputed'
  - *or* None, not for optimizer

    deprecated
  - *or* callable, not for optimizer

  See also *constraint-1*.

Notes

constraint-1 : union type

affinity, if linkage is "ward", only "euclidean" is accepted

- affinity : 'euclidean' *or* None
- *or* metric : 'euclidean' *or* None
- *or* linkage : negated type *of* 'ward'

constraint-2 : negated type *of* 'X/isSparse'

A sparse matrix was passed, but dense data is required. Use X.toarray() to convert to a dense numpy array.

constraint-3 : union type

n_clusters must be None if distance_threshold is not None.

- n_clusters : None
- *or* distance_threshold : None

constraint-4 : union type

compute_full_tree must be True if distance_threshold is not None.

- compute_full_tree : 'True'
- *or* distance_threshold : None

fit(*X*, *y=None*, ***fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

- **X** (array *of* items : array *of* items : float) – The data
- **y** (*any type, optional*) – Ignored

transform(*X*, *y=None*)

Transform the data.

Note: The transform method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (array *of* items : array *of* items : float) – A M by N array of M observations in N dimensions or a length

Returns

result – The pooled values for each feature cluster.

Return type

array *of* items : array *of* items : float

lale.lib.sklearn.fit_spec_proxy module

lale.lib.sklearn.function_transformer module

class lale.lib.sklearn.function_transformer.**FunctionTransformer**(*, *func=None*, *inverse_func=None*, *validate=False*, *accept_sparse=False*, *check_inverse=True*, *kw_args=None*, *inv_kw_args=None*, *feature_names_out=None*)

Bases: *PlannedIndividualOp*

FunctionTransformer from scikit-learn constructs a transformer from an arbitrary callable that operates at the level of an entire dataset.

This documentation is auto-generated from JSON schemas.

Parameters

- **func** (*union type, not for optimizer, default None*) – The callable to use for the transformation.
  - callable
  - *or* None
- **inverse_func** (*union type, not for optimizer, default None*) – The callable to use for the inverse transformation.
  - callable
  - *or* None
- **validate** (*boolean, not for optimizer, default False*) – Indicate that the input X array should be checked before calling `func`.

  See also *constraint-1*.
- **accept_sparse** (*boolean, not for optimizer, default False*) – Indicate that func accepts a sparse matrix as input.

  See also *constraint-1*.
- **check_inverse** (*boolean, not for optimizer, default True*) – Whether to check that `func` followed by `inverse_func` leads to the original inputs.
- **kw_args** (*union type, not for optimizer, default None*) – Dictionary of additional keyword arguments to pass to func.

  - dict
  - *or* None
- **inv_kw_args** (*union type, not for optimizer, default None*) – Dictionary of additional keyword arguments to pass to inverse_func.
  - dict
  - *or* None
- **feature_names_out** (*union type, optional, not for optimizer, default None*) – Determines the list of feature names that will be returned by the `get_feature_names_out` method. If it is 'one-to-one', then the output feature names will be equal to the input feature names. If it is a callable, then it must take two positional arguments: this `FunctionTransformer` (`self`) and an array-like of input feature names (`input_features`). It must return an array-like of output feature names. The `get_feature_names_out` method is only defined if `feature_names_out` is not None.
  - callable
  - *or* 'one-to-one' *or* None

Notes

constraint-1 : union type

If validate is False, then accept_sparse has no effect. Otherwise, if accept_sparse is false, sparse matrix inputs will cause an exception to be raised.

- validate : False
- *or* negated type *of* 'X/isSparse'
- *or* accept_sparse : True

fit(*X*, *y=None*, ***fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

- **X** (*array*) –
  - items : array
    - items : union type
      - float
      - *or* string
- **y** (*Any, optional*) –

transform(*X*, *y=None*)

Transform the data.

Note: The transform method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (*array*) –

- items : array
  - items : union type
    - float
    - *or* string

Returns

result

Return type

array *of* items : Any

lale.lib.sklearn.gaussian_nb module

class lale.lib.sklearn.gaussian_nb.**GaussianNB**(**, priors=None, var_smoothing=1e-09*)

Bases: *PlannedIndividualOp*

Gaussian Naive Bayes classifier from scikit-learn.

This documentation is auto-generated from JSON schemas.

Parameters

- **priors** (*union type, not for optimizer, default None*) – Prior probabilities of the classes. If specified the priors are not
  - array *of* items : float
  - *or* None
- **var_smoothing** (*float, >=0.0 for optimizer, <=1.0 for optimizer, optional, not for optimizer, default 1e-09*) – Portion of the largest variance of all features that is added to variances for calculation stability.

Notes

constraint-1 : negated type *of* 'X/isSparse'

A sparse matrix was passed, but dense data is required. Use X.toarray() to convert to a dense numpy array.

fit(*X, y=None, **fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

- **X** (array *of* items : array *of* items : float) –
- **y** (*union type*) –
  - array *of* items : string
  - *or* array *of* items : float
  - *or* array *of* items : boolean
- **sample_weight** (*union type, optional, default None*) – Weights applied to individual samples.
  - array *of* items : float
  - *or* None

    Uniform weights.

partial_fit(*X, y=None, **fit_params*)

Incremental fit to train train the operator on a batch of samples.

Note: The partial_fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

- **X** (array *of* items : array *of* items : float) –
- **y** (*union type*) –
  - array *of* items : string
  - *or* array *of* items : float
  - *or* array *of* items : boolean
- **classes** (*union type, optional*) –
  - array *of* items : string
  - *or* array *of* items : float
  - *or* array *of* items : boolean

- **sample_weight** (*union type, optional, default None*) – Weights applied to individual samples.
  - array *of* items : float
  - *or* None

    Uniform weights.

predict(*X*, ***predict_params*)

Make predictions.

Note: The predict method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (array *of* items : array *of* items : float) –

Returns

result –

- array *of* items : string
- *or* array *of* items : float
- *or* array *of* items : boolean

Return type

union type

predict_proba(*X*)

Probability estimates for all classes.

Note: The predict_proba method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (array *of* items : array *of* items : float) –

Returns

result – Returns the probability of the samples for each class in

Return type

array *of* items : array *of* items : float

lale.lib.sklearn.gradient_boosting_classifier module

class lale.lib.sklearn.gradient_boosting_classifier.**GradientBoostingClassifier**(*, *loss='log_loss'*, *learning_rate=0.1*, *n_estimators=100*, *subsample=1.0*, *criterion='friedman_mse'*, *min_samples_split=2*, *min_samples_leaf=1*, *min_weight_fraction_leaf=0.0*, *max_depth=3*, *min_impurity_decrease=0.0*, *init=None*, *random_state=None*, *max_features=None*, *verbose=0*, *max_leaf_nodes=None*, *warm_start=False*, *validation_fraction=0.1*, *n_iter_no_change=None*, *tol=0.0001*, *ccp_alpha=0.0*)

Bases: `PlannedIndividualOp`

Gradient boosting classifier random forest from scikit-learn.

This documentation is auto-generated from JSON schemas.

Parameters

- **loss** ('log_loss' *or* 'exponential', optional, default 'log_loss') – TThe loss function to be optimized. 'log_loss' refers to binomial and multinomial deviance, the same as used in logistic regression. It is a good choice for classification with probabilistic outputs. For loss 'exponential', gradient boosting recovers the AdaBoost algorithm.
- **learning_rate** (*float, >=0.01 for optimizer, <=1.0 for optimizer, loguniform distribution, optional, not for optimizer, default 0.1*) – learning rate shrinks the contribution of each tree by *learning_rate*.
- **n_estimators** (*integer, >=1, >=10 for optimizer, <=100 for optimizer, uniform distribution, optional, default 100*) – The number of boosting stages to perform.
- **subsample** (*float, >0.0, >=0.01 for optimizer, <=1.0, <=1.0 for optimizer, uniform distribution, optional, not for optimizer, default 1.0*) – The fraction of samples to be used for fitting the individual base learners.
- **criterion** (union type, optional, not for optimizer, default 'friedman_mse' *of* 'squared_error' *or* 'friedman_mse') – The function to measure the quality of a split. Supported criteria are *friedman_mse* for the mean squared error with improvement score by Friedman, *squared_error* for mean squared error. The default value of *friedman_mse* is generally the best as it can provide a better approximation in some cases.
- **min_samples_split** (*union type, optional, default 2*) – The minimum number of samples required to split an internal node:
  - integer, >=2, uniform distribution, not for optimizer
  - *or* float, >0.0, >=0.01 for optimizer, <=1.0, <=0.5 for optimizer, default 0.05

- **min_samples_leaf** (*union type, optional, default 1*) – The minimum number of samples required to be at a leaf node.
  - integer, >=1, not for optimizer
  - *or* float, >0.0, >=0.01 for optimizer, <=0.5, default 0.05
- **min_weight_fraction_leaf** (*float, >=0.0, <=0.5, optional, not for optimizer, default 0.0*) – The minimum weighted fraction of the sum total of weights (of all the input samples) required to be at a leaf node. Samples have equal weight when sample_weight is not provided.
- **max_depth** (*integer, >=3 for optimizer, <=5 for optimizer, optional, default 3*) – Maximum depth of the individual regression estimators.
- **min_impurity_decrease** (*float, >=0.0, <=10.0 for optimizer, optional, not for optimizer, default 0.0*) – A node will be split if this split induces a decrease of the impurity greater than or equal to this value.
- **init** (*union type, not for optimizer, default None*) – An estimator object that is used to compute the initial predictions.
  - operator
  - *or* 'zero' *or* None
- **random_state** (*union type, optional, not for optimizer, default None*) – If int, random_state is the seed used by the random number generator;
  - integer
  - *or* numpy.random.RandomState
  - *or* None
- **max_features** (*union type, optional, default None*) – The number of features to consider when looking for the best split.
  - integer, >=2, <='X/items/maxItems', not for optimizer

    Consider max_features features at each split.
  - *or* float, >0.0, >=0.01 for optimizer, <1.0, uniform distribution, default 0.5
  - *or* 'auto', 'sqrt', 'log2', *or* None
- **verbose** (*integer, optional, not for optimizer, default 0*) – Enable verbose output. If 1 then it prints progress and performance
- **max_leaf_nodes** (*union type, optional, not for optimizer, default None*) – Grow trees with `max_leaf_nodes` in best-first fashion.
  - integer, >=1, >=3 for optimizer, <=1000 for optimizer
  - *or* None

    Unlimited number of leaf nodes.
- **warm_start** (*boolean, optional, not for optimizer, default False*) – When set to True, reuse the solution of the previous call to fit and add more estimators to the ensemble, otherwise, just erase the previous solution.
- **validation_fraction** (*float, >=0.0, <=1.0, optional, not for optimizer, default 0.1*) – The proportion of training data to set aside as validation set for early stopping.
- **n_iter_no_change** (*union type, optional, not for optimizer, default None*) – `n_iter_no_change` is used to decide if early stopping will be used
  - integer, >=5 for optimizer, <=10 for optimizer
  - *or* None
- **tol** (*float, >=1e-08 for optimizer, <=0.01 for optimizer, optional, not for optimizer, default 0.0001*) – Tolerance for the early stopping. When the loss is not improving
- **ccp_alpha** (*float, >=0.0, <=0.1 for optimizer, optional, not for optimizer, default 0.0*) – Complexity parameter used for Minimal Cost-Complexity Pruning. The subtree with the largest cost complexity that is smaller than ccp_alpha will be chosen. By default, no pruning is performed.

decision_function(*X*)

Confidence scores for all classes.

Note: The decision_function method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters
: **X** (array *of* items : array *of* items : float) – Features; the outer array is over samples.

Returns
: **result** – Confidence scores for samples for each class in the model.
  - array *of* items : array *of* items : float
    In the multi-way case, score per (sample, class) combination.
  - *or* array *of* items : float
    In the binary case, score for *self._classes[1]*.

Return type
: union type

`fit`(*X*, *y=None*, ***fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters
- **X** (array *of* items : array *of* items : float) – The input samples. Internally, it will be converted to
- **y** (*union type*) – Target values (strings or integers in classification, real numbers
  - array *of* items : float
  - *or* array *of* items : string
  - *or* array *of* items : boolean
- **sample_weight** (*union type, optional, default None*) – Sample weights. If None, then samples are equally weighted. Splits
  - array *of* items : float
  - *or* None
- **monitor** (*union type, optional, default None*) – The monitor is called after each iteration with the current the current iteration, a reference to the estimator and the local variables of _fit_stages as keyword arguments callable(i, self, locals()).
  - callable
  - *or* None

`predict`(*X*, ***predict_params*)

Make predictions.

Note: The predict method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters
: **X** (array, optional *of* items : array *of* items : float) – The input samples. Internally, it will be converted to

Returns
: **result** – The predicted values.
  - array *of* items : float
  - *or* array *of* items : string
  - *or* array *of* items : boolean

Return type
: union type

predict_proba(*X*)

Probability estimates for all classes.

Note: The predict_proba method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (array, optional *of* items : array *of* items : float) – The input samples. Internally, it will be converted to

Returns

result – The class probabilities of the input samples. The order of the

Return type

array *of* items : array *of* items : float

lale.lib.sklearn.gradient_boosting_regressor module

class lale.lib.sklearn.gradient_boosting_regressor.**GradientBoostingRegressor**(**, loss='squared_error', learning_rate=0.1, n_estimators=100, subsample=1.0, criterion='friedman_mse', min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_depth=3, min_impurity_decrease=0.0, init=None, random_state=None, max_features=None, alpha=0.9, verbose=0, max_leaf_nodes=None, warm_start=False, validation_fraction=0.1, n_iter_no_change=None, tol=0.0001, ccp_alpha=0.0*)

Bases: `PlannedIndividualOp`

Gradient boosting regressor random forest from scikit-learn.

This documentation is auto-generated from JSON schemas.

Parameters

- **loss** (union type, optional, default 'squared_error' *of* 'squared_error', 'absolute_error', 'huber', *or* 'quantile') – Loss function to be optimized. 'squared_error' refers to the squared error for regression. 'absolute_error' refers to the absolute error of regression and is a robust loss function. 'huber' is a combination of the two. 'quantile' allows quantile regression (use alpha to specify the quantile).
- **learning_rate** (*float, >=0.01 for optimizer, <=1.0 for optimizer, loguniform distribution, optional, not for optimizer, default 0.1*) – learning rate shrinks the contribution of each tree by *learning_rate*.

- **n_estimators** (*integer, >=1, >=10 for optimizer, <=100 for optimizer, uniform distribution, optional, default 100*) – The number of boosting stages to perform. Gradient boosting
- **subsample** (*float, >0.0, >=0.01 for optimizer, <=1.0, <=1.0 for optimizer, uniform distribution, optional, not for optimizer, default 1.0*) – The fraction of samples to be used for fitting the individual base
- **criterion** (union type, optional, not for optimizer, default 'friedman_mse' *of* 'squared_error' *or* 'friedman_mse') – Function to measure the quality of a split.
- **min_samples_split** (*union type, optional, default 2*) – The minimum number of samples required to split an internal node:
  - integer, >=2, uniform distribution, not for optimizer
  - *or* float, >0.0, >=0.01 for optimizer, <=1.0, <=0.5 for optimizer, default 0.05
- **min_samples_leaf** (*union type, optional, default 1*) – The minimum number of samples required to be at a leaf node.
  - integer, >=1, not for optimizer
  - *or* float, >0.0, >=0.01 for optimizer, <=0.5, default 0.05
- **min_weight_fraction_leaf** (*float, >=0.0, <=0.5, optional, not for optimizer, default 0.0*) – The minimum weighted fraction of the sum total of weights (of all the input samples) required to be at a leaf node. Samples have equal weight when sample_weight is not provided.
- **max_depth** (*integer, >=3 for optimizer, <=5 for optimizer, optional, default 3*) – maximum depth of the individual regression estimators.
- **min_impurity_decrease** (*float, >=0.0, <=10.0 for optimizer, optional, not for optimizer, default 0.0*) – A node will be split if this split induces a decrease of the impurity greater than or equal to this value.
- **init** (*union type, not for optimizer, default None*) – An estimator object that is used to compute the initial predictions.
  - operator
  - *or* 'zero' *or* None
- **random_state** (*union type, optional, not for optimizer, default None*) – If int, random_state is the seed used by the random number generator;
  - integer
  - *or* numpy.random.RandomState
  - *or* None
- **max_features** (*union type, optional, default None*) – The number of features to consider when looking for the best split.
  - integer, >=2, <='X/items/maxItems', not for optimizer

    Consider max_features features at each split.
  - *or* float, >0.0, >=0.01 for optimizer, <1.0, uniform distribution, default 0.5
  - *or* 'auto', 'sqrt', 'log2', *or* None
- **alpha** (*float, >=1e-10 for optimizer, <=0.9999999999 for optimizer, loguniform distribution, optional, default 0.9*) – The alpha-quantile of the huber loss function and the quantile
- **verbose** (*integer, optional, not for optimizer, default 0*) – Enable verbose output. If 1 then it prints progress and performance
- **max_leaf_nodes** (*union type, optional, not for optimizer, default None*) – Grow trees with `max_leaf_nodes` in best-first fashion.
  - integer, >=1, >=3 for optimizer, <=1000 for optimizer
  - *or* None

    Unlimited number of leaf nodes.
- **warm_start** (*boolean, optional, not for optimizer, default False*) – When set to `True`, reuse the solution of the previous call to fit
- **validation_fraction** (*float, >=0.0, <=1.0, optional, not for*

optimizer, default 0.1) – The proportion of training data to set aside as validation set for early stopping.

- **n_iter_no_change** (*union type, optional, not for optimizer, default None*) – n_iter_no_change is used to decide if early stopping will be used
  - integer, >=5 for optimizer, <=10 for optimizer
  - *or* None
- **tol** (*float, >=1e-08 for optimizer, <=0.01 for optimizer, optional, not for optimizer, default 0.0001*) – Tolerance for the early stopping. When the loss is not improving
- **ccp_alpha** (*float, >=0.0, <=0.1 for optimizer, optional, not for optimizer, default 0.0*) – Complexity parameter used for Minimal Cost-Complexity Pruning. The subtree with the largest cost complexity that is smaller than ccp_alpha will be chosen. By default, no pruning is performed.

fit(*X*, *y=None*, ***fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

- **X** (array *of* items : array *of* items : float) – The input samples. Internally, it will be converted to
- **y** (array *of* items : float) – Target values (strings or integers in classification, real numbers
- **sample_weight** (*union type, optional, default None*) – Sample weights. If None, then samples are equally weighted. Splits
  - array *of* items : float
  - *or* None
- **monitor** (*union type, optional, default None*) – The monitor is called after each iteration with the current the current iteration, a reference to the estimator and the local variables of _fit_stages as keyword arguments callable(i, self, locals()).
  - callable
  - *or* None

predict(*X*, ***predict_params*)

Make predictions.

Note: The predict method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (array, optional *of* items : array *of* items : float) – The input samples. Internally, it will be converted to

Returns

result – The predicted values.

Return type

array *of* items : float

lale.lib.sklearn.isolation_forest module

class lale.lib.sklearn.isolation_forest.**IsolationForest**(**, n_estimators=100, max_samples='auto', contamination='auto', max_features=1.0, bootstrap=True, n_jobs=None, random_state=None, verbose=0, warm_start=False*)

Bases: `PlannedIndividualOp`

Isolation forest from scikit-learn for getting the anomaly score of each sample using the IsolationForest algorithm.

This documentation is auto-generated from JSON schemas.

Parameters

- **n_estimators** (*integer, >=10 for optimizer, <=100 for optimizer, uniform distribution, default 100*) – The number of base estimators in the ensemble.
- **max_samples** (*union type, default 'auto'*) – The number of samples to draw from X to train each base estimator.
  - integer, >=2, <='X/maxItems', not for optimizer
    Draw max_samples samples.
  - *or* float, >0.0, >=0.2 for optimizer, <=1.0, <=1.0 for optimizer
    Draw max_samples * X.shape[0] samples.
  - *or* 'auto'
    Draw max_samples=min(256, n_samples) samples.
- **contamination** (*union type, not for optimizer, default 'auto'*) – The amount of contamination of the data set, i.e. the proportion of outliers in the data set. Used when fitting to define the threshold on the scores of the samples.
  - float, >=0.0, <=0.5
  - *or* 'auto'
- **max_features** (*union type, default 1.0*) – The number of features to draw from X to train each base estimator.
  - integer, >=2, <='X/items/maxItems', not for optimizer
    Draw max_features features.
  - *or* float, >0.0, >=0.01 for optimizer, <=1.0, <=1.0 for optimizer
    Draw max_samples * X.shape[1] features.
- **bootstrap** (*boolean, default True*) – Whether samples are drawn with (True) or without (False) replacement.
- **n_jobs** (*union type, not for optimizer, default None*) – The number of jobs to run in parallel for both *fit* and *predict*.
  - None
    1 unless in joblib.parallel_backend context.
  - *or* -1
    Use all processors.
  - *or* integer, >=1
    Number of CPU cores.
- **random_state** (*union type, not for optimizer, default None*) – Controls the pseudo-randomness of the selection of the feature and split values for each branching step and each tree in the forest. If int, random_state is the seed used by the random number generator
  - integer
  - *or* numpy.random.RandomState
  - *or* None
- **verbose** (*integer, not for optimizer, default 0*) – Controls the verbosity of the tree building process.

- **warm_start** (*boolean, not for optimizer, default False*) – When set to True, reuse the solution of the previous call to fit and add more estimators to the ensemble, otherwise, just fit a whole new ensemble.

decision_function(*X*)

Confidence scores for all classes.

Note: The decision_function method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters
: **X** (array *of* items : array *of* items : float) – Features; the outer array is over samples.

Returns
: **result**

Return type
: array *of* items : float

fit(*X*, *y=None*, ***fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

- **X** (array *of* items : array *of* items : float) – The training input samples. Sparse matrices are accepted only if
- **y** (*union type, optional*) –
  - array *of* items : float
    The target values (class labels in classification, real numbers in
  - *or* None
- **sample_weight** (*union type, optional*) – Sample weights. If None, then samples are equally weighted.
  - array *of* items : float
  - *or* None

predict(*X*, ***predict_params*)

Make predictions.

Note: The predict method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters
: **X** (array *of* items : array *of* items : float) –

Returns
: **result**

Return type
: array *of* items : float

lale.lib.sklearn.isomap module

class lale.lib.sklearn.isomap.**Isomap**(**, n_neighbors=5, n_components=2, eigen_solver='auto', tol=0, max_iter=None, path_method='auto', neighbors_algorithm='auto', n_jobs=None, metric='minkowski', p=2, metric_params=None, radius=None*)

Bases: `PlannedIndividualOp`

"Isomap embedding from scikit-learn.

This documentation is auto-generated from JSON schemas.

Parameters

- **n_neighbors** (*integer, >=5 for optimizer, <=20 for optimizer, uniform distribution, default 5*) – number of neighbors to consider for each point.
- **n_components** (*integer, >=2 for optimizer, <='X/items/maxItems', <=256 for optimizer, uniform distribution, default 2*) – number of coordinates for the manifold
- **eigen_solver** ('auto', 'arpack', *or* 'dense', default 'auto') – 'auto' : Attempt to choose the most efficient solver for the given problem
- **tol** (*float, >=0 for optimizer, <=1 for optimizer, uniform distribution, default 0*) – Convergence tolerance passed to arpack or lobpcg
- **max_iter** (*union type, not for optimizer, default None*) – Maximum number of iterations for the arpack solver
  - integer
  - *or* None
- **path_method** ('auto', 'FW', *or* 'D', default 'auto') – Method to use in finding shortest path
- **neighbors_algorithm** ('auto', 'brute', 'kd_tree', *or* 'ball_tree', default 'auto') – Algorithm to use for nearest neighbors search, passed to neighbors.NearestNeighbors instance.
- **n_jobs** (*union type, not for optimizer, default None*) – The number of parallel jobs to run
  - integer
  - *or* None
- **metric** (*Any, not for optimizer, default 'minkowski'*) – The metric to use when calculating distance between instances in a feature array. If metric is a string or callable, it must be one of the options allowed by sklearn.metrics.pairwise_distances for its metric parameter. If metric is "precomputed", X is assumed to be a distance matrix and must be square.
- **p** (*integer, not for optimizer, default 2*) – Parameter for the Minkowski metric from sklearn.metrics.pairwise.pairwise_distances. When p = 1, this is equivalent to using manhattan_distance (l1), and euclidean_distance (l2) for p = 2. For arbitrary p, minkowski_distance (l_p) is used.
- **metric_params** (*union type, not for optimizer, default None*) – Additional keyword arguments for the metric function
  - dict
  - *or* None
- **radius** (*union type, optional, not for optimizer, default None*) – Limiting distance of neighbors to return. If `radius` is a float, then `n_neighbors` must be set to `None`.
  - float
  - *or* None

Notes

constraint-1 : union type

- intersection type
  - dict *of* n_neighbors : integer
  - *and* dict *of* radius : None
- *or* intersection type
  - dict *of* n_neighbors : None
  - *and* dict *of* radius : float

fit(*X*, *y=None*, ***fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

- **X** (*Any*) – Sample data, shape = (n_samples, n_features), in the form of a numpy array, precomputed tree, or NearestNeighbors object.
- **y** (*any type, optional*) –

transform(*X*, *y=None*)

Transform the data.

Note: The transform method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters
X (*Any*) –

Returns
result – Transform X.

Return type
Any

lale.lib.sklearn.k_means module

class lale.lib.sklearn.k_means.**KMeans**(**, n_clusters=8, init='k-means++', n_init=10, max_iter=300, tol=0.0001, verbose=0, random_state=None, copy_x=True, algorithm='lloyd'*)

Bases: *PlannedIndividualOp*

KMeans from scikit-learn.

This documentation is auto-generated from JSON schemas.

Parameters

- **n_clusters** (*integer, >=2 for optimizer, <=8 for optimizer, uniform distribution, default 8*) – The number of clusters to form as well as the number of centroids to generate.
- **init** (*union type, default 'k-means++'*) – Method for initialization, defaults to *k-means++*. *k-means++* : selects initial cluster centers for k-mean clustering in a smart way to speed up convergence. See section Notes in k_init for more details. *random*: choose n_clusters observations (rows) at random from data for the initial centroids. If an array is passed, it should be of shape (n_clusters, n_features) and gives the initial centers. If a callable is passed, it should take arguments X, n_clusters and a random state and return an initialization.
  - 'k-means++' *or* 'random'
  - *or* callable, not for optimizer

    - *or* array, not for optimizer *of* items : array *of* items : float
- **n_init** (*union type, default 10*) – Number of time the k-means algorithm will be run with different centroid seeds. The final results will be the best output of n_init consecutive runs in terms of inertia. When n_init='auto', the number of runs will be 10 if using init='random', and 1 if using init='kmeans++'.
    - integer, >=3 for optimizer, <=10 for optimizer, uniform distribution
    - *or* 'auto'
- **max_iter** (*integer, >=10 for optimizer, <=1000 for optimizer, uniform distribution, default 300*) – Maximum number of iterations of the k-means algorithm for a single run.
- **tol** (*float, >=1e-08 for optimizer, <=0.01 for optimizer, default 0.0001*) – Relative tolerance with regards to Frobenius norm of the difference in the cluster centers of two consecutive iterations to declare convergence.
- **verbose** (*integer, not for optimizer, default 0*) – Verbosity mode.
- **random_state** (*union type, not for optimizer, default None*) – Determines random number generation for centroid initialization
    - integer
    - *or* numpy.random.RandomState
    - *or* None
- **copy_x** (*boolean, default True*) – When pre-computing distances it is more numerically accurate to center the data first. If copy_x is True (default), then the original data is not modified. If False, the original data is modified, and put back before the function returns, but small numerical differences may be introduced by subtracting and then adding the data mean. Note that if the original data is not C-contiguous, a copy will be made even if copy_x is False. If the original data is sparse, but not in CSR format, a copy will be made even if copy_x is False.
- **algorithm** ('lloyd' *or* 'elkan', default 'lloyd') – K-means algorithm to use. The classical EM-style algorithm is "lloyd". The "elkan" variation is more efficient on data with well-defined clusters, by using the triangle inequality. However it's more memory intensive due to the allocation of an extra array of shape (n_samples, n_clusters).

fit(*X*, *y=None*, ***fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

- **X** (array *of* items : array *of* items : float) – Training instances to cluster. Array-like or sparse matrix, shape=(n_samples, n_features)
- **y** (*any type, optional*) – not used, present here for API consistency by convention.
- **sample_weight** (*union type, optional, default 'deprecated'*) – The parameter *sample_weight* is deprecated in version 1.3 and will be removed in 1.5.
    - array *of* items : float
    - *or* None *or* 'deprecated'

predict(*X*, ***predict_params*)

Make predictions.

Note: The predict method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

- **X** (array *of* items : array *of* items : float) – New data to predict.

- **sample_weight** (*union type, optional, default None*) – The weights for each observation in X
  - array *of* items : float
  - *or* None

Returns
: **result** – Index of the cluster each sample belongs to.

Return type
: array *of* items : float

transform(*X*, *y=None*)

Transform the data.

Note: The transform method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters
: **X** (array *of* items : array *of* items : float) – New data to transform.

Returns
: **result** – X transformed in the new space.

Return type
: array *of* items : array *of* items : float

lale.lib.sklearn.k_neighbors_classifier module

class lale.lib.sklearn.k_neighbors_classifier.**KNeighborsClassifier**(**, n_neighbors=5, weights='uniform', algorithm='auto', leaf_size=30, p=2, metric='minkowski', metric_params=None, n_jobs=None*)

Bases: *PlannedIndividualOp*

K nearest neighbors classifier from scikit-learn.

This documentation is auto-generated from JSON schemas.

Parameters

- **n_neighbors** (*integer, >=1, <='X/maxItems', <=100 for optimizer, uniform distribution, default 5*) – Number of neighbors to use by default for kneighbors queries.
- **weights** ('uniform' *or* 'distance', default 'uniform') – Weight function used in prediction.
- **algorithm** ('ball_tree', 'kd_tree', 'brute', *or* 'auto', default 'auto') – Algorithm used to compute the nearest neighbors.
- **leaf_size** (*integer, >=1, <=100 for optimizer, uniform distribution, not for optimizer, default 30*) – Leaf size passed to BallTree or KDTree.
- **p** (*integer, >=1, <=3 for optimizer, uniform distribution, default 2*) – Power parameter for the Minkowski metric.
- **metric** ('euclidean', 'manhattan', *or* 'minkowski', default 'minkowski') – The distance metric to use for the tree.
- **metric_params** (*union type, not for optimizer, default None*) – Additional keyword arguments for the metric function.
  - None
  - *or* dict

- **n_jobs** (*union type, not for optimizer, default None*) – Number of parallel jobs to run for the neighbor search.
  - None
    1 unless in joblib.parallel_backend context.
  - *or* -1
    Use all processors.
  - *or* integer, >=1
    Number of CPU cores.

`fit`(*X*, *y=None*, ***fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

- **X** (array *of* items : array *of* items : float) – Features; the outer array is over samples.
- **y** (*union type*) – Target class labels; the array is over samples.
  - array *of* items : float
  - *or* array *of* items : array *of* items : float
  - *or* array *of* items : string
  - *or* array *of* items : boolean

`predict`(*X*, ***predict_params*)

Make predictions.

Note: The predict method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (array *of* items : array *of* items : float) – Features; the outer array is over samples.

Returns

result – Predicted class label per sample.

- array *of* items : float
- *or* array *of* items : array *of* items : float
- *or* array *of* items : string
- *or* array *of* items : boolean

Return type

union type

`predict_proba`(*X*)

Probability estimates for all classes.

Note: The predict_proba method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (array *of* items : array *of* items : float) – Features; the outer array is over samples.

Returns

result – Probability of the sample for each class in the model.

Return type

array *of* items : array *of* items : float

lale.lib.sklearn.k_neighbors_regressor module

class lale.lib.sklearn.k_neighbors_regressor.**KNeighborsRegressor**(**, n_neighbors=5, weights='uniform', algorithm='auto', leaf_size=30, p=2, metric='minkowski', metric_params=None, n_jobs=None*)

Bases: `PlannedIndividualOp`

K nearest neighbors regressor from scikit-learn.

This documentation is auto-generated from JSON schemas.

Parameters

- **n_neighbors** (*integer, >=1, <='X/maxItems', <=100 for optimizer, uniform distribution, default 5*) – Number of neighbors to use by default for kneighbors queries.
- **weights** ('uniform' *or* 'distance', default 'uniform') – Weight function used in prediction.
- **algorithm** ('ball_tree', 'kd_tree', 'brute', *or* 'auto', default 'auto') – Algorithm used to compute the nearest neighbors.
- **leaf_size** (*integer, >=1, <=100 for optimizer, uniform distribution, not for optimizer, default 30*) – Leaf size passed to BallTree or KDTree.
- **p** (*integer, >=1, <=3 for optimizer, uniform distribution, default 2*) – Power parameter for the Minkowski metric.
- **metric** (*union type, default 'minkowski'*) – The distance metric to use for the tree.
  - 'cityblock', 'cosine', 'euclidean', 'haversine', 'l1', 'l2', 'manhattan', 'nan_euclidean', *or* 'precomputed'
  - *or* callable, not for optimizer

    It takes two arrays representing 1D vectors as inputs and must return one value indicating the distance between those vectors. This works for Scipy's metrics, but is less efficient than passing the metric name as a string.
  - *or* Any, not for optimizer

    It will be passed directly to the underlying computation routines.
- **metric_params** (*union type, not for optimizer, default None*) – Additional keyword arguments for the metric function.
  - None
  - *or* dict
- **n_jobs** (*union type, not for optimizer, default None*) – Number of parallel jobs to run for the neighbor search.
  - None

    1 unless in joblib.parallel_backend context.
  - *or* -1

    Use all processors.
  - *or* integer, >=1

    Number of CPU cores.

fit(*X, y=None, **fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

- **X** (array *of* items : array *of* items : float) – Features; the outer array is over samples.
- **y** (*union type*) – Target class labels; the array is over samples.
  - array *of* items : float
  - *or* array *of* items : array *of* items : float

`predict`(*X*, ***predict_params*)

Make predictions.

Note: The predict method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (array *of* items : array *of* items : float) – Features; the outer array is over samples.

Returns

result – Returns predicted values.

- array *of* items : float
- *or* array *of* items : array *of* items : float

Return type

union type

lale.lib.sklearn.linear_regression module

`class lale.lib.sklearn.linear_regression.LinearRegression`(*, *fit_intercept=True*, *copy_X=True*, *n_jobs=None*, *positive=False*)

Bases: *PlannedIndividualOp*

Linear regression linear model from scikit-learn for classification.

This documentation is auto-generated from JSON schemas.

Parameters

- **fit_intercept** (*boolean, default True*) – Whether to calculate the intercept for this model.
- **copy_X** (*boolean, default True*) – If True, X will be copied; else, it may be overwritten.
- **n_jobs** (*union type, optional, not for optimizer, default None*) – The number of jobs to run in parallel.
  - None

    1 unless in joblib.parallel_backend context.
  - *or* -1

    Use all processors.
  - *or* integer, >=1

    Number of CPU cores.
- **positive** (*boolean, optional, not for optimizer, default False*) – When set to True, forces the coefficients to be positive.

  See also *constraint-1*.

Notes

constraint-1 : union type
Setting positive=True is only supported for dense arrays.
- positive : False
- *or* negated type *of* 'X/isSparse'

fit(*X*, *y=None*, ***fit_params*)
Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters
- **X** (array *of* items : array *of* items : float) – Features; the outer array is over samples.
- **y** (*union type*) – Target values. Will be cast to X's dtype if necessary
  - array *of* items : array *of* items : float
  - *or* array *of* items : float
- **sample_weight** (*union type, optional*) – Sample weights.
  - array *of* items : float
  - *or* None
    Samples are equally weighted.

predict(*X*, ***predict_params*)
Make predictions.

Note: The predict method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters
X (array, optional *of* items : array *of* items : float) – Samples.

Returns
result – Returns predicted values.
- array *of* items : array *of* items : float
- *or* array *of* items : float

Return type
union type

lale.lib.sklearn.linear_svc module

class lale.lib.sklearn.linear_svc.**LinearSVC**(*, *penalty='l2'*, *loss='squared_hinge'*, *dual=True*, *tol=0.0001*, *C=1.0*, *multi_class='ovr'*, *fit_intercept=True*, *intercept_scaling=1.0*, *class_weight=None*, *verbose=0*, *random_state=None*, *max_iter=1000*)

Bases: *PlannedIndividualOp*

Linear Support Vector Classification from scikit-learn.

This documentation is auto-generated from JSON schemas.

Parameters
- **penalty** ('l1' *or* 'l2', default 'l2') – Norm used in the penalization.

  See also *constraint-1*, *constraint-2*, *constraint-3*.
- **loss** ('hinge' *or* 'squared_hinge', default 'squared_hinge') – Loss function.

  See also *constraint-1*, *constraint-2*, *constraint-3*.

- **dual** (*union type, default True*) – Select the algorithm to either solve the dual or primal optimization problem.
  - boolean
    Prefer dual=False when n_samples > n_features.
  - *or* 'auto'
    Choose the value of the parameter automatically, based on the values of n_samples, n_features, loss, multi_class and penalty. If n_samples < n_features and optimizer supports chosen loss, multi_class and penalty, then dual will be set to True, otherwise it will be set to False.

  See also *constraint-2*, *constraint-3*.
- **tol** (*float, >0.0, <=0.01 for optimizer, default 0.0001*) – Tolerance for stopping criteria.
- **C** (*float, >0.0, >=0.03125 for optimizer, <=32768 for optimizer, loguniform distribution, default 1.0*) – Penalty parameter C of the error term.
- **multi_class** ('ovr' *or* 'crammer_singer', default 'ovr') – Determines the multi-class strategy if *y* contains more than two classes.

  See also *constraint-1*, *constraint-2*, *constraint-3*.
- **fit_intercept** (*boolean, default True*) – Whether to calculate the intercept for this model.
- **intercept_scaling** (*float, >0.0, <=1.0 for optimizer, not for optimizer, default 1.0*) – Append a constant feature with constant value intercept_scaling to the instance vector.
- **class_weight** (*union type, not for optimizer, default None*) –
  - None
    By default, all classes have weight 1.
  - *or* 'balanced'
    Adjust weights by inverse frequency.
  - *or* dict, not for optimizer
    Dictionary mapping class labels to weights.
- **verbose** (*integer, not for optimizer, default 0*) – Enable verbose output.
- **random_state** (*union type, not for optimizer, default None*) – Seed of pseudo-random number generator.
  - numpy.random.RandomState
  - *or* None
    RandomState used by np.random
  - *or* integer
    Explicit seed.
- **max_iter** (*integer, >=1, >=10 for optimizer, <=1000 for optimizer, not for optimizer, default 1000*) – The maximum number of iterations to be run.

Notes

constraint-1 : union type
The combination of penalty=`l1` and loss=`hinge` is not supported. If multi_class='crammer_singer', the options loss, penalty and dual will be ignored.

- penalty : 'l2'
- *or* loss : 'squared_hinge'
- *or* multi_class : 'crammer_singer'

constraint-2 : union type
The combination of penalty=`l2` and loss=`hinge` is not supported when dual=False. If multi_class='crammer_singer', the options loss, penalty and dual will be ignored.

- penalty : 'l1'
- *or* loss : 'squared_hinge'
- *or* dual : True
- *or* multi_class : 'crammer_singer'

constraint-3 : union type
The combination of penalty=`l1` and loss=`squared_hinge` is not supported when dual=True. If multi_class='crammer_singer', the options loss, penalty and dual will be ignored.

- penalty : 'l2'
- *or* loss : 'hinge'
- *or* dual : False
- *or* multi_class : 'crammer_singer'

decision_function(*X*)

Confidence scores for all classes.

Note: The decision_function method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (*array*) – The outer array is over samples aka rows.

- items : array *of* items : float
  The inner array is over features aka columns.

Returns

result – Confidence scores for samples for each class in the model.

- array *of* items : array *of* items : float
  In the multi-way case, score per (sample, class) combination.
- *or* array *of* items : float
  In the binary case, score for *self._classes[1]*.

Return type

union type

fit(*X*, *y=None*, ***fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

- **X** (*array*) – The outer array is over samples aka rows.
  - items : array *of* items : float
    The inner array is over features aka columns.
- **y** (*union type*) – The predicted classes.
  - array *of* items : float
  - *or* array *of* items : string
  - *or* array *of* items : boolean

- **sample_weight** (*union type, optional*) – Sample weights.
  - array *of* items : float
  - *or* None
    Samples are equally weighted.

predict(*X*, ***predict_params*)

Make predictions.

Note: The predict method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (*array, optional*) – The outer array is over samples aka rows.
- items : array *of* items : float
  The inner array is over features aka columns.

Returns

result – Predict class labels for samples in X.
- array *of* items : float
- *or* array *of* items : string
- *or* array *of* items : boolean

Return type

union type

lale.lib.sklearn.linear_svr module

class lale.lib.sklearn.linear_svr.**LinearSVR**(**, epsilon=0.0, tol=0.0001, C=1.0, loss='epsilon_insensitive', fit_intercept=True, intercept_scaling=1.0, dual=True, verbose=0, random_state=None, max_iter=1000*)

Bases: *PlannedIndividualOp*

LinearSVR from scikit-learn.

This documentation is auto-generated from JSON schemas.

Parameters

- **epsilon** (*float, >=1e-08 for optimizer, <=1.35 for optimizer, loguniform distribution, default 0.0*) – Epsilon parameter in the epsilon-insensitive loss function. Note that the value of this parameter depends on the scale of the target variable y. If unsure, set epsilon=0.
- **tol** (*float, >=1e-08 for optimizer, <=0.01 for optimizer, default 0.0001*) – Tolerance for stopping criteria.
- **C** (*float, not for optimizer, default 1.0*) – Regularization parameter. The strength of the regularization is inversely proportional to C. Must be strictly positive.
- **loss** ('squared_epsilon_insensitive' *or* 'epsilon_insensitive', default 'epsilon_insensitive') –

  Specifies the loss function.

  The epsilon-insensitive loss (standard SVR) is the L1 loss, while the squared epsilon-insensitive loss ('squared_epsilon_insensitive') is the L2 loss.

  See also *constraint-1*.
- **fit_intercept** (*boolean, default True*) – Whether to calculate the intercept for this model. If set to false, no intercept will be used in calculations (i.e. data is expected to be already centered).
- **intercept_scaling** (*float, not for optimizer, default 1.0*) – When self.fit_intercept is True, instance vector x becomes [x, self.intercept_scaling], i.e. a

"synthetic" feature with constant value equals to intercept_scaling is appended to the instance vector. The intercept becomes intercept_scaling * synthetic feature weight Note! the synthetic feature weight is subject to l1/l2 regularization as all other features. To lessen the effect of regularization on synthetic feature weight (and therefore on the intercept) intercept_scaling has to be increased.

- **dual** (*union type, default True*) – Select the algorithm to either solve the dual or primal optimization problem.
  - boolean
    Prefer dual=False when n_samples > n_features.
  - *or* 'auto'
    Choose the value of the parameter automatically, based on the values of n_samples, n_features, loss, multi_class and penalty. If n_samples < n_features and optimizer supports chosen loss, multi_class and penalty, then dual will be set to True, otherwise it will be set to False.

  See also *constraint-1*.
- **verbose** (*integer, not for optimizer, default 0*) – Enable verbose output. Note that this setting takes advantage of a per-process runtime setting in liblinear that, if enabled, may not work properly in a multithreaded context.
- **random_state** (*union type, not for optimizer, default None*) – Seed of pseudo-random number generator.
  - numpy.random.RandomState
  - *or* None
    RandomState used by np.random
  - *or* integer
    Explicit seed.
- **max_iter** (*integer, >=10 for optimizer, <=1000 for optimizer, uniform distribution, default 1000*) – The maximum number of iterations to be run.

Notes

constraint-1 : union type
    loss='epsilon_insensitive' is not supported when dual=False.
- loss : 'squared_epsilon_insensitive'
- *or* dual : True

fit(*X*, *y=None*, ***fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

- **X** (array *of* items : array *of* items : float) – Training vector, where n_samples in the number of samples and n_features is the number of features.
- **y** (array *of* items : float) – Target vector relative to X
- **sample_weight** (*union type, optional, default None*) – Array of weights that are assigned to individual samples
  - array *of* items : float
  - *or* None

predict(*X*, ***predict_params*)

Make predictions.

Note: The predict method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (*array*) – The outer array is over samples aka rows.

- items : array *of* items : float

  The inner array is over features aka columns.

Returns

result – Returns predicted values.

Return type

array *of* items : float

lale.lib.sklearn.logistic_regression module

class lale.lib.sklearn.logistic_regression.**LogisticRegression**(**, solver='lbfgs', penalty='l2', dual=False, C=1.0, tol=0.0001, fit_intercept=True, intercept_scaling=1.0, class_weight=None, random_state=None, max_iter=100, multi_class='auto', verbose=0, warm_start=False, n_jobs=None, l1_ratio=None*)

Bases: *PlannedIndividualOp*

Logistic regression linear model from scikit-learn for classification.

This documentation is auto-generated from JSON schemas.

Parameters

- **solver** ('lbfgs', 'liblinear', 'newton-cg', 'newton-cholesky', 'sag', *or* 'saga', default 'lbfgs') –

  Algorithm to use in the optimization problem. Default is 'lbfgs'. To choose a solver, you might want to consider the following aspects:

  For small datasets, 'liblinear' is a good choice, whereas 'sag' and 'saga' are faster for large ones; For multiclass problems, only 'newton-cg', 'sag', 'saga' and 'lbfgs' handle multinomial loss; 'liblinear' and is limited to one-versus-rest schemes. 'newton-cholesky' is a good choice for n_samples >> n_features, especially with one-hot encoded categorical features with rare categories. Note that it is limited to binary classification and the one-versus-rest reduction for multiclass classification. Be aware that the memory usage of this solver has a quadratic dependency on n_features because it explicitly computes the Hessian matrix.

  See also *constraint-1*, *constraint-2*, *constraint-3*, *constraint-4*, *constraint-6*.
- **penalty** ('l1', 'l2', 'elasticnet', *or* None, not for optimizer, default 'l2') – Norm used in the penalization.

  See also *constraint-1*, *constraint-2*, *constraint-4*, *constraint-5*, *constraint-6*.
- **dual** (*boolean, default False*) – Dual or primal formulation. Prefer dual=False when n_samples > n_features.

  See also *constraint-2*.
- **C** (*float, >0.0, >=0.03125 for optimizer, <=32768 for optimizer, loguniform distribution, not for optimizer, default 1.0*) – Inverse regularization strength. Smaller values specify stronger regularization.
- **tol** (*float, >0.0, >=1e-08 for optimizer, <=0.01 for optimizer, default 0.0001*) – Tolerance for stopping criteria.
- **fit_intercept** (*boolean, default True*) – Specifies whether a constant (bias or intercept) should be added to the decision function.

- **intercept_scaling** (*float, >=0.0, <=1.0, uniform distribution, default 1.0*) – Useful only when the solver 'liblinear' is used and self.fit_intercept is set to True. In this case, X becomes [X, self.intercept_scaling], i.e. a "synthetic" feature with constant value equal to intercept_scaling is appended to the instance vector. The intercept becomes "intercept_scaling * synthetic_feature_weight". Note! the synthetic feature weight is subject to l1/l2 regularization as all other features. To lessen the effect of regularization on synthetic feature weight (and therefore on the intercept) intercept_scaling has to be increased.
- **class_weight** (*union type, not for optimizer, default None*) –
  - None

    By default, all classes have weight 1.
  - *or* 'balanced'

    Uses the values of y to automatically adjust weights inversely proportional to class frequencies in the input data as "n_samples / (n_classes * np.bincount(y))".
  - *or* dict, not for optimizer

    Weights associated with classes in the form "{class_label: weight}".
- **random_state** (*union type, not for optimizer, default None*) – Seed of pseudo-random number generator for shuffling data when solver == 'sag', 'saga' or 'liblinear'.
  - None

    RandomState used by np.random
  - *or* numpy.random.RandomState

    Use the provided random state, only affecting other users of that same random state instance.
  - *or* integer

    Explicit seed.
- **max_iter** (*integer, >=1, >=10 for optimizer, <=1000 for optimizer, uniform distribution, default 100*) – Maximum number of iterations for solvers to converge.
- **multi_class** ('auto', 'ovr', *or* 'multinomial', default 'auto') – If the option chosen is *ovr*, then a binary problem is fit for each label. For *multinomial* the loss minimised is the multinomial loss fit across the entire probability distribution, even when the data is binary. *multinomial* is unavailable when solver=`liblinear`. *auto* selects *ovr* if the data is binary, or if solver=`liblinear`, and otherwise selects *multinomial*.

  See also *constraint-3*.
- **verbose** (*integer, not for optimizer, default 0*) – For the liblinear and lbfgs solvers set verbose to any positive number for verbosity.
- **warm_start** (*boolean, not for optimizer, default False*) – When set to True, reuse the solution of the previous call to fit as initialization, otherwise, just erase the previous solution. Useless for liblinear solver.
- **n_jobs** (*union type, not for optimizer, default None*) – Number of CPU cores when parallelizing over classes if multi_class is ovr. This parameter is ignored when the "solver" is set to 'liblinear' regardless of whether 'multi_class' is specified or not.
  - None

    1 unless in joblib.parallel_backend context.
  - *or* -1

    Use all processors.
  - *or* integer, >=1

    Number of CPU cores.
- **l1_ratio** (*union type, optional, not for optimizer, default None*) – The Elastic-Net mixing parameter.

  - float, >=0.0, <=1.0
  - *or* None

  See also *constraint-5*.

Notes

constraint-1 : union type
: The newton-cg, sag, and lbfgs solvers support only l2 or no penalties.
  - solver : negated type *of* 'newton-cg', 'newton-cholesky', 'sag', *or* 'lbfgs'
  - *or* penalty : 'l2', 'none', *or* None

constraint-2 : union type
: The dual formulation is only implemented for l2 penalty with the liblinear solver.
  - dual : False
  - *or* dict
    - penalty : 'l2'
    - solver : 'liblinear'

constraint-3 : union type
: The multi_class multinomial option is unavailable when the solver is liblinear or newton-cholesky.
  - multi_class : negated type *of* 'multinomial'
  - *or* solver : negated type *of* 'liblinear' *or* 'newton-cholesky'

constraint-4 : union type, not for optimizer
: penalty='none' is not supported for the liblinear solver
  - solver : negated type *of* 'liblinear'
  - *or* penalty : negated type *of* 'none' *or* None

constraint-5 : union type, not for optimizer
: When penalty is elasticnet, l1_ratio must be between 0 and 1.
  - penalty : negated type *of* 'elasticnet'
  - *or* l1_ratio : float, >0.0, <=1.0

constraint-6 : union type, not for optimizer
: Only 'saga' solver supports elasticnet penalty
  - penalty : negated type *of* 'elasticnet'
  - *or* solver : 'saga'

decision_function(*X*)

Confidence scores for all classes.

Note: The decision_function method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (array *of* items : array *of* items : float) – Features; the outer array is over samples.

Returns

result – Confidence scores for samples for each class in the model.

- array *of* items : array *of* items : float
  In the multi-way case, score per (sample, class) combination.
- *or* array *of* items : float
  In the binary case, score for *self._classes[1]*.

Return type

union type

fit(*X*, *y=None*, ***fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

- **X** (array *of* items : array *of* items : float) – Features; the outer array is over samples.
- **y** (*union type*) – Target class labels; the array is over samples.
  - array *of* items : float
  - *or* array *of* items : string
  - *or* array *of* items : boolean

`predict`(*X*, ***predict_params*)

Make predictions.

Note: The predict method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (array *of* items : array *of* items : float) – Features; the outer array is over samples.

Returns

result – Predicted class label per sample.

- array *of* items : float
- *or* array *of* items : string
- *or* array *of* items : boolean

Return type

union type

`predict_proba`(*X*)

Probability estimates for all classes.

Note: The predict_proba method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (array *of* items : array *of* items : float) – Features; the outer array is over samples.

Returns

result – Probability of the sample for each class in the model.

Return type

array *of* items : array *of* items : float

lale.lib.sklearn.min_max_scaler module

class `lale.lib.sklearn.min_max_scaler.MinMaxScaler`(**, feature_range='(0, 1)', copy=True, clip=False*)

Bases: `PlannedIndividualOp`

Min-max scaler transformer from scikit-learn.

This documentation is auto-generated from JSON schemas.

Parameters

- **feature_range** (*tuple, >=2 items, <=2 items, not for optimizer, default (0, 1)*) – Desired range of transformed data.
  - item 0 : float, >=-1 for optimizer, <=0 for optimizer
  - item 1 : float, >=0.001 for optimizer, <=1 for optimizer
- **copy** (*boolean, not for optimizer, default True*) – Set to False to perform inplace row normalization and avoid a copy (if the input is already a numpy array).
- **clip** (*boolean, optional, not for optimizer, default False*) – Set to True to clip transformed values of held-out data to provided feature range.

Notes

constraint-1 : negated type *of* 'X/isSparse'
MinMaxScaler does not support sparse input. Consider using MaxAbsScaler instead.

fit(*X*, *y=None*, ***fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

- **X** (array *of* items : array *of* items : float) – Features; the outer array is over samples.
- **y** (*any type, optional*) –

partial_fit(*X*, *y=None*, ***fit_params*)

Incremental fit to train train the operator on a batch of samples.

Note: The partial_fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

transform(*X*, *y=None*)

Transform the data.

Note: The transform method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters
X (array *of* items : array *of* items : float) – Features; the outer array is over samples.

Returns
result – Output data schema for transformed data.

Return type
array *of* items : array *of* items : float

lale.lib.sklearn.missing_indicator module

class lale.lib.sklearn.missing_indicator.**MissingIndicator**(**, missing_values=nan, features='missing-only', sparse='auto', error_on_new=True*)

Bases: *PlannedIndividualOp*

Missing values indicator transformer from scikit-learn.

This documentation is auto-generated from JSON schemas.

Parameters

- **missing_values** (*union type, not for optimizer, default nan*) – The placeholder for the missing values.
  - float
  - *or* string
  - *or* nan
  - *or* None

  See also *constraint-2*.
- **features** ('missing-only' *or* 'all', not for optimizer, default 'missing-only') – Whether the imputer mask should represent all or a subset of features.

  See also *constraint-1*.

- **sparse** (*union type, not for optimizer, default 'auto'*) – Whether the imputer mask format should be sparse or dense.
  - boolean
  - *or* 'auto'
- **error_on_new** (*boolean, not for optimizer, default True*) – If True (default), transform will raise an error when there are

  See also *constraint-1*.

Notes

constraint-1 : union type
  error_on_new, only when features="missing-only"
  - error_on_new : True
  - *or* features : 'missing-only'

constraint-2 : union type
  Sparse input with missing_values=0 is not supported. Provide a dense array instead.
  - negated type *of* 'X/isSparse'
  - *or* missing_values : negated type *of* 0

fit(*X*, *y=None*, ***fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

X (array *of* items : array *of* items : float) – Input data, where `n_samples` is the number of samples and

transform(*X*, *y=None*)

Transform the data.

Note: The transform method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (array *of* items : array *of* items : float) – The input data to complete.

Returns

result – The missing indicator for input data.

Return type

array *of* items : array *of* items : boolean

lale.lib.sklearn.mlp_classifier module

class lale.lib.sklearn.mlp_classifier.**MLPClassifier**(**, hidden_layer_sizes=['100'], activation='relu', solver='adam', alpha=0.0001, batch_size='auto', learning_rate='constant', learning_rate_init=0.001, power_t=0.5, max_iter=200, shuffle=True, random_state=None, tol=0.0001, verbose=False, warm_start=False, momentum=0.9, nesterovs_momentum=True, early_stopping=False, validation_fraction=0.1, beta_1=0.9, beta_2=0.999, epsilon=1e-08, n_iter_no_change=10, max_fun=15000*)

Bases: *PlannedIndividualOp*

Multi-layer perceptron dense deep neural network from scikit-learn for classification.

This documentation is auto-generated from JSON schemas.

Parameters

- **hidden_layer_sizes** (tuple, >=1 items for optimizer, <=20 items for optimizer, default [100] *of* items : integer, >=1 for optimizer, <=500 for optimizer) – The ith element represents the number of neurons in the ith hidden layer.
- **activation** ('identity', 'logistic', 'tanh', *or* 'relu', default 'relu') – Activation function for the hidden layer.
- **solver** ('lbfgs', 'sgd', *or* 'adam', default 'adam') – The solver for weight optimization.
- **alpha** (*float, >=1e-10 for optimizer, <=1 for optimizer, loguniform distribution, default 0.0001*) – L2 penalty (regularization term) parameter.
- **batch_size** (*union type, default 'auto'*) – Size of minibatches for stochastic optimizers.
  - integer, >=3 for optimizer, <=128 for optimizer, uniform distribution
    Size of minibatches
  - *or* 'auto'
    Automatic selection, batch_size=min(200, n_samples)
- **learning_rate** ('constant', 'invscaling', *or* 'adaptive', default 'constant') – Learning rate schedule for weight updates.
- **learning_rate_init** (*float, >0, <=0.1 for optimizer, not for optimizer, default 0.001*) – The initial learning rate used. It controls the step-size in updating the weights.
- **power_t** (*float, >=0.01 for optimizer, <=10 for optimizer, not for optimizer, default 0.5*) – The exponent for inverse scaling learning rate.
- **max_iter** (*integer, >=1, >=10 for optimizer, <=1000 for optimizer, uniform distribution, not for optimizer, default 200*) – Maximum number of iterations. The solver iterates until convergence (determined by "tol") or this number of iterations.
- **shuffle** (*boolean, not for optimizer, default True*) – Whether to shuffle samples in each iteration.
- **random_state** (*union type, not for optimizer, default None*) – Random generator selection
  - integer
    seed used by the random number generators
  - *or* numpy.random.RandomState
    Random number generator
  - *or* None
    RandomState instance used by np.random
- **tol** (*float, >=1e-08 for optimizer, <=0.01 for optimizer, default 0.0001*) – Tolerance for the optimization. When the loss or score is not improving by at least tol for n_iter_no_change consecutive iterations, unless learning_rate is set to "adaptive", convergence is considered to be reached and training stops.
- **verbose** (*boolean, not for optimizer, default False*) – Whether to print progress messages to stdout.
- **warm_start** (*boolean, not for optimizer, default False*) – When set to True, reuse the solution of the previous call to fit as initialization, otherwise, just erase the previous solution.
- **momentum** (*float, >=0, <=1, default 0.9*) – Momentum for gradient descent update.
- **nesterovs_momentum** (*boolean, default True*) – Whether to use Nesterov's momentum.

- **early_stopping** (*boolean, default False*) – Whether to use early stopping to terminate training when validation score is not improving. If set to true, it will automatically set aside 10% of training data as validation and terminate training when validation score is not improving by at least tol for n_iter_no_change consecutive epochs.
- **validation_fraction** (*float, >=0.0, <=1.0, default 0.1*) – The proportion of training data to set aside as validation set for early stopping.
- **beta_1** (*float, >=0.0, <1.0, default 0.9*) – Exponential decay rate for estimates of first moment vector in adam.
- **beta_2** (*float, >=0.0, <1.0, default 0.999*) – Exponential decay rate for estimates of second moment vector in adam.
- **epsilon** (*float, >=1e-08 for optimizer, <=1.35 for optimizer, loguniform distribution, default 1e-08*) – Value for numerical stability in adam.
- **n_iter_no_change** (*integer, >=1, <=50 for optimizer, not for optimizer, default 10*) – Maximum number of epochs to not meet tol improvement.
- **max_fun** (*integer, >=0, optional, not for optimizer, default 15000*) – Maximum number of loss function calls.

fit(*X*, *y=None*, ***fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

- **X** (array *of* items : array *of* items : float) –
- **y** (*union type*) –
  - array *of* items : string
  - *or* array *of* items : float
  - *or* array *of* items : boolean

partial_fit(*X*, *y=None*, ***fit_params*)

Incremental fit to train train the operator on a batch of samples.

Note: The partial_fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

- **X** (array *of* items : array *of* items : float) –
- **y** (*union type*) –
  - array *of* items : string
  - *or* array *of* items : float
  - *or* array *of* items : boolean
- **classes** (*union type, optional*) –
  - array *of* items : string
  - *or* array *of* items : float
  - *or* array *of* items : boolean

predict(*X*, ***predict_params*)

Make predictions.

Note: The predict method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (array *of* items : array *of* items : float) –

Returns

result –
- array *of* items : string
- *or* array *of* items : float
- *or* array *of* items : boolean

Return type
union type

`predict_proba`(*X*)

Probability estimates for all classes.

Note: The predict_proba method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters
X (array *of* items : array *of* items : float) –

Returns
result

Return type
array *of* items : array *of* items : float

lale.lib.sklearn.multi_output_regressor module

class `lale.lib.sklearn.multi_output_regressor.MultiOutputRegressor`(*, *estimator=None*, *n_jobs=None*)

Bases: *PlannedIndividualOp*

Multi-output regressor from scikit-learn for multi target regression.

This documentation is auto-generated from JSON schemas.

Parameters
- **estimator** (*union type, not for optimizer, default None*) – An estimator object implementing *fit* and *predict*.
  - operator
  - *or* None
- **n_jobs** (*union type, not for optimizer, default None*) – The number of jobs to run in parallel for *fit*, *predict*, and *partial_fit* (if supported by the passed estimator).
  - None
    1 unless in joblib.parallel_backend context.
  - *or* -1
    Use all processors.
  - *or* integer, >=1
    Number of CPU cores.

`fit`(*X*, *y=None*, ***fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters
- **X** (array *of* items : array *of* items : float) –
- **y** (array *of* items : array *of* items : float) – The target values (real numbers).
- **sample_weight** (*union type, optional, default None*) – Sample weights. If None, then samples are equally weighted. Only supported if the underlying regressor supports sample weights.
  - array *of* items : float

– *or* None

partial_fit(*X*, *y=None*, ***fit_params*)

Incremental fit to train train the operator on a batch of samples.

Note: The partial_fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

predict(*X*, ***predict_params*)

Make predictions.

Note: The predict method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (array, optional *of* items : array *of* items : float) –

Returns

result – The predicted regression values.

Return type

array *of* items : array *of* items : float

lale.lib.sklearn.multinomial_nb module

class lale.lib.sklearn.multinomial_nb.**MultinomialNB**(***, *alpha=1.0*, *fit_prior=True*, *class_prior=None*, *force_alpha=True*)

Bases: *PlannedIndividualOp*

Multinomial Naive Bayes classifier from scikit-learn.

This documentation is auto-generated from JSON schemas.

Parameters

- **alpha** (*float, >=1e-10 for optimizer, <=1.0 for optimizer, loguniform distribution, default 1.0*) – Additive (Laplace/Lidstone) smoothing parameter (0 for no smoothing).
- **fit_prior** (*boolean, default True*) – Whether to learn class prior probabilities or not.
- **class_prior** (*union type, optional, not for optimizer, default None*) – Prior probabilities of the classes. If specified the priors are not adjusted according to the data.
  - array *of* items : float
  - *or* None
- **force_alpha** (*boolean, optional, not for optimizer, default True*) – If False and alpha is less than 1e-10, it will set alpha to 1e-10. If True, alpha will remain unchanged. This may cause numerical errors if alpha is too close to 0.

fit(*X*, *y=None*, ***fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

- **X** (array *of* items : array *of* items : float) –
- **y** (*union type*) –
  - array *of* items : string
  - *or* array *of* items : float
  - *or* array *of* items : boolean

- **sample_weight** (*union type, optional, default None*) – Weights applied to individual samples.
  - array *of* items : float
  - *or* None
    Uniform weights.

partial_fit(*X*, *y=None*, ***fit_params*)

Incremental fit to train train the operator on a batch of samples.

Note: The partial_fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

- **X** (array *of* items : array *of* items : float) –
- **y** (*union type*) –
  - array *of* items : string
  - *or* array *of* items : float
  - *or* array *of* items : boolean
- **classes** (*union type, optional*) –
  - array *of* items : string
  - *or* array *of* items : float
  - *or* array *of* items : boolean
- **sample_weight** (*union type, optional, default None*) – Weights applied to individual samples.
  - array *of* items : float
  - *or* None
    Uniform weights.

predict(*X*, ***predict_params*)

Make predictions.

Note: The predict method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (array *of* items : array *of* items : float) –

Returns

result –
- array *of* items : string
- *or* array *of* items : float
- *or* array *of* items : boolean

Return type

union type

predict_proba(*X*)

Probability estimates for all classes.

Note: The predict_proba method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (array *of* items : array *of* items : float) –

Returns

result – Returns the probability of the samples for each class in the model. The columns correspond to the classes in sorted order, as they appear in the attribute *classes_*.

Return type

array *of* items : array *of* items : float

lale.lib.sklearn.nmf module

class lale.lib.sklearn.nmf.**NMF**(*, *n_components=None*, *init=None*, *solver='cd'*, *beta_loss='frobenius'*, *tol=0.0001*, *max_iter=200*, *random_state=None*, *l1_ratio=0.0*, *verbose=0*, *shuffle=False*, *alpha_W=0.0*, *alpha_H='same'*)

Bases: `PlannedIndividualOp`

Non-negative matrix factorization transformer from scikit-learn for linear dimensionality reduction.

This documentation is auto-generated from JSON schemas.

Parameters

- **n_components** (*union type, default None*) – Number of components.
  - integer, >=1, >=2 for optimizer, <='X/items/maxItems', <=256 for optimizer, uniform distribution
  - *or* 'auto'

    The number of components is automatically inferred from W or H shapes.
  - *or* None

    If not set, keep all components.
- **init** ('custom', 'nndsvd', 'nndsvda', 'nndsvdar', 'random', *or* None, not for optimizer, default None) – Method used to initialize the procedure.
- **solver** ('cd' *or* 'mu', not for optimizer, default 'cd') – Numerical solver to use:

  See also *constraint-1*.
- **beta_loss** (*union type, not for optimizer, default 'frobenius'*) – Beta divergence to be minimized, measuring the distance between X and the dot product WH.
  - float, >=-1 for optimizer, <=1 for optimizer
  - *or* 'frobenius', 'kullback-leibler', *or* 'itakura-saito'

  See also *constraint-1*.
- **tol** (*float, >=0.0, >=1e-08 for optimizer, <=0.01 for optimizer, default 0.0001*) – Tolerance of the stopping condition.
- **max_iter** (*integer, >=1, >=10 for optimizer, <=1000 for optimizer, uniform distribution, default 200*) – Maximum number of iterations before timing out.
- **random_state** (*union type, not for optimizer, default None*) – Used for initialization and in coordinate descent.
  - integer
  - *or* numpy.random.RandomState
  - *or* None
- **l1_ratio** (*float, >=0.0, <=1.0, not for optimizer, default 0.0*) – The regularization mixing parameter.
- **verbose** (*union type, not for optimizer, default 0*) – Whether to be verbose.
  - boolean
  - *or* integer
- **shuffle** (*boolean, default False*) – If true, randomize the order of coordinates in the CD solver.
- **alpha_W** (*float, >=1e-10 for optimizer, <=1.0 for optimizer, loguniform distribution, optional, not for optimizer, default 0.0*) – Constant that multiplies the regularization terms of W. Set it to zero (default) to have no regularization on W.
- **alpha_H** (*union type, optional, not for optimizer, default 'same'*) – Constant that multiplies the regularization terms of H. Set it to zero to have no regularization on H. If "same" (default), it takes the same value as alpha_W.

– 'same'
– *or* float, >=1e-10 for optimizer, <=1.0 for optimizer, loguniform distribution

Notes

constraint-1 : union type
beta_loss, only in 'mu' solver
- beta_loss : 'frobenius'
- *or* solver : 'mu'

fit(*X*, *y=None*, ***fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters
- **X** (array *of* items : array *of* items : float, >=0.0) –
- **y** (*Any, optional*) –

transform(*X*, *y=None*)

Transform the data.

Note: The transform method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters
X (array *of* items : array *of* items : float, >=0.0) –

Returns
result – Transformed data

Return type
array *of* items : array *of* items : float

lale.lib.sklearn.normalizer module

class lale.lib.sklearn.normalizer.**Normalizer**(***, *norm='l2'*, *copy=True*)

Bases: *PlannedIndividualOp*

Normalizer transformer from scikit-learn.

This documentation is auto-generated from JSON schemas.

Parameters
- **norm** ('l1', 'l2', *or* 'max', default 'l2') – The norm to use to normalize each non zero sample.
- **copy** (*boolean, optional, not for optimizer, default True*) – Set to False to perform inplace row normalization and avoid a copy.

fit(*X*, *y=None*, ***fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters
- **X** (array *of* items : array *of* items : float) – Features; the outer array is over samples.
- **y** (*any type, optional*) – Target class labels; the array is over samples.

transform(*X*, *y=None*)

Transform the data.

Note: The transform method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

- **X** (array *of* items : array *of* items : float) – The data to normalize, row by row. scipy.sparse matrices should be
- **copy** (*union type, optional, default None*) – Copy the input X or not.
  - boolean
  - *or* None

Returns

result – Scale each non zero row of X to unit norm

Return type

array *of* items : array *of* items : float

lale.lib.sklearn.nystroem module

class lale.lib.sklearn.nystroem.**Nystroem**(**, kernel='rbf', gamma=None, coef0=None, degree=None, kernel_params=None, n_components=100, random_state=None, n_jobs=None*)

Bases: *PlannedIndividualOp*

Nystroem transformer from scikit-learn.

This documentation is auto-generated from JSON schemas.

Parameters

- **kernel** (*union type, default 'rbf'*) – Kernel map to be approximated.
  - 'additive_chi2', 'chi2', 'cosine', 'linear', 'poly', 'polynomial', 'rbf', 'laplacian', *or* 'sigmoid'

    keys of sklearn.metrics.pairwise.KERNEL_PARAMS
  - *or* callable, not for optimizer
- **gamma** (*union type, default None*) – Gamma parameter.
  - None
  - *or* float, >=3.0517578125e-05 for optimizer, <=8 for optimizer, loguniform distribution
- **coef0** (*union type, default None*) – Zero coefficient.
  - None
  - *or* float, >=-1, <=1 for optimizer, uniform distribution
- **degree** (*union type, default None*) – Degree of the polynomial kernel.
  - None
  - *or* integer, >=2 for optimizer, <=5 for optimizer
- **kernel_params** (*union type, optional, not for optimizer, default None*) – Additional parameters (keyword arguments) for kernel function passed as callable object.
  - dict
  - *or* None
- **n_components** (*integer, >=1, >=10 for optimizer, <=256 for optimizer, uniform distribution, default 100*) – Number of features to construct. How many data points will be used to construct the mapping.
- **random_state** (*union type, not for optimizer, default None*) – Seed of pseudo-random number generator.
  - integer

- *or* numpy.random.RandomState
- *or* None

- **n_jobs** (*union type, optional, not for optimizer, default None*) – The number of jobs to use for the computation.
  - None
    1 unless in joblib.parallel_backend context.
  - *or* -1
    Use all processors.
  - *or* integer, >=1
    Number of CPU cores.

fit(*X*, *y=None*, ***fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters
- **X** (array *of* items : array *of* items : float) – Features; the outer array is over samples.
- **y** (*any type, optional*) – Target class labels; the array is over samples.

transform(*X*, *y=None*)

Transform the data.

Note: The transform method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters
X (array *of* items : array *of* items : float) – Features; the outer array is over samples.

Returns
result – Output data schema for predictions (projected data) using the Nystroem model from scikit-learn.

Return type
array *of* items : array *of* items : float

lale.lib.sklearn.one_hot_encoder module

class lale.lib.sklearn.one_hot_encoder.**OneHotEncoder**(**, categories='auto', dtype='float64', handle_unknown='error', drop=None, sparse_output=True, feature_name_combiner='concat'*)

Bases: *PlannedIndividualOp*

One-hot encoder transformer from scikit-learn that encodes categorical features as numbers.

This documentation is auto-generated from JSON schemas.

Parameters
- **categories** (*union type, not for optimizer, default 'auto'*) –
  - ‘auto’ *or* None
    Determine categories automatically from training data.
  - *or* array
    The ith list element holds the categories expected in the ith column.
    * items : union type
      · array *of* items : string
      · *or* array *of* items : float
        Should be sorted.

- **dtype** (*Any, not for optimizer, default 'float64'*) – Desired dtype of output, must be number. See https://docs.scipy.org/doc/numpy-1.14.0/reference/arrays.scalars.html#arrays-scalars-built-in
- **handle_unknown** (*union type, not for optimizer, default 'error'*) – Specifies the way unknown categories are handled during transform.
  - 'error'
    Raise an error if an unknown category is present during transform.
  - *or* 'ignore'
    When an unknown category is encountered during transform, the resulting one-hot encoded columns for this feature will be all zeros. In the inverse transform, an unknown category will be denoted as None.
  - *or* 'infrequent_if_exist'
    When an unknown category is encountered during transform, the resulting one-hot encoded columns for this feature will map to the infrequent category if it exists. The infrequent category will be mapped to the last position in the encoding. During inverse transform, an unknown category will be mapped to the category denoted `'infrequent'` if it exists. If the `'infrequent'` category does not exist, then transform and inverse_transform will handle an unknown category as with `handle_unknown='ignore'`. Infrequent categories exist based on `min_frequency` and `max_categories`. Read more in the User Guide.
- **drop** (*union type, optional, not for optimizer, default None*) – Specifies a methodology to use to drop one of the categories per feature.
  - 'first' *or* 'if_binary'
  - *or* array, not for optimizer *of* items : float
  - *or* None
- **sparse_output** (*boolean, optional, not for optimizer, default True*) – Will return sparse matrix if set true, else will return an array.
- **feature_name_combiner** (*union type, optional, not for optimizer, default 'concat'*) – Used to create feature names to be returned by get_feature_names_out.
  - 'concat'
    concatenates encoded feature name and category with feature + "_" + str(category).E.g. feature X with values 1, 6, 7 create feature names X_1, X_6, X_7.
  - *or* callable, not for optimizer
    Callable with signature def callable(input_feature, category) that returns a string

fit(*X*, *y=None*, ***fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

- **X** (*array*) – Features; the outer array is over samples.
  - items : array
    - items : union type
      - float
      - *or* string
- **y** (*any type, optional*) – Target class labels; the array is over samples.

transform(*X*, *y=None*)

Transform the data.

Note: The transform method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (*array*) – Features; the outer array is over samples.

- items : array
  - items : union type
    - float
    - *or* string

Returns

result – One-hot codes.

Return type

array *of* items : array *of* items : float

lale.lib.sklearn.ordinal_encoder module

class lale.lib.sklearn.ordinal_encoder.**OrdinalEncoder**(**, categories='auto', dtype='float64', handle_unknown='error', encode_unknown_with='auto', unknown_value=None, encoded_missing_value=nan, max_categories=None, min_frequency=None*)

Bases: *PlannedIndividualOp*

Ordinal encoder transformer from scikit-learn that encodes categorical features as numbers.

This documentation is auto-generated from JSON schemas.

Parameters

- **categories** (*union type, not for optimizer, default 'auto'*) –
  - 'auto' *or* None

    Determine categories automatically from training data.
  - *or* array

    The ith list element holds the categories expected in the ith column.
    - items : union type
      - array *of* items : string
      - *or* array *of* items : float

        Should be sorted.
- **dtype** (*Any, not for optimizer, default 'float64'*) – Desired dtype of output, must be number. See https://docs.scipy.org/doc/numpy-1.14.0/reference/arrays.scalars.html#arrays-scalars-built-in
- **handle_unknown** ('error', 'ignore', *or* 'use_encoded_value', optional, not for optimizer, default 'error') –

  When set to 'error' an error will be raised in case an unknown categorical feature is present during transform.

  When set to 'use_encoded_value', the encoded value of unknown categories will be set to the value given for the parameter unknown_value. In inverse_transform, an unknown category will be denoted as None. When this parameter is set to *ignore* and an unknown category is encountered during transform, the resulting encoding with be set to the value indicated by *encode_unknown_with* (this functionality is added by lale).

  See also *constraint-1*, *constraint-2*.

- **encode_unknown_with** (*union type, optional, not for optimizer, default 'auto'*) – When an unknown categorical feature value is found during transform, and 'handle_unknown' is set to 'ignore', that value is encoded with this value. Default of 'auto' sets it to an integer equal to n+1, where n is the maximum encoding value based on known categories.
  - integer
  - *or* 'auto'
- **unknown_value** (*union type, optional, not for optimizer, default None*) –

  When the parameter handle_unknown is set to 'use_encoded_value', this parameter is required and will set the encoded value of unknown categories.

  It has to be distinct from the values used to encode any of the categories in fit.
  - integer
  - *or* nan
  - *or* None

  See also *constraint-1*, *constraint-1*, *constraint-2*.
- **encoded_missing_value** (*union type, optional, not for optimizer, default nan*) – Encoded value of missing categories. If set to `np.nan`, then the `dtype` parameter must be a float dtype.
  - integer
  - *or* nan
  - *or* None
- **max_categories** (*union type, optional, not for optimizer, default None*) – Specifies an upper limit to the number of output categories for each input feature when considering infrequent categories. If there are infrequent categories, max_categories includes the category representing the infrequent categories along with the frequent categories. If None, there is no limit to the number of output features.

  max_categories do not take into account missing or unknown categories. Setting unknown_value or encoded_missing_value to an integer will increase the number of unique integer codes by one each. This can result in up to max_categories + 2 integer codes.
  - integer, >1
  - *or* None
- **min_frequency** (*union type, optional, not for optimizer, default None*) – Specifies the minimum frequency below which a category will be considered infrequent.
  - integer, >=1

    Categories with a smaller cardinality will be considered infrequent.
  - *or* float, >=0.0, <=1.0

    Categories with a smaller cardinality than min_frequency * n_samples will be considered infrequent.
  - *or* None

Notes

constraint-1 : union type

unknown_value should be an integer or np.nan when handle_unknown is 'use_encoded_value'.
- handle_unknown : negated type *of* 'use_encoded_value'
- *or* unknown_value : nan
- *or* unknown_value : integer

constraint-2 : union type

unknown_value should only be set when handle_unknown is 'use_encoded_value'.
- handle_unknown : 'use_encoded_value'

- *or* unknown_value : None

fit(*X*, *y=None*, ***fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

- **X** (*array*) – Features; the outer array is over samples.
  - items : union type
    - array *of* items : float
    - *or* array *of* items : string
- **y** (*any type, optional*) – Target class labels; the array is over samples.

transform(*X*, *y=None*)

Transform the data.

Note: The transform method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (*array*) – Features; the outer array is over samples.

- items : union type
  - array *of* items : float
  - *or* array *of* items : string

Returns

result – Ordinal codes.

Return type

array *of* items : array *of* items : float

lale.lib.sklearn.passive_aggressive_classifier module

class lale.lib.sklearn.passive_aggressive_classifier.**PassiveAggressiveClassifier**(**, C=1.0, fit_intercept=False, max_iter=1000, tol=None, early_stopping=False, validation_fraction=0.1, n_iter_no_change=5, shuffle=True, verbose=0, loss='hinge', n_jobs=None, random_state=None, warm_start=False, class_weight=None, average=False*)

Bases: *PlannedIndividualOp*

Passive aggressive classifier from scikit-learn.

This documentation is auto-generated from JSON schemas.

Parameters

- **C** (*float, >=1e-05 for optimizer, <=10 for optimizer, loguniform distribution, default 1.0*) – Maximum step size (regularization). Defaults to 1.0.
- **fit_intercept** (*boolean, default False*) – Whether the intercept should be estimated or not. If False, thethe data is assumed to be already centered.
- **max_iter** (*integer, >=5 for optimizer, <=1000 for optimizer, uniform distribution, default 1000*) – The maximum number of passes over the training data (aka epochs).
- **tol** (*union type, default None*) – The stopping criterion. If it is not None, the iterations will stop
  - float, >=1e-08 for optimizer, <=0.01 for optimizer
  - *or* None
- **early_stopping** (*boolean, default False*) – Whether to use early stopping to terminate training when validation.
- **validation_fraction** (*float, >=0, <=1, optional, not for optimizer, default 0.1*) – The proportion of training data to set aside as validation set for early stopping.
- **n_iter_no_change** (*integer, >=5 for optimizer, <=10 for optimizer, optional, not for optimizer, default 5*) – Number of iterations with no improvement to wait before early stopping.
- **shuffle** (*boolean, default True*) – Whether or not the training data should be shuffled after each epoch.
- **verbose** (*union type, optional, not for optimizer, default 0*) – The verbosity level
  - integer
  - *or* None
- **loss** ('hinge' *or* 'squared_hinge', default 'hinge') – The loss function to be used:
- **n_jobs** (*union type, optional, not for optimizer, default None*) – The number of CPUs to use to do the OVA (One Versus All, for
  - integer
  - *or* None
- **random_state** (*union type, optional, not for optimizer, default None*) – The seed of the pseudo random number generator to use when shuffling
  - integer
  - *or* numpy.random.RandomState
  - *or* None
- **warm_start** (*boolean, optional, not for optimizer, default False*) – When set to True, reuse the solution of the previous call to fit as initialization, otherwise, just erase the previous solution.
- **class_weight** (*union type, optional, not for optimizer, default None*) – Preset for the class_weight fit parameter.
  - dict
  - *or* 'balanced' *or* None
- **average** (*union type, default False*) – When set to True, computes the averaged SGD weights and stores the result in the `coef_` attribute.
  - boolean
  - *or* integer, not for optimizer

decision_function(*X*)

Confidence scores for all classes.

Note: The decision_function method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters
: **X** (array *of* items : array *of* items : float) –

Returns
: **result** – Confidence scores for samples for each class in the model.

  - array *of* items : array *of* items : float
    In the multi-way case, score per (sample, class) combination.
  - *or* array *of* items : float
    In the binary case, score for *self._classes[1]*.

Return type
: union type

`fit`(*X*, *y=None*, ***fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

- **X** (array *of* items : array *of* items : float) –
- **y** (*union type*) –
  - array *of* items : string
  - *or* array *of* items : float
  - *or* array *of* items : boolean
- **coef_init** (array, optional *of* items : array *of* items : float) – The initial coefficients to warm-start the optimization.
- **intercept_init** (array, optional *of* items : float) – The initial intercept to warm-start the optimization.

`partial_fit`(*X*, *y=None*, ***fit_params*)

Incremental fit to train train the operator on a batch of samples.

Note: The partial_fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

- **X** (array *of* items : array *of* items : float) –
- **y** (*union type*) –
  - array *of* items : string
  - *or* array *of* items : float
  - *or* array *of* items : boolean
- **classes** (*union type, optional*) –
  - array *of* items : string
  - *or* array *of* items : float
  - *or* array *of* items : boolean

`predict`(*X*, ***predict_params*)

Make predictions.

Note: The predict method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters
: **X** (array *of* items : array *of* items : float) –

Returns
: **result** –

  - array *of* items : string
  - *or* array *of* items : float

- *or* array *of* items : boolean

Return type
union type

lale.lib.sklearn.pca module

class lale.lib.sklearn.pca.**PCA**(*, *n_components=None*, *copy=True*, *whiten=False*, *svd_solver='auto'*, *tol=0.0*, *iterated_power='auto'*, *random_state=None*, *n_oversamples=10*, *power_iteration_normalizer='auto'*)

Bases: `PlannedIndividualOp`

Principal component analysis transformer from scikit-learn for linear dimensionality reduction.

This documentation is auto-generated from JSON schemas.

Parameters

- **n_components** (*union type, default None*) –
  - None
    If not set, keep all components.
  - *or* 'mle'
    Use Minka's MLE to guess the dimension.
  - *or* float, >0.0, <1.0
    Select the number of components such that the amount of variance that needs to be explained is greater than the specified percentage.
  - *or* integer, >=1, <='X/items/maxItems', not for optimizer
    Number of components to keep.

  See also *constraint-2*, *constraint-3*.
- **copy** (*boolean, not for optimizer, default True*) – If false, overwrite data passed to fit.
- **whiten** (*boolean, default False*) – When true, multiply the components vectors by the square root of n_samples and then divide by the singular values to ensure uncorrelated outputs with unit component-wise variances.
- **svd_solver** ('auto', 'full', 'arpack', *or* 'randomized', default 'auto') – Algorithm to use.

  See also *constraint-2*, *constraint-3*, *constraint-4*.
- **tol** (*float, >=0.0, <=1 for optimizer, not for optimizer, default 0.0*) – Tolerance for singular values computed by svd_solver arpack.
- **iterated_power** (*union type, not for optimizer, default 'auto'*) –
  - integer, >=0, <=10 for optimizer
    Number of iterations for the power method computed by svd_solver randomized.
  - *or* 'auto'
    Pick automatically.

  See also *constraint-4*.
- **random_state** (*union type, not for optimizer, default None*) – Seed of pseudo-random number generator for shuffling data.
  - None
    RandomState used by np.random
  - *or* numpy.random.RandomState
    Use the provided random state, only affecting other users of that same random state instance.
  - *or* integer
    Explicit seed.

- **n_oversamples** (*integer, >=0, <=1000 for optimizer, optional, not for optimizer, default 10*) – This parameter is only relevant when `svd_solver="randomized"`. It corresponds to the additional number of random vectors to sample the range of X so as to ensure proper conditioning. See randomized_svd for more details.
- **power_iteration_normalizer** ('auto', 'QR', 'LU', *or* 'none', optional, not for optimizer, default 'auto') – Power iteration normalizer for randomized SVD solver. Not used by ARPACK. See `randomized_svd` for more details.

Notes

constraint-1 : negated type *of* 'X/isSparse'
    This class does not support sparse input. See TruncatedSVD for an alternative with sparse data.
constraint-2 : union type
    Option n_components mle can only be set for svd_solver full or auto.
    - n_components : negated type *of* 'mle'
    - *or* svd_solver : 'full' *or* 'auto'

constraint-3 : union type
    Setting 0 < n_components < 1 only works for svd_solver full.
    - n_components : negated type *of* float, >0.0, <1.0
    - *or* svd_solver : 'full'

constraint-4 : union type
    Option iterated_power can be set for svd_solver randomized.
    - iterated_power : 'auto'
    - *or* svd_solver : 'randomized'

fit(*X*, *y=None*, ***fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters
- **X** (array *of* items : array *of* items : float) – Features; the outer array is over samples.
- **y** (*Any, optional*) – Target for supervised learning (ignored).

transform(*X*, *y=None*)

Transform the data.

Note: The transform method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters
X (array *of* items : array *of* items : float) – Features; the outer array is over samples.

Returns
result – Features; the outer array is over samples.

Return type
array *of* items : array *of* items : float

lale.lib.sklearn.perceptron module

class lale.lib.sklearn.perceptron.**Perceptron**(**, penalty=None, alpha=0.0001, fit_intercept=True, max_iter=1000, tol=0.001, shuffle=True, verbose=0, eta0=1.0, n_jobs=None, random_state=None, early_stopping=False, validation_fraction=0.1, n_iter_no_change=5, class_weight, warm_start=False*)

Bases: `PlannedIndividualOp`

Perceptron classifier from scikit-learn.

This documentation is auto-generated from JSON schemas.

Parameters

- **penalty** ('l2', 'l1', 'elasticnet', *or* None, not for optimizer, default None) – The penalty (aka regularization term) to be used.
- **alpha** (*float, >=1e-10 for optimizer, <=1.0 for optimizer, loguniform distribution, default 0.0001*) – Constant that multiplies the regularization term if regularization is used.
- **fit_intercept** (*boolean, default True*) – Whether the intercept should be estimated or not. If False, the data is assumed to be already centered.
- **max_iter** (*integer, >=10 for optimizer, <=10000 for optimizer, loguniform distribution, default 1000*) – The maximum number of passes over the training data (aka epochs).
- **tol** (*union type, default 0.001*) – The stopping criterion
  - float, >=1e-08 for optimizer, <=0.01 for optimizer
    If not None, the iterations will stop when (loss > previous_loss - tol).
  - *or* None
- **shuffle** (*boolean, default True*) – Whether or not the training data should be shuffled after each epoch.
- **verbose** (*integer, not for optimizer, default 0*) – The verbosity level.
- **eta0** (*float, >=0.01 for optimizer, <=1.0 for optimizer, loguniform distribution, default 1.0*) – Constant by which the updates are multiplied.
- **n_jobs** (*union type, not for optimizer, default None*) – The number of CPUs to use to do the OVA (One Versus All, for multi-class problems) computation.
  - None
    1 unless in joblib.parallel_backend context.
  - *or* -1
    Use all processors.
  - *or* integer, >=1
    Number of CPU cores.
- **random_state** (*union type, not for optimizer, default None*) – If int, random_state is the seed used by the random number generator;
  - integer
  - *or* numpy.random.RandomState
  - *or* None
- **early_stopping** (*boolean, not for optimizer, default False*) – Whether to use early stopping to terminate training when validation score is not improving.
- **validation_fraction** (*float, >=0, <=1, not for optimizer, default 0.1*) – The proportion of training data to set aside as validation set for early stopping.
- **n_iter_no_change** (*integer, not for optimizer, default 5*) – Number of iterations with no improvement to wait before early stopping.
- **class_weight** (*union type, not for optimizer*) – Weights associated with classes in the form `{class_label: weight}`.
  - dict

  - *or* array *of* items : dict
  - *or* 'balanced' *or* None
- **warm_start** (*boolean, not for optimizer, default False*) – When set to True, reuse the solution of the previous call to fit as initialization, otherwise, just erase the previous solution.

decision_function(*X*)

Confidence scores for all classes.

Note: The decision_function method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (array *of* items : array *of* items : float) –

Returns

result –

- array *of* items : array *of* items : float
  In the multi-way case, score per (sample, class) combination.
- *or* array *of* items : float
  In the binary case, score for *self._classes[1]*.

Return type

union type

fit(*X*, *y=None*, ***fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

- **X** (array *of* items : array *of* items : float) –
- **y** (*union type*) –
  - array *of* items : string
  - *or* array *of* items : float
  - *or* array *of* items : boolean
- **coef_init** (*union type, optional*) – The initial coefficients to warm-start the optimization.
  - array *of* items : array *of* items : float
  - *or* None
- **intercept_init** (*union type, optional*) – The initial intercept to warm-start the optimization.
  - array *of* items : float
  - *or* None
- **sample_weight** (*union type, optional, default None*) – Weights applied to individual samples.
  - array *of* items : float
  - *or* None
    Uniform weights.

partial_fit(*X*, *y=None*, ***fit_params*)

Incremental fit to train train the operator on a batch of samples.

Note: The partial_fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

- **X** (array *of* items : array *of* items : float) –
- **y** (*union type*) –

- array *of* items : string
- *or* array *of* items : float
- *or* array *of* items : boolean
- **classes** (*union type, optional*) –
  - array *of* items : string
  - *or* array *of* items : float
  - *or* array *of* items : boolean
- **sample_weight** (*union type, optional, default None*) – Weights applied to individual samples.
  - array *of* items : float
  - *or* None

    Uniform weights.

predict(*X*, ***predict_params*)

Make predictions.

Note: The predict method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (array *of* items : array *of* items : float) –

Returns

result –
- array *of* items : string
- *or* array *of* items : float
- *or* array *of* items : boolean

Return type

union type

lale.lib.sklearn.pipeline module

class lale.lib.sklearn.pipeline.**Pipeline**(*, *steps*, *memory=None*, *verbose=False*)

Bases: *PlannedIndividualOp*

Pipeline from scikit-learn creates a sequential list of operators.

This documentation is auto-generated from JSON schemas.

Parameters

- **steps** (*array, not for optimizer*) – List of (name, transform) tuples (implementing fit/transform) that are chained, in the order in which they are chained, with the last object an estimator.
  - items : tuple, >=2 items, <=2 items

    Tuple of (name, transform).
    - item 0 : string

      Name.
    - item 1 : union type
      - operator

        Transform.
      - *or* None *or* 'passthrough'

        NoOp
- **memory** (*union type, optional, not for optimizer, default None*) – Used to cache the fitted transformers of the pipeline.
  - string

    Path to the caching directory.

  - *or* dict, not for optimizer
    Object with the joblib.Memory interface
  - *or* None
    No caching.
- **verbose** (*boolean, optional, not for optimizer, default False*) – If True, the time elapsed while fitting each step will be printed as it is completed.

fit(*X*, *y=None*, ***fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters
- **X** (*Any*) – Features.
- **y** (*Any*) – Target for supervised learning.

predict(*X*, ***predict_params*)

Make predictions.

Note: The predict method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters
X (*Any*) – Features.

Returns
result – Predictions.

Return type
Any

predict_proba(*X*)

Probability estimates for all classes.

Note: The predict_proba method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters
X (*Any*) – Features.

Returns
result – Probability of the sample for each class in the model.

Return type
array *of* items : array *of* items : float

transform(*X*, *y=None*)

Transform the data.

Note: The transform method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters
X (*Any*) – Features.

Returns
result – Features.

Return type
Any

lale.lib.sklearn.polynomial_features module

class lale.lib.sklearn.polynomial_features.**PolynomialFeatures**(**, degree=2, interaction_only=False, include_bias=True, order='C'*)

Bases: *PlannedIndividualOp*

Polynomial features transformer from scikit-learn.

This documentation is auto-generated from JSON schemas.

Parameters

- **degree** (*integer, >=2 for optimizer, <=3 for optimizer, optional, default 2*) – The degree of the polynomial features.
- **interaction_only** (*boolean, optional, default False*) – If true, only interaction features are produced: features that are products of at most degree distinct input features (so not x[1] ** 2, x[0] * x[2] ** 3, etc.).
- **include_bias** (*boolean, default True*) – If True (default), then include a bias column, the feature in which all polynomial powers are zero (i.e. a column of ones - acts as an intercept term in a linear model).
- **order** ('C' *or* 'F', optional, not for optimizer, default 'C') – Order of output array in the dense case. 'F' order is faster to compute, but may slow down subsequent estimators.

fit(*X, y=None, **fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

- **X** (array *of* items : array *of* items : float) – The data.
- **y** (*any type, optional*) –

transform(*X, y=None*)

Transform the data.

Note: The transform method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (array *of* items : array *of* items : float) – The data to transform, row by row.

Returns

result – The matrix of features, where NP is the number of polynomial

Return type

array *of* items : array *of* items : float

lale.lib.sklearn.quadratic_discriminant_analysis module

class lale.lib.sklearn.quadratic_discriminant_analysis.**QuadraticDiscriminantAnalysis**(**, priors=None, reg_param=0.0, store_covariance=False, tol=0.0001*)

Bases: *PlannedIndividualOp*

Quadratic discriminant analysis classifier with a quadratic decision boundary from scikit-learn.

This documentation is auto-generated from JSON schemas.

Parameters

- **priors** (*union type, not for optimizer, default None*) – Priors on classes
  - array *of* items : float
  - *or* None
- **reg_param** (*float, >=0.0 for optimizer, <=1.0 for optimizer, uniform distribution, optional, default 0.0*) – Regularizes the covariance estimate as
- **store_covariance** (*boolean, not for optimizer, default False*) – If True the covariance matrices are computed and stored in the
- **tol** (*float, >=1e-08 for optimizer, <=0.01 for optimizer, optional, default 0.0001*) – Threshold used for rank estimation.

decision_function(*X*)

Confidence scores for all classes.

Note: The decision_function method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (array *of* items : array *of* items : float) – Features; the outer array is over samples.

Returns

result – Confidence scores for samples for each class in the model.

- array *of* items : array *of* items : float
  In the multi-way case, score per (sample, class) combination.
- *or* array *of* items : float
  In the binary case, score for *self._classes[1]*.

Return type

union type

fit(*X*, *y=None*, ***fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

- **X** (array *of* items : array *of* items : float) – Training vector, where n_samples is the number of samples and
- **y** (*union type*) – Target values (integers)
  - array *of* items : float
  - *or* array *of* items : string
  - *or* array *of* items : boolean

predict(*X*, ***predict_params*)

Make predictions.

Note: The predict method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (array, optional *of* items : array *of* items : float) –

Returns

result – Perform classification on an array of test vectors X.

- array *of* items : float
- *or* array *of* items : string
- *or* array *of* items : boolean

Return type
union type

predict_proba(*X*)

Probability estimates for all classes.

Note: The predict_proba method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters
X (array, optional *of* items : array *of* items : float) – Array of samples/test vectors.

Returns
result – Posterior probabilities of classification per class.

Return type
array *of* items : array *of* items : float

lale.lib.sklearn.quantile_transformer module

class lale.lib.sklearn.quantile_transformer.**QuantileTransformer**(*, *n_quantiles=1000*, *output_distribution='uniform'*, *ignore_implicit_zeros=False*, *subsample=100000*, *random_state=None*, *copy=True*)

Bases: *PlannedIndividualOp*

Quantile transformer from scikit-learn.

This documentation is auto-generated from JSON schemas.

Parameters

- **n_quantiles** (*integer, >=10 for optimizer, <=2000 for optimizer, uniform distribution, default 1000*) – Number of quantiles to be computed. It corresponds to the number
- **output_distribution** ('normal' *or* 'uniform', default 'uniform') – Marginal distribution for the transformed data. The choices are
- **ignore_implicit_zeros** (*boolean, not for optimizer, default False*) – Only applies to sparse matrices. If True, the sparse entries of the
- **subsample** (*integer, >=1 for optimizer, <=100000 for optimizer, uniform distribution, default 100000*) – Maximum number of samples used to estimate the quantiles for
- **random_state** (*union type, not for optimizer, default None*) – If int, random_state is the seed used by the random number generator;
  - integer
  - *or* numpy.random.RandomState
  - *or* None
- **copy** (*boolean, not for optimizer, default True*) – Set to False to perform inplace transformation and avoid a copy (if the

fit(*X*, *y=None*, ***fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters
X (array *of* items : array *of* items : float) – The data used to scale along the features axis.

If a sparse matrix is provided, it will be converted into a sparse csc_matrix. Additionally, the sparse matrix needs to be nonnegative if ignore_implicit_zeros is False.

transform(*X*, *y=None*)

Transform the data.

Note: The transform method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (array *of* items : array *of* items : float) – The data used to scale along the features axis. If a sparse matrix is provided, it will be converted into a sparse csc_matrix. Additionally, the sparse matrix needs to be nonnegative if ignore_implicit_zeros is False.

Returns

result – The projected data.

Return type

array *of* items : array *of* items : float

lale.lib.sklearn.random_forest_classifier module

class lale.lib.sklearn.random_forest_classifier.**RandomForestClassifier**(*, *n_estimators=100*, *criterion='gini'*, *max_depth=None*, *min_samples_split=2*, *min_samples_leaf=1*, *min_weight_fraction_leaf=0.0*, *max_features=None*, *max_leaf_nodes=None*, *min_impurity_decrease=0.0*, *bootstrap=True*, *oob_score=False*, *n_jobs=None*, *random_state=None*, *verbose=0*, *warm_start=False*, *class_weight=None*, *ccp_alpha=0.0*, *max_samples=None*, *monotonic_cst=None*)

Bases: `PlannedIndividualOp`

Random forest classifier from scikit-learn.

This documentation is auto-generated from JSON schemas.

Parameters

- **n_estimators** (*integer, >=1, >=10 for optimizer, <=100 for optimizer, optional, default 100*) – The number of trees in the forest.
- **criterion** ('gini' *or* 'entropy', optional, default 'gini') – The function to measure the quality of a split.
- **max_depth** (*union type, optional, default None*) – The maximum depth of the tree.
  - integer, >=1, >=3 for optimizer, <=5 for optimizer
  - *or* None

    Nodes are expanded until all leaves are pure or until all leaves contain less than min_samples_split samples.

- **min_samples_split** (*union type, optional, default 2*) – The minimum number of samples required to split an internal node.
  - integer, >=2, >=2 for optimizer, <='X/maxItems', <=5 for optimizer, default 2
    Consider min_samples_split as the minimum number.
  - *or* float, >0.0, >=0.01 for optimizer, <=1.0, <=0.5 for optimizer, default 0.05
    min_samples_split is a fraction and ceil(min_samples_split * n_samples) are the minimum number of samples for each split.
- **min_samples_leaf** (*union type, optional, default 1*) – The minimum number of samples required to be at a leaf node.
  - integer, >=1, >=1 for optimizer, <='X/maxItems', <=5 for optimizer, default 1
    Consider min_samples_leaf as the minimum number.
  - *or* float, >0.0, >=0.01 for optimizer, <=0.5, default 0.05
    min_samples_leaf is a fraction and ceil(min_samples_leaf * n_samples) are the minimum number of samples for each node.
- **min_weight_fraction_leaf** (*float, >=0.0, <=0.5, optional, not for optimizer, default 0.0*) – The minimum weighted fraction of the sum total of weights (of all the input samples) required to be at a leaf node. Samples have equal weight when sample_weight is not provided.
- **max_features** (*union type, optional, default None*) – The number of features to consider when looking for the best split.
  - integer, >=2, <='X/items/maxItems', not for optimizer
    Consider max_features features at each split.
  - *or* float, >0.0, >=0.01 for optimizer, <=1.0, uniform distribution, default 0.5
    max_features is a fraction and int(max_features * n_features) features are considered at each split.
  - *or* 'sqrt', 'log2', *or* None
- **max_leaf_nodes** (*union type, optional, not for optimizer, default None*) – Grow trees with max_leaf_nodes in best-first fashion. Best nodes are defined as relative reduction in impurity.
  - integer, >=1, >=3 for optimizer, <=1000 for optimizer
  - *or* None
    Unlimited number of leaf nodes.
- **min_impurity_decrease** (*float, >=0.0, <=10.0 for optimizer, optional, not for optimizer, default 0.0*) – A node will be split if this split induces a decrease of the impurity greater than or equal to this value.
- **bootstrap** (*boolean, optional, not for optimizer, default True*) – Whether bootstrap samples are used when building trees. If False, the whole datset is used to build each tree.

  See also *constraint-2*.
- **oob_score** (*union type, optional, not for optimizer, default False*) – Whether to use out-of-bag samples to estimate the generalization accuracy.
  - callable, not for optimizer
    A callable with signature metric(y_true, y_pred).
  - *or* boolean

  See also *constraint-2*.
- **n_jobs** (*union type, optional, not for optimizer, default None*) – The number of jobs to run in parallel for both fit and predict.
  - None
    1 unless in joblib.parallel_backend context.
  - *or* -1
    Use all processors.
  - *or* integer, >=1
    Number of CPU cores.

- **random_state** (*union type, optional, not for optimizer, default None*) – Seed of pseudo-random number generator.
  - numpy.random.RandomState
  - *or* None
    RandomState used by np.random
  - *or* integer
    Explicit seed.
- **verbose** (*integer, optional, not for optimizer, default 0*) – Controls the verbosity when fitting and predicting.
- **warm_start** (*boolean, optional, not for optimizer, default False*) – When set to True, reuse the solution of the previous call to fit and add more estimators to the ensemble, otherwise, just fit a whole new forest.
- **class_weight** (*union type, not for optimizer, default None*) – Weights associated with classes in the form `{class_label: weight}`.
  - dict
  - *or* array *of* items : dict
  - *or* 'balanced', 'balanced_subsample', *or* None
- **ccp_alpha** (*float, >=0.0, <=0.1 for optimizer, optional, not for optimizer, default 0.0*) – Complexity parameter used for Minimal Cost-Complexity Pruning. The subtree with the largest cost complexity that is smaller than ccp_alpha will be chosen. By default, no pruning is performed.
- **max_samples** (*union type, optional, not for optimizer, default None*) – If bootstrap is True, the number of samples to draw from X to train each base estimator.
  - None
    Draw X.shape[0] samples.
  - *or* integer, >=1
    Draw max_samples samples.
  - *or* float, >0.0, <1.0
    Draw max_samples * X.shape[0] samples.
- **monotonic_cst** (*union type, optional, not for optimizer, default None*) – Indicates the monotonicity constraint to enforce on each feature. Monotonicity constraints are not supported for: multioutput regressions (i.e. when n_outputs > 1),

  regressions trained on data with missing values.
  - array *of* items : -1, 0, *or* 1
    array-like of int of shape (n_features)
  - *or* None
    No constraints are applied.

Notes

constraint-1 : negated type *of* 'y/isSparse'
    This classifier does not support sparse labels.
constraint-2 : union type
    Out of bag estimation only available if bootstrap=True.
    - bootstrap : True
    - *or* oob_score : False

fit(*X*, *y=None*, ***fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

- **X** (*array*) – The outer array is over samples aka rows.
  - items : array *of* items : float

    The inner array is over features aka columns.
- **y** (*union type*) – The predicted classes.
  - array *of* items : float
  - *or* array *of* items : string
  - *or* array *of* items : boolean
- **sample_weight** (*union type, optional*) – Sample weights.
  - array *of* items : float
  - *or* None

    Samples are equally weighted.

predict(*X*, ***predict_params*)

Make predictions.

Note: The predict method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (*array, optional*) – The outer array is over samples aka rows.

- items : array *of* items : float

  The inner array is over features aka columns.

Returns

result – The predicted classes.

- array *of* items : float
- *or* array *of* items : string
- *or* array *of* items : boolean

Return type

union type

predict_proba(*X*)

Probability estimates for all classes.

Note: The predict_proba method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (*array, optional*) – The outer array is over samples aka rows.

- items : array *of* items : float

  The inner array is over features aka columns.

Returns

result – The outer array is over samples aka rows.

- items : array *of* items : float

  The inner array has items corresponding to each class.

Return type

array

lale.lib.sklearn.random_forest_regressor module

class lale.lib.sklearn.random_forest_regressor.**RandomForestRegressor**(**, n_estimators=100, criterion='squared_error', max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features=None, max_leaf_nodes=None, min_impurity_decrease=0.0, bootstrap=True, oob_score=False, n_jobs=None, random_state=None, verbose=0, warm_start=False, ccp_alpha=0.0, max_samples=None, monotonic_cst=None*)

Bases: *PlannedIndividualOp*

Random forest regressor from scikit-learn.

This documentation is auto-generated from JSON schemas.

Parameters

- **n_estimators** (*integer, >=1, >=10 for optimizer, <=100 for optimizer, default 100*) – The number of trees in the forest.
- **criterion** (*union type, default 'squared_error'*) – The function to measure the quality of a split. Supported criteria are "squared_error" for the mean squared error, which is equal to variance reduction as feature selection criterion, "absolute_error" for the mean absolute error, and "poisson" which uses reduction in Poisson deviance to find splits. Training using "absolute_error" is significantly slower than when using "squared_error".
  - 'squared_error', 'absolute_error', *or* 'poisson'
  - *or* 'mse' *or* 'mae', not for optimizer
- **max_depth** (*union type, default None*) – The maximum depth of the tree.
  - integer, >=1, >=3 for optimizer, <=5 for optimizer
  - *or* None

    Nodes are expanded until all leaves are pure or until all leaves contain less than min_samples_split samples.
- **min_samples_split** (*union type, default 2*) – The minimum number of samples required to split an internal node.
  - integer, >=2, >=2 for optimizer, <='X/maxItems', <=5 for optimizer, not for optimizer

    Consider min_samples_split as the minimum number.
  - *or* float, >0.0, >=0.01 for optimizer, <=1.0, <=0.5 for optimizer, default 0.05

    min_samples_split is a fraction and ceil(min_samples_split * n_samples) are the minimum number of samples for each split.
- **min_samples_leaf** (*union type, default 1*) – The minimum number of samples required to be at a leaf node.
  - integer, >=1, >=1 for optimizer, <='X/maxItems', <=5 for optimizer, not for optimizer

    Consider min_samples_leaf as the minimum number.

- *or* float, >0.0, >=0.01 for optimizer, <=0.5, default 0.05
  min_samples_leaf is a fraction and ceil(min_samples_leaf * n_samples) are the minimum number of samples for each node.
- **min_weight_fraction_leaf** (*float, >=0.0, <=0.5, optional, not for optimizer, default 0.0*) – The minimum weighted fraction of the sum total of weights (of all the input samples) required to be at a leaf node. Samples have equal weight when sample_weight is not provided.
- **max_features** (*union type, default None*) – The number of features to consider when looking for the best split.
  - integer, >=2, <='X/items/maxItems', not for optimizer
    Consider max_features features at each split.
  - *or* float, >0.0, >=0.01 for optimizer, <=1.0, uniform distribution, default 0.5
    max_features is a fraction and int(max_features * n_features) features are considered at each split.
  - *or* 'sqrt', 'log2', *or* None
- **max_leaf_nodes** (*union type, optional, not for optimizer, default None*) – Grow trees with max_leaf_nodes in best-first fashion. Best nodes are defined as relative reduction in impurity.
  - integer, >=1, >=3 for optimizer, <=1000 for optimizer
  - *or* None
    Unlimited number of leaf nodes.
- **min_impurity_decrease** (*float, >=0.0, <=10.0 for optimizer, optional, not for optimizer, default 0.0*) – A node will be split if this split induces a decrease of the impurity greater than or equal to this value.
- **bootstrap** (*boolean, default True*) – Whether bootstrap samples are used when building trees. If False, the whole datset is used to build each tree.

  See also *constraint-2*.
- **oob_score** (*union type, optional, not for optimizer, default False*) – Whether to use out-of-bag samples to estimate the generalization accuracy.
  - callable, not for optimizer
    A callable with signature metric(y_true, y_pred).
  - *or* boolean

  See also *constraint-2*.
- **n_jobs** (*union type, optional, not for optimizer, default None*) – The number of jobs to run in parallel for both fit and predict.
  - None
    1 unless in joblib.parallel_backend context.
  - *or* -1
    Use all processors.
  - *or* integer, >=1
    Number of CPU cores.
- **random_state** (*union type, optional, not for optimizer, default None*) – Seed of pseudo-random number generator.
  - numpy.random.RandomState
  - *or* None
    RandomState used by np.random
  - *or* integer
    Explicit seed.
- **verbose** (*integer, optional, not for optimizer, default 0*) – Controls the verbosity when fitting and predicting.
- **warm_start** (*boolean, optional, not for optimizer, default False*) – When set to True, reuse the solution of the previous call to fit and add more estimators to the ensemble, otherwise, just fit a whole new forest.

- **ccp_alpha** (*float, >=0.0, <=0.1 for optimizer, optional, not for optimizer, default 0.0*) – Complexity parameter used for Minimal Cost-Complexity Pruning. The subtree with the largest cost complexity that is smaller than ccp_alpha will be chosen. By default, no pruning is performed.
- **max_samples** (*union type, optional, not for optimizer, default None*) – If bootstrap is True, the number of samples to draw from X to train each base estimator.
  - None
    Draw X.shape[0] samples.
  - *or* integer, >=1
    Draw max_samples samples.
  - *or* float, >0.0, <1.0
    Draw max_samples * X.shape[0] samples.
- **monotonic_cst** (*union type, optional, not for optimizer, default None*) – Indicates the monotonicity constraint to enforce on each feature. Monotonicity constraints are not supported for: multioutput regressions (i.e. when n_outputs > 1),

  regressions trained on data with missing values.
  - array *of* items : -1, 0, *or* 1
    array-like of int of shape (n_features)
  - *or* None
    No constraints are applied.

Notes

constraint-1 : negated type *of* 'y/isSparse'
    This classifier does not support sparse labels.
constraint-2 : union type
    Out of bag estimation only available if bootstrap=True.
    - bootstrap : True
    - *or* oob_score : False

fit(*X*, *y=None*, ***fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

- **X** (*array*) – The outer array is over samples aka rows.
  - items : array *of* items : float
    The inner array is over features aka columns.
- **y** (*union type*) – The predicted classes.
  - array *of* items : array *of* items : float
  - *or* array *of* items : float
- **sample_weight** (*union type, optional*) – Sample weights.
  - array *of* items : float
  - *or* None
    Samples are equally weighted.

predict(*X*, ***predict_params*)

Make predictions.

Note: The predict method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (*array, optional*) – The outer array is over samples aka rows.

- items : array *of* items : float

  The inner array is over features aka columns.

Returns

result – The predicted values.

- array *of* items : array *of* items : float
- *or* array *of* items : float

Return type

union type

lale.lib.sklearn.rfe module

`class lale.lib.sklearn.rfe.RFE(*, estimator, n_features_to_select=None, step=1, verbose=0, importance_getter='auto')`

Bases: `PlannedIndividualOp`

Recursive feature elimination transformer from scikit-learn.

This documentation is auto-generated from JSON schemas.

Parameters

- **estimator** (*operator, not for optimizer*) – A supervised learning estimator with a fit method that provides information about feature importance either through a *coef_* attribute or through a *feature_importances_* attribute.
- **n_features_to_select** (*union type, not for optimizer, default None*) –
  - None

    Half of the features are selected.
  - *or* integer, >=1, <='X/items/maxItems', not for optimizer

    Absolute number of features to select.
  - *or* float, >0.0, <1.0

    Fraction of features to select
- **step** (*union type, not for optimizer, default 1*) – If greater than or equal to 1, then step corresponds to the (integer) number of features to remove at each iteration. If within (0.0, 1.0), then step corresponds to the percentage (rounded down) of features to remove at each iteration.
  - integer, >=1, not for optimizer
  - *or* float, >0, <1, <=0.5 for optimizer
- **verbose** (*union type, not for optimizer, default 0*) – Controls verbosity of output.
  - boolean
  - *or* integer
- **importance_getter** (*union type, optional, not for optimizer, default 'auto'*) –
  - 'auto'

    Use the feature importance either through a *coef_* or *feature_importances_* attributes of estimator.
  - *or* string

    Attribute name/path for extracting feature importance (implemented with attrgetter).
  - *or* callable

    The callable is passed with the fitted estimator and it should return importance for each feature.

decision_function(*X*)

Confidence scores for all classes.

Note: The decision_function method is not available until this operator is trained.

Once this method is available, it will have the following signature:

fit(*X*, *y=None*, ***fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

- **X** (array *of* items : array *of* items : float) – Features; the outer array is over samples.
- **y** (array *of* items : float) – Target class labels; the array is over samples.

predict(*X*, ***predict_params*)

Make predictions.

Note: The predict method is not available until this operator is trained.

Once this method is available, it will have the following signature:

predict_proba(*X*)

Probability estimates for all classes.

Note: The predict_proba method is not available until this operator is trained.

Once this method is available, it will have the following signature:

transform(*X*, *y=None*)

Transform the data.

Note: The transform method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (array, optional *of* items : array *of* items : float) – The input samples.

Returns

result

Return type

array *of* items : array *of* items : float

lale.lib.sklearn.ridge module

class lale.lib.sklearn.ridge.**Ridge**(***, *alpha=1.0*, *fit_intercept=True*, *copy_X=True*, *max_iter=None*, *tol=0.0001*, *solver='auto'*, *random_state=None*, *positive=False*)

Bases: *PlannedIndividualOp*

Ridge regression estimator from scikit-learn.

This documentation is auto-generated from JSON schemas.

Parameters

- **alpha** (*union type, default 1.0*) – Regularization strength; larger values specify stronger regularization.
  - float, >0.0, >=1e-10 for optimizer, <=1.0 for optimizer, loguniform distribution, default 1.0

  - *or* array, not for optimizer *of* items : float, >0.0
    Penalties specific to the targets.
- **fit_intercept** (*boolean, default True*) – Whether to calculate the intercept for this model.

  See also *constraint-1*.
- **copy_X** (*boolean, optional, default True*) – If True, X will be copied; else, it may be overwritten.
- **max_iter** (*union type, optional, default None*) – Maximum number of iterations for conjugate gradient solver.
  - integer, >=1, >=10 for optimizer, <=1000 for optimizer
  - *or* None
- **tol** (*float, >=1e-08 for optimizer, <=0.01 for optimizer, optional, default 0.0001*) – Precision of the solution.
- **solver** ('auto', 'svd', 'cholesky', 'lsqr', 'sparse_cg', 'sag', 'saga', *or* 'lbfgs', default 'auto') – Solver to use in the computational routines:
  - 'auto' chooses the solver automatically based on the type of data.
  - 'svd' uses a Singular Value Decomposition of X to compute the Ridge coefficients. More stable for singular matrices than 'cholesky'.
  - 'cholesky' uses the standard scipy.linalg.solve function to obtain a closed-form solution.
  - 'sparse_cg' uses the conjugate gradient solver as found in scipy.sparse.linalg.cg. As an iterative algorithm, this solver is more appropriate than 'cholesky' for large-scale data (possibility to set *tol* and *max_iter*).
  - 'lsqr' uses the dedicated regularized least-squares routine scipy.sparse.linalg.lsqr. It is the fastest and uses an iterative procedure.
  - 'sag' uses a Stochastic Average Gradient descent, and 'saga' uses its improved, unbiased version named SAGA. Both methods also use an iterative procedure, and are often faster than other solvers when both n_samples and n_features are large. Note that 'sag' and 'saga' fast convergence is only guaranteed on features with approximately the same scale. You can preprocess the data with a scaler from sklearn.preprocessing.
  - 'lbfgs' uses L-BFGS-B algorithm implemented in *scipy.optimize.minimize*. It can be used only when *positive* is True.

  All last six solvers support both dense and sparse data. However, only 'sag', 'sparse_cg', and 'lbfgs' support sparse input when *fit_intercept* is True.

  See also *constraint-1*, *constraint-2*, *constraint-3*, *constraint-4*.
- **random_state** (*union type, optional, not for optimizer, default None*) – The seed of the pseudo random number generator to use when shuffling
  - integer
  - *or* numpy.random.RandomState
  - *or* None
- **positive** (*boolean, optional, not for optimizer, default False*) – When set to True, forces the coefficients to be positive. Only 'lbfgs' solver is supported in this case.

  See also *constraint-3*, *constraint-4*.

Notes

constraint-1 : union type
: solver {svd, lsqr, cholesky, saga} does not support fitting the intercept on sparse data. Please set the solver to 'auto' or 'sparse_cg', 'sag', or set *fit_intercept=False*.
  - negated type *of* 'X/isSparse'
  - *or* fit_intercept : False
  - *or* solver : 'auto', 'sparse_cg', *or* 'sag'

constraint-2 : union type
: SVD solver does not support sparse inputs currently.
  - negated type *of* 'X/isSparse'
  - *or* solver : negated type *of* 'svd'

constraint-3 : union type
: Only 'lbfgs' solver is supported when positive is True. *auto* works too when tested.
  - positive : False
  - *or* solver : 'lbfgs' *or* 'auto'

constraint-4 : union type
: *lbfgs* solver can be used only when positive=True.
  - positive : True
  - *or* solver : negated type *of* 'lbfgs'

`fit`(*X*, *y=None*, ***fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

- **X** (array *of* items : array *of* items : float) – Training data
- **y** (*union type*) – Target values
  - array *of* items : array *of* items : float
  - *or* array *of* items : float
- **sample_weight** (*union type, optional*) – Individual weights for each sample
  - float
  - *or* array *of* items : float
  - *or* None

`predict`(*X*, ***predict_params*)

Make predictions.

Note: The predict method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (*union type, optional*) – Samples.
- array *of* items : float
- *or* array *of* items : array *of* items : float

Returns

result – Returns predicted values.
- array *of* items : float
- *or* array *of* items : array *of* items : float

Return type

union type

lale.lib.sklearn.ridge_classifier module

class lale.lib.sklearn.ridge_classifier.**RidgeClassifier**(**, alpha=1.0, fit_intercept=True, copy_X=True, max_iter=None, tol=0.0001, solver='auto', class_weight=None, random_state=None*)

Bases: *PlannedIndividualOp*

Ridge classifier from scikit-learn.

This documentation is auto-generated from JSON schemas.

Parameters

- **alpha** (*union type, default 1.0*) – Regularization strength; larger values specify stronger regularization.
  - float, >0.0, >=1e-05 for optimizer, <=10.0 for optimizer, loguniform distribution
  - *or* array, not for optimizer *of* items : float, >0.0

    Penalties specific to the targets.
- **fit_intercept** (*boolean, default True*) – Whether to calculate the intercept for this model.
- **copy_X** (*boolean, optional, default True*) – If True, X will be copied; else, it may be overwritten.
- **max_iter** (*union type, optional, default None*) – Maximum number of iterations for conjugate gradient solver.
  - integer, >=10 for optimizer, <=1000 for optimizer
  - *or* None
- **tol** (*float, >=1e-08 for optimizer, <=0.01 for optimizer, optional, default 0.0001*) – Precision of the solution.
- **solver** ('auto', 'svd', 'cholesky', 'lsqr', 'sparse_cg', 'sag', *or* 'saga', default 'auto') – Solver to use in the computational routines.
- **class_weight** (*union type, optional, not for optimizer, default None*) – Weights associated with classes in the form `{class_label: weight}`.
  - dict
  - *or* 'balanced' *or* None
- **random_state** (*union type, optional, not for optimizer, default None*) – The seed of the pseudo random number generator to use when shuffling
  - integer
  - *or* numpy.random.RandomState
  - *or* None

decision_function(*X*)

Confidence scores for all classes.

Note: The decision_function method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (array *of* items : array *of* items : float) – Features; the outer array is over samples.

Returns

result – Confidence scores for samples for each class in the model.

- array *of* items : array *of* items : float

  In the multi-way case, score per (sample, class) combination.
- *or* array *of* items : float

  In the binary case, score for *self._classes[1]*.

Return type

union type

fit(*X*, *y=None*, ***fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

- **X** (array *of* items : array *of* items : float) – Training data
- **y** (*union type*) – Target values
  - array *of* items : array *of* items : float
  - *or* array *of* items : float
  - *or* array *of* items : string
  - *or* array *of* items : boolean
- **sample_weight** (*union type, optional*) – Individual weights for each sample
  - float
  - *or* array *of* items : float
  - *or* None

predict(*X*, ***predict_params*)

Make predictions.

Note: The predict method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (*union type, optional*) – Samples.
- array *of* items : float
- *or* array *of* items : array *of* items : float

Returns

result – Predicted class label per sample.
- array *of* items : float
- *or* array *of* items : string
- *or* array *of* items : boolean

Return type

union type

lale.lib.sklearn.robust_scaler module

class lale.lib.sklearn.robust_scaler.**RobustScaler**(**, with_centering=True, with_scaling=True, quantile_range='(0.25, 0.75)', copy=True, unit_variance=False*)

Bases: *PlannedIndividualOp*

Robust scaler transformer from scikit-learn.

This documentation is auto-generated from JSON schemas.

Parameters

- **with_centering** (*boolean, optional, default True*) – If True, center the data before scaling.

  See also *constraint-1*.
- **with_scaling** (*boolean, optional, default True*) – If True, scale the data to interquartile range.

- **quantile_range** (*tuple, >=2 items for optimizer, <=2 items for optimizer, default (0.25, 0.75)*) – Default: (25.0, 75.0) = (1st quantile, 3rd quantile) = IQR
  - item 0 : float, >=0.001 for optimizer, <=0.3 for optimizer
  - item 1 : float, >=0.7 for optimizer, <=0.999 for optimizer
- **copy** (*boolean, not for optimizer, default True*) – If False, try to avoid a copy and do inplace scaling instead.
- **unit_variance** (*boolean, optional, not for optimizer, default False*) – If True, scale data so that normally distributed features have a variance of 1. In general, if the difference between the x-values of q_max and q_min for a standard normal distribution is greater than 1, the dataset will be scaled down. If less than 1, the dataset will be scaled up.

Notes

constraint-1 : union type

Cannot center sparse matrices: use *with_centering=False* instead. See docstring for motivation and alternatives.

- with_centering : False
- *or* negated type *of* 'X/isSparse'

fit(*X*, *y=None*, ***fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

- **X** (array *of* items : array *of* items : float) – The data used to compute the median and quantiles
- **y** (*any type, optional*) –

transform(*X*, *y=None*)

Transform the data.

Note: The transform method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (array, optional *of* items : array *of* items : float) – The data used to scale along the specified axis.

Returns

result – Center and scale the data.

Return type

array *of* items : array *of* items : float

lale.lib.sklearn.select_k_best module

class `lale.lib.sklearn.select_k_best.SelectKBest`(**, score_func='<function f_classif>', k=10*)

Bases: `PlannedIndividualOp`

Select k best feature selection transformer from scikit-learn.

This documentation is auto-generated from JSON schemas.

Parameters

- **score_func** (*callable, not for optimizer, default <function f_classif at 0x7f3272b1f160>*) – Function taking two arrays X and y, and returning a pair of arrays (scores, pvalues) or a single array with scores.
- **k** (*union type, default 10*) – Number of top features to select
  - integer, >=1, >=2 for optimizer, <='X/items/maxItems', <=15 for optimizer
  - *or* 'all'

fit(*X, y=None, **fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

- **X** (array *of* items : array *of* items : float) – Training input samples.
- **y** (*union type*) – Target values (class labels in classification, real numbers in regression).
  - array *of* items : float
  - *or* array *of* items : string
  - *or* array *of* items : boolean

transform(*X, y=None*)

Transform the data.

Note: The transform method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (array *of* items : array *of* items : float) – The input samples

Returns

result – The input samples with only the selected features.

Return type

array *of* items : array *of* items : float

lale.lib.sklearn.sgd_classifier module

class `lale.lib.sklearn.sgd_classifier.SGDClassifier`(**, loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=1000, tol=0.001, shuffle=True, verbose=0, epsilon=0.1, n_jobs=None, random_state=None, learning_rate='optimal', eta0=0.0, power_t=0.5, early_stopping=False, validation_fraction=0.1, n_iter_no_change=5, class_weight=None, warm_start=False, average=False*)

Bases: `PlannedIndividualOp`

SGD classifier from scikit-learn uses linear classifiers (SVM, logistic regression, a.o.) with stochastic gradient descent training.

This documentation is auto-generated from JSON schemas.

Parameters

- **loss** (*union type, default 'hinge'*) – The loss function to be used. Defaults to 'hinge', which gives a linear SVM. The possible options are 'hinge', 'log', 'modified_huber', 'squared_hinge', 'perceptron', or a regression loss: 'squared_error', 'huber', 'epsilon_insensitive', or 'squared_epsilon_insensitive'. The 'log_loss' loss gives logistic regression, a probabilistic classifier. 'modified_huber' is another smooth loss that brings tolerance to outliers as well as probability estimates. 'squared_hinge' is like hinge but is quadratically penalized. 'perceptron' is the linear loss used by the perceptron algorithm. The other losses are designed for regression but can be useful in classification as well; see SGDRegressor for a description. More details about the losses formulas can be found in the scikit-learn User Guide.
  - 'hinge', 'log_loss', 'modified_huber', 'squared_hinge', 'perceptron', 'squared_error', 'huber', 'epsilon_insensitive', *or* 'squared_epsilon_insensitive'
  - *or* 'squared_loss', not for optimizer
- **penalty** ('elasticnet', 'l1', *or* 'l2', default 'l2') – The penalty (aka regularization term) to be used. Defaults to 'l2'
- **alpha** (*float, >=1e-10 for optimizer, <=1.0 for optimizer, loguniform distribution, default 0.0001*) – Constant that multiplies the regularization term. Defaults to 0.0001
- **l1_ratio** (*float, >=1e-09 for optimizer, <=1.0 for optimizer, loguniform distribution, default 0.15*) – The Elastic Net mixing parameter, with 0 <= l1_ratio <= 1.
- **fit_intercept** (*boolean, default True*) – Whether the intercept should be estimated or not. If False, the
- **max_iter** (*integer, >=10 for optimizer, <=1000 for optimizer, uniform distribution, default 1000*) – The maximum number of passes over the training data (aka epochs).
- **tol** (*union type, default 0.001*) – The stopping criterion.
  - float, >=1e-08 for optimizer, <=0.01 for optimizer
  - *or* None
- **shuffle** (*boolean, default True*) – Whether or not the training data should be shuffled after each epoch.
- **verbose** (*integer, not for optimizer, default 0*) – The verbosity level
- **epsilon** (*float, >=1e-08 for optimizer, <=1.35 for optimizer, loguniform distribution, default 0.1*) – Epsilon in the epsilon-insensitive loss functions; only if *loss* is
- **n_jobs** (*union type, not for optimizer, default None*) – The number of CPUs to use to do the OVA (One Versus All, for
  - integer
  - *or* None
- **random_state** (*union type, not for optimizer, default None*) – The seed of the pseudo random number generator to use when shuffling
  - integer
  - *or* numpy.random.RandomState
  - *or* None
- **learning_rate** ('optimal', 'constant', 'invscaling', *or* 'adaptive', default 'optimal') – The learning rate schedule:

  See also *constraint-1*.
- **eta0** (*float, >=0.01 for optimizer, <=1.0 for optimizer, loguniform distribution, default 0.0*) – The initial learning rate for the 'constant', 'invs-

caling' or

See also *constraint-1*.

- **power_t** (*float, >=1e-05 for optimizer, <=1.0 for optimizer, uniform distribution, default 0.5*) – The exponent for inverse scaling learning rate [default 0.5].
- **early_stopping** (*boolean, not for optimizer, default False*) – Whether to use early stopping to terminate training when validation
- **validation_fraction** (*float, >=0.0, <=1.0, not for optimizer, default 0.1*) – The proportion of training data to set aside as validation set for
- **n_iter_no_change** (*integer, >=5 for optimizer, <=10 for optimizer, not for optimizer, default 5*) – Number of iterations with no improvement to wait before early stopping.
- **class_weight** (*union type, not for optimizer, default None*) – Preset for the class_weight fit parameter.
  - dict
  - *or* 'balanced' *or* None
- **warm_start** (*boolean, not for optimizer, default False*) – When set to True, reuse the solution of the previous call to fit as
- **average** (*union type, not for optimizer, default False*) – When set to True, computes the averaged SGD weights and stores the result in the `coef_` attribute.
  - boolean
  - *or* integer, not for optimizer

Notes

constraint-1 : union type

eta0 must be greater than 0 if the learning_rate is not 'optimal'.

- learning_rate : 'optimal'
- *or* eta0 : float, >0.0

decision_function(*X*)

Confidence scores for all classes.

Note: The decision_function method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (array *of* items : array *of* items : float) –

Returns

result – Confidence scores for samples for each class in the model.

- array *of* items : array *of* items : float
  In the multi-way case, score per (sample, class) combination.
- *or* array *of* items : float
  In the binary case, score for *self._classes[1]*.

Return type

union type

fit(*X*, *y=None*, ***fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

- **X** (array *of* items : array *of* items : float) –
- **y** (*union type*) –
  - array *of* items : string
  - *or* array *of* items : float
  - *or* array *of* items : boolean
- **coef_init** (array, optional *of* items : array *of* items : float) – The initial coefficients to warm-start the optimization.
- **intercept_init** (array, optional *of* items : float) – The initial intercept to warm-start the optimization.
- **sample_weight** (*union type, optional, default None*) – Weights applied to individual samples.
  - array *of* items : float
  - *or* None

    Uniform weights.

partial_fit(*X*, *y=None*, ***fit_params*)

Incremental fit to train train the operator on a batch of samples.

Note: The partial_fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

- **X** (array *of* items : array *of* items : float) –
- **y** (*union type*) –
  - array *of* items : string
  - *or* array *of* items : float
  - *or* array *of* items : boolean
- **classes** (*union type, optional*) –
  - array *of* items : string
  - *or* array *of* items : float
  - *or* array *of* items : boolean
- **sample_weight** (*union type, optional, default None*) – Weights applied to individual samples.
  - array *of* items : float
  - *or* None

    Uniform weights.

predict(*X*, ***predict_params*)

Make predictions.

Note: The predict method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (array *of* items : array *of* items : float) –

Returns

result –

- array *of* items : string
- *or* array *of* items : float
- *or* array *of* items : boolean

Return type

union type

predict_proba(*X*)

Probability estimates for all classes.

Note: The predict_proba method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (array *of* items : array *of* items : float) –

Returns

result – Returns the probability of the sample for each class in the model,

Return type

array *of* items : array *of* items : float

lale.lib.sklearn.sgd_regressor module

class lale.lib.sklearn.sgd_regressor.**SGDRegressor**(**, loss='squared_error', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=1000, tol=None, shuffle=True, verbose=0, epsilon=0.1, random_state=None, learning_rate='invscaling', eta0=0.01, power_t=0.25, early_stopping=False, validation_fraction=0.1, n_iter_no_change=5, warm_start=False, average=False*)

Bases: *PlannedIndividualOp*

SGD regressor from scikit-learn uses linear regressors (SVM, logistic regression, a.o.) with stochastic gradient descent training.

This documentation is auto-generated from JSON schemas.

Parameters

- **loss** ('squared_error', 'huber', 'epsilon_insensitive', *or* 'squared_epsilon_insensitive', default 'squared_error') – The loss function to be used. The possible values are 'squared_error', 'huber', 'epsilon_insensitive', or 'squared_epsilon_insensitive'. The 'squared_error' refers to the ordinary least squares fit. 'huber' modifies 'squared_error' to focus less on getting outliers correct by switching from squared to linear loss past a distance of epsilon. 'epsilon_insensitive' ignores errors less than epsilon and is linear past that; this is the loss function used in SVR. 'squared_epsilon_insensitive' is the same but becomes squared loss past a tolerance of epsilon. More details about the losses formulas can be found in the scikit-learn User Guide.
- **penalty** ('elasticnet', 'l1', *or* 'l2', default 'l2') – The penalty (aka regularization term) to be used. Defaults to 'l2'
- **alpha** (*float, >=1e-10 for optimizer, <=1.0 for optimizer, loguniform distribution, default 0.0001*) – Constant that multiplies the regularization term. Defaults to 0.0001
- **l1_ratio** (*float, >=1e-09 for optimizer, <=1.0 for optimizer, loguniform distribution, default 0.15*) – The Elastic Net mixing parameter, with 0 <= l1_ratio <= 1.
- **fit_intercept** (*boolean, default True*) – Whether the intercept should be estimated or not. If False, the
- **max_iter** (*integer, >=5 for optimizer, <=1000 for optimizer, uniform distribution, default 1000*) – The maximum number of passes over the training data (aka epochs).
- **tol** (*union type, default None*) – The stopping criterion. If it is not None, the iterations will stop
  - float, >=1e-08 for optimizer, <=0.01 for optimizer
  - *or* None
- **shuffle** (*boolean, default True*) – Whether or not the training data should be shuffled after each epoch.
- **verbose** (*integer, not for optimizer, default 0*) – The verbosity level.

- **epsilon** (*float, >=1e-08 for optimizer, <=1.35 for optimizer, loguniform distribution, default 0.1*) – Epsilon in the epsilon-insensitive loss functions; only if *loss* is
- **random_state** (*union type, not for optimizer, default None*) – The seed of the pseudo random number generator to use when shuffling
  - integer
  - *or* numpy.random.RandomState
  - *or* None
- **learning_rate** ('optimal', 'constant', 'invscaling', *or* 'adaptive', default 'invscaling') – The learning rate schedule:

  See also *constraint-1*.
- **eta0** (*float, >=0.01 for optimizer, <=1.0 for optimizer, loguniform distribution, default 0.01*) – The initial learning rate for the 'constant', 'invscaling' or

  See also *constraint-1*.
- **power_t** (*float, >=1e-05 for optimizer, <=1.0 for optimizer, uniform distribution, default 0.25*) – The exponent for inverse scaling learning rate [default 0.5].
- **early_stopping** (*boolean, not for optimizer, default False*) – Whether to use early stopping to terminate training when validation
- **validation_fraction** (*float, >=0.0, <=1.0, not for optimizer, default 0.1*) – The proportion of training data to set aside as validation set for
- **n_iter_no_change** (*integer, >=5 for optimizer, <=10 for optimizer, not for optimizer, default 5*) – Number of iterations with no improvement to wait before early stopping.
- **warm_start** (*boolean, not for optimizer, default False*) – When set to True, reuse the solution of the previous call to fit as
- **average** (*union type, not for optimizer, default False*) – When set to True, computes the averaged SGD weights and stores the result in the `coef_` attribute.
  - boolean
  - *or* integer, not for optimizer

Notes

constraint-1 : union type

eta0 must be greater than 0 if the learning_rate is not 'optimal'.

- learning_rate : 'optimal'
- *or* eta0 : float, >0.0

fit(*X*, *y=None*, ***fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

- **X** (array *of* items : array *of* items : float) –
- **y** (array *of* items : float) –
- **coef_init** (array, optional *of* items : float) – The initial coefficients to warm-start the optimization.
- **intercept_init** (array, optional *of* items : float) – The initial intercept to warm-start the optimization.

- **sample_weight** (*union type, optional, default None*) – Weights applied to individual samples.
  - array *of* items : float
  - *or* None
    Uniform weights.

partial_fit(*X*, *y=None*, ***fit_params*)

Incremental fit to train train the operator on a batch of samples.

Note: The partial_fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

- **X** (array *of* items : array *of* items : float) –
- **y** (array *of* items : float) –
- **classes** (array, optional *of* items : float) –
- **sample_weight** (*union type, optional, default None*) – Weights applied to individual samples.
  - array *of* items : float
  - *or* None
    Uniform weights.

predict(*X*, ***predict_params*)

Make predictions.

Note: The predict method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters
X (array *of* items : array *of* items : float) –

Returns
result

Return type
array *of* items : float

lale.lib.sklearn.simple_imputer module

class lale.lib.sklearn.simple_imputer.**SimpleImputer**(**, missing_values=nan, strategy='mean', fill_value=None, copy=True, add_indicator=False, keep_empty_features=False*)

Bases: *PlannedIndividualOp*

Simple imputer transformer from scikit-learn for completing missing values.

This documentation is auto-generated from JSON schemas.

Parameters

- **missing_values** (*union type, not for optimizer, default nan*) – The placeholder for the missing values.
  - float
  - *or* string
  - *or* nan
  - *or* None
- **strategy** (*union type, default 'mean'*) – The imputation strategy.
  - 'constant', not for optimizer
  - *or* 'mean', 'median', *or* 'most_frequent'

- **fill_value** (*union type, not for optimizer, default None*) – When strategy == "constant", fill_value is used to replace all occurrences of missing_values
  - float
  - *or* string
  - *or* None
- **copy** (*boolean, not for optimizer, default True*) – If True, a copy of X will be created.
- **add_indicator** (*boolean, not for optimizer, default False*) – If True, a MissingIndicator transform will stack onto output of the imputer's transform.
- **keep_empty_features** (*boolean, optional, not for optimizer, default False*) – If True, features that consist exclusively of missing values when fit is called are returned in results when transform is called. The imputed value is always 0 except when strategy="constant" in which case fill_value will be used instead.

fit(*X*, *y=None*, ***fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

- **X** (*array*) – Input data, where `n_samples` is the number of samples and `n_features` is the number of features.
  - items : array
    - items : union type
      - float
      - *or* string
- **y** (*any type, optional*) –

transform(*X*, *y=None*)

Transform the data.

Note: The transform method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (*array*) – The input data to complete.

- items : array
  - items : union type
    - float
    - *or* string

Returns

result – The input data to complete.

- items : array
  - items : union type
    - float
    - *or* string

Return type

array

lale.lib.sklearn.stacking_classifier module

class `lale.lib.sklearn.stacking_classifier.`**`StackingClassifier`**(*, *estimators*, *final_estimator=None*, *cv=5*, *stack_method='auto'*, *n_jobs=None*, *passthrough=False*)

Bases: `PlannedIndividualOp`

Stacking classifier from scikit-learn for stacking ensemble.

This documentation is auto-generated from JSON schemas.

Parameters

- **estimators** (*array*) – Base estimators which will be stacked together. Each element of the list is defined as a tuple of string (i.e. name) and an estimator instance. An estimator can be set to 'drop' using set_params.
  - items : tuple
    - item 0 : string
    - item 1 : union type
      - operator
      - *or* None
- **final_estimator** (*union type, default None*) – A classifier which will be used to combine the base estimators. The default classifier is a 'LogisticRegression'
  - operator
  - *or* None
- **cv** (*union type, default 5*) – Determines the cross-validation splitting strategy used in cross_val_predict to train final_estimator.
  - union type
    - integer, >=2, >=3 for optimizer, <=4 for optimizer, uniform distribution, default 5

      Number of folds for cross-validation.
    - *or* None, not for optimizer

      to use the default 5-fold cross validation
  - *or* 'prefit', not for optimizer

    "prefit" to assume the estimators are prefit. In this case, the estimators will not be refitted.
  - *or* CrossvalGenerator, not for optimizer

    Object with split function: generator yielding (train, test) splits as arrays of indices. Can use any of the iterators from https://scikit-learn.org/stable/modules/cross_validation.html#cross-validation-iterators
- **stack_method** ('auto', 'predict_proba', 'decision_function', *or* 'predict', not for optimizer, default 'auto') – Methods called for each base estimator. If 'auto', it will try to invoke, for each estimator, 'predict_proba', 'decision_function' or 'predict' in that order. Otherwise, one of 'predict_proba', 'decision_function' or 'predict'. If the method is not implemented by the estimator, it will raise an error.
- **n_jobs** (*union type, not for optimizer, default None*) – The number of jobs to run in parallel for `fit`.
  - integer
  - *or* None
- **passthrough** (*boolean, default False*) – When False, only the predictions of estimators will be used as training data for 'final_estimator'. When True, the 'final_estimator' is trained on the predictions as well as the original training data.

decision_function(*X*)

Confidence scores for all classes.

Note: The decision_function method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters
: **X** (array *of* items : array *of* items : float) – Training vectors, where n_samples is the number of samples and n_features is the number of features.

Returns
: **result** – The decision function computed by the final estimator.

Return type
: array *of* items : array *of* items : float

`fit`(*X*, *y=None*, ***fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

- **X** (array *of* items : array *of* items : float) – Training vectors, where n_samples is the number of samples and n_features is the number of features.
- **y** (*union type*) – The target values (class labels).
  - array *of* items : float
  - *or* array *of* items : string
  - *or* array *of* items : boolean
- **sample_weight** (*union type, optional*) – Sample weights. If None, then samples are equally weighted.
  - array *of* items : float
  - *or* None

`predict`(*X*, ***predict_params*)

Make predictions.

Note: The predict method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters
: **X** (array, optional *of* items : array *of* items : float) – The input samples.

Returns
: **result** – Predicted targets.

Return type
: array *of* items : float

`predict_proba`(*X*)

Probability estimates for all classes.

Note: The predict_proba method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters
: **X** (array, optional *of* items : array *of* items : float) – The input samples.

Returns
: **result** – Class probabilities of the input samples.

Return type
: array *of* items : array *of* items : float

`transform`(*X*, *y=None*)

Transform the data.

Note: The transform method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (array, optional *of* items : array *of* items : float) – Training vectors, where n_samples is the number of samples and n_features is the number of features

Returns

result – Transformed array

- items : array
  - items : union type
    - float
    - *or* array *of* items : float

Return type

array

lale.lib.sklearn.stacking_regressor module

class lale.lib.sklearn.stacking_regressor.**StackingRegressor**(***, *estimators*, *final_estimator=None*, *cv=5*, *n_jobs=None*, *passthrough=False*)

Bases: *PlannedIndividualOp*

Stacking regressor from scikit-learn for stacking ensemble.

This documentation is auto-generated from JSON schemas.

Parameters

- **estimators** (*array*) – Base estimators which will be stacked together. Each element of the list is defined as a tuple of string (i.e. name) and an estimator instance. An estimator can be set to 'drop' using set_params.
  - items : tuple
    - item 0 : string
    - item 1 : union type
      - operator
      - *or* None
- **final_estimator** (*union type, default None*) – A regressor which will be used to combine the base estimators. The default classifier is a 'RidgeCV'
  - operator
  - *or* None
- **cv** (*union type, default 5*) – Determines the cross-validation splitting strategy used in cross_val_predict to train final_estimator.
  - union type
    - integer, >=2, >=3 for optimizer, <=4 for optimizer, uniform distribution, default 5

      Number of folds for cross-validation.
    - *or* None, not for optimizer

      to use the default 5-fold cross validation
  - *or* 'prefit', not for optimizer

    "prefit" to assume the estimators are prefit. In this case, the estimators will not be refitted.
  - *or* CrossvalGenerator, not for optimizer

    Object with split function: generator yielding (train, test) splits as arrays of indices. Can use any of the iterators from https://scikit-learn.org/stable/modules/cross_validation.html#cross-validation-iterators

- `n_jobs` (*union type, not for optimizer, default None*) – The number of jobs to run in parallel for `fit`.
  - integer
  - *or* None
- `passthrough` (*boolean, default False*) – When False, only the predictions of estimators will be used as training data for 'final_estimator'. When True, the 'final_estimator' is trained on the predictions as well as the original training data.

`fit`(*X*, *y=None*, ***fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

- **X** (array *of* items : array *of* items : float) – Training vectors, where n_samples is the number of samples and n_features is the number of features.
- **y** (array *of* items : float) – Target values.
- **sample_weight** (*union type, optional*) – Sample weights. If None, then samples are equally weighted.
  - array *of* items : float
  - *or* None

`predict`(*X*, ***predict_params*)

Make predictions.

Note: The predict method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (array, optional *of* items : array *of* items : float) – The input samples.

Returns

result – Predicted targets.

Return type

array *of* items : float

`transform`(*X*, *y=None*)

Transform the data.

Note: The transform method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (array, optional *of* items : array *of* items : float) – Training vectors, where n_samples is the number of samples and n_features is the number of features

Returns

result – Transformed array

- items : array
  - items : union type
    - float
    - *or* array *of* items : float

Return type

array

lale.lib.sklearn.stacking_utils module

lale.lib.sklearn.standard_scaler module

class lale.lib.sklearn.standard_scaler.**StandardScaler**(*, *copy=True*, *with_mean=True*, *with_std=True*)

Bases: *PlannedIndividualOp*

Standard scaler transformer from scikit-learn.

This documentation is auto-generated from JSON schemas.

Parameters

- **copy** (*boolean, not for optimizer, default True*) – If False, try to avoid a copy and do inplace scaling instead.
- **with_mean** (*boolean, default True*) – If True, center the data before scaling.

  See also *constraint-1*.
- **with_std** (*boolean, default True*) – If True, scale the data to unit variance (or equivalently, unit standard deviation).

Notes

constraint-1 : union type

Setting *with_mean* to True does not work on sparse matrices, because centering them entails building a dense matrix which in common use cases is likely to be too large to fit in memory.

- with_mean : False
- *or* negated type *of* 'X/isSparse'

fit(*X*, *y=None*, ***fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

- **X** (array *of* items : array *of* items : float) – The data used to compute the mean and standard deviation
- **y** (*any type, optional*) – Ignored

partial_fit(*X*, *y=None*, ***fit_params*)

Incremental fit to train train the operator on a batch of samples.

Note: The partial_fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

transform(*X*, *y=None*)

Transform the data.

Note: The transform method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

- **X** (array *of* items : array *of* items : float) – The data used to scale along the features axis.
- **copy** (*union type, optional, default None*) – Copy the input X or not.
  - boolean
  - *or* None

Returns
result – Perform standardization by centering and scaling

Return type
array *of* items : array *of* items : float

lale.lib.sklearn.svc module

`class lale.lib.sklearn.svc.SVC(*, C=1.0, kernel='rbf', degree=3, gamma='scale', coef0=0.0, shrinking=True, probability=False, tol=0.001, cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_shape='ovr', random_state=None, break_ties=False)`

Bases: `PlannedIndividualOp`

Support Vector Classification from scikit-learn.

This documentation is auto-generated from JSON schemas.

Parameters

- **C** (*float, >0.0, >=0.03125 for optimizer, <=32768 for optimizer, loguniform distribution, optional, not for optimizer, default 1.0*) – Penalty parameter C of the error term.
- **kernel** (*union type, default 'rbf'*) – Specifies the kernel type to be used in the algorithm.
  - 'precomputed', not for optimizer
  - *or* 'linear', 'poly', 'rbf', *or* 'sigmoid'
  - *or* callable, not for optimizer

  See also *constraint-1*.
- **degree** (*integer, >=0, >=2 for optimizer, <=5 for optimizer, default 3*) – Degree of the polynomial kernel function ('poly').
- **gamma** (*union type, default 'scale'*) – Kernel coefficient for 'rbf', 'poly' and 'sigmoid'.
  - 'scale' *or* 'auto'
  - *or* float, >0.0, >=3.0517578125e-05 for optimizer, <=8 for optimizer, loguniform distribution
- **coef0** (*float, >=-1 for optimizer, <=1 for optimizer, optional, not for optimizer, default 0.0*) – Independent term in kernel function.
- **shrinking** (*boolean, default True*) – Whether to use the shrinking heuristic.
- **probability** (*boolean, optional, default False*) – Whether to enable probability estimates.
- **tol** (*float, >0.0, <=0.01 for optimizer, default 0.001*) – Tolerance for stopping criteria.
- **cache_size** (*integer, >=0, <=1000 for optimizer, not for optimizer, default 200*) – Specify the size of the kernel cache (in MB).
- **class_weight** (*union type, optional, not for optimizer, default None*) –
  - None

    By default, all classes have weight 1.
  - *or* 'balanced'

    Adjust weights by inverse frequency.
  - *or* dict, not for optimizer

    Dictionary mapping class labels to weights.
- **verbose** (*boolean, optional, not for optimizer, default False*) – Enable verbose output.
- **max_iter** (*integer, >=1 for optimizer, <=1000 for optimizer, not for optimizer, default -1*) – Hard limit on iterations within solver, or -1 for no

limit.

- **decision_function_shape** ('ovo' *or* 'ovr', not for optimizer, default 'ovr') – Whether to return a one-vs-rest ('ovr') decision function of shape (n_samples, n_classes) as all other classifiers, or the original one-vs-one ('ovo') decision function of libsvm which has shape (n_samples, n_classes * (n_classes - 1) / 2).
- **random_state** (*union type, optional, not for optimizer, default None*) – Seed of pseudo-random number generator.
  - numpy.random.RandomState
  - *or* None

    RandomState used by np.random
  - *or* integer

    Explicit seed.
- **break_ties** (*boolean, optional, not for optimizer, default False*) – If true, decision_function_shape='ovr', and number of classes > 2, predict will break ties according to the confidence values of decision_function; otherwise the first class among the tied classes is returned.

Notes

constraint-1 : union type

Sparse precomputed kernels are not supported.

- negated type *of* 'X/isSparse'
- *or* kernel : negated type *of* 'precomputed'

decision_function(*X*)

Confidence scores for all classes.

Note: The decision_function method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (*array*) – The outer array is over samples aka rows.

- items : array *of* items : float

  The inner array is over features aka columns.

Returns

result – Confidence scores for samples for each class in the model.

- array *of* items : array *of* items : float

  In the multi-way case, score per (sample, class) combination.
- *or* array *of* items : float

  In the binary case, score for *self._classes[1]*.

Return type

union type

fit(*X*, *y=None*, ***fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

- **X** (*array*) – The outer array is over samples aka rows.
  - items : array *of* items : float

    The inner array is over features aka columns.
- **y** (*union type*) – The predicted classes.
  - array *of* items : float
  - *or* array *of* items : string

    - *or* array *of* items : boolean
  - **sample_weight** (*union type, optional*) – Sample weights.
    - array *of* items : float
    - *or* None

      Samples are equally weighted.

predict(*X*, ***predict_params*)

Make predictions.

Note: The predict method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (*array, optional*) – The outer array is over samples aka rows.
- items : array *of* items : float

  The inner array is over features aka columns.

Returns

result – The predicted classes.
- array *of* items : float
- *or* array *of* items : string
- *or* array *of* items : boolean

Return type

union type

predict_proba(*X*)

Probability estimates for all classes.

Note: The predict_proba method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (*array, optional*) – The outer array is over samples aka rows.
- items : array *of* items : float

  The inner array is over features aka columns.

Returns

result – The outer array is over samples aka rows.
- items : array *of* items : float

  The inner array has items corresponding to each class.

Return type

array

lale.lib.sklearn.svr module

class lale.lib.sklearn.svr.**SVR**(*, *kernel='rbf'*, *degree=3*, *gamma='scale'*, *coef0=0.0*, *tol=0.001*, *C=1.0*, *epsilon=0.1*, *shrinking=True*, *cache_size=200.0*, *verbose=False*, *max_iter=-1*)

Bases: *PlannedIndividualOp*

Support Vector Classification from scikit-learn.

This documentation is auto-generated from JSON schemas.

Parameters

- **kernel** (*union type, default 'rbf'*) – Specifies the kernel type to be used in the algorithm.
  - 'precomputed', not for optimizer

  - *or* 'linear', 'poly', 'rbf', *or* 'sigmoid'
  - *or* callable, not for optimizer

  See also *constraint-1*.
- **degree** (*integer, >=0, >=2 for optimizer, <=5 for optimizer, default 3*) – Degree of the polynomial kernel function ('poly').
- **gamma** (*union type, default 'scale'*) – Kernel coefficient for 'rbf', 'poly' and 'sigmoid'.
  - 'scale' *or* 'auto'
  - *or* float, >0.0, >=3.0517578125e-05 for optimizer, <=8 for optimizer, loguniform distribution
- **coef0** (*float, >=-1 for optimizer, <=1 for optimizer, not for optimizer, default 0.0*) – Independent term in kernel function.
- **tol** (*float, >0.0, <=0.01 for optimizer, default 0.001*) – Tolerance for stopping criteria.
- **C** (*float, >0.0, >=0.03125 for optimizer, <=32768 for optimizer, loguniform distribution, default 1.0*) – Penalty parameter C of the error term.
- **epsilon** (*float, >=0.0, >=1e-05 for optimizer, <=10000.0 for optimizer, not for optimizer, default 0.1*) – Epsilon in the epsilon-SVR model. It specifies the epsilon-tube within which no penalty is associated in the training loss function with points predicted within a distance epsilon from the actual value.
- **shrinking** (*boolean, default True*) – Whether to use the shrinking heuristic.
- **cache_size** (*float, >=0, <=1000 for optimizer, not for optimizer, default 200.0*) – Specify the size of the kernel cache (in MB).
- **verbose** (*boolean, not for optimizer, default False*) – Enable verbose output.
- **max_iter** (*integer, >=1 for optimizer, <=1000 for optimizer, not for optimizer, default -1*) – Hard limit on iterations within solver, or -1 for no limit.

Notes

constraint-1 : union type

Sparse precomputed kernels are not supported.

- negated type *of* 'X/isSparse'
- *or* kernel : negated type *of* 'precomputed'

fit(*X*, *y=None*, ***fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

- **X** (*array*) – The outer array is over samples aka rows.
  - items : array *of* items : float

    The inner array is over features aka columns.
- **y** (array *of* items : float) –

predict(*X*, ***predict_params*)

Make predictions.

Note: The predict method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters
X (*array, optional*) – The outer array is over samples aka rows.
- items : array *of* items : float
  The inner array is over features aka columns.

Returns
result – The predicted classes.

Return type
array *of* items : float

lale.lib.sklearn.target_encoder module

`class lale.lib.sklearn.target_encoder.TargetEncoder`(*, *categories='auto'*, *target_type='auto'*, *smooth='auto'*, *cv=5*, *shuffle=True*, *random_state=None*)

Bases: `PlannedIndividualOp`

Target encoder for regression and classification targets..

This documentation is auto-generated from JSON schemas.

Parameters
- **categories** (*union type, not for optimizer, default 'auto'*) – Categories (unique values) per feature.
  - 'auto'
    Determine categories automatically from training data.
  - *or* array
    The ith list element holds the categories expected in the ith column.
    * items : union type
      · array *of* items : string
      · *or* array *of* items : float
        Should be sorted.
- **target_type** (*union type, not for optimizer, default 'auto'*) – Type of target.
  - 'auto'
    Type of target is inferred with type_of_target.
  - *or* 'continuous'
    Continuous target
  - *or* 'binary'
    Binary target
  - *or* 'multiclass'
    Multiclass target
- **smooth** (*union type, optional, not for optimizer, default 'auto'*) – The amount of mixing of the target mean conditioned on the value of the category with the global target mean.
  - 'auto'
    Set to an empirical Bayes estimate.
  - *or* float, >=0.0, <=1.0
    A larger smooth value will put more weight on the global target mean
- **cv** (*integer, >=1, optional, not for optimizer, default 5*) – Determines the number of folds in the cross fitting strategy used in fit_transform. For classification targets, StratifiedKFold is used and for continuous targets, KFold is used.
- **shuffle** (*boolean, optional, not for optimizer, default True*) – Whether to shuffle the data in fit_transform before splitting into folds. Note that the samples within each split will not be shuffled.

- **random_state** (*union type, optional, not for optimizer, default None*) – When shuffle is True, random_state affects the ordering of the indices, which controls the randomness of each fold. Otherwise, this parameter has no effect. Pass an int for reproducible output across multiple function calls.
  - None
  - *or* numpy.random.RandomState
    Use the provided random state, only affecting other users of that same random state instance.
  - *or* integer
    Explicit seed.

fit(*X*, *y=None*, ***fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

- **X** (*array*) – Features; the outer array is over samples.
  - items : union type
    - array *of* items : float
    - *or* array *of* items : string
- **y** (*array, optional*) – The target data used to encode the categories.

transform(*X*, *y=None*)

Transform the data.

Note: The transform method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (*array*) – Features; the outer array is over samples.

- items : union type
  - array *of* items : float
  - *or* array *of* items : string

Returns

result – Transformed input; the outer array is over samples.

- items : union type
  - array *of* items : float
  - *or* array *of* items : string

Return type

array

lale.lib.sklearn.tfidf_vectorizer module

class lale.lib.sklearn.tfidf_vectorizer.**TfidfVectorizer**(**, input='content', encoding='utf-8', decode_error='strict', strip_accents=None, lowercase=True, preprocessor=None, tokenizer=None, analyzer='word', stop_words=None, token_pattern='(?u)\x08\\w\\w+\x08', ngram_range='(1, 1)', max_df=1.0, min_df=1, max_features=None, vocabulary=None, binary=False, dtype='float64', norm='l2', use_idf=True, smooth_idf=True, sublinear_tf=False*)

Bases: `PlannedIndividualOp`

TF-IDF vectorizer transformer from scikit-learn for turning text into term frequency - inverse document frequency numeric features.

This documentation is auto-generated from JSON schemas.

Parameters

- **input** ('filename', 'file', *or* 'content', not for optimizer, default 'content') –
- **encoding** (*string, not for optimizer, default 'utf-8'*) –
- **decode_error** ('strict', 'ignore', *or* 'replace', not for optimizer, default 'strict') –
- **strip_accents** ('ascii', 'unicode', *or* None, not for optimizer, default None) –
- **lowercase** (*boolean, not for optimizer, default True*) –
- **preprocessor** (*union type, not for optimizer, default None*) –
  - callable, not for optimizer
  - *or* None
- **tokenizer** (*union type, not for optimizer, default None*) –
  - callable, not for optimizer
    - *or* None

  See also *constraint-1*.
- **analyzer** (*union type, default 'word'*) –
  - 'word', 'char', *or* 'char_wb'
    - *or* callable, not for optimizer

  See also *constraint-1*, *constraint-2*.
- **stop_words** (*union type, not for optimizer, default None*) –
  - None *or* 'english'
    - *or* array *of* items : string

  See also *constraint-2*.
- **token_pattern** (*string, optional, not for optimizer, default '(?u)\b\w\w+\b'*) –
- **ngram_range** (*union type, default (1, 1)*) –
  - tuple, >=2 items for optimizer, <=2 items for optimizer, not for optimizer *of* items : integer, >=1 for optimizer, <=3 for optimizer
  - *or* (1, 1), (1, 2), (1, 3), (2, 2), (2, 3), *or* (3, 3)
- **max_df** (*union type, default 1.0*) –
  - float, >=0.0, >=0.8 for optimizer, <=1.0, <=0.9 for optimizer, uniform distribution float in range [0.0, 1.0]
  - *or* integer, not for optimizer
- **min_df** (*union type, default 1*) –
  - float, >=0.0, >=0.0 for optimizer, <=1.0, <=0.1 for optimizer, uniform distribution float in range [0.0, 1.0]
  - *or* integer, not for optimizer
- **max_features** (*union type, not for optimizer, default None*) –
  - integer, >=1, <=10000 for optimizer
  - *or* None
- **vocabulary** (*union type, not for optimizer, default None*) – XXX TODO XXX, Mapping or iterable, optional
  - dict
  - *or* None
- **binary** (*boolean, default False*) –
- **dtype** (*string, not for optimizer, default 'float64'*) – XXX TODO XXX, type, optional
- **norm** ('l1', 'l2', *or* None, default 'l2') –
- **use_idf** (*boolean, default True*) –
- **smooth_idf** (*boolean, default True*) –

- **sublinear_tf** (*boolean, default False*) –

Notes

constraint-1 : union type
tokenizer, only applies if analyzer == 'word'
- analyzer : 'word'
- *or* tokenizer : None

constraint-2 : union type
stop_words can be a list only if analyzer == 'word'
- stop_words : negated type *of* array *of* items : string
- *or* analyzer : 'word'

fit(*X*, *y=None*, ***fit_params*)
Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters
- **X** (*union type*) – Features; the outer array is over samples.
  - array *of* items : string
  - *or* array *of* items : array, >=1 items, <=1 items *of* items : string
- **y** (*any type, optional*) – Target class labels; the array is over samples.

transform(*X*, *y=None*)
Transform the data.

Note: The transform method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters
X (*union type*) – Features; the outer array is over samples.
- array *of* items : string
- *or* array *of* items : array, >=1 items, <=1 items *of* items : string

Returns
result – Output data schema for predictions (projected data) using the TfidfVectorizer model from scikit-learn.

Return type
array *of* items : array *of* items : float

lale.lib.sklearn.variance_threshold module

class lale.lib.sklearn.variance_threshold.**VarianceThreshold**(**, threshold=0*)
Bases: *PlannedIndividualOp*

VarianceThreshold transformer from scikit-learn.

This documentation is auto-generated from JSON schemas.

Parameters
threshold (*union type, default 0*) – Features with a training-set variance lower than this threshold will be removed. The default is to keep all features with non-zero variance, i.e. remove the features that have the same value in all samples.
- float, >0, <=1 for optimizer, loguniform distribution, default 0

Features with a training-set variance lower than this threshold will be removed. The default is to keep all features with non-zero variance, i.e. remove the features that have the same value in all samples.

- *or* 0

  Keep all features with non-zero variance, i.e. remove the features that have the same value in all samples

fit(*X*, *y=None*, ***fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

- **X** (array *of* items : array *of* items : float) – Features; the outer array is over samples.
- **y** (`any type, optional`) – Target class labels (unused).

transform(*X*, *y=None*)

Transform the data.

Note: The transform method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (array *of* items : array *of* items : float) –

Returns

result

Return type

array *of* items : array *of* items : float

lale.lib.sklearn.voting_classifier module

class lale.lib.sklearn.voting_classifier.**VotingClassifier**(***, *estimators*, *voting='hard'*, *weights=None*, *n_jobs=None*, *flatten_transform=True*, *verbose=False*)

Bases: `PlannedIndividualOp`

Voting classifier from scikit-learn for voting ensemble.

This documentation is auto-generated from JSON schemas.

Parameters

- **estimators** (`array, not for optimizer`) – List of (string, estimator) tuples. Invoking the `fit` method on the `VotingClassifier` will fit clones.
  - items : tuple
    - item 0 : string
    - item 1 : union type
      - operator
      - *or* 'drop'
- **voting** ('hard' *or* 'soft', default 'hard') – If 'hard', uses predicted class labels for majority rule voting.

  See also *constraint-1*.
- **weights** (`union type, not for optimizer, default None`) – Sequence of weights (*float* or *int*) to weight the occurrences of
  - array *of* items : float
  - *or* None

- **n_jobs** (*union type, not for optimizer, default None*) – The number of jobs to run in parallel for `fit`.
  - integer
  - *or* None
- **flatten_transform** (*boolean, not for optimizer, default True*) – Affects shape of transform output only when voting='soft'

  See also *constraint-1*.
- **verbose** (*boolean, optional, not for optimizer, default False*) – If True, the time elapsed while fitting will be printed as it is completed.

Notes

constraint-1 : union type

Parameter: flatten_transform > only when voting='soft' if voting='soft' and flatten_transform=true

- voting : 'soft'
- *or* flatten_transform : True

fit(*X*, *y=None*, ***fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

- **X** (array *of* items : array *of* items : float) – Training vectors, where n_samples is the number of samples and n_features is the number of features.
- **y** (*union type*) – The target values (class labels).
  - array *of* items : float
  - *or* array *of* items : string
  - *or* array *of* items : boolean
- **sample_weight** (*union type, optional*) – Sample weights. If None, then samples are equally weighted.
  - array *of* items : float
  - *or* None

predict(*X*, ***predict_params*)

Make predictions.

Note: The predict method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (array, optional *of* items : array *of* items : float) – The input samples.

Returns

result – Predicted class labels.

Return type

array *of* items : float

predict_proba(*X*)

Probability estimates for all classes.

Note: The predict_proba method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (array, optional *of* items : array *of* items : float) – The input samples.

Returns
result – Weighted average probability for each class per sample.

Return type
array *of* items : array *of* items : float

`transform`(*X*, *y=None*)

Transform the data.

Note: The transform method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters
X (array, optional *of* items : array *of* items : float) – Training vectors, where n_samples is the number of samples and

Returns
result – If *voting='soft'* and *flatten_transform=True*:

- items : array
  - items : union type
    - float
    - *or* array *of* items : float

Return type
array

lale.lib.sklearn.voting_regressor module

class `lale.lib.sklearn.voting_regressor.VotingRegressor`(**, estimators, weights=None, n_jobs=None, verbose=False*)

Bases: `PlannedIndividualOp`

Voting classifier from scikit-learn for voting ensemble.

This documentation is auto-generated from JSON schemas.

Parameters

- **estimators** (*array, not for optimizer*) – List of (string, estimator) tuples. Invoking the `fit` method on the `VotingClassifier` will fit clones.
  - items : tuple
    - item 0 : string
    - item 1 : union type
      - operator
      - *or* 'drop'
- **weights** (*union type, default None*) – Sequence of weights (*float* or *int*) to weight the occurrences of
  - array *of* items : float
  - *or* None
- **n_jobs** (*union type, not for optimizer, default None*) – The number of jobs to run in parallel for `fit`.
  - integer
  - *or* None
- **verbose** (*boolean, optional, not for optimizer, default False*) – If True, the time elapsed while fitting will be printed as it is completed.

`fit`(*X*, *y=None*, ***fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

- **X** (array *of* items : array *of* items : float) – Input samples.
- **y** (array *of* items : float) – Target values.
- **sample_weight** (*union type, optional*) – Sample weights. If None, then samples are equally weighted.
  - array *of* items : float
  - *or* None

predict(*X*, ***predict_params*)

Make predictions.

Note: The predict method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (array, optional *of* items : array *of* items : float) – The input samples.

Returns

result – Predicted class labels.

Return type

array *of* items : float

transform(*X*, *y=None*)

Transform the data.

Note: The transform method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (array, optional *of* items : array *of* items : float) – Input samples

Returns

result – Values predicted by each regressor

- items : array
  - items : union type
    - float
    - *or* array *of* items : float

Return type

array

Module contents

Schema-enhanced versions of some of the operators from scikit-learn to enable hyperparameter tuning.

Operators

Classifiers:

- lale.lib.sklearn. AdaBoostClassifier
- lale.lib.sklearn. BaggingClassifier
- lale.lib.sklearn. DecisionTreeClassifier
- lale.lib.sklearn. DummyClassifier
- lale.lib.sklearn. ExtraTreesClassifier

- lale.lib.sklearn. GaussianNB
- lale.lib.sklearn. GradientBoostingClassifier
- lale.lib.sklearn. KNeighborsClassifier
- lale.lib.sklearn. LinearSVC
- lale.lib.sklearn. LogisticRegression
- lale.lib.sklearn. MLPClassifier
- lale.lib.sklearn. MultinomialNB
- lale.lib.sklearn. PassiveAggressiveClassifier
- lale.lib.sklearn. Perceptron
- lale.lib.sklearn. RandomForestClassifier
- lale.lib.sklearn. RidgeClassifier
- lale.lib.sklearn. SGDClassifier
- lale.lib.sklearn. StackingClassifier
- lale.lib.sklearn. SVC
- lale.lib.sklearn. VotingClassifier

Regressors:

- lale.lib.sklearn. AdaBoostRegressor
- lale.lib.sklearn. BaggingRegressor
- lale.lib.sklearn. DecisionTreeRegressor
- lale.lib.sklearn. DummyRegressor
- lale.lib.sklearn. ExtraTreesRegressor
- lale.lib.sklearn. GradientBoostingRegressor
- lale.lib.sklearn. KNeighborsRegressor
- lale.lib.sklearn. LinearRegression
- lale.lib.sklearn. LinearSVR
- lale.lib.sklearn. MultiOutputRegressor
- lale.lib.sklearn. RandomForestRegressor
- lale.lib.sklearn. Ridge
- lale.lib.sklearn. SGDRegressor
- lale.lib.sklearn. StackingRegressor
- lale.lib.sklearn. SVR
- lale.lib.sklearn. VotingRegressor

Transformers:

- lale.lib.sklearn. ColumnTransformer
- lale.lib.sklearn. FeatureAgglomeration
- lale.lib.sklearn. FunctionTransformer

- lale.lib.sklearn. IsolationForest
- lale.lib.sklearn. Isomap
- lale.lib.sklearn. MinMaxScaler
- lale.lib.sklearn. MissingIndicator
- lale.lib.sklearn. NMF
- lale.lib.sklearn. Normalizer
- lale.lib.sklearn. Nystroem
- lale.lib.sklearn. OneHotEncoder
- lale.lib.sklearn. OrdinalEncoder
- lale.lib.sklearn. PCA
- lale.lib.sklearn. PolynomialFeatures
- lale.lib.sklearn. QuadraticDiscriminantAnalysis
- lale.lib.sklearn. QuantileTransformer
- lale.lib.sklearn. RFE
- lale.lib.sklearn. RobustScaler
- lale.lib.sklearn. SelectKBest
- lale.lib.sklearn. SimpleImputer
- lale.lib.sklearn. StandardScaler
- lale.lib.sklearn. TargetEncoder
- lale.lib.sklearn. TfidfVectorizer
- lale.lib.sklearn. VarianceThreshold

Estimators and transformers:

- lale.lib.sklearn. Pipeline

Clustering:

- lale.lib.sklearn. KMeans

lale.lib.snapml package

Submodules

lale.lib.snapml.batched_tree_ensemble_classifier module

class `lale.lib.snapml.batched_tree_ensemble_classifier.`**`BatchedTreeEnsembleClassifier`**

Bases: *`PlannedIndividualOp`*

Batched Tree Ensemble Classifier from Snap ML.

This documentation is auto-generated from JSON schemas.

fit(*X*, *y=None*, ***fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

- **X** (*array*) – The outer array is over samples aka rows.
  - items : array *of* items : float

    The inner array is over features aka columns.
- **y** (*union type*) – The classes.
  - array *of* items : float
  - *or* array *of* items : string
  - *or* array *of* items : boolean
- **sample_weight** (*union type, optional, default None*) – Sample weights.
  - array *of* items : float
  - *or* None

    Samples are equally weighted.

partial_fit(*X*, *y=None*, ***fit_params*)

Incremental fit to train train the operator on a batch of samples.

Note: The partial_fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

predict(*X*, ***predict_params*)

Make predictions.

Note: The predict method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (*array*) – The outer array is over samples aka rows.

- items : array *of* items : float

  The inner array is over features aka columns.

Returns

result – The predicted classes.

- array *of* items : float
- *or* array *of* items : string
- *or* array *of* items : boolean

Return type

union type

predict_proba(*X*)

Probability estimates for all classes.

Note: The predict_proba method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (*array, optional*) – The outer array is over samples aka rows.

- items : array *of* items : float

  The inner array is over features aka columns.

Returns

result – The outer array is over samples aka rows.

- items : array *of* items : float
  The inner array contains probabilities corresponding to each class.

Return type
array

lale.lib.snapml.batched_tree_ensemble_regressor module

class lale.lib.snapml.batched_tree_ensemble_regressor.**BatchedTreeEnsembleRegressor**

Bases: *PlannedIndividualOp*

Batched Tree Ensemble Regressor from Snap ML.

This documentation is auto-generated from JSON schemas.

fit(*X*, *y=None*, ***fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

- **X** (*array*) – The outer array is over samples aka rows.
  - items : array *of* items : float
    The inner array is over features aka columns.
- **y** (*union type*) – The classes.
  - array *of* items : float
  - *or* array *of* items : string
  - *or* array *of* items : boolean
- **sample_weight** (*union type, optional, default None*) – Sample weights.
  - array *of* items : float
  - *or* None
    Samples are equally weighted.

partial_fit(*X*, *y=None*, ***fit_params*)

Incremental fit to train train the operator on a batch of samples.

Note: The partial_fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

predict(*X*, ***predict_params*)

Make predictions.

Note: The predict method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (*array*) – The outer array is over samples aka rows.

- items : array *of* items : float
  The inner array is over features aka columns.

Returns

result – The predicted classes.

- array *of* items : float
- *or* array *of* items : string
- *or* array *of* items : boolean

Return type
union type

`predict_proba`(*X*)

Probability estimates for all classes.

Note: The predict_proba method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (*array, optional*) – The outer array is over samples aka rows.

- items : array *of* items : float

  The inner array is over features aka columns.

Returns

result – The outer array is over samples aka rows.

- items : array *of* items : float

  The inner array contains probabilities corresponding to each class.

Return type
array

lale.lib.snapml.snap_boosting_machine_classifier module

class lale.lib.snapml.snap_boosting_machine_classifier.**SnapBoostingMachineClassifier**(*, *num_round=100*, *learning_rate=0.1*, *random_state=0*, *colsample_bytree=1.0*, *subsample=1.0*, *verbose=False*, *lambda_l2=0.0*, *early_stopping_rounds=*, *compress_trees=False*, *base_score=None*, *class_weight=None*, *max_depth=None*, *min_max_depth=1*, *max_max_depth=5*, *n_jobs=1*, *use_histograms=True*, *hist_nbins=256*, *use_gpu=False*, *gpu_id=0*, *tree_select_probability=*, *regularizer=1.0*, *fit_intercept=False*, *gamma=1.0*, *n_components=10*)

Bases: `PlannedIndividualOp`

Boosting machine classifier from Snap ML. It can be used for binary classification problems.

This documentation is auto-generated from JSON schemas.

Parameters

- **num_round** (*integer, >=1, >=100 for optimizer, <=1000 for optimizer, default 100*) – Number of boosting iterations.
- **learning_rate** (*float, >0.0, >=0.01 for optimizer, <=0.3 for optimizer, uniform distribution, default 0.1*) – Learning rate / shrinkage factor.
- **random_state** (*integer, not for optimizer, default 0*) – Random seed.
- **colsample_bytree** (*float, >0.0, <=1.0, not for optimizer, default 1.0*) – Fraction of feature columns used at each boosting iteration.
- **subsample** (*float, >0.0, <=1.0, not for optimizer, default 1.0*) – Fraction of training examples used at each boosting iteration.
- **verbose** (*boolean, not for optimizer, default False*) – Print off information during training.
- **lambda_l2** (*float, >=0.0, not for optimizer, default 0.0*) – L2-reguralization penalty used during tree-building.

- **early_stopping_rounds** (*integer, >=1, not for optimizer, default 10*) – When a validation set is provided, training will stop if the validation loss does not increase after a fixed number of rounds.
- **compress_trees** (*boolean, not for optimizer, default False*) – Compress trees after training for fast inference.
- **base_score** (*union type, not for optimizer, default None*) – Base score to initialize boosting algorithm. If None then the algorithm will initialize the base score to be the the logit of the probability of the positive class.
  - float
  - *or* None
- **class_weight** ('balanced' *or* None, not for optimizer, default None) – If set to 'balanced' samples weights will be applied to account for class imbalance, otherwise no sample weights will be used.
- **max_depth** (*union type, not for optimizer, default None*) – If set, will set min_max_depth = max_depth = max_max_depth
  - integer, >=1
  - *or* None
- **min_max_depth** (*integer, >=1, >=1 for optimizer, <=5 for optimizer, default 1*) – Minimum max_depth of trees in the ensemble.
- **max_max_depth** (*integer, >=1, >=5 for optimizer, <=10 for optimizer, default 5*) – Maximum max_depth of trees in the ensemble.
- **n_jobs** (*integer, >=1, not for optimizer, default 1*) – Number of threads to use during training.
- **use_histograms** (*boolean, not for optimizer, default True*) – Use histograms to accelerate tree-building.

  See also *constraint-1*.
- **hist_nbins** (*integer, not for optimizer, default 256*) – Number of histogram bins.
- **use_gpu** (*boolean, not for optimizer, default False*) – Use GPU for tree-building.

  See also *constraint-1*.
- **gpu_id** (*integer, not for optimizer, default 0*) – Device ID for GPU to use during training.
- **tree_select_probability** (*float, >=0.0, <=1.0, not for optimizer, default 1.0*) – Probability of selecting a tree (rather than a kernel ridge regressor) at each boosting iteration.
- **regularizer** (*float, >=0.0, not for optimizer, default 1.0*) – L2-regularization penality for the kernel ridge regressor.
- **fit_intercept** (*boolean, not for optimizer, default False*) – Include intercept term in the kernel ridge regressor.
- **gamma** (*float, >=0.0, not for optimizer, default 1.0*) – Guassian kernel parameter.
- **n_components** (*integer, >=1, not for optimizer, default 10*) – Number of components in the random projection.

Notes

constraint-1 : union type

GPU only supported for histogram-based splits.

- use_gpu : False
- *or* use_histograms : True

fit(*X*, *y=None*, ***fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

- **X** (*array*) – The outer array is over samples aka rows.
  - items : array *of* items : float

    The inner array is over features aka columns.
- **y** (*union type*) – The classes.
  - array *of* items : float
  - *or* array *of* items : string
  - *or* array *of* items : boolean
- **sample_weight** (*union type, optional, default None*) – Sample weights.
  - array *of* items : float
  - *or* None

    Samples are equally weighted.
- **X_val** (*union type, optional, default None*) –
  - array

    The outer array is over validation samples aka rows.
    - items : array *of* items : float

      The inner array is over features aka columns.
  - *or* None

    No validation set provided.
- **y_val** (*union type, optional, default None*) – The validation classes.
  - array *of* items : float
  - *or* array *of* items : string
  - *or* array *of* items : boolean
  - *or* None

    No validation set provided.
- **sample_weight_val** (*union type, optional, default None*) – Validation sample weights.
  - array *of* items : float
  - *or* None

    Validation samples are equally weighted.

predict(*X*, ***predict_params*)

Make predictions.

Note: The predict method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

- **X** (*array*) – The outer array is over samples aka rows.
  - items : array *of* items : float

    The inner array is over features aka columns.

- **n_jobs** (*integer, >=1, optional, default 1*) – Number of threads used to run inference.

Returns

result – The predicted classes.

- array *of* items : float
- *or* array *of* items : string
- *or* array *of* items : boolean

Return type
union type

`predict_proba(X)`

Probability estimates for all classes.

Note: The predict_proba method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

- **X** (*array, optional*) – The outer array is over samples aka rows.
  - items : array *of* items : float
    The inner array is over features aka columns.
- **n_jobs** (*integer, >=1, optional, default 1*) – Number of threads used to run inference.

Returns

result – The outer array is over samples aka rows.

- items : array *of* items : float
  The inner array contains probabilities corresponding to each class.

Return type
array

lale.lib.snapml.snap_boosting_machine_regressor module

class lale.lib.snapml.snap_boosting_machine_regressor.**SnapBoostingMachineRegressor**(**, num_round=100, objective='mse', learning_rate=0.1, random_state=0, colsample_bytree=1.0, subsample=1.0, verbose=False, lambda_l2=0.0, early_stopping_rounds=1, compress_trees=False, base_score=None, max_depth=None, min_max_depth=1, max_max_depth=5, n_jobs=1, use_histograms=True, hist_nbins=256, use_gpu=False, gpu_id=0, tree_select_probability=1., regularizer=1.0, fit_intercept=False, gamma=1.0, n_components=10*)

Bases: *PlannedIndividualOp*

Boosting machine Regressor from Snap ML.

This documentation is auto-generated from JSON schemas.

Parameters

- **num_round** (*integer, >=1, >=100 for optimizer, <=1000 for optimizer, optional, default 100*) – Number of boosting iterations.
- **objective** ('mse' *or* 'cross_entropy', optional, not for optimizer, default 'mse') – Training objective.
- **learning_rate** (*float, >0.0, >=0.01 for optimizer, <=0.3 for optimizer, uniform distribution, optional, default 0.1*) – Learning rate / shrinkage factor.
- **random_state** (*integer, optional, not for optimizer, default 0*) – Random seed.
- **colsample_bytree** (*float, >0.0, <=1.0, optional, not for optimizer, default 1.0*) – Fraction of feature columns used at each boosting iteration.
- **subsample** (*float, >0.0, <=1.0, optional, not for optimizer, default 1.0*) – Fraction of training examples used at each boosting iteration.
- **verbose** (*boolean, optional, not for optimizer, default False*) – Print off information during training.
- **lambda_l2** (*float, >=0.0, optional, not for optimizer, default 0.0*)

– L2-reguralization penalty used during tree-building.

- **early_stopping_rounds** (*integer, >=1, optional, not for optimizer, default 10*) – When a validation set is provided, training will stop if the validation loss does not increase after a fixed number of rounds.
- **compress_trees** (*boolean, optional, not for optimizer, default False*) – Compress trees after training for fast inference.
- **base_score** (*union type, optional, not for optimizer, default None*) – Base score to initialize boosting algorithm. If None then the algorithm will initialize the base score to be the the logit of the probability of the positive class.
  - float
  - *or* None
- **max_depth** (*union type, optional, not for optimizer, default None*) – If set, will set min_max_depth = max_depth = max_max_depth
  - integer, >=1
  - *or* None
- **min_max_depth** (*integer, >=1, >=1 for optimizer, <=5 for optimizer, optional, default 1*) – Minimum max_depth of trees in the ensemble.
- **max_max_depth** (*integer, >=1, >=5 for optimizer, <=10 for optimizer, optional, default 5*) – Maximum max_depth of trees in the ensemble.
- **n_jobs** (*integer, >=1, optional, not for optimizer, default 1*) – Number of threads to use during training.
- **use_histograms** (*boolean, optional, not for optimizer, default True*) – Use histograms to accelerate tree-building.

  See also *constraint-1*.
- **hist_nbins** (*integer, optional, not for optimizer, default 256*) – Number of histogram bins.
- **use_gpu** (*boolean, optional, not for optimizer, default False*) – Use GPU for tree-building.

  See also *constraint-1*.
- **gpu_id** (*integer, optional, not for optimizer, default 0*) – Device ID for GPU to use during training.
- **tree_select_probability** (*float, >=0.0, <=1.0, optional, not for optimizer, default 1.0*) – Probability of selecting a tree (rather than a kernel ridge regressor) at each boosting iteration.
- **regularizer** (*float, >=0.0, optional, not for optimizer, default 1.0*) – L2-regularization penality for the kernel ridge regressor.
- **fit_intercept** (*boolean, optional, not for optimizer, default False*) – Include intercept term in the kernel ridge regressor.
- **gamma** (*float, >=0.0, optional, not for optimizer, default 1.0*) – Guassian kernel parameter.
- **n_components** (*integer, >=1, optional, not for optimizer, default 10*) – Number of components in the random projection.

Notes

constraint-1 : union type

GPU only supported for histogram-based splits.

- use_gpu : False
- *or* use_histograms : True

`fit`(*X*, *y=None*, ***fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

- **X** (*array*) – The outer array is over samples aka rows.
  - items : array *of* items : float
    The inner array is over features aka columns.
- **y** (union type *of* array *of* items : float) – The regression target.
- **sample_weight** (*union type, optional, default None*) – Sample weights.
  - array *of* items : float
  - *or* None
    Samples are equally weighted.
- **X_val** (*union type, optional, default None*) –
  - array
    The outer array is over validation samples aka rows.
    * items : array *of* items : float
      The inner array is over features aka columns.
  - *or* None
    No validation set provided.
- **y_val** (*union type, optional, default None*) – The validation regression target.
  - array *of* items : float
  - *or* None
    No validation set provided.
- **sample_weight_val** (*union type, optional, default None*) – Validation sample weights.
  - array *of* items : float
  - *or* None
    Validation samples are equally weighted.

`predict`(*X*, ***predict_params*)

Make predictions.

Note: The predict method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

- **X** (*array*) – The outer array is over samples aka rows.
  - items : array *of* items : float
    The inner array is over features aka columns.
- **n_jobs** (*integer, >=1, optional, default 1*) – Number of threads used to run inference.

Returns

result – The predicted values.

Return type
union type *of* array *of* items : float

lale.lib.snapml.snap_decision_tree_classifier module

class lale.lib.snapml.snap_decision_tree_classifier.**SnapDecisionTreeClassifier**(**, criterion='gini', splitter='best', max_depth=None, min_samples_leaf=1, max_features=None, random_state=None, n_jobs=1, use_histograms=True, hist_nbins=256, use_gpu=False, gpu_id=0, verbose=False*)

Bases: *PlannedIndividualOp*

Decision tree classifier from Snap ML. It can be used for binary classification problems.

This documentation is auto-generated from JSON schemas.

Parameters

- **criterion** (*'gini', optional, not for optimizer, default 'gini'*) – Function to measure the quality of a split.
- **splitter** (*'best', optional, not for optimizer, default 'best'*) – The strategy used to choose the split at each node.
- **max_depth** (*union type, optional, default None*) – The maximum depth of the tree.
  - integer, >=1, >=3 for optimizer, <=5 for optimizer
  - *or* None
    Nodes are expanded until all leaves are pure or until all leaves contain less than min_samples_leaf samples.
- **min_samples_leaf** (*union type, optional, not for optimizer, default 1*) – The minimum number of samples required to be at a leaf node.
  - integer, >=1, <='X/maxItems', not for optimizer
    Consider min_samples_leaf as the minimum number.
  - *or* float, >0.0, <=0.5
    min_samples_leaf is a fraction and ceil(min_samples_leaf * n_samples) are the minimum number of samples for each node.
- **max_features** (*union type, optional, default None*) – The number of features to consider when looking for the best split.
  - integer, >=1, <='X/items/maxItems', not for optimizer
    Consider max_features features at each split.
  - *or* float, >0.0, >=0.1 for optimizer, <=1.0, <=0.9 for optimizer, uniform distribution
    max_features is a fraction and int(max_features * n_features) features are considered at each split.
  - *or* 'auto', 'sqrt', 'log2', *or* None

- **random_state** (*union type, optional, not for optimizer, default None*) – Seed of pseudo-random number generator.
  - None
    RandomState used by np.random
  - *or* integer
    Explicit seed.
- **n_jobs** (*integer, >=1, optional, not for optimizer, default 1*) – Number of CPU threads to use.
- **use_histograms** (*boolean, optional, not for optimizer, default True*) – Use histogram-based splits rather than exact splits.

  See also *constraint-1*.
- **hist_nbins** (*integer, >=1, >=16 for optimizer, <=256, <=256 for optimizer, optional, default 256*) – Number of histogram bins.
- **use_gpu** (*boolean, optional, not for optimizer, default False*) – Use GPU acceleration (only supported for histogram-based splits).

  See also *constraint-1*.
- **gpu_id** (*integer, optional, not for optimizer, default 0*) – Device ID of the GPU which will be used when GPU acceleration is enabled.
- **verbose** (*boolean, optional, not for optimizer, default False*) – If True, it prints debugging information while training. Warning: this will increase the training time. For performance evaluation, use verbose=False.

Notes

constraint-1 : union type
GPU only supported for histogram-based splits.
- use_gpu : False
- *or* use_histograms : True

fit(*X, y=None, **fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters
- **X** (*array*) – The outer array is over samples aka rows.
  - items : array *of* items : float
    The inner array is over features aka columns.
- **y** (*union type*) – The classes.
  - array *of* items : float
  - *or* array *of* items : string
  - *or* array *of* items : boolean
- **sample_weight** (*union type, optional, default None*) – Sample weights.
  - array *of* items : float
  - *or* None
    Samples are equally weighted.

predict(*X, **predict_params*)

Make predictions.

Note: The predict method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

- **X** (*array*) – The outer array is over samples aka rows.
  - items : array *of* items : float
    The inner array is over features aka columns.
- **n_jobs** (*integer, >=0, optional, default 0*) – Number of threads used to run inference. By default inference runs with maximum number of available threads.

Returns

result – The predicted classes.

- array *of* items : float
- *or* array *of* items : string
- *or* array *of* items : boolean

Return type

union type

`predict_proba(X)`

Probability estimates for all classes.

Note: The predict_proba method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

- **X** (*array, optional*) – The outer array is over samples aka rows.
  - items : array *of* items : float
    The inner array is over features aka columns.
- **n_jobs** (*integer, >=0, optional, default 0*) – Number of threads used to run inference. By default inference runs with maximum number of available threads..

Returns

result – The outer array is over samples aka rows.

- items : array *of* items : float
  The inner array contains probabilities corresponding to each class.

Return type

array

lale.lib.snapml.snap_decision_tree_regressor module

`class lale.lib.snapml.snap_decision_tree_regressor.SnapDecisionTreeRegressor(*, criterion='mse', splitter='best', max_depth=None, min_samples_leaf=1, max_features=None, random_state=None, n_jobs=1, use_histograms=True, hist_nbins=256, use_gpu=False, gpu_id=0, verbose=False)`

Bases: `PlannedIndividualOp`

Decision tree Regressor from Snap ML.

This documentation is auto-generated from JSON schemas.

Parameters

- `criterion` (*'mse', optional, not for optimizer, default 'mse'*) – Function to measure the quality of a split.
- `splitter` (*'best', optional, not for optimizer, default 'best'*) – The strategy used to choose the split at each node.
- `max_depth` (*union type, optional, default None*) – The maximum depth of the tree.
  - integer, >=1, >=3 for optimizer, <=5 for optimizer
  - *or* None

    Nodes are expanded until all leaves are pure or until all leaves contain less than min_samples_leaf samples.
- `min_samples_leaf` (*union type, optional, not for optimizer, default 1*) – The minimum number of samples required to be at a leaf node.
  - integer, >=1, <='X/maxItems', not for optimizer

    Consider min_samples_leaf as the minimum number.
  - *or* float, >0.0, <=0.5

    min_samples_leaf is a fraction and ceil(min_samples_leaf * n_samples) are the minimum number of samples for each node.
- `max_features` (*union type, optional, default None*) – The number of features to consider when looking for the best split.
  - integer, >=1, <='X/items/maxItems', not for optimizer

    Consider max_features features at each split.
  - *or* float, >0.0, >=0.1 for optimizer, <=1.0, <=0.9 for optimizer, uniform distribution

    max_features is a fraction and int(max_features * n_features) features are considered at each split.
  - *or* 'auto', 'sqrt', 'log2', *or* None
- `random_state` (*union type, optional, not for optimizer, default None*) – Seed of pseudo-random number generator.
  - None

    RandomState used by np.random
  - *or* integer

    Explicit seed.
- `n_jobs` (*integer, >=1, optional, not for optimizer, default 1*) – Number of CPU threads to use.
- `use_histograms` (*boolean, optional, not for optimizer, default True*) – Use histogram-based splits rather than exact splits.

  See also *constraint-1*.
- `hist_nbins` (*integer, >=1, >=16 for optimizer, <=256, <=256 for optimizer, optional, default 256*) – Number of histogram bins.
- `use_gpu` (*boolean, optional, not for optimizer, default False*) – Use GPU acceleration (only supported for histogram-based splits).

  See also *constraint-1*.
- `gpu_id` (*integer, optional, not for optimizer, default 0*) – Device ID of the GPU which will be used when GPU acceleration is enabled.
- `verbose` (*boolean, optional, not for optimizer, default False*) – If True, it prints debugging information while training. Warning: this will increase the training time. For performance evaluation, use verbose=False.

Notes

constraint-1 : union type

GPU only supported for histogram-based splits.

- use_gpu : False
- *or* use_histograms : True

fit(*X*, *y=None*, ***fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

- **X** (*array*) – The outer array is over samples aka rows.
  - items : array *of* items : float

    The inner array is over features aka columns.
- **y** (union type *of* array *of* items : float) – The regression target.
- **sample_weight** (*union type, optional, default None*) – Sample weights.
  - array *of* items : float
  - *or* None

    Samples are equally weighted.

predict(*X*, ***predict_params*)

Make predictions.

Note: The predict method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

- **X** (*array*) – The outer array is over samples aka rows.
  - items : array *of* items : float

    The inner array is over features aka columns.
- **n_jobs** (*integer, >=0, optional, default 0*) – Number of threads used to run inference. By default inference runs with maximum number of available threads.

Returns

result – The predicted values.

Return type

union type *of* array *of* items : float

lale.lib.snapml.snap_linear_regression module

class lale.lib.snapml.snap_linear_regression.**SnapLinearRegression**(*, *max_iter=100*, *regularizer=1.0*, *use_gpu=False*, *device_ids=None*, *dual=True*, *verbose=False*, *n_jobs=1*, *penalty='l2'*, *tol=0.001*, *generate_training_history=None*, *privacy=False*, *eta=0.3*, *batch_size=100*, *privacy_epsilon=10.0*, *grad_clip=1.0*, *fit_intercept=False*, *intercept_scaling=1.0*, *normalize=False*, *kernel='linear'*, *gamma=1.0*, *n_components=100*, *random_state=None*)

Bases: `PlannedIndividualOp`

Linear Regression from Snap ML.

This documentation is auto-generated from JSON schemas.

Parameters

- **max_iter** (*integer, >=1, >=10 for optimizer, <=1000 for optimizer, optional, default 100*) – Maximum number of iterations used by the solver to converge.
- **regularizer** (*float, >0.0, >=1.0 for optimizer, <=100.0 for optimizer, uniform distribution, optional, default 1.0*) – Larger regularization values imply stronger regularization.
- **use_gpu** (*boolean, optional, not for optimizer, default False*) – Use GPU Acceleration.
- **device_ids** (*union type, optional, not for optimizer, default None*) – Device IDs of the GPUs which will be used when GPU acceleration is enabled.
  - None

    Use [0].
  - *or* array *of* items : integer
- **dual** (*boolean, optional, not for optimizer, default True*) – Use dual formulation (rather than primal).

  See also *constraint-1*, *constraint-2*.
- **verbose** (*boolean, optional, not for optimizer, default False*) – If True, it prints the training cost, one per iteration. Warning: this will increase the training time. For performance evaluation, use verbose=False.
- **n_jobs** (*integer, >=1, optional, not for optimizer, default 1*) – The number of threads used for running the training. The value of this parameter should be a multiple of 32 if the training is performed on GPU (use_gpu=True).
- **penalty** ('l1' *or* 'l2', optional, not for optimizer, default 'l2') – The regularization / penalty type. Possible values are 'l2' for L2 regularization (LinearRegression) or 'l1' for L1 regularization (SparseLinearRegression). L1 regularization is possible only for the primal optimization problem (dual=False).

  See also *constraint-1*, *constraint-3*.
- **tol** (*float, >0.0, optional, not for optimizer, default 0.001*) – The

tolerance parameter. Training will finish when maximum change in model coefficients is less than tol.

- **generate_training_history** ('summary', 'full', *or* None, optional, not for optimizer, default None) – Determines the level of summary statistics that are generated during training.
- **privacy** (*boolean, optional, not for optimizer, default False*) – Train the model using a differentially private algorithm.

  See also *constraint-2*, *constraint-3*, *constraint-4*.
- **eta** (*float, >0.0, optional, not for optimizer, default 0.3*) – Learning rate for the differentially private training algorithm.
- **batch_size** (*integer, >=1, optional, not for optimizer, default 100*) – Mini-batch size for the differentially private training algorithm.
- **privacy_epsilon** (*float, >0.0, optional, not for optimizer, default 10.0*) – Target privacy gaurantee. Learned model will be (privacy_epsilon, 0.01)-private.
- **grad_clip** (*float, >=0.0, optional, not for optimizer, default 1.0*) – Gradient clipping parameter for the differentially private training algorithm.
- **fit_intercept** (*boolean, optional, default False*) – Add bias term – note, may affect speed of convergence, especially for sparse datasets.

  See also *constraint-4*.
- **intercept_scaling** (*float, >0.0, optional, not for optimizer, default 1.0*) – Scaling of bias term. The inclusion of a bias term is implemented by appending an additional feature to the dataset. This feature has a constant value, that can be set using this parameter.
- **normalize** (*boolean, optional, not for optimizer, default False*) – Normalize rows of dataset (recommended for fast convergence).
- **kernel** ('rbf' *or* 'linear', optional, not for optimizer, default 'linear') – Approximate feature map of a specified kernel function.
- **gamma** (*float, >0.0, optional, not for optimizer, default 1.0*) – Parameter of RBF kernel: exp(-gamma * x^2).
- **n_components** (*integer, >=1, optional, not for optimizer, default 100*) – Dimensionality of the feature space when approximating a kernel function.
- **random_state** (*union type, optional, not for optimizer, default None*) – Seed of pseudo-random number generator.
  - None

    RandomState used by np.random
  - *or* integer

    Explicit seed.

Notes

constraint-1 : union type

L1 regularization is supported only for primal optimization problems.

- penalty : 'l2'
- *or* dual : False

constraint-2 : union type

Privacy only supported for primal objective functions.

- privacy : False
- *or* dual : False

constraint-3 : union type

Privacy only supported for L2-regularized objective functions.

- privacy : False

  - *or* penalty : 'l2'

constraint-4 : union type

Privacy not supported with fit_intercept=True.

  - privacy : False
  - *or* fit_intercept : False

fit(*X*, *y=None*, ***fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

- **X** (*array*) – The outer array is over samples aka rows.
  - **–** items : array *of* items : float

    The inner array is over features aka columns.
- **y** (union type *of* array *of* items : float) – The regression target.

predict(*X*, ***predict_params*)

Make predictions.

Note: The predict method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

- **X** (*array*) – The outer array is over samples aka rows.
  - **–** items : array *of* items : float

    The inner array is over features aka columns.
- **n_jobs** (*integer, >=0, optional, default 0*) – Number of threads used to run inference. By default inference runs with maximum number of available threads.

Returns

result – The predicted values.

Return type

union type *of* array *of* items : float

lale.lib.snapml.snap_logistic_regression module

class lale.lib.snapml.snap_logistic_regression.**SnapLogisticRegression**(*, *max_iter=100*, *regularizer=1.0*, *use_gpu=False*, *device_ids=None*, *class_weight=None*, *dual=True*, *verbose=False*, *n_jobs=1*, *penalty='l2'*, *tol=0.001*, *generate_training_history=None*, *privacy=False*, *eta=0.3*, *batch_size=100*, *privacy_epsilon=10.0*, *grad_clip=1.0*, *fit_intercept=True*, *intercept_scaling=1.0*, *normalize=True*, *kernel='linear'*, *gamma=1.0*, *n_components=100*, *random_state=None*)

Bases: `PlannedIndividualOp`

Logistic Regression from Snap ML.

This documentation is auto-generated from JSON schemas.

Parameters

- **max_iter** (*integer, >=1, >=10 for optimizer, <=1000 for optimizer, optional, default 100*) – Maximum number of iterations used by the solver to converge.
- **regularizer** (*float, >0.0, >=1.0 for optimizer, <=100.0 for optimizer, uniform distribution, optional, default 1.0*) – Larger regularization values imply stronger regularization.
- **use_gpu** (*boolean, optional, not for optimizer, default False*) – Use GPU Acceleration.
- **device_ids** (*union type, optional, not for optimizer, default None*) – Device IDs of the GPUs which will be used when GPU acceleration is enabled.
  - None

    Use [0].
  - *or* array *of* items : integer
- **class_weight** ('balanced' *or* None, optional, not for optimizer, default None) – If set to 'balanced' samples weights will be applied to account for class imbalance, otherwise no sample weights will be used.
- **dual** (*boolean, optional, not for optimizer, default True*) – Use dual formulation (rather than primal).

  See also *constraint-1*, *constraint-2*.
- **verbose** (*boolean, optional, not for optimizer, default False*) – If True, it prints the training cost, one per iteration. Warning: this will increase the training time. For performance evaluation, use verbose=False.
- **n_jobs** (*integer, >=1, optional, not for optimizer, default 1*) – The number of threads used for running the training. The value of this parameter should be a multiple of 32 if the training is performed on GPU (use_gpu=True).
- **penalty** ('l1' *or* 'l2', optional, not for optimizer, default 'l2') – The regularization / penalty type. Possible values are 'l2' for L2 regularization (LogisticRegression) or 'l1'

for L1 regularization (SparseLogisticRegression). L1 regularization is possible only for the primal optimization problem (dual=False).

See also *constraint-1*, *constraint-3*.

- **tol** (*float, >0.0, optional, not for optimizer, default 0.001*) – The tolerance parameter. Training will finish when maximum change in model coefficients is less than tol.
- **generate_training_history** ('summary', 'full', *or* None, optional, not for optimizer, default None) – Determines the level of summary statistics that are generated during training.
- **privacy** (*boolean, optional, not for optimizer, default False*) – Train the model using a differentially private algorithm.

  See also *constraint-2*, *constraint-3*, *constraint-4*.
- **eta** (*float, >0.0, optional, not for optimizer, default 0.3*) – Learning rate for the differentially private training algorithm.
- **batch_size** (*integer, >=1, optional, not for optimizer, default 100*) – Mini-batch size for the differentially private training algorithm.
- **privacy_epsilon** (*float, >0.0, optional, not for optimizer, default 10.0*) – Target privacy gaurantee. Learned model will be (privacy_epsilon, 0.01)-private.
- **grad_clip** (*float, >=0.0, optional, not for optimizer, default 1.0*) – Gradient clipping parameter for the differentially private training algorithm.
- **fit_intercept** (*boolean, optional, always print, default True*) – Add bias term – note, may affect speed of convergence, especially for sparse datasets.

  See also *constraint-4*.
- **intercept_scaling** (*float, >0.0, optional, not for optimizer, default 1.0*) – Scaling of bias term. The inclusion of a bias term is implemented by appending an additional feature to the dataset. This feature has a constant value, that can be set using this parameter.
- **normalize** (*boolean, optional, not for optimizer, always print, default True*) – Normalize rows of dataset (recommended for fast convergence).
- **kernel** ('rbf' *or* 'linear', optional, not for optimizer, default 'linear') – Approximate feature map of a specified kernel function.
- **gamma** (*float, >0.0, optional, not for optimizer, default 1.0*) – Parameter of RBF kernel: exp(-gamma * x^2).
- **n_components** (*integer, >=1, optional, not for optimizer, default 100*) – Dimensionality of the feature space when approximating a kernel function.
- **random_state** (*union type, optional, not for optimizer, default None*) – Seed of pseudo-random number generator.
  - None

    RandomState used by np.random
  - *or* integer

    Explicit seed.

Notes

constraint-1 : union type
L1 regularization is supported only for primal optimization problems.
- penalty : 'l2'
- *or* dual : False

constraint-2 : union type
Privacy only supported for primal objective functions.
- privacy : False
- *or* dual : False

constraint-3 : union type
Privacy only supported for L2-regularized objective functions.
- privacy : False
- *or* penalty : 'l2'

constraint-4 : union type
Privacy not supported with fit_intercept=True.
- privacy : False
- *or* fit_intercept : False

fit(*X*, *y=None*, ***fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters
- **X** (*array*) – The outer array is over samples aka rows.
  - items : array *of* items : float
    The inner array is over features aka columns.
- **y** (*union type*) – The classes.
  - array *of* items : float
  - *or* array *of* items : string
  - *or* array *of* items : boolean

predict(*X*, ***predict_params*)

Make predictions.

Note: The predict method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters
- **X** (*array*) – The outer array is over samples aka rows.
  - items : array *of* items : float
    The inner array is over features aka columns.
- **n_jobs** (*integer, >=0, optional, default 0*) – Number of threads used to run inference. By default inference runs with maximum number of available threads.

Returns

result – The predicted classes.
- array *of* items : float
- *or* array *of* items : string
- *or* array *of* items : boolean

Return type
union type

predict_proba(*X*)

Probability estimates for all classes.

Note: The predict_proba method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

- **X** (*array, optional*) – The outer array is over samples aka rows.
  - items : array *of* items : float
    The inner array is over features aka columns.
- **n_jobs** (*integer, >=0, optional, default 0*) – Number of threads used to run inference. By default inference runs with maximum number of available threads.

Returns

result – The outer array is over samples aka rows.

- items : array *of* items : float
  The inner array contains probabilities corresponding to each class.

Return type

array

lale.lib.snapml.snap_random_forest_classifier module

class lale.lib.snapml.snap_random_forest_classifier.**SnapRandomForestClassifier**(**, n_estimators=10, criterion='gini', max_depth=None, min_samples_leaf=1, max_features='auto', bootstrap=True, n_jobs=1, random_state=None, verbose=False, use_histograms=False, hist_nbins=256, use_gpu=False, gpu_ids=None*)

Bases: *PlannedIndividualOp*

Random forest classifier from Snap ML. It can be used for binary classification problems.

This documentation is auto-generated from JSON schemas.

Parameters

- **n_estimators** (*integer, >=1, >=10 for optimizer, <=100 for optimizer, optional, default 10*) – The number of trees in the forest.
- **criterion** (*'gini', optional, not for optimizer, default 'gini'*) – Function to measure the quality of a split.
- **max_depth** (*union type, optional, default None*) – The maximum depth of the tree.
  - integer, >=1, >=3 for optimizer, <=5 for optimizer
  - *or* None
    Nodes are expanded until all leaves are pure or until all leaves contain less than min_samples_leaf samples.

- **min_samples_leaf** (*union type, optional, not for optimizer, default 1*) – The minimum number of samples required to be at a leaf node.
  - integer, >=1, <='X/maxItems', not for optimizer
    Consider min_samples_leaf as the minimum number.
  - *or* float, >0.0, <=0.5
    min_samples_leaf is a fraction and ceil(min_samples_leaf * n_samples) are the minimum number of samples for each node.
- **max_features** (*union type, optional, default 'auto'*) – The number of features to consider when looking for the best split.
  - integer, >=1, <='X/items/maxItems', not for optimizer
    Consider max_features features at each split.
  - *or* float, >0.0, >=0.1 for optimizer, <=1.0, <=0.9 for optimizer, uniform distribution
    max_features is a fraction and int(max_features * n_features) features are considered at each split.
  - *or* 'auto', 'sqrt', 'log2', *or* None
- **bootstrap** (*boolean, optional, not for optimizer, default True*) – Whether bootstrap samples are used when building trees.
- **n_jobs** (*integer, >=1, optional, not for optimizer, default 1*) – Number of CPU threads to use.
- **random_state** (*union type, optional, not for optimizer, default None*) – Seed of pseudo-random number generator.
  - None
    RandomState used by np.random
  - *or* integer
    Explicit seed.
- **verbose** (*boolean, optional, not for optimizer, default False*) – If True, it prints debugging information while training. Warning: this will increase the training time. For performance evaluation, use verbose=False.
- **use_histograms** (*boolean, optional, not for optimizer, default False*) – Use histogram-based splits rather than exact splits.

  See also *constraint-1*.
- **hist_nbins** (*integer, optional, not for optimizer, default 256*) – Number of histogram bins.
- **use_gpu** (*boolean, optional, not for optimizer, default False*) – Use GPU acceleration (only supported for histogram-based splits).

  See also *constraint-1*.
- **gpu_ids** (*union type, optional, not for optimizer, default None*) – Device IDs of the GPUs which will be used when GPU acceleration is enabled.
  - None
    Use [0].
  - *or* array *of* items : integer

Notes

constraint-1 : union type
GPU only supported for histogram-based splits.
- use_gpu : False
- *or* use_histograms : True

fit(*X*, *y=None*, ***fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters
- **X** (*array*) – The outer array is over samples aka rows.
  - items : array *of* items : float
    The inner array is over features aka columns.
- **y** (*union type*) – The classes.
  - array *of* items : float
  - *or* array *of* items : string
  - *or* array *of* items : boolean
- **sample_weight** (*union type, optional, default None*) – Sample weights.
  - array *of* items : float
  - *or* None
    Samples are equally weighted.

predict(*X*, ***predict_params*)

Make predictions.

Note: The predict method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters
- **X** (*array*) – The outer array is over samples aka rows.
  - items : array *of* items : float
    The inner array is over features aka columns.
- **n_jobs** (*integer, >=0, optional, default 0*) – Number of threads used to run inference. By default inference runs with maximum number of available threads.

Returns

result – The predicted classes.
- array *of* items : float
- *or* array *of* items : string
- *or* array *of* items : boolean

Return type
union type

predict_proba(*X*)

Probability estimates for all classes.

Note: The predict_proba method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters
- **X** (*array, optional*) – The outer array is over samples aka rows.
  - items : array *of* items : float
    The inner array is over features aka columns.

- **n_jobs** (*integer, >=0, optional, default 0*) – Number of threads used to run inference. By default inference runs with maximum number of available threads..

Returns

result – The outer array is over samples aka rows.

- items : array *of* items : float

    The inner array contains probabilities corresponding to each class.

Return type

array

lale.lib.snapml.snap_random_forest_regressor module

`class lale.lib.snapml.snap_random_forest_regressor.SnapRandomForestRegressor(*, n_estimators=10, criterion='mse', max_depth=None, min_samples_leaf=1, max_features='auto', bootstrap=True, n_jobs=1, random_state=None, verbose=False, use_histograms=False, hist_nbins=256, use_gpu=False, gpu_ids=None)`

Bases: *PlannedIndividualOp*

Random forest regressor from Snap ML.

This documentation is auto-generated from JSON schemas.

Parameters

- **n_estimators** (*integer, >=1, >=10 for optimizer, <=100 for optimizer, optional, default 10*) – The number of trees in the forest.
- **criterion** (*'mse', optional, not for optimizer, default 'mse'*) – Function to measure the quality of a split.
- **max_depth** (*union type, optional, default None*) – The maximum depth of the tree.
    - integer, >=1, >=3 for optimizer, <=5 for optimizer
    - *or* None

        Nodes are expanded until all leaves are pure or until all leaves contain less than min_samples_leaf samples.
- **min_samples_leaf** (*union type, optional, not for optimizer, default 1*) – The minimum number of samples required to be at a leaf node.
    - integer, >=1, <='X/maxItems', not for optimizer

        Consider min_samples_leaf as the minimum number.
    - *or* float, >0.0, <=0.5

        min_samples_leaf is a fraction and ceil(min_samples_leaf * n_samples) are the minimum number of samples for each node.
- **max_features** (*union type, optional, default 'auto'*) – The number of features to consider when looking for the best split.
    - integer, >=1, <='X/items/maxItems', not for optimizer

        Consider max_features features at each split.

   - *or* float, >0.0, >=0.1 for optimizer, <=1.0, <=0.9 for optimizer, uniform distribution

     max_features is a fraction and int(max_features * n_features) features are considered at each split.
   - *or* 'auto', 'sqrt', 'log2', *or* None
- **bootstrap** (*boolean, optional, not for optimizer, default True*) – Whether bootstrap samples are used when building trees.
- **n_jobs** (*integer, >=1, optional, not for optimizer, default 1*) – Number of CPU threads to use.
- **random_state** (*union type, optional, not for optimizer, default None*) – Seed of pseudo-random number generator.
   - None

     RandomState used by np.random
   - *or* integer

     Explicit seed.
- **verbose** (*boolean, optional, not for optimizer, default False*) – If True, it prints debugging information while training. Warning: this will increase the training time. For performance evaluation, use verbose=False.
- **use_histograms** (*boolean, optional, not for optimizer, default False*) – Use histogram-based splits rather than exact splits.

  See also *constraint-1*.
- **hist_nbins** (*integer, optional, not for optimizer, default 256*) – Number of histogram bins.
- **use_gpu** (*boolean, optional, not for optimizer, default False*) – Use GPU acceleration (only supported for histogram-based splits).

  See also *constraint-1*.
- **gpu_ids** (*union type, optional, not for optimizer, default None*) – Device IDs of the GPUs which will be used when GPU acceleration is enabled.
   - None

     Use [0].
   - *or* array *of* items : integer

Notes

constraint-1 : union type

GPU only supported for histogram-based splits.
- use_gpu : False
- *or* use_histograms : True

fit(*X*, *y=None*, ***fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters
- **X** (*array*) – The outer array is over samples aka rows.
   - items : array *of* items : float

     The inner array is over features aka columns.
- **y** (union type *of* array *of* items : float) – The regression target.
- **sample_weight** (*union type, optional, default None*) – Sample weights.
   - array *of* items : float
   - *or* None

Samples are equally weighted.

predict(*X*, ***predict_params*)

Make predictions.

Note: The predict method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

- **X** (*array*) – The outer array is over samples aka rows.
  - items : array *of* items : float

    The inner array is over features aka columns.
- **n_jobs** (*integer, >=0, optional, default 0*) – Number of threads used to run inference. By default inference runs with maximum number of available threads.

Returns

result – The predicted values.

Return type

union type *of* array *of* items : float

lale.lib.snapml.snap_svm_classifier module

class lale.lib.snapml.snap_svm_classifier.**SnapSVMClassifier**(*, *max_iter=100*, *regularizer=1.0*, *use_gpu=False*, *device_ids=None*, *class_weight=None*, *verbose=False*, *n_jobs=1*, *tol=0.001*, *generate_training_history=None*, *fit_intercept=True*, *intercept_scaling=1.0*, *normalize=True*, *kernel='rbf'*, *gamma=1.0*, *n_components=100*, *random_state=None*)

Bases: *PlannedIndividualOp*

Support Vector Machine from Snap ML.

This documentation is auto-generated from JSON schemas.

Parameters

- **max_iter** (*integer, >=1, >=10 for optimizer, <=1000 for optimizer, optional, default 100*) – Maximum number of iterations used by the solver to converge.
- **regularizer** (*float, >0.0, >=1.0 for optimizer, <=100.0 for optimizer, uniform distribution, optional, default 1.0*) – Larger regularization values imply stronger regularization.
- **use_gpu** (*boolean, optional, not for optimizer, default False*) – Use GPU Acceleration.
- **device_ids** (*union type, optional, not for optimizer, default None*) – Device IDs of the GPUs which will be used when GPU acceleration is enabled.
  - None

    Use [0].
  - *or* array *of* items : integer
- **class_weight** ('balanced' *or* None, optional, not for optimizer, default None) – If set to 'balanced' samples weights will be applied to account for class imbalance, otherwise no sample weights will be used.

- **verbose** (*boolean, optional, not for optimizer, default False*) – If True, it prints the training cost, one per iteration. Warning: this will increase the training time. For performance evaluation, use verbose=False.
- **n_jobs** (*integer, >=1, optional, not for optimizer, default 1*) – The number of threads used for running the training. The value of this parameter should be a multiple of 32 if the training is performed on GPU (use_gpu=True).
- **tol** (*float, >0.0, optional, not for optimizer, default 0.001*) – The tolerance parameter. Training will finish when maximum change in model coefficients is less than tol.
- **generate_training_history** ('summary', 'full', *or* None, optional, not for optimizer, default None) – Determines the level of summary statistics that are generated during training.
- **fit_intercept** (*boolean, optional, default True*) – Add bias term – note, may affect speed of convergence, especially for sparse datasets.
- **intercept_scaling** (*float, >0.0, optional, not for optimizer, default 1.0*) – Scaling of bias term. The inclusion of a bias term is implemented by appending an additional feature to the dataset. This feature has a constant value, that can be set using this parameter.
- **normalize** (*boolean, optional, not for optimizer, default True*) – Normalize rows of dataset (recommended for fast convergence).
- **kernel** ('rbf' *or* 'linear', optional, default 'rbf') – Approximate feature map of a specified kernel function.
- **gamma** (*float, >0.0, >=0.01 for optimizer, <=100.0 for optimizer, uniform distribution, optional, default 1.0*) – Parameter of RBF kernel: exp(-gamma * x^2).
- **n_components** (*integer, >=1, >=10 for optimizer, <=200 for optimizer, optional, default 100*) – Dimensionality of the feature space when approximating a kernel function.
- **random_state** (*union type, optional, not for optimizer, default None*) – Seed of pseudo-random number generator.
  - None

    RandomState used by np.random
  - *or* integer

    Explicit seed.

decision_function(*X*)

Confidence scores for all classes.

Note: The decision_function method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

- **X** (*array, optional*) – The outer array is over samples aka rows.
  - items : array *of* items : float

    The inner array is over features aka columns.
- **n_jobs** (*integer, >=0, optional, default 0*) – Number of threads used to run inference. By default inference runs with maximum number of available threads.

Returns

result – The outer array is over samples aka rows.

- items : array *of* items : float

  The inner array contains confidence scores corresponding to each class.

Return type

array

fit(*X*, *y=None*, ***fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

- **X** (*array*) – The outer array is over samples aka rows.
  - items : array *of* items : float

    The inner array is over features aka columns.
- **y** (*union type*) – The classes.
  - array *of* items : float
  - *or* array *of* items : string
  - *or* array *of* items : boolean

predict(*X*, ***predict_params*)

Make predictions.

Note: The predict method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

- **X** (*array*) – The outer array is over samples aka rows.
  - items : array *of* items : float

    The inner array is over features aka columns.
- **n_jobs** (*integer, >=0, optional, default 0*) – Number of threads used to run inference. By default inference runs with maximum number of available threads.

Returns

result – The predicted classes.

- array *of* items : float
- *or* array *of* items : string
- *or* array *of* items : boolean

Return type

union type

Module contents

Schema-enhanced versions of the operators from Snap ML to enable hyperparameter tuning.

Operators

Classifiers:

- lale.lib.snapml. BatchedTreeEnsembleClassifier
- lale.lib.snapml. SnapBoostingMachineClassifier
- lale.lib.snapml. SnapDecisionTreeClassifier
- lale.lib.snapml. SnapLogisticRegression
- lale.lib.snapml. SnapRandomForestClassifier
- lale.lib.snapml. SnapSVMClassifier

Regressors:

- lale.lib.snapml. BatchedTreeEnsembleRegressor
- lale.lib.snapml. SnapBoostingMachineRegressor
- lale.lib.snapml. SnapDecisionTreeRegressor
- lale.lib.snapml. SnapLinearRegression
- lale.lib.snapml. SnapRandomForestRegressor

lale.lib.xgboost package

Submodules

lale.lib.xgboost.xgb_classifier module

class lale.lib.xgboost.xgb_classifier.**XGBClassifier**(**, max_depth=None, learning_rate=None, n_estimators, verbosity=None, objective='binary:logistic', booster=None, tree_method=None, n_jobs=1, nthread=None, gamma=None, min_child_weight=None, max_delta_step=None, subsample=None, colsample_bytree=None, colsample_bylevel=None, colsample_bynode=None, reg_alpha=None, reg_lambda=None, scale_pos_weight=None, base_score=None, random_state=0, missing=nan, silent=None, seed=None, monotone_constraints=None, interaction_constraints=None, num_parallel_tree=None, validate_parameters=None, gpu_id=None, importance_type='gain', use_label_encoder=False, enable_categorical=False, predictor=None, max_leaves=None, max_bin=None, grow_policy=None, sampling_method=None, max_cat_to_onehot=None, eval_metric=None, early_stopping_rounds=None, callbacks=None, feature_types, max_cat_threshold=None, device=None, multi_strategy=None*)

Bases: `PlannedIndividualOp`

XGBClassifier gradient boosted decision trees.

This documentation is auto-generated from JSON schemas.

Parameters

- **max_depth** (*union type, default None*) – Maximum tree depth for base learners.
  - integer, >=0, >=1 for optimizer, <=7 for optimizer, uniform distribution
  - *or* None, not for optimizer
- **learning_rate** (*union type, default None*) – Boosting learning rate (xgb's "eta")
  - float, >=0.02 for optimizer, <=1 for optimizer, loguniform distribution
  - *or* None, not for optimizer
- **n_estimators** (*union type*) – Number of trees to fit.

  - integer, >=50 for optimizer, <=1000 for optimizer, default 200
  - *or* None
- **verbosity** (*union type, >=0, <=3, not for optimizer, default None*) – The degree of verbosity.
  - integer
  - *or* None
- **objective** (*union type, not for optimizer, default 'binary:logistic'*) – Specify the learning task and the corresponding learning objective or a custom objective function to be used.
  - 'binary:hinge', 'binary:logistic', 'binary:logitraw', 'multi:softmax', *or* 'multi:softprob'
  - *or* callable
- **booster** ('gbtree', 'gblinear', 'dart', *or* None, not for optimizer, default None) – Specify which booster to use.
- **tree_method** ('auto', 'exact', 'approx', 'hist', 'gpu_hist', *or* None, not for optimizer, default None) – Specify which tree method to use. Default to auto. If this parameter is set to default, XGBoost will choose the most conservative option available. Refer to https://xgboost.readthedocs.io/en/latest/parameter.html.
- **n_jobs** (*union type, not for optimizer, default 1*) – Number of parallel threads used to run xgboost. (replaces `nthread`)
  - integer
  - *or* None
- **nthread** (*union type, optional, not for optimizer, default None*) – Number of parallel threads used to run xgboost. Deprecated, please use n_jobs
  - integer
  - *or* None
- **gamma** (*union type, default None*) – Minimum loss reduction required to make a further partition on a leaf node of the tree.
  - float, >=0, <=1.0 for optimizer
  - *or* None, not for optimizer
- **min_child_weight** (*union type, default None*) – Minimum sum of instance weight(hessian) needed in a child.
  - integer, >=2 for optimizer, <=20 for optimizer, uniform distribution
  - *or* None, not for optimizer
- **max_delta_step** (*union type, not for optimizer, default None*) – Maximum delta step we allow each tree's weight estimation to be.
  - None
  - *or* integer
- **subsample** (*union type, default None*) – Subsample ratio of the training instance.
  - float, >0, >=0.01 for optimizer, <=1.0 for optimizer, uniform distribution
  - *or* None, not for optimizer
- **colsample_bytree** (*union type, not for optimizer, default None*) – Subsample ratio of columns when constructing each tree.
  - float, >0, >=0.1 for optimizer, <=1, <=1.0 for optimizer, uniform distribution
  - *or* None, not for optimizer
- **colsample_bylevel** (*union type, not for optimizer, default None*) – Subsample ratio of columns for each split, in each level.
  - float, >0, >=0.1 for optimizer, <=1, <=1.0 for optimizer, uniform distribution
  - *or* None, not for optimizer
- **colsample_bynode** (*union type, not for optimizer, default None*) – Subsample ratio of columns for each split.
  - float, >0, <=1
  - *or* None, not for optimizer

- **reg_alpha** (*union type, default None*) – L1 regularization term on weights
  - float, >=0 for optimizer, <=1 for optimizer, uniform distribution
  - *or* None, not for optimizer
- **reg_lambda** (*union type, default None*) – L2 regularization term on weights
  - float, >=0.1 for optimizer, <=1 for optimizer, uniform distribution
  - *or* None, not for optimizer
- **scale_pos_weight** (*union type, not for optimizer, default None*) – Balancing of positive and negative weights.
  - float
  - *or* None, not for optimizer
- **base_score** (*union type, not for optimizer, default None*) – The initial prediction score of all instances, global bias.
  - float
  - *or* None, not for optimizer
- **random_state** (*union type, not for optimizer, default 0*) – Random number seed. (replaces seed)
  - integer
  - *or* None
- **missing** (*union type, not for optimizer, default nan*) – Value in the data which needs to be present as a missing value. If If None, defaults to np.nan.
  - float
  - *or* None *or* nan
- **silent** (*union type, optional, not for optimizer, default None*) – Deprecated and replaced with verbosity, but adding to be backward compatible.
  - boolean
  - *or* None
- **seed** (*any type, optional, not for optimizer, default None*) – deprecated and replaced with random_state, but adding to be backward compatible.
- **monotone_constraints** (*union type, optional, not for optimizer, default None*) – Constraint of variable monotonicity.
  - None
  - *or* string
- **interaction_constraints** (*union type, optional, not for optimizer, default None*) – Constraints for interaction representing permitted interactions. The constraints must be specified in the form of a nest list, e.g. [[0, 1], [2, 3, 4]], where each inner list is a group of indices of features that are allowed to interact with each other.
  - None
  - *or* string
- **num_parallel_tree** (*union type, optional, not for optimizer, default None*) – Used for boosting random forest.
  - None
  - *or* integer
- **validate_parameters** (*union type, optional, not for optimizer, default None*) – Give warnings for unknown parameter.
  - None
  - *or* boolean
  - *or* integer
- **gpu_id** (*union type, optional, not for optimizer, default None*) – Device ordinal.
  - integer
  - *or* None
- **importance_type** ('gain', 'weight', 'cover', 'total_gain', 'total_cover', *or* None, optional, not for optimizer, default 'gain') – The feature importance type for the *fea-*

*ture_importances_* property.

- **use_label_encoder** (*boolean, optional, not for optimizer, default False*) –

  (Deprecated) Use the label encoder from scikit-learn to encode the labels.

  For new code, we recommend that you set this parameter to False.
- **enable_categorical** (*boolean, optional, not for optimizer, default False*) – Experimental support for categorical data. Do not set to true unless you are interested in development. Only valid when gpu_hist and dataframe are used.
- **predictor** (*union type, optional, not for optimizer, default None*) – Force XGBoost to use specific predictor, available choices are [cpu_predictor, gpu_predictor].
  - string
  - *or* None
- **max_leaves** (*union type, optional, not for optimizer, default None*) – Maximum number of leaves; 0 indicates no limit.
  - integer
  - *or* None, not for optimizer
- **max_bin** (*union type, optional, not for optimizer, default None*) – If using histogram-based algorithm, maximum number of bins per feature.
  - integer
  - *or* None, not for optimizer
- **grow_policy** (0, 1, 'depthwise', 'lossguide', *or* None, optional, not for optimizer, default None) –

  Tree growing policy.

  0 or depthwise: favor splitting at nodes closest to the node, i.e. grow depth-wise.
  1 or lossguide: favor splitting at nodes with highest loss change.
- **sampling_method** ('uniform', 'gadient_based', *or* None, optional, not for optimizer, default None) –

  Sampling method. Used only by gpu_hist tree method.
  - uniform: select random training instances uniformly.
  - gradient_based select random training instances with higher probability when the gradient and hessian are larger. (cf. CatBoost)
- **max_cat_to_onehot** (*union type, optional, not for optimizer, default None*) –

  A threshold for deciding whether XGBoost should use

  one-hot encoding based split for categorical data.
  - integer
  - *or* None
- **eval_metric** (*union type, optional, not for optimizer, default None*) – Metric used for monitoring the training result and early stopping.
  - string
  - *or* array *of* items : string
  - *or* array *of* items : callable
  - *or* None
- **early_stopping_rounds** (*union type, optional, not for optimizer, default None*) –

  Activates early stopping.

  Validation metric needs to improve at least once in every early_stopping_rounds round(s) to continue training.
  - integer
  - *or* None
- **callbacks** (*union type, optional, not for optimizer, default None*) –

  List of callback functions that are applied at end of each iteration.

It is possible to use predefined callbacks by using Callback API.

- array *of* items : callable
- *or* None

- **feature_types** (*Any, optional, not for optimizer*) – Used for specifying feature types without constructing a dataframe. See DMatrix for details.
- **max_cat_threshold** (*union type, optional, not for optimizer, default None*) –

  Maximum number of categories considered for each split.

  Used only by partition-based splits for preventing over-fitting. Also, enable_categorical needs to be set to have categorical feature support. See Categorical Data and Parameters for Categorical Feature for details.

  - integer, >=0, >=1 for optimizer, <=10 for optimizer, uniform distribution
  - *or* None

- **device** (*union type, optional, not for optimizer, default None*) – Device ordinal
  - 'cpu', 'cuda', *or* 'gpu'
  - *or* None
- **multi_strategy** (*union type, optional, not for optimizer, default None*) –

  The strategy used for training multi-target models,

  including multi-target regression and multi-class classification. See Multiple Outputs for more information.

  - 'one_output_per_tree'
    One model for each target.
  - *or* 'multi_output_tree'
    Use multi-target trees.
  - *or* None

fit(*X, y=None, **fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

- **X** (array *of* items : array *of* items : float) – Feature matrix
- **y** (*union type*) – Labels
  - array *of* items : float
  - *or* array *of* items : string
  - *or* array *of* items : boolean
- **sample_weight** (*union type, optional, default None*) – Weight for each instance
  - array *of* items : float
  - *or* None
- **eval_set** (*union type, optional, default None*) – A list of (X, y) pairs to use as a validation set for
  - array
  - *or* None
- **sample_weight_eval_set** (*union type, optional, default None*) – A list of the form [L_1, L_2, …, L_n], where each L_i is a list of
  - array
  - *or* None
- **eval_metric** (*union type, optional, default None*) – If a str, should be a built-in evaluation metric to use. See

- array *of* items : string
- *or* string
- *or* None
- *or* dict

- **early_stopping_rounds** (*union type, optional, default None*) – Activates early stopping. Validation error needs to decrease at
  - integer
  - *or* None
- **verbose** (*boolean, optional, default True*) – If *verbose* and an evaluation set is used, writes the evaluation
- **xgb_model** (*union type, optional, default None*) – file name of stored xgb model or 'Booster' instance Xgb model to be
  - string
  - *or* None
- **callbacks** (*union type, optional, default None*) – List of callback functions that are applied at each iteration.
  - array *of* items : dict
  - *or* None

partial_fit(*X*, *y=None*, ***fit_params*)

Incremental fit to train train the operator on a batch of samples.

Note: The partial_fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

- **X** (array *of* items : array *of* items : float) – Feature matrix
- **y** (*union type*) – Labels
  - array *of* items : float
  - *or* array *of* items : string
  - *or* array *of* items : boolean
- **sample_weight** (*union type, optional, default None*) – Weight for each instance
  - array *of* items : float
  - *or* None
- **eval_set** (*union type, optional, default None*) – A list of (X, y) pairs to use as a validation set for
  - array
  - *or* None
- **sample_weight_eval_set** (*union type, optional, default None*) – A list of the form [L_1, L_2, …, L_n], where each L_i is a list of
  - array
  - *or* None
- **eval_metric** (*union type, optional, default None*) – If a str, should be a built-in evaluation metric to use. See
  - array *of* items : string
  - *or* string
  - *or* None
  - *or* dict
- **early_stopping_rounds** (*union type, optional, default None*) – Activates early stopping. Validation error needs to decrease at
  - integer
  - *or* None
- **verbose** (*boolean, optional, default True*) – If *verbose* and an evaluation set is used, writes the evaluation

- **xgb_model** (*union type, optional, default None*) – file name of stored xgb model or 'Booster' instance Xgb model to be
  - string
  - *or* None
- **callbacks** (*union type, optional, default None*) – List of callback functions that are applied at each iteration.
  - array *of* items : dict
  - *or* None

predict(*X*, ***predict_params*)

Make predictions.

Note: The predict method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

- **X** (array *of* items : array *of* items : float) – The dmatrix storing the input.
- **output_margin** (*boolean, optional, default False*) – Whether to output the raw untransformed margin value.
- **ntree_limit** (*union type, optional*) – Limit number of trees in the prediction; defaults to best_ntree_limit if defined
  - integer
  - *or* None
- **validate_features** (*boolean, optional, default True*) – When this is True, validate that the Booster's and data's feature_names are identical.

Returns

result – Predicted class label per sample.

- array *of* items : float
- *or* array *of* items : string
- *or* array *of* items : boolean

Return type

union type

predict_proba(*X*)

Probability estimates for all classes.

Note: The predict_proba method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

X (array *of* items : array *of* items : float) –

Returns

result – Probability of the sample for each class in the model.

Return type

array *of* items : array *of* items : float

lale.lib.xgboost.xgb_regressor module

class lale.lib.xgboost.xgb_regressor.**XGBRegressor**(**, max_depth=None, learning_rate=None, n_estimators, verbosity=None, silent=None, objective='reg:linear', booster=None, tree_method=None, n_jobs=1, nthread=None, gamma=None, min_child_weight=None, max_delta_step=None, subsample=None, colsample_bytree=None, colsample_bylevel=None, colsample_bynode=None, reg_alpha=None, reg_lambda=None, scale_pos_weight=None, base_score=None, random_state=0, missing=nan, importance_type='gain', seed=None, monotone_constraints=None, interaction_constraints=None, num_parallel_tree=None, validate_parameters=None, gpu_id=None, enable_categorical=False, predictor=None, max_leaves=None, max_bin=None, grow_policy=None, sampling_method=None, max_cat_to_onehot=None, eval_metric=None, early_stopping_rounds=None, callbacks=None, feature_types, max_cat_threshold=None, device=None, multi_strategy=None*)

Bases: *PlannedIndividualOp*

XGBRegressor gradient boosted decision trees.

This documentation is auto-generated from JSON schemas.

Parameters

- **max_depth** (*union type, default None*) – Maximum tree depth for base learners.
  - integer, >=0, >=1 for optimizer, <=7 for optimizer, uniform distribution
  - *or* None, not for optimizer
- **learning_rate** (*union type, default None*) – Boosting learning rate (xgb's "eta")
  - float, >=0.02 for optimizer, <=1 for optimizer, loguniform distribution
  - *or* None, not for optimizer
- **n_estimators** (*union type*) – Number of trees to fit.
  - integer, >=50 for optimizer, <=1000 for optimizer, default 200
  - *or* None
- **verbosity** (*union type, >=0, <=3, not for optimizer, default None*) – The degree of verbosity.
  - integer
  - *or* None
- **silent** (*union type, optional, not for optimizer, default None*) – Deprecated and replaced with verbosity, but adding to be backward compatible.
  - boolean
  - *or* None
- **objective** (*union type, not for optimizer, default 'reg:linear'*) – Specify the learning task and the corresponding learning objective or a custom objective function to be used.
  - 'reg:linear', 'reg:logistic', 'reg:gamma', 'reg:tweedie', *or* 'reg:squarederror'
  - *or* callable
- **booster** ('gbtree', 'gblinear', 'dart', *or* None, not for optimizer, default None) – Spec-

ify which booster to use.

- **tree_method** ('auto', 'exact', 'approx', 'hist', 'gpu_hist', *or* None, not for optimizer, default None) – Specify which tree method to use. Default to auto. If this parameter is set to default, XGBoost will choose the most conservative option available. Refer to https://xgboost.readthedocs.io/en/latest/parameter.html.
- **n_jobs** (*union type, not for optimizer, default 1*) – Number of parallel threads used to run xgboost. (replaces `nthread`)
  - integer
  - *or* None
- **nthread** (*union type, optional, not for optimizer, default None*) – Number of parallel threads used to run xgboost. Deprecated, please use n_jobs
  - integer
  - *or* None
- **gamma** (*union type, default None*) – Minimum loss reduction required to make a further partition on a leaf node of the tree.
  - float, >=0, <=1.0 for optimizer
  - *or* None, not for optimizer
- **min_child_weight** (*union type, default None*) – Minimum sum of instance weight(hessian) needed in a child.
  - integer, >=2 for optimizer, <=20 for optimizer, uniform distribution
  - *or* None, not for optimizer
- **max_delta_step** (*union type, not for optimizer, default None*) – Maximum delta step we allow each tree's weight estimation to be.
  - None
  - *or* integer
- **subsample** (*union type, default None*) – Subsample ratio of the training instance.
  - float, >0, >=0.01 for optimizer, <=1.0 for optimizer, uniform distribution
  - *or* None, not for optimizer
- **colsample_bytree** (*union type, not for optimizer, default None*) – Subsample ratio of columns when constructing each tree.
  - float, >0, >=0.1 for optimizer, <=1, <=1.0 for optimizer, uniform distribution
  - *or* None, not for optimizer
- **colsample_bylevel** (*union type, not for optimizer, default None*) – Subsample ratio of columns for each split, in each level.
  - float, >0, >=0.1 for optimizer, <=1, <=1.0 for optimizer, uniform distribution
  - *or* None, not for optimizer
- **colsample_bynode** (*union type, not for optimizer, default None*) – Subsample ratio of columns for each split.
  - float, >0, <=1
  - *or* None, not for optimizer
- **reg_alpha** (*union type, default None*) – L1 regularization term on weights
  - float, >=0 for optimizer, <=1 for optimizer, uniform distribution
  - *or* None, not for optimizer
- **reg_lambda** (*union type, default None*) – L2 regularization term on weights
  - float, >=0.1 for optimizer, <=1 for optimizer, uniform distribution
  - *or* None, not for optimizer
- **scale_pos_weight** (*union type, not for optimizer, default None*) – Balancing of positive and negative weights.
  - float
  - *or* None, not for optimizer
- **base_score** (*union type, not for optimizer, default None*) – The initial prediction score of all instances, global bias.
  - float

  - *or* None, not for optimizer
- **random_state** (*union type, not for optimizer, default 0*) – Random number seed. (replaces seed)
  - integer
  - *or* None
- **missing** (*union type, not for optimizer, default nan*) – Value in the data which needs to be present as a missing value. If If None, defaults to np.nan.
  - float
  - *or* None *or* nan
- **importance_type** ('gain', 'weight', 'cover', 'total_gain', 'total_cover', *or* None, optional, not for optimizer, default 'gain') – The feature importance type for the *feature_importances_* property.
- **seed** (*any type, optional, not for optimizer, default None*) – deprecated and replaced with random_state, but adding to be backward compatible.
- **monotone_constraints** (*union type, optional, not for optimizer, default None*) – Constraint of variable monotonicity.
  - None
  - *or* string
- **interaction_constraints** (*union type, optional, not for optimizer, default None*) – Constraints for interaction representing permitted interactions. The constraints must be specified in the form of a nest list, e.g. [[0, 1], [2, 3, 4]], where each inner list is a group of indices of features that are allowed to interact with each other.
  - None
  - *or* string
- **num_parallel_tree** (*union type, optional, not for optimizer, default None*) – Used for boosting random forest.
  - None
  - *or* integer
- **validate_parameters** (*union type, optional, not for optimizer, default None*) – Give warnings for unknown parameter.
  - None
  - *or* boolean
  - *or* integer
- **gpu_id** (*union type, optional, not for optimizer, default None*) – Device ordinal.
  - integer
  - *or* None
- **enable_categorical** (*boolean, optional, not for optimizer, default False*) – Experimental support for categorical data. Do not set to true unless you are interested in development. Only valid when gpu_hist and dataframe are used.
- **predictor** (*union type, optional, not for optimizer, default None*) – Force XGBoost to use specific predictor, available choices are [cpu_predictor, gpu_predictor].
  - string
  - *or* None
- **max_leaves** (*union type, optional, not for optimizer, default None*) – Maximum number of leaves; 0 indicates no limit.
  - integer
  - *or* None, not for optimizer
- **max_bin** (*union type, optional, not for optimizer, default None*) – If using histogram-based algorithm, maximum number of bins per feature.
  - integer
  - *or* None, not for optimizer

- **grow_policy** (0, 1, 'depthwise', 'lossguide', *or* None, optional, not for optimizer, default None) –
  Tree growing policy.
  0 or depthwise: favor splitting at nodes closest to the node, i.e. grow depth-wise. 1 or lossguide: favor splitting at nodes with highest loss change.
- **sampling_method** ('uniform', 'gadient_based', *or* None, optional, not for optimizer, default None) –
  Sampling method. Used only by gpu_hist tree method.
  - uniform: select random training instances uniformly.
  - gradient_based select random training instances with higher probability when the gradient and hessian are larger. (cf. CatBoost)
- **max_cat_to_onehot** (*union type, optional, not for optimizer, default None*) –
  A threshold for deciding whether XGBoost should use
  one-hot encoding based split for categorical data.
  - integer
  - *or* None
- **eval_metric** (*union type, optional, not for optimizer, default None*) – Metric used for monitoring the training result and early stopping.
  - string
  - *or* array *of* items : string
  - *or* array *of* items : callable
  - *or* None
- **early_stopping_rounds** (*union type, optional, not for optimizer, default None*) –
  Activates early stopping.
  Validation metric needs to improve at least once in every early_stopping_rounds round(s) to continue training.
  - integer
  - *or* None
- **callbacks** (*union type, optional, not for optimizer, default None*) –
  List of callback functions that are applied at end of each iteration.
  It is possible to use predefined callbacks by using Callback API.
  - array *of* items : callable
  - *or* None
- **feature_types** (*Any, optional, not for optimizer*) – Used for specifying feature types without constructing a dataframe. See DMatrix for details.
- **max_cat_threshold** (*union type, optional, not for optimizer, default None*) –
  Maximum number of categories considered for each split.
  Used only by partition-based splits for preventing over-fitting. Also, enable_categorical needs to be set to have categorical feature support. See Categorical Data and Parameters for Categorical Feature for details.
  - integer, >=0, >=1 for optimizer, <=10 for optimizer, uniform distribution
  - *or* None
- **device** (*union type, optional, not for optimizer, default None*) – Device ordinal
  - 'cpu', 'cuda', *or* 'gpu'
  - *or* None
- **multi_strategy** (*union type, optional, not for optimizer, default None*) –
  The strategy used for training multi-target models,

including multi-target regression and multi-class classification. See Multiple Outputs for more information.

- ’one_output_per_tree’
  One model for each target.
- *or* ‘multi_output_tree’
  Use multi-target trees.
- *or* None

fit(*X*, *y=None*, ***fit_params*)

Train the operator.

Note: The fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

- **X** (array *of* items : array *of* items : float) – Feature matrix
- **y** (array *of* items : float) – Labels
- **sample_weight** (*union type, optional, default None*) – Weight for each instance
  - array *of* items : float
  - *or* None
- **eval_set** (*union type, optional, default None*) – A list of (X, y) pairs to use as a validation set for
  - array
  - *or* None
- **sample_weight_eval_set** (*union type, optional, default None*) – A list of the form [L_1, L_2, . . . , L_n], where each L_i is a list of
  - array
  - *or* None
- **eval_metric** (*union type, optional, default None*) – If a str, should be a built-in evaluation metric to use. See
  - array *of* items : string
  - *or* string
  - *or* None
  - *or* dict
- **early_stopping_rounds** (*union type, optional, default None*) – Activates early stopping. Validation error needs to decrease at
  - integer
  - *or* None
- **verbose** (*boolean, optional, default True*) – If *verbose* and an evaluation set is used, writes the evaluation
- **xgb_model** (*union type, optional, default None*) – file name of stored xgb model or ‘Booster’ instance Xgb model to be
  - string
  - *or* None
- **callbacks** (*union type, optional, default None*) – List of callback functions that are applied at each iteration.
  - array *of* items : dict
  - *or* None

partial_fit(*X*, *y=None*, ***fit_params*)

Incremental fit to train train the operator on a batch of samples.

Note: The partial_fit method is not available until this operator is trainable.

Once this method is available, it will have the following signature:

Parameters

- **X** (array *of* items : array *of* items : float) – Feature matrix
- **y** (array *of* items : float) – Labels
- **sample_weight** (*union type, optional, default None*) – Weight for each instance
  - array *of* items : float
  - *or* None
- **eval_set** (*union type, optional, default None*) – A list of (X, y) pairs to use as a validation set for
  - array
  - *or* None
- **sample_weight_eval_set** (*union type, optional, default None*) – A list of the form [L_1, L_2, …, L_n], where each L_i is a list of
  - array
  - *or* None
- **eval_metric** (*union type, optional, default None*) – If a str, should be a built-in evaluation metric to use. See
  - array *of* items : string
  - *or* string
  - *or* None
  - *or* dict
- **early_stopping_rounds** (*union type, optional, default None*) – Activates early stopping. Validation error needs to decrease at
  - integer
  - *or* None
- **verbose** (*boolean, optional, default True*) – If *verbose* and an evaluation set is used, writes the evaluation
- **xgb_model** (*union type, optional, default None*) – file name of stored xgb model or 'Booster' instance Xgb model to be
  - string
  - *or* None
- **callbacks** (*union type, optional, default None*) – List of callback functions that are applied at each iteration.
  - array *of* items : dict
  - *or* None

predict(*X*, ***predict_params*)

Make predictions.

Note: The predict method is not available until this operator is trained.

Once this method is available, it will have the following signature:

Parameters

- **X** (array *of* items : array *of* items : float) – The dmatrix storing the input.
- **output_margin** (*boolean, optional, default False*) – Whether to output the raw untransformed margin value.
- **ntree_limit** (*union type, optional*) – Limit number of trees in the prediction; defaults to best_ntree_limit if defined
  - integer
  - *or* None
- **validate_features** (*boolean, optional, default True*) – When this is True, validate that the Booster's and data's feature_names are identical.

Returns

result – Output data schema for predictions (target class labels).

Return type

array *of* items : float

Module contents

Scikit-learn compatible wrappers for XGBoost along with schemas to enable hyperparameter tuning.

Operators:

- XGBClassifier
- XGBRegressor

Submodules

lale.lib.dataframe module

Common interface to manipulate different type of dataframes supported in Lale.

`lale.lib.dataframe.`**`count`**(*df*)

`lale.lib.dataframe.`**`get_columns`**(*df*) → List[Union[str, int]]

`lale.lib.dataframe.`**`make_series_concat`**(*df1*, *df2*)

`lale.lib.dataframe.`**`make_series_distinct`**(*df*)

`lale.lib.dataframe.`**`select_col`**(*df*, *col: Union[str, int]*)

Module contents

lale.search package

Submodules

lale.search.PGO module

class `lale.search.PGO.`**`DefaultValue`**(*value*)

Bases: `Enum`

An enumeration.

token = 0

class `lale.search.PGO.`**`FrequencyDistribution`**(*freqs: ~typing.Iterable[~typing.Tuple[~typing.Union[~lale.search.PGO.DefaultValue, ~lale.search.PGO.T], int]], dtype=<class 'object'>*)

Bases: `Generic`[T]

Represents the distribution implied by a histogram

classmethod asEnumValues(*freqs: Iterable[Tuple[Any, int]], values: List[Any]*) → *FrequencyDistribution*[Any]

classmethod asFloatValues(*freqs: Iterable[Tuple[Any, int]], inclusive_min: Optional[float] = None, inclusive_max: Optional[float] = None*) → *FrequencyDistribution*[float]

classmethod asIntegerValues(*freqs: Iterable[Tuple[Any, int]], inclusive_min: Optional[float] = None, inclusive_max: Optional[float] = None*) → *FrequencyDistribution*[int]

cumulative_freqs: ndarray

freq_dist: ndarray

sample() → T

samples(*count: int*) → Sequence[T]

vals: ndarray

lale.search.PGO.**freqsAsEnumValues**(*freqs: Iterable[Tuple[Any, int]], values: List[Any]*) → Iterator[Tuple[Union[*DefaultValue*, Any], int]]

only keeps things that match the string representation of values in the enumeration. converts from the string to the value as represented in the enumeration.

lale.search.PGO.**freqsAsFloatValues**(*freqs: Iterable[Tuple[Any, int]], inclusive_min: Optional[float] = None, inclusive_max: Optional[float] = None*) → Iterator[Tuple[Union[*DefaultValue*, float], int]]

maps the str values to integers, and skips anything that does not look like an integer

lale.search.PGO.**freqsAsIntegerValues**(*freqs: Iterable[Tuple[Any, int]], inclusive_min: Optional[float] = None, inclusive_max: Optional[float] = None*) → Iterator[Tuple[Union[*DefaultValue*, int], int]]

maps the str values to integers, and skips anything that does not look like an integer

lale.search.PGO.**load_pgo_data**(*json_data*) → Dict[str, Dict[str, Dict[str, int]]]

lale.search.PGO.**load_pgo_file**(*filepath*) → Dict[str, Dict[str, Dict[str, int]]]

lale.search.PGO.**normalize_pgo_type**(*data: Dict[str, Dict[str, Union[int, Dict[str, Union[str, int]]]]]*) → Dict[str, Dict[str, Dict[str, int]]]

lale.search.PGO.**remove_defaults_dict**(*d: Dict[XDK, Union[*DefaultValue*, XDV]]*) → Dict[XDK, XDV]

lale.search.lale_grid_search_cv module

lale.search.lale_grid_search_cv.**HPValuetoGSValue**(*key: str, hp:* SearchSpace, *num_samples: Optional[int] = None*) → List[Any]

lale.search.lale_grid_search_cv.**SearchSpaceGridstoGSGrids**(*hp_grids: List[Dict[str,* SearchSpacePrimitive*]], num_samples: Optional[int] = None*) → List[Dict[str, List[Any]]]

lale.search.lale_grid_search_cv.**SearchSpaceGridtoGSGrid**(*hp: Dict[str,* SearchSpacePrimitive*], num_samples: Optional[int] = None*) → Dict[str, List[Any]] lale.search.lale_grid_search_cv.**SearchSpaceNumberToGSValues**(*key: str*, *hp:* SearchSpaceNumber, *num_samples: Optional[int] = None*) → List[Any]

Returns either a list of values intended to be sampled uniformly

lale.search.lale_grid_search_cv.**get_defaults_as_param_grid**(*op:* IndividualOp)

lale.search.lale_grid_search_cv.**get_grid_search_parameter_grids**(*op:* PlannedOperator, *num_samples: Optional[int] = None*, *num_grids: Optional[float] = None*, *pgo: Optional[Dict[str, Dict[str, Dict[str, int]]]] = None*, *data_schema: Optional[Dict[str, Any]] = None*) → List[Dict[str, List[Any]]]

Top level function: given a lale operator, returns a list of parameter grids suitable for passing to GridSearchCV. Note that you will need to wrap the lale operator for sklearn compatibility to call GridSearchCV directly. The lale GridSearchCV wrapper takes care of that for you

lale.search.lale_grid_search_cv.**get_lale_gridsearchcv_op**(*op*, *params*, ***kwargs*)

lale.search.lale_grid_search_cv.**get_parameter_grids**(*op:* PlannedOperator, *num_samples: Optional[int] = None*, *num_grids: Optional[float] = None*, *pgo: Optional[Dict[str, Dict[str, Dict[str, int]]]] = None*, *data_schema: Optional[Dict[str, Any]] = None*)

Parameters

- **op** (*The lale PlannedOperator*) –
- **num_samples** (*integer, optional*) – If set, will limit the number of samples for each distribution
- **num_grids** (*integer or float, optional*) – if set to an integer => 1, it will determine how many parameter grids will be returned (at most) if set to an float between 0 and 1, it will determine what fraction should be returned note that setting it to 1 is treated as in integer. To return all results, use None
- **pgo** (*Optional profile guided optimization data that guides discretization*) –
- **data_schema** (*Optional schema for the input data. which is used for hyperparameter schema data constraints*) –

lale.search.lale_grid_search_cv.**gridsearchcv_grid_to_string**(*grid: Dict[str, List[Any]]*) → str

lale.search.lale_grid_search_cv.**gridsearchcv_grids_to_string**(*grids: List[Dict[str, List[Any]]]*) → str

lale.search.lale_hyperopt module

class lale.search.lale_hyperopt.**SearchSpaceHPExprVisitor**(*name: str*)

Bases: *Visitor*

array_single_expr_(*space: SearchSpaceArray*, *path: str*, *num*)

get_unique_name(*name: str*) → str

mk_label(*label*, *counter*, *useCounter=True*)

names: Dict[str, int]

classmethod run(*space: SearchSpace*, *name: str*)

visitSearchSpaceArray(*space: SearchSpaceArray*, *path: str*, *counter=None*)

visitSearchSpaceBool(*space: SearchSpaceEnum*, *path: str*, *counter=None*)

visitSearchSpaceConstant(*space: SearchSpaceEnum*, *path: str*, *counter=None*)

visitSearchSpaceDict(*sd: SearchSpaceDict*, *path: str*, *counter=None*)

visitSearchSpaceEmpty(*op: SearchSpaceEmpty*, *path: str*, *counter=None*)

visitSearchSpaceEnum(*space: SearchSpaceEnum*, *path: str*, *counter=None*)

visitSearchSpaceNumber(*space: SearchSpaceNumber*, *path: str*, *counter=None*)

visitSearchSpaceObject(*space: SearchSpaceObject*, *path: str*, *counter=None*)

visitSearchSpaceOperator(*op: SearchSpaceOperator*, *path: str*, *counter=None*)

visitSearchSpaceProduct(*prod: SearchSpaceProduct*, *path: str*, *counter=None*)

visitSearchSpaceSum(*space_sum: SearchSpaceSum*, *path: str*, *counter=None*)

class lale.search.lale_hyperopt.**SearchSpaceHPStrVisitor**(*name: str*)

Bases: *Visitor*

array_single_str_(*space: SearchSpaceArray*, *path: str*, *num*, *useCounter=True*) → str

decls: str

get_unique_name(*name: str*) → str

get_unique_variable_name(*name: str*) → str

mk_label(*label*, *counter*, *useCounter=True*)

names: Dict[str, int]

nested_header: Optional[str]

pgo_dict: Dict[str, *FrequencyDistribution*]

pgo_header: Optional[str]

classmethod run(*space: SearchSpace*, *name: str*, *counter=None*, *useCounter=True*)

visitSearchSpaceArray(*space:* SearchSpaceArray, *path:* *str*, *counter=None*, *useCounter=True*) → str

visitSearchSpaceBool(*space:* SearchSpaceEnum, *path:* *str*, *counter=None*, *useCounter=True*)

visitSearchSpaceConstant(*space:* SearchSpaceEnum, *path:* *str*, *counter=None*, *useCounter=True*)

visitSearchSpaceDict(*sd:* SearchSpaceDict, *path:* *str*, *counter=None*, *useCounter=True*)

visitSearchSpaceEmpty(*op:* SearchSpaceEmpty, *path:* *str*, *counter=None*, *useCounter=True*) → str

visitSearchSpaceEnum(*space:* SearchSpaceEnum, *path:* *str*, *counter=None*, *useCounter=True*)

visitSearchSpaceNumber(*space:* SearchSpaceNumber, *path:* *str*, *counter=None*, *useCounter=True*)

visitSearchSpaceObject(*space:* SearchSpaceObject, *path:* *str*, *counter=None*, *useCounter=True*)

visitSearchSpaceOperator(*op:* SearchSpaceOperator, *path:* *str*, *counter=None*, *useCounter=True*)

visitSearchSpaceProduct(*prod:* SearchSpaceProduct, *path:* *str*, *counter=None*, *useCounter=True*)

visitSearchSpaceSum(*sum_space:* SearchSpaceSum, *path:* *str*, *counter=None*, *useCounter=True*)

lale.search.lale_hyperopt.**make_nested_hyperopt**(*space*)

lale.search.lale_hyperopt.**pgo_sample**(*pgo*, *sample*)

lale.search.lale_hyperopt.**search_space_to_hp_expr**(*space:* SearchSpace, *name:* *str*)

lale.search.lale_hyperopt.**search_space_to_hp_str**(*space:* SearchSpace, *name:* *str*) → str

lale.search.lale_hyperopt.**search_space_to_str_for_comparison**(*space:* SearchSpace, *name:* *str*) → str

lale.search.lale_smac module

class lale.search.lale_smac.**FakeNone**

Bases: `object`

lale.search.lale_smac.**HPValuetoSMAC**(*key:* *str*, *hp:* SearchSpace) → Hyperparameter

lale.search.lale_smac.**SearchSpaceGridtoSMAC**(*hp:* *Dict[str, SearchSpacePrimitive]*, *disc:* *int*) → Iterable[Hyperparameter]

lale.search.lale_smac.**SearchSpaceNumberToSMAC**(*key:* *str*, *hp:* SearchSpaceNumber) → Hyperparameter

Returns either a list of values intended to be sampled uniformly or a frozen scipy.stats distribution

lale.search.lale_smac.**addSearchSpaceGrid**(*hp:* *Dict[str, SearchSpacePrimitive]*, *disc:* *int*, *parent_disc:* *Hyperparameter*, *cs:* *ConfigurationSpace*) → None

lale.search.lale_smac.**addSearchSpaceGrids**(*grids:* *List[Dict[str, SearchSpacePrimitive]]*, *cs:* *ConfigurationSpace*) → None

lale.search.lale_smac.**get_smac_space**(*op:* *Ops.PlannedOperator*, *lale_num_grids:* *Optional[float] = None*, *lale_pgo:* *Optional[Dict[str, Dict[str, Dict[str, int]]]] = None*, *data_schema:* *Optional[Dict[str, Any]] = None*) → ConfigurationSpace

Top level function: given a lale operator, returns a ConfigurationSpace for use with SMAC.

Parameters

- **op** (*The lale PlannedOperator*) –
- **lale_num_grids** (*integer or float, optional*) – if set to an integer => 1, it will determine how many parameter grids will be returned (at most) if set to an float between 0 and 1, it will determine what fraction should be returned note that setting it to 1 is treated as in integer. To return all results, use None
- **lale_pgo** (*Optional profile guided optimization data that guides discretization*) –
- **data_schema** (*Optional schema for the input data. which is used for hyperparameter schema data constraints*) –

lale.search.lale_smac.**hp_grids_to_smac_cs**(*grids: List[Dict[str, SearchSpacePrimitive]]*) → ConfigurationSpace

lale.search.lale_smac.**lale_op_smac_tae**(*op: Ops.PlannedOperator*, *f_min*)

lale.search.lale_smac.**lale_trainable_op_from_config**(*op: Ops.PlannedOperator*, *cfg*) → Ops.TrainableOperator

lale.search.lale_smac.**smac_fixup_params**(*cfg*)

lale.search.op2hp module

lale.search.op2hp.**hyperopt_search_space**(*op: PlannedOperator*, *schema=None*, *pgo: Optional[Dict[str, Dict[str, Dict[str, int]]]] = None*, *data_schema: Optional[Dict[str, Any]] = None*)

lale.search.schema2search_space module

class lale.search.schema2search_space.**FreqsWrapper**(*base: Optional[Dict[str, Dict[str, int]]]*)

Bases: object

base: Optional[Dict[str, Dict[str, int]]]

exception lale.search.schema2search_space.**OperatorSchemaError**(*sub_path: Any*, *message: Optional[str] = None*)

Bases: *VisitorPathError*

get_message_str() → str

class lale.search.schema2search_space.**SearchSpaceOperatorVisitor**(*pgo: Optional[Dict[str, Dict[str, Dict[str, int]]]] = None*, *data_schema: Optional[Dict[str, Any]] = None*)

Bases: *Visitor*

JsonSchemaToSearchSpaceHelper(*longName: str*, *path: str*, *schema: Dict[str, Any]*, *relevantFields: Optional[Set[str]]*, *pgo_freqs: Optional[Union[FreqsWrapper, Dict[str, int]]] = None*, *sub_space: bool = True*) → Dict[str, *SearchSpace*]

data_schema: Optional[Dict[str, Any]]

pgo: Optional[Dict[str, Dict[str, Dict[str, int]]]]

classmethod **run**(*op: PlannedOperator, pgo: Optional[Dict[str, Dict[str, Dict[str, int]]]] = None, data_schema: Optional[Dict[str, Any]] = None*) → *SearchSpace*

schemaToSearchSpace(*longName: str, name: str, schema: Dict[str, Any]*) → Optional[*SearchSpace*]

schemaToSearchSpaceHelper(*longName, schema: Optional[Dict[str, Any]], relevantFields: Optional[Set[str]], pgo_freqs: Optional[Union[FreqsWrapper, Dict[str, int]]] = None, sub_space: bool = True*) → Optional[*SearchSpace*]

schemaToSearchSpaceHelper_(*longName, path: str, schema: Dict[str, Any], relevantFields: Optional[Set[str]], pgo_freqs: Optional[Union[FreqsWrapper, Dict[str, int]]] = None, sub_space: bool = True*) → Optional[*SearchSpace*]

schemaToSimplifiedAndSearchSpace(*longName: str, name: str, schema: Dict[str, Any]*) → Tuple[Optional[Dict[str, Any]], Optional[*SearchSpace*]]

visitOperatorChoice(*op: OperatorChoice*) → *SearchSpace*

visitPlannedIndividualOp(*op: PlannedIndividualOp*) → *SearchSpace*

visitPlannedPipeline(*op: PlannedPipeline*) → *SearchSpace*

visitTrainableIndividualOp(*op: PlannedIndividualOp*) → *SearchSpace*

visitTrainablePipeline(*op: PlannedPipeline*) → *SearchSpace*

visitTrainedIndividualOp(*op: PlannedIndividualOp*) → *SearchSpace*

visitTrainedPipeline(*op: PlannedPipeline*) → *SearchSpace*

lale.search.schema2search_space.**add_sub_space**(*space, k, v*)

Given a search space and a "key", if the defined subschema does not exist, set it to be the constant v space

lale.search.schema2search_space.**asFreqs**(*part: Optional[Union[FreqsWrapper, Dict[str, int]]]*) → Optional[Iterable[Tuple[Any, int]]]

lale.search.schema2search_space.**freqs_wrapper_lookup**(*part: Optional[Union[FreqsWrapper, Dict[str, int]]], k: str*) → Optional[Union[*FreqsWrapper*, Dict[str, int]]]

lale.search.schema2search_space.**get_default**(*schema*) → Optional[Any]

lale.search.schema2search_space.**op_to_search_space**(*op: PlannedOperator, pgo: Optional[Dict[str, Dict[str, Dict[str, int]]]] = None, data_schema: Optional[Dict[str, Any]] = None*) → *SearchSpace*

Given an operator, this method compiles its schemas into a SearchSpace

lale.search.schema2search_space.**pgo_lookup**(*pgo: Optional[Dict[str, Dict[str, Dict[str, int]]]], name: str*) → Optional[*FreqsWrapper*]

lale.search.search_space module

class lale.search.search_space.**SearchSpace**(*default: Optional[Any] = None*)

Bases: object

default() → Optional[Any]

Return an optional default value, if None. if not None, the default value should be in the search space

classmethod focused_path_string(*path: List[SearchSpace]*) → str

str_with_focus(*path: Optional[List[SearchSpace]] = None, default: Optional[Any] = None*) → Union[str, Any]

Given a path list, returns a string for the focused path. If the path is None, returns everything, without focus. If the path does not start with self, returns None

class lale.search.search_space.**SearchSpaceArray**(*prefix: Optional[List[SearchSpace]], minimum: int = 0, *, maximum: int, additional: Optional[SearchSpace] = None, is_tuple=False*)

Bases: *SearchSpace*

items(*max_elts: Optional[int] = None*) → Iterable[*SearchSpace*]

class lale.search.search_space.**SearchSpaceBool**(*pgo: Optional[Union[FrequencyDistribution, Iterable[Tuple[Any, int]]]] = None, default: Optional[Any] = None*)

Bases: *SearchSpaceEnum*

pgo: Optional[*FrequencyDistribution*]

vals: List[Any]

class lale.search.search_space.**SearchSpaceConstant**(*v, pgo: Optional[Union[FrequencyDistribution, Iterable[Tuple[Any, int]]]] = None*)

Bases: *SearchSpaceEnum*

pgo: Optional[*FrequencyDistribution*]

vals: List[Any]

class lale.search.search_space.**SearchSpaceDict**(*d: Dict[str, SearchSpace]*)

Bases: *SearchSpace*

class lale.search.search_space.**SearchSpaceEmpty**

Bases: *SearchSpace*

class lale.search.search_space.**SearchSpaceEnum**(*vals: Iterable[Any], pgo: Optional[Union[FrequencyDistribution, Iterable[Tuple[Any, int]]]] = None, default: Optional[Any] = None*)

Bases: *SearchSpacePrimitive*

pgo: Optional[*FrequencyDistribution*]

vals: List[Any]

exception lale.search.search_space.**SearchSpaceError**(*sub_path: Any, message: Optional[str] = None*)

Bases: *VisitorPathError*

get_message_str() → str

path_string() → str

class lale.search.search_space.**SearchSpaceNumber**(*minimum=None*, *exclusiveMinimum: bool = False*, *maximum=None*, *exclusiveMaximum: bool = False*, *discrete: bool = False*, *distribution='uniform'*, *pgo: Optional[Union[FrequencyDistribution, Iterable[Tuple[Any, int]]]] = None*, *default: Optional[Any] = None*)

Bases: *SearchSpacePrimitive*

discrete: bool

distribution: str

exclusiveMaximum: bool

exclusiveMinumum: bool

getInclusiveMax() → Optional[float]

Return the maximum as an inclusive maximum (exclusive maxima are adjusted accordingly)

getInclusiveMin() → Optional[float]

Return the maximum as an inclusive minimum (exclusive minima are adjusted accordingly)

maximum: Optional[float]

minimum: Optional[float]

pgo: Optional[*FrequencyDistribution*]

class lale.search.search_space.**SearchSpaceObject**(*longName: str*, *keys: List[str]*, *choices: Iterable[Any]*)

Bases: *SearchSpace*

class lale.search.search_space.**SearchSpaceOperator**(*sub_space: SearchSpace*, *default: Optional[Any] = None*)

Bases: *SearchSpace*

sub_space: *SearchSpace*

class lale.search.search_space.**SearchSpacePrimitive**(*default: Optional[Any] = None*)

Bases: *SearchSpace*

class lale.search.search_space.**SearchSpaceProduct**(*sub_spaces: List[Tuple[str, SearchSpace]]*, *default: Optional[Any] = None*)

Bases: *SearchSpace*

get_indexed_spaces() → Iterable[Tuple[str, int, *SearchSpace*]]

sub_spaces: List[Tuple[str, *SearchSpace*]]

class lale.search.search_space.**SearchSpaceSum**(*sub_spaces: List[SearchSpace]*, *default: Optional[Any] = None*)

Bases: *SearchSpace*

sub_spaces: List[*SearchSpace*]

lale.search.search_space.**should_print_search_space**(**s: str*)

lale.search.search_space_grid module

class lale.search.search_space_grid.**SearchSpaceToGridVisitor**

Bases: *Visitor*

classmethod fixupDegenerateSearchSpaces(*space: Union[List[Dict[str, SearchSpacePrimitive]], SearchSpacePrimitive]*) → List[Dict[str, *SearchSpacePrimitive*]]

classmethod run(*space:* SearchSpace) → List[Dict[str, *SearchSpacePrimitive*]]

visitSearchSpaceArray(*space:* SearchSpaceArray) → List[Dict[str, *SearchSpacePrimitive*]]

visitSearchSpaceBool(*space:* SearchSpacePrimitive) → *SearchSpacePrimitive*

visitSearchSpaceConstant(*space:* SearchSpacePrimitive) → *SearchSpacePrimitive*

visitSearchSpaceDict(*op:* SearchSpaceDict) → Union[List[Dict[str, *SearchSpacePrimitive*]], *SearchSpacePrimitive*]

visitSearchSpaceEmpty(*op:* SearchSpaceEmpty)

visitSearchSpaceEnum(*space:* SearchSpacePrimitive) → *SearchSpacePrimitive*

visitSearchSpaceNumber(*space:* SearchSpacePrimitive) → *SearchSpacePrimitive*

visitSearchSpaceObject(*space:* SearchSpaceObject) → List[Dict[str, *SearchSpacePrimitive*]]

visitSearchSpaceOperator(*op:* SearchSpaceOperator) → Union[List[Dict[str, *SearchSpacePrimitive*]], *SearchSpacePrimitive*]

visitSearchSpacePrimitive(*space:* SearchSpacePrimitive) → *SearchSpacePrimitive*

visitSearchSpaceProduct(*op:* SearchSpaceProduct) → Union[List[Dict[str, *SearchSpacePrimitive*]], *SearchSpacePrimitive*]

visitSearchSpaceSum(*op:* SearchSpaceSum) → Union[List[Dict[str, *SearchSpacePrimitive*]], *SearchSpacePrimitive*]

lale.search.search_space_grid.**get_search_space_grids**(*op:* PlannedOperator, *num_grids: Optional[float] = None*, *pgo: Optional[Dict[str, Dict[str, Dict[str, int]]]] = None*, *data_schema: Optional[Dict[str, Any]] = None*) → List[Dict[str, *SearchSpacePrimitive*]]

Top level function: given a lale operator, returns a list of hp grids.

Parameters

- **op** (*The lale PlannedOperator*) –
- **num_grids** (*integer or float, optional*) – if set to an integer => 1, it will determine how many parameter grids will be returned (at most) if set to an float between 0 and 1, it will determine what fraction should be returned note that setting it to 1 is treated as in integer. To return all results, use None
- **pgo** (*Optional Profile Guided Optimization data that can be used when discretizing continuous parameters*) –
- **data_schema** (*A schema for the actual data. If provided, it is used to instantiate data dependent schema hyperamparameter specifications.*) –

lale.search.search_space_grid.**op_to_search_space_grids**(*op: PlannedOperator*, *pgo: Optional[Dict[str, Dict[str, Dict[str, int]]]] = None*, *data_schema: Optional[Dict[str, Any]] = None*) → List[Dict[str, SearchSpacePrimitive]]

lale.search.search_space_grid.**search_space_grid_to_string**(*grid: Dict[str, SearchSpacePrimitive]*) → str

lale.search.search_space_grid.**search_space_grids_to_string**(*grids: List[Dict[str, SearchSpacePrimitive]]*) → str

lale.search.search_space_grid.**search_space_to_grids**(*hp: SearchSpace*) → List[Dict[str, SearchSpacePrimitive]]

Module contents

lale.util package

Submodules

lale.util.Visitor module

class lale.util.Visitor.**Visitor**

Bases: `object`

defaultVisit(*node*, **args*, ***kwargs*)

lale.util.Visitor.**accept**(*obj: Any*, *v: Visitor*, **args*, ***kwargs*)

lale.util.VisitorMeta module

class lale.util.VisitorMeta.**AbstractVisitorMeta**(*name*, *bases*, *namespace*, ***kwargs*)

Bases: `VisitorMeta`, `ABCMeta`

This meta class adds an _accept method that calls visitCLASSNAME on the visitor. It does not currently support inheritance: you need to define the visitC method for subclasses explicitly. The private _accept method should be called via the Visitor#acccept method.

class lale.util.VisitorMeta.**VisitorMeta**(**args*, ***kwargs*)

Bases: `type`

This meta class adds a private _accept method that calls visitCLASSNAME on the visitor. It does not currently support inheritance: you need to define the visitC method for subclasses explicitly. The private _accept method should be called via the Visitor#acccept method

lale.util.VisitorPathError module

exception lale.util.VisitorPathError.**VisitorPathError**(*path: List[Any]*, *message: Optional[str] = None*)

Bases: ValueError

get_message_str() → str

property path: Iterator[Any]

path_string() → str

push_parent_path(*part: Any*) → None

lale.util.batch_data_dictionary_dataset module

class lale.util.batch_data_dictionary_dataset.**BatchDataDict**(**args: Any*, ***kwargs: Any*)

Bases: Dataset

Pytorch Dataset subclass that takes a dictionary of format {'<batch_idx>': <batch_data>}.

X is the dictionary dataset and y is ignored.

Parameters

- **X** (*dict*) – Dictionary of format {'<batch_idx>': <batch_data>}
- **y** (*None*) – Ignored.

lale.util.hdf5_to_torch_dataset module

class lale.util.hdf5_to_torch_dataset.**HDF5TorchDataset**(**args: Any*, ***kwargs: Any*)

Bases: Dataset

Pytorch Dataset subclass that takes a hdf5 file pointer.

.

Parameters

file (*file is an object of class h5py.File*) –

get_data()

lale.util.numpy_to_torch_dataset module

class lale.util.numpy_to_torch_dataset.**NumpyTorchDataset**(**args: Any*, ***kwargs: Any*)

Bases: Dataset

Pytorch Dataset subclass that takes a numpy array and an optional label array.

X and y are the dataset and labels respectively.

Parameters

- **X** (*numpy array*) – Two dimensional dataset of input features.
- **y** (*numpy array*) – Labels

get_data()

lale.util.numpy_to_torch_dataset.**numpy_collate_fn**(*batch*)

lale.util.numpy_torch_dataset module

class lale.util.numpy_torch_dataset.**NumpyTorchDataset**(**args: Any*, ***kwargs: Any*)

Bases: `Dataset`

Pytorch Dataset subclass that takes a numpy array and an optional label array.

X and y are the dataset and labels respectively.

Parameters

- **X** (*numpy array*) – Two dimensional dataset of input features.
- **y** (*numpy array*) – Labels

get_data()

lale.util.numpy_torch_dataset.**numpy_collate_fn**(*batch*)

lale.util.pandas_to_torch_dataset module

class lale.util.pandas_to_torch_dataset.**PandasTorchDataset**(**args: Any*, ***kwargs: Any*)

Bases: `Dataset`

Pytorch Dataset subclass that takes a pandas DataFrame and an optional label pandas Series.

X and y are the dataset and labels respectively.

Parameters

- **X** (*pandas DataFrame*) – Two dimensional dataset of input features.
- **y** (*pandas Series*) – Labels

get_data()

lale.util.pandas_to_torch_dataset.**pandas_collate_fn**(*batch*)

lale.util.pandas_torch_dataset module

class lale.util.pandas_torch_dataset.**PandasTorchDataset**(**args: Any*, ***kwargs: Any*)

Bases: `Dataset`

Pytorch Dataset subclass that takes a pandas DataFrame and an optional label pandas Series.

X and y are the dataset and labels respectively.

Parameters

- **X** (*pandas DataFrame*) – Two dimensional dataset of input features.
- **y** (*pandas Series*) – Labels

get_data()

lale.util.pandas_torch_dataset.**pandas_collate_fn**(*batch*)

Module contents

Submodules

lale.docstrings module

lale.docstrings.**set_docstrings**(*lale_op: IndividualOp*)

If we are running under sphinx, this will take a variable whose value is a lale operator and change it to a value of an artificial class with appropriately documented methods.

lale.expressions module

class lale.expressions.**Expr**(*expr: Union[Num, Str, List, Tuple, Set, Dict, Constant, Name, Expr, UnaryOp, BinOp, BoolOp, Compare, Call, Attribute, Subscript], istrue=None*)

Bases: `object`

property expr

class lale.expressions.**FixUnparser**(*tree, file=sys.stdout*)

Bases: `Unparser`

Print the source for tree to file.

lale.expressions.**asc**(*column: Union[Expr, str]*) → *Expr*

lale.expressions.**astype**(*dtype, subject: Expr*) → *Expr*

lale.expressions.**collect_set**(*group: Expr*) → *Expr*

lale.expressions.**count**(*group: Expr*) → *Expr*

lale.expressions.**day_of_month**(*subject: Expr, fmt: Optional[str] = None*) → *Expr*

lale.expressions.**day_of_week**(*subject: Expr, fmt: Optional[str] = None*) → *Expr*

lale.expressions.**day_of_year**(*subject: Expr, fmt: Optional[str] = None*) → *Expr*

lale.expressions.**desc**(*column: Union[Expr, str]*) → *Expr*

lale.expressions.**distinct_count**(*group: Expr*) → *Expr*

lale.expressions.**first**(*group: Expr*) → *Expr*

lale.expressions.**fixedUnparse**(*tree*)

lale.expressions.**hash**(*hash_method: str, subject: Expr*) → *Expr*

lale.expressions.**hash_mod**(*hash_method: str, subject: Expr, n: Expr*) → *Expr*

lale.expressions.**hour**(*subject: Expr, fmt: Optional[str] = None*) → *Expr*

lale.expressions.**identity**(*subject: Expr*) → *Expr*

lale.expressions.**isnan**(*column: Expr*) → *Expr*

lale.expressions.**isnotnan**(*column: Expr*) → *Expr* `lale.expressions.isnotnull`(*column: Expr*) → *Expr*

`lale.expressions.isnull`(*column: Expr*) → *Expr*

`lale.expressions.ite`(*cond: Expr*, *v1: Union[Expr, int, float, bool, str]*, *v2: Union[Expr, int, float, bool, str]*) → *Expr*

`lale.expressions.item`(*group: Expr*, *value: Union[int, str]*) → *Expr*

`lale.expressions.max`(*group: Expr*) → *Expr*

`lale.expressions.max_gap_to_cutoff`(*group: Expr*, *cutoff: Expr*) → *Expr*

`lale.expressions.mean`(*group: Expr*) → *Expr*

`lale.expressions.median`(*group: Expr*) → *Expr*

`lale.expressions.min`(*group: Expr*) → *Expr*

`lale.expressions.minute`(*subject: Expr*, *fmt: Optional[str] = None*) → *Expr*

`lale.expressions.mode`(*group: Expr*) → *Expr*

`lale.expressions.month`(*subject: Expr*, *fmt: Optional[str] = None*) → *Expr*

`lale.expressions.normalized_count`(*group: Expr*) → *Expr*

`lale.expressions.normalized_sum`(*group: Expr*) → *Expr*

`lale.expressions.recent`(*series: Expr*, *age: int*) → *Expr*

`lale.expressions.recent_gap_to_cutoff`(*series: Expr*, *cutoff: Expr*, *age: int*) → *Expr*

`lale.expressions.replace`(*subject: Expr*, *old2new: Dict[Any, Any]*, *handle_unknown: str = 'identity'*, *unknown_value=None*) → *Expr*

`lale.expressions.string_indexer`(*subject: Expr*) → *Expr*

`lale.expressions.sum`(*group: Expr*) → *Expr*

`lale.expressions.trend`(*series: Expr*) → *Expr*

`lale.expressions.variance`(*group: Expr*) → *Expr*

`lale.expressions.window_max`(*series: Expr*, *size: int*) → *Expr*

`lale.expressions.window_max_trend`(*series: Expr*, *size: int*) → *Expr*

`lale.expressions.window_mean`(*series: Expr*, *size: int*) → *Expr*

`lale.expressions.window_mean_trend`(*series: Expr*, *size: int*) → *Expr*

`lale.expressions.window_min`(*series: Expr*, *size: int*) → *Expr*

`lale.expressions.window_min_trend`(*series: Expr*, *size: int*) → *Expr*

`lale.expressions.window_variance`(*series: Expr*, *size: int*) → *Expr*

`lale.expressions.window_variance_trend`(*series: Expr*, *size: int*) → *Expr*

lale.grammar module

class lale.grammar.**Grammar**(*variables: Optional[Dict[str,* Operator*]] = None*)

Bases: `Operator`

Base class for Lale grammars.

get_params(*deep: bool = True*) → Dict[str, Any]

For scikit-learn compatibility

input_schema_fit()

Input schema for the fit method.

is_classifier() → bool

Checks if this operator is a clasifier.

Returns
True if the classifier tag is set.

Return type
bool

is_supervised()

Checks if this operator needs labeled data for learning.

Returns
True if the fit method requires a y argument.

Return type
bool

sample(*n: int*) → *PlannedOperator*

Sample the grammar *g* starting from *g.start*, that is, choose one element at random for each possible choices.

Parameters
n (`int`) – number of derivations

Return type
PlannedOperator

transform_schema(*s_X*)

Return the output schema given the input schema.

Parameters
s_X – Input dataset or schema.

Returns
Schema of the output data given the input data schema.

Return type
JSON schema

unfold(*n: int*) → *PlannedOperator*

Explore this grammar *self.start* and generate all possible choices after *n* derivations.

Parameters
n (`int`) – number of derivations

Return type
PlannedOperator

validate_schema(*X, y=None*)

Validate that X and y are valid with respect to the input schema of this operator.

Parameters

- **X** – Features.
- **y** – Target class labels or None for unsupervised operators.

Raises
`ValueError` – If X or y are invalid as inputs.

class `lale.grammar.NonTerminal`(*name*)

Bases: `Operator`

Abstract operator for non-terminal grammar rules.

get_params(*deep: bool = True*) → Dict[str, Any]

For scikit-learn compatibility

input_schema_fit()

Input schema for the fit method.

is_classifier() → bool

Checks if this operator is a clasifier.

Returns
True if the classifier tag is set.

Return type
bool

is_supervised()

Checks if this operator needs labeled data for learning.

Returns
True if the fit method requires a y argument.

Return type
bool

transform_schema(*s_X*)

Return the output schema given the input schema.

Parameters
s_X – Input dataset or schema.

Returns
Schema of the output data given the input data schema.

Return type
JSON schema

validate_schema(*X*, *y=None*)

Validate that X and y are valid with respect to the input schema of this operator.

Parameters
- **X** – Features.
- **y** – Target class labels or None for unsupervised operators.

Raises
`ValueError` – If X or y are invalid as inputs.

lale.helpers module

class `lale.helpers.GenSym`(*names: Set[str]*)

Bases: `object`

`lale.helpers.`**add_missing_values**(*orig_X*, *missing_rate=0.1*, *seed=None*)

`lale.helpers.`**append_batch**(*data*, *batch_data*)

`lale.helpers.`**are_hyperparameters_equal**(*hyperparam1*, *hyperparam2*) `lale.helpers.arg_name`(*pos=0*, *level=1*) → Optional[str]

`lale.helpers.assignee_name`(*level=1*) → Optional[str]

`lale.helpers.create_data_loader`(*X: Any*, *y: Optional[Any] = None*, *batch_size: int = 1*, *num_workers: int = 0*, *shuffle: bool = True*)

A function that takes a dataset as input and outputs a Pytorch dataloader.

Parameters

- **X** (`Input data.`) – The formats supported are Pandas DataFrame, Numpy array, a sparse matrix, torch.tensor, torch.utils.data.Dataset, path to a HDF5 file, lale.util.batch_data_dictionary_dataset.BatchDataDict, a Python dictionary of the format *{"dataset": torch.utils.data.Dataset, "collate_fn":collate_fn for torch.utils.data.DataLoader}*
- **y** (`Labels., optional`) – Supported formats are Numpy array or Pandas series, by default None
- **batch_size** (`int, optional`) – Number of samples in each batch, by default 1
- **num_workers** (`int, optional`) – Number of workers used by the data loader, by default 0
- **shuffle** (`boolean, optional, default True`) – Whether to use SequentialSampler or RandomSampler for creating batches

Return type

torch.utils.data.DataLoader

Raises

`TypeError` – Raises a TypeError if the input format is not supported.

`lale.helpers.create_individual_op_using_reflection`(*class_name*, *operator_name*, *param_dict*)

`lale.helpers.create_instance_from_hyperopt_search_space`(*lale_object*, *hyperparams*) → *Operator*

Hyperparams is a n-tuple of dictionaries of hyper-parameters, each dictionary corresponds to an operator in the pipeline

`lale.helpers.cross_val_score`(*estimator*, *X*, *y=None*, *scoring: ~typing.Any = <function accuracy_score>*, *cv: ~typing.Any = 5*)

Use the given estimator to perform fit and predict for splits defined by 'cv' and compute the given score on each of the splits.

Parameters

- **estimator** (`A valid sklearn_wrapper estimator`) –
- **X** (`Valid data value that works with the estimator`) –
- **y** (`Valid target value that works with the estimator`) –
- **scoring** (a scorer object from sklearn.metrics (https://scikit-learn.org/stable/modules/classes.html#module-sklearn.metrics)) – Default value is accuracy_score.
- **cv** (`an integer or an object that has a split function as a generator yielding (train, test) splits as arrays of indices.`) – Integer value is used as number of folds in sklearn.model_selection.StratifiedKFold, default is 5. Note that any of the iterators from https://scikit-learn.org/stable/modules/cross_validation.html#cross-validation-iterators can be used here.

Returns

cv_results

Return type

a list of scores corresponding to each cross validation fold

`lale.helpers.cross_val_score_track_trials`(*estimator*, *X*, *y=None*, *scoring: ~typing.Any = <function accuracy_score>*, *cv: ~typing.Any = 5*, *args_to_scorer: ~typing.Optional[~typing.Dict[str, ~typing.Any]] = None*, *args_to_cv: ~typing.Optional[~typing.Dict[str, ~typing.Any]] = None*, ***fit_params*)

Use the given estimator to perform fit and predict for splits defined by 'cv' and compute the given score on each of the splits.

Parameters

- **estimator** (*A valid sklearn_wrapper estimator*) –
- **X** (*Valid data that works with the estimator*) –
- **y** (*Valid target that works with the estimator*) –
- **scoring** (*string or a scorer object created using*) – https://scikit-learn.org/stable/modules/generated/sklearn.metrics.make_scorer.html#sklearn.metrics.make_scorer. A string from sklearn.metrics.SCORERS.keys() can be used or a scorer created from one of sklearn.metrics (https://scikit-learn.org/stable/modules/classes.html#module-sklearn.metrics). A completely custom scorer object can be created from a python function following the example at https://scikit-learn.org/stable/modules/model_evaluation.html The metric has to return a scalar value,
- **cv** (*an integer or an object that has a split function as a generator yielding (train, test) splits as arrays of indices.*) – Integer value is used as number of folds in sklearn.model_selection.StratifiedKFold, default is 5. Note that any of the iterators from https://scikit-learn.org/stable/modules/cross_validation.html#cross-validation-iterators can be used here.
- **args_to_scorer** (*A dictionary of additional keyword arguments to pass to the scorer.*) – Used for cases where the scorer has a signature such as `scorer(estimator, X, y, **kwargs)`.
- **args_to_cv** (*A dictionary of additional keyword arguments to pass to the split method of cv.*) – This is only applicable when cv is not an integer.
- **fit_params** (*Additional parameters that should be passed when calling fit on the estimator*) –

Returns

cv_results

Return type

a list of scores corresponding to each cross validation fold

`lale.helpers.`**`data_to_json`**(*data*, *subsample_array: bool = True*) → Union[list, dict, int, float]

`lale.helpers.`**`dict_without`**(*orig_dict: Dict[str, Any]*, *key: str*) → Dict[str, Any]

`lale.helpers.`**`find_lale_wrapper`**(*sklearn_obj: Any*) → Optional[Any]

Parameters

sklearn_obj – An sklearn compatible object that may have a lale wrapper

Returns

The lale wrapper type, or None if one could not be found

`lale.helpers.`**`fold_schema`**(*X*, *y*, *cv=1*, *is_classifier=True*)

`lale.helpers.`**`get_estimator_param_name_from_hyperparams`**(*hyperparams*)

`lale.helpers.`**`get_name_and_index`**(*name: str*) → Tuple[str, int]

given a name of the form "name@i", returns (name, i) if given a name of the form "name", returns (name, 0)

`lale.helpers.`**`get_sklearn_estimator_name`**() → str

Some higher order sklearn operators changed the name of the nested estimatator in later versions. This returns the appropriate version dependent paramater name

`lale.helpers.`**`import_from_sklearn`**(*sklearn_obj: Any*, *fitted: bool = True*, *in_place: bool = False*)

This method take an object and tries to wrap sklearn objects (at the top level or contained within hyperparameters of other sklearn objects). It will modify the object to add in the appropriate lale wrappers. It may also return a wrapper or different object than given.

Parameters

- **sklearn_obj** – the object that we are going to try and wrap
- **fitted** – should we return a TrainedOperator
- **in_place** – should we try to mutate what we can in place, or should we aggressively deepcopy everything

Returns

The wrapped object (or the input object if we could not wrap it)

`lale.helpers.`**`import_from_sklearn_pipeline`**(*sklearn_pipeline: Any*, *fitted: bool = True*)

Note: Same as import_from_sklearn. This alternative name exists for backwards compatibility.

This method take an object and tries to wrap sklearn objects (at the top level or contained within hyperparameters of other sklearn objects). It will modify the object to add in the appropriate lale wrappers. It may also return a wrapper or different object than given.

Parameters

- **sklearn_pipeline** – the object that we are going to try and wrap
- **fitted** – should we return a TrainedOperator

Returns

The wrapped object (or the input object if we could not wrap it)

`lale.helpers.`**`instantiate_from_hyperopt_search_space`**(*obj_hyperparams*, *new_hyperparams*)

`lale.helpers.`**`is_empty_dict`**(*val*) → bool

`lale.helpers.`**`is_numeric_structure`**(*structure_type: str*)

`lale.helpers.`**`json_lookup`**(*ptr*, *jsn*, *default=None*)

`lale.helpers.`**`make_array_index_name`**(*index*, *is_tuple: bool = False*)

`lale.helpers.`**`make_degen_indexed_name`**(*name*, *index*)

`lale.helpers.`**`make_indexed_name`**(*name*, *index*)

`lale.helpers.`**`make_nested_hyperopt_space`**(*sub_space*)

`lale.helpers.`**`ndarray_to_json`**(*arr: ndarray*, *subsample_array: bool = True*) → Union[list, dict]

`lale.helpers.`**`nest_HPparam`**(*name: str*, *key: str*)

`lale.helpers.`**`nest_HPparams`**(*name: str*, *grid: Mapping[str, V]*) → Dict[str, V]

`lale.helpers.`**`nest_all_HPparams`**(*name: str*, *grids: Iterable[Mapping[str, V]]*) → List[Dict[str, V]]

Given the name of an operator in a pipeline, this transforms every key(parameter name) in the grids to use the operator name as a prefix (separated by __). This is the convention in scikit-learn pipelines.

`lale.helpers.`**`nest_choice_HPparam`**(*key: str*)

`lale.helpers.`**`nest_choice_HPparams`**(*grid: Mapping[str, V]*) → Dict[str, V]

`lale.helpers.`**`nest_choice_all_HPparams`**(*grids: Iterable[Mapping[str, V]]*) → List[Dict[str, V]]

this transforms every key(parameter name) in the grids to be nested under a choice, using a ? as a prefix (separated by __). This is the convention in scikit-learn pipelines.

`lale.helpers.`**`partition_sklearn_choice_params`**(*d: Dict[str, Any]*) → Tuple[int, Dict[str, Any]] `lale.helpers.`**`partition_sklearn_params`**`(`*d: Dict[str, Any]*`)` → Tuple[Dict[str, Any], Dict[str, Dict[str, Any]]]

`lale.helpers.`**`split_with_schemas`**`(`*estimator*, *all_X*, *all_y*, *indices*, *train_indices=None*`)`

`lale.helpers.`**`to_graphviz`**`(`*lale_operator:* Operator, *ipython_display: bool = True*, *call_depth: int = 1*, ***dot_graph_attr*`)`

`lale.helpers.`**`unnest_HPparams`**`(`*k: str*`)` → List[str]

`lale.helpers.`**`unnest_choice`**`(`*k: str*`)` → str

class `lale.helpers.`**`val_wrapper`**`(`*base*`)`

Bases: `object`

This is used to wrap values that cause problems for hyper-optimizer backends lale will unwrap these when given them as the value of a hyper-parameter

classmethod **`unwrap`**`(`*obj*`)`

`unwrap_self``()`

`lale.helpers.`**`with_fixed_estimator_name`**`(`***kwargs*`)`

Some higher order sklearn operators changed the name of the nested estimator in later versions. This fixes up the arguments, renaming estimator and base_estimator appropriately.

`lale.helpers.`**`write_batch_output_to_file`**`(`*file_obj*, *file_path*, *total_len*, *batch_idx*, *batch_X*, *batch_y*, *batch_out_X*, *batch_out_y*`)`

lale.json_operator module

`lale.json_operator.`**`from_json`**`(`*jsn: Dict[str, Any]*`)` → *Operator*

`lale.json_operator.`**`json_op_kind`**`(`*jsn: Dict[str, Any]*`)` → str

`lale.json_operator.`**`to_json`**`(`*op:* Operator, *call_depth: int = 1*, *add_custom_default: bool = False*`)` → Dict[str, Any]

lale.operator_wrapper module

`lale.operator_wrapper.`**`get_lale_wrapper_modules`**`()` → Set[str]

`lale.operator_wrapper.`**`register_lale_wrapper_modules`**`(`*m: str*`)` → None

Register a module with lale's import system so that `lale.helpers.import_from_sklearn_pipeline()` will look for replacement classes in that module.

Example: (in *__init__.py* file for the module):

```
from lale import register_lale_wrapper_modules

register_lale_wrapper_modules(__name__)
```

Parameters

m (*[str]*) – The module name

lale.operator_wrapper.**wrap_imported_operators**(*exclude_classes: Optional[Container[str]] = None, wrapper_modules: Optional[List[str]] = None*) → None

Wrap the currently imported operators from the symbol table to their lale wrappers.

Parameters

- **exclude_classes** (*string, optional, default None*) – List of class names to exclude from wrapping, alias names if they are used while importing.
- **wrapper_modules** (*set of string, optional, default None*) – Set of Lale modules to use for wrapping operators.

lale.operators module

Classes for Lale operators including individual operators, pipelines, and operator choice.

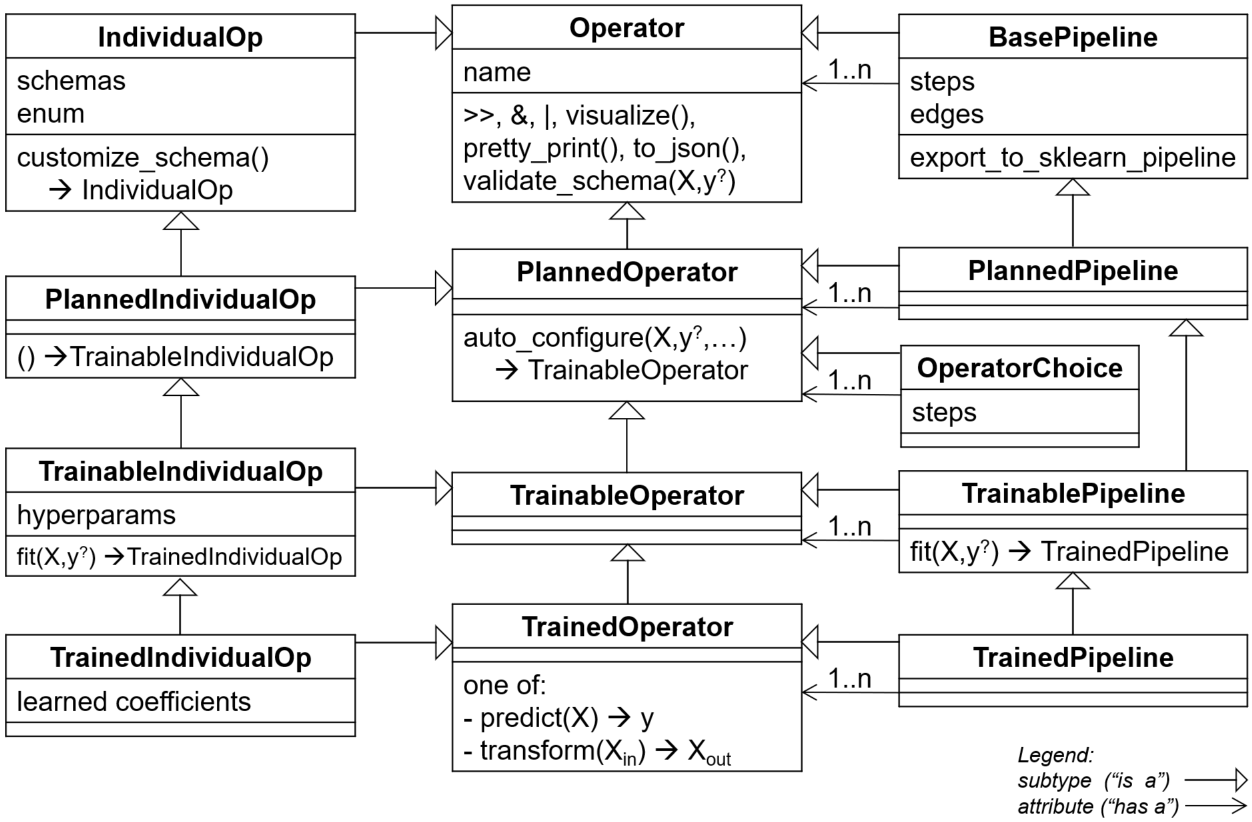
This module declares several functions for constructing individual operators, pipelines, and operator choices.

- Functions make_pipeline and Pipeline compose linear sequential pipelines, where each step has an edge to the next step. Instead of these functions you can also use the >> combinator.
- Functions make_union_no_concat and make_union compose pipelines that operate over the same data without edges between their steps. Instead of these functions you can also use the & combinator.
- Function *make_choice* creates an operator choice. Instead of this function you can also use the | combinator.
- Function make_pipeline_graph creates a pipeline from steps and edges, thus supporting any arbitrary acyclic directed graph topology.
- Function make_operator creates an individual Lale operator from a schema and an implementation class or object. This is called for each of the operators in module lale.lib when it is being imported.
- Functions get_available_operators, get_available_estimators, and get_available_transformers return lists of individual operators previously registered by *make_operator*.

The root of the hierarchy is the abstract class Operator, all other Lale operators inherit from this class, either directly or indirectly.

- The abstract classes Operator, PlannedOperator, TrainableOperator, and TrainedOperator correspond to lifecycle states.
- The concrete classes IndividualOp, PlannedIndividualOp, TrainableIndividualOp, and TrainedIndividualOp inherit from the corresponding abstract operator classes and encapsulate implementations of individual operators from machine-learning libraries such as scikit-learn.
- The concrete classes BasePipeline, PlannedPipeline, TrainablePipeline, and TrainedPipeline inherit from the corresponding abstract operator classes and represent directed acyclic graphs of operators. The steps of a pipeline can be any operators, including individual operators, other pipelines, or operator choices, whose lifecycle state is at least that of the pipeline.
- The concrete class OperatorChoice represents a planned operator that offers a choice for automated algorithm selection. The steps of a choice can be any planned operators, including individual operators, pipelines, or other operator choices.

The following picture illustrates the core operator class hierarchy.



scikit-learn compatibility:

Lale operators attempt to behave like reasonable sckit-learn operators when possible. In particular, operators support:

- get_params to return the hyperparameter settings for an operator.
- set_params for updating them (in-place). This is only supported by TrainableIndividualOps and Pipelines. Note that while set_params is supported for compatibility, but its use is not encouraged, since it mutates the operator in-place. Instead, we recommend using with_params, a functional alternative that is supported by all operators. It returns a new operator with updated parameters.
- sklearn.base.clone works for Lale operators, cloning them as expected. Note that cloning a TrainedOperator will return a TrainableOperator, since the cloned version does not have the result of training.

There also some known differences (that we are not currently planning on changing):

- Lale operators do not inherit from any sklearn base class.
- The Operator class constructors do not explicitly declare their set of hyperparameters. However, the do implement get_params, (just not using sklearn style reflection).

There may also be other incompatibilities: our testing currently focuses on ensuring that clone works.

parameter path format:

scikit-learn uses a simple addressing scheme to refer to nested hyperparameter: *name__param* refers to the *param* hyperparameter nested under the *name* object. Since lale supports richer structures, we conservatively extend this scheme as follows:

- __ : separates nested components (as-in sklearn).
- *?* : is the discriminant (choice made) for a choice.
- *?* : is also a prefix for the nested parts of the chosen branch.
- *x@n* : In a pipeline, if multiple components have identical names, everything but the first are suffixed with a number (starting with 1) indicating which one we are talking about. For example, given *(x >> y >> x)*, we would treat this much the same as *(x >> y >> x@1)*.
- *$* : is used in the rare case that sklearn would expect the key of an object, but we allow (and have) a non-object schema. In that case, $ is used as the key. This should only happen at the top level, since nested occurrences should be removed.
- *#* : is a structure indicator, and the value should be one of 'list', 'tuple', or 'dict'.
- *n* : is used to represent the nth component in an array or tuple.

class lale.operators.**BasePipeline**(*steps: List[OpType_co], edges: Optional[Iterable[Tuple[OpType_co, OpType_co]]] = None, _lale_preds: Optional[Union[Dict[int, List[int]], Dict[OpType_co, List[OpType_co]]]] = None, ordered: bool = False*)

Bases: `Operator`, `Generic[OpType_co]`

This is a concrete class that can instantiate a new pipeline operator and provide access to its meta data.

edges() → List[Tuple[OpType_co, OpType_co]]

export_to_sklearn_pipeline()

get_defaults() → Dict[str, Any]

get_last() → Optional[OpType_co]

get_params(*deep: Union[bool, Literal[0]] = True*) → Dict[str, Any]

If deep is False, additional '_lale_XXX' fields are added to support cloning. If these are not desires, deep=0 can be used to disable this

input_schema_fit() → Dict[str, Any]

Input schema for the fit method.

is_classifier() → bool

Checks if this operator is a clasifier.

Returns

True if the classifier tag is set.

Return type

bool

is_supervised() → bool

Checks if this operator needs labeled data for learning.

Returns

True if the fit method requires a y argument.

Return type

bool

remove_last(*inplace: bool = False*) → *BasePipeline*[OpType_co]

set_params(***impl_params*)

This implements the set_params, as per the scikit-learn convention, extended as documented in the module docstring

property steps: List[Tuple[str, OpType_co]]

This is meant to function similarly to the scikit-learn steps property and for linear pipelines, should behave the same

steps_list() → List[OpType_co]

transform_schema(*s_X: Dict[str, Any]*)

Return the output schema given the input schema.

Parameters

s_X – Input dataset or schema.

Returns

Schema of the output data given the input data schema.

Return type

JSON schema

validate_schema(*X: Any*, *y: Optional[Any] = None*)

Validate that X and y are valid with respect to the input schema of this operator.

Parameters

- **X** – Features.
- **y** – Target class labels or None for unsupervised operators.

Raises

ValueError – If X or y are invalid as inputs.

class lale.operators.**IndividualOp**(*_lale_name: str*, *_lale_impl*, *_lale_schemas*, *_lale_frozen_hyperparameters=None*, ***hp*)

Bases: *Operator*

This is a concrete class that can instantiate a new individual operator and provide access to its metadata. The enum property can be used to access enumerations for hyper-parameters, auto-generated from the operator's schema. For example, *LinearRegression.enum.solver.saga* As a short-hand, if the hyper-parameter name does not conflict with any fields of this class, the auto-generated enums can also be accessed directly. For example, *LinearRegression.solver.saga*

Create a new IndividualOp.

Parameters

- **name** (String) – Name of the operator.
- **impl** – An instance of operator implementation class. This is a class that contains fit, predict/transform methods implementing an underlying algorithm.
- **schemas** (*dict*) – This is a dictionary of json schemas for the operator.

MAX_FIX_DEPTH: int = 2

MAX_FIX_SUGGESTIONS: int = 3

class_name() → str

Fully qualified Python class name of this operator.

documentation_url()

property enum: _DictionaryObjectForEnum

frozen_hyperparams() → Optional[List[str]]

get_defaults() → Mapping[str, Any]

Returns the default values of hyperparameters for the operator.

Returns
A dictionary with names of the hyperparamers as keys and their default values as values.

Return type
dict

get_forwards() → Union[bool, List[str]]

Returns the list of attributes (methods/properties) the schema has asked to be forwarded. A boolean value is a blanket opt-in or out of forwarding

get_param_dist(*size=10*) → Dict[str, List[Any]]

Returns a dictionary for discretized hyperparameters.

Each entry is a list of values. For continuous hyperparameters, it returns up to *size* uniformly distributed values.

Warning: ignores side constraints, unions, and distributions.

get_param_ranges() → Tuple[Dict[str, Any], Dict[str, Any]]

Returns two dictionaries, ranges and cat_idx, for hyperparameters.

The ranges dictionary has two kinds of entries. Entries for numeric and Boolean hyperparameters are tuples of the form (min, max, default). Entries for categorical hyperparameters are lists of their values.

The cat_idx dictionary has (min, max, default) entries of indices into the corresponding list of values.

Warning: ignores side constraints and unions.

get_params(*deep: Union[bool, Literal[0]] = True*) → Dict[str, Any]

Get parameters for this operator.

This method follows scikit-learn's convention that all operators have a constructor which takes a list of keyword arguments. This is not required for operator impls which do not desire scikit-compatibility.

Parameters
deep (*boolean, optional*) – If True, will return the parameters for this operator, and their nested parameters If False, will return the parameters for this operator, along with '_lale_XXX` fields needed to support cloning

Returns
params – Parameter names mapped to their values.

Return type
mapping of string to any

get_schema(*schema_kind: str*) → Dict[str, Any]

Return a schema of the operator.

Parameters
schema_kind (*string, 'hyperparams' or 'input_fit' or 'input_partial_fit' or 'input_transform' or 'input_transform_X_y' or 'input_predict' or 'input_predict_proba' or 'input_decision_function' or 'output_transform' or 'output_transform_X_y' or 'output_predict' or 'output_predict_proba' or 'output_decision_function'*) – Type of the schema to be returned.

Returns
The Python object containing the JSON schema of the operator. For all the schemas currently present, this would be a dictionary.

Return type
dict

get_tags() → Dict[str, List[str]]

Return the tags of an operator.

Returns

A list of tags describing the operator.

Return type

list

has_method(*method_name: str*) → bool

has_schema(*schema_kind: str*) → bool

Return true if the operator has the schema kind.

Parameters

schema_kind (*string, 'hyperparams' or 'input_fit' or 'input_partial_fit' or 'input_transform' or 'input_transform_X_y' or 'input_predict' or 'input_predict_proba' or 'input_decision_function' or 'output_transform' or 'output_transform_X_y' or 'output_predict' or 'output_predict_proba' or 'output_decision_function' or 'input_score_samples' or 'output_score_samples'*) – Type of the schema to be returned.

Return type

True if the json schema is present, False otherwise.

has_tag(*tag: str*) → bool

Check the presence of a tag for an operator.

Parameters

tag (*string*) –

Returns

Flag indicating the presence or absence of the given tag in this operator's schemas.

Return type

boolean

hyperparam_schema(*name: Optional[str] = None*) → Dict[str, Any]

Returns the hyperparameter schema for the operator.

Parameters

name (*string, optional*) – Name of the hyperparameter.

Returns

Full hyperparameter schema for this operator or part of the schema corresponding to the hyperparameter given by parameter *name*.

Return type

dict

hyperparams() → Dict[str, Any]

hyperparams_all() → Optional[Dict[str, Any]]

This is the hyperparameters that are currently set. Some of them may not have been set explicitly (e.g. if this is a clone of an operator, some of these may be defaults. To get the hyperparameters that were actually set, use `hyperparams()`

property impl: Any

Returns the underlying impl. This can be used to access additional field and methods not exposed by Lale. If only the type of the impl is needed, please use self.impl_class instead, as it can be more efficient.

If the found impl has a _wrapped_model, it will be returned instead

property impl_class: type

Returns the class of the underlying impl. This should return the same thing as self.impl.__class__, but can be more efficient.

input_schema_decision_function() → Dict[str, Any]

Input schema for the decision_function method.

input_schema_fit() → Dict[str, Any]

Input schema for the fit method.

input_schema_partial_fit() → Dict[str, Any]

Input schema for the partial_fit method.

input_schema_predict() → Dict[str, Any]

Input schema for the predict method.

input_schema_predict_log_proba() → Dict[str, Any]

Input schema for the predict_log_proba method. We assume that it is the same as the predict_proba method if none has been defined explicitly.

input_schema_predict_proba() → Dict[str, Any]

Input schema for the predict_proba method.

input_schema_score_samples() → Dict[str, Any]

Input schema for the score_samples method. We assume that it is the same as the predict method if none has been defined explicitly.

input_schema_transform() → Dict[str, Any]

Input schema for the transform method.

input_schema_transform_X_y() → Dict[str, Any]

Input schema for the transform_X_y method.

is_classifier() → bool

Checks if this operator is a clasifier.

Returns
True if the classifier tag is set.

Return type
bool

is_regressor() → bool

is_supervised(*default_if_missing=True*) → bool

Checks if this operator needs labeled data for learning.

Returns
True if the fit method requires a y argument.

Return type
bool

is_transformer() → bool

Checks if the operator is a transformer

output_schema_decision_function() → Dict[str, Any]

Output schema for the decision_function method.

output_schema_predict() → Dict[str, Any]

Output schema for the predict method.

output_schema_predict_log_proba() → Dict[str, Any]

Output schema for the predict_log_proba method. We assume that it is the same as the predict_proba method if none has been defined explicitly.

output_schema_predict_proba() → Dict[str, Any]

Output schema for the predict_proba method.

output_schema_score_samples() → Dict[str, Any]

Output schema for the score_samples method. We assume that it is the same as the predict method if none has been defined explicitly.

output_schema_transform() → Dict[str, Any]

Oputput schema for the transform method.

output_schema_transform_X_y() → Dict[str, Any]

Oputput schema for the transform_X_y method.

reduced_hyperparams()

property **shallow_impl**: Any

Returns the underlying impl. This can be used to access additional field and methods not exposed by Lale. If only the type of the impl is needed, please use self.impl_class instead, as it can be more efficient.

transform_schema(*s_X: Dict[str, Any]*) → Dict[str, Any]

Return the output schema given the input schema.

Parameters
: **s_X** – Input dataset or schema.

Returns
: Schema of the output data given the input data schema.

Return type
: JSON schema

validate_schema(*X: Any*, *y: Optional[Any] = None*)

Validate that X and y are valid with respect to the input schema of this operator.

Parameters
- **X** – Features.
- **y** – Target class labels or None for unsupervised operators.

Raises
: ValueError – If X or y are invalid as inputs.

class lale.operators.**Operator**

Bases: object

Abstract base class for all Lale operators.

Pipelines and individual operators extend this.

step_1 >> step_2 -> PlannedPipeline

Pipe combinator, create two-step pipeline with edge from step_1 to step_2.

If step_1 is a pipeline, create edges from all of its sinks. If step_2 is a pipeline, create edges to all of its sources.

Parameters
- **step_1** (Operator) – The origin of the edge(s).
- **step_2** (Operator) – The destination of the edge(s).

Returns
: Pipeline with edge from step_1 to step_2.

Return type
: *BasePipeline*

step_1 & step_2 -> PlannedPipeline

And combinator, create two-step pipeline without an edge between step_1 and step_2.

Parameters

- **step_1** (Operator) – The first step.
- **step_2** (Operator) – The second step.

Returns

Pipeline without any additional edges beyond those already inside of step_1 or step_2.

Return type

BasePipeline

step_1 | step_2 -> OperatorChoice

Or combinator, create operator choice between step_1 and step_2.

Parameters

- **step_1** (Operator) – The first step.
- **step_2** (Operator) – The second step.

Returns

Algorithmic coice between step_1 or step_2.

Return type

OperatorChoice

class_name() → str

Fully qualified Python class name of this operator.

property classes_

clone() → *Operator*

Return a copy of this operator, with the same hyper-parameters but without training data This behaves the same as calling sklearn.base.clone(self)

property coef_

diff(*other: Operator*, *show_imports: bool = True*, *customize_schema: bool = False*, *ipython_display: Literal[False] = False*) → str

diff(*other: Operator*, *show_imports: bool = True*, *customize_schema: bool = False*, *ipython_display: bool = False*) → Optional[str]

Displays a diff between this operator and the given other operator.

Parameters

- **other** (Operator) – Operator to diff against
- **show_imports** (*bool, default True*) – Whether to include import statements in the pretty-printed code.
- **customize_schema** (*bool, default False*) – If True, then individual operators whose schema differs from the lale.lib version of the operator will be printed with calls to *customize_schema* that reproduce this difference.
- **ipython_display** (*bool, default False*) – If True, will display Markdown-formatted diff string in Jupyter notebook. If False, returns pretty-printing diff as Python string.

Returns

If called with ipython_display=False, return pretty-printed diff as a Python string.

Return type

str or None

property feature_importances_

get_defaults() → Mapping[str, Any]

get_forwards() → Union[bool, List[str]]

Returns the list of attributes (methods/properties) the schema has asked to be forwarded. A boolean value is a blanket opt-in or out of forwarding

get_param_dist(*size=10*) → Dict[str, List[Any]]

Returns a dictionary for discretized hyperparameters.

Each entry is a list of values. For continuous hyperparameters, it returns up to *size* uniformly distributed values.

Warning: ignores side constraints, unions, and distributions.

get_param_ranges() → Tuple[Dict[str, Any], Dict[str, Any]]

Returns two dictionaries, ranges and cat_idx, for hyperparameters.

The ranges dictionary has two kinds of entries. Entries for numeric and Boolean hyperparameters are tuples of the form (min, max, default). Entries for categorical hyperparameters are lists of their values.

The cat_idx dictionary has (min, max, default) entries of indices into the corresponding list of values.

Warning: ignores side constraints and unions.

abstract **get_params**(*deep: bool = True*) → Dict[str, Any]

For scikit-learn compatibility

abstract **input_schema_fit**() → Dict[str, Any]

Input schema for the fit method.

abstract **is_classifier**() → bool

Checks if this operator is a clasifier.

Returns
True if the classifier tag is set.

Return type
bool

is_frozen_trainable() → bool

Return true if all hyperparameters are bound, in other words, search spaces contain no free hyperparameters to be tuned.

is_frozen_trained() → bool

Return true if all learnable coefficients are bound, in other words, there are no free parameters to be learned by fit.

abstract **is_supervised**() → bool

Checks if this operator needs labeled data for learning.

Returns
True if the fit method requires a y argument.

Return type
bool

property **n_classes_**

name() → str

Get the name of this operator instance.

pretty_print(**, show_imports: bool = True, combinators: bool = True, assign_nested: bool = True, customize_schema: bool = False, astype: Literal['lale', 'sklearn'] = 'lale', ipython_display: Literal[False] = False*) → str `pretty_print`(*, *show_imports: bool = True*, *combinators: bool = True*, *assign_nested: bool = True*, *customize_schema: bool = False*, *astype: Literal['lale', 'sklearn'] = 'lale'*, *ipython_display: Union[bool, Literal['input']] = False*) → Optional[str]

Returns the Python source code representation of the operator.

Parameters

- **show_imports** (*bool, default True*) – Whether to include import statements in the pretty-printed code.
- **combinators** (*bool, default True*) – If True, pretty-print with combinators (>>, |, &). Otherwise, pretty-print with functions (*make_pipeline*, *make_choice*, *make_union*) instead. Always False when astype is 'sklearn'.
- **assign_nested** (*bool, default True*) – If True, then nested operators, such as the base estimator for an ensemble, get assigned to fresh intermediate variables if configured with non-trivial arguments of their own.
- **customize_schema** (*bool, default False*) – If True, then individual operators whose schema differs from the lale.lib version of the operator will be printed with calls to *customize_schema* that reproduce this difference.
- **astype** (*union type, default 'lale'*) –
  - 'lale'

    Use *lale.operators.make_pipeline* and *lale.operators.make_union* when pretty-printing wth functions.
  - 'sklearn'

    Set combinators to False and use *sklearn.pipeline.make_pipeline* and *sklearn.pipeline.make_union* for pretty-printed functions.
- **ipython_display** (*union type, default False*) –
  - False

    Return the pretty-printed code as a plain old Python string.
  - True:

    Pretty-print in notebook cell output with syntax highlighting.
  - 'input'

    Create a new notebook cell with pretty-printed code as input.

Returns

If called with ipython_display=False, return pretty-printed Python source code as a Python string.

Return type

str or None

`replace`(*original_op: Operator*, *replacement_op: Operator*) → *Operator*

Replaces an original operator with a replacement operator for the given operator. Replacement also occurs for all operators within the given operator's steps (i.e. pipelines and choices). If a planned operator is given as original_op, all derived operators (including trainable and trained versions) will be replaced. Otherwise, only the exact operator instance will be replaced.

Parameters

- **original_op** – Operator to replace within given operator. If operator is a planned operator, all derived operators (including trainable and trained versions) will be replaced. Otherwise, only the exact operator instance will be replaced.
- **replacement_op** – Operator to replace the original with.

Returns

Modified operator where original operator is replaced with replacement throughout.

Return type

modified_operator

to_json() → Dict[str, Any]

Returns the JSON representation of the operator.

Returns
JSON representation that describes this operator and is valid with respect to lale.json_operator.SCHEMA.

Return type
JSON document

to_lale()

This is a deprecated method for backward compatibility and will be removed soon

abstract **transform_schema**(*s_X: Dict[str, Any]*) → Dict[str, Any]

Return the output schema given the input schema.

Parameters
s_X – Input dataset or schema.

Returns
Schema of the output data given the input data schema.

Return type
JSON schema

abstract **validate_schema**(*X: Any*, *y: Optional[Any] = None*)

Validate that X and y are valid with respect to the input schema of this operator.

Parameters
- **X** – Features.
- **y** – Target class labels or None for unsupervised operators.

Raises
ValueError – If X or y are invalid as inputs.

visualize(*ipython_display: bool = True*)

Visualize the operator using graphviz (use in a notebook).

Parameters
ipython_display (*bool, default True*) – If True, proactively ask Jupyter to render the graph. Otherwise, the graph will only be rendered when visualize() was called in the last statement in a notebook cell.

Returns
Digraph object from the graphviz package.

Return type
Digraph

with_params(***impl_params*) → *Operator*

This implements a functional version of set_params which returns a new operator instead of modifying the original

class lale.operators.**OperatorChoice**(*steps*, *name: Optional[str] = None*)

Bases: *PlannedOperator*, Generic[OperatorChoiceType_co]

fit(*X: Any*, *y: Optional[Any] = None*, ***fit_params*)

get_defaults() → Mapping[str, Any]

get_params(*deep: bool = True*) → Dict[str, Any]

For scikit-learn compatibility

input_schema_fit() → Dict[str, Any]

Input schema for the fit method.

`is_classifier()` → bool

Checks if this operator is a clasifier.

Returns
True if the classifier tag is set.

Return type
bool

`is_frozen_trainable()` → bool

Return true if all hyperparameters are bound, in other words, search spaces contain no free hyperparameters to be tuned.

`is_supervised()` → bool

Checks if this operator needs labeled data for learning.

Returns
True if the fit method requires a y argument.

Return type
bool

`set_params`(***impl_params*)

This implements the set_params, as per the scikit-learn convention, extended as documented in the module docstring

`property steps: List[Tuple[str, OperatorChoiceType_co]]`

This is meant to function similarly to the scikit-learn steps property and for linear pipelines, should behave the same

`steps_list()` → List[OperatorChoiceType_co]

`transform_schema`(*s_X: Dict[str, Any]*)

Return the output schema given the input schema.

Parameters
s_X – Input dataset or schema.

Returns
Schema of the output data given the input data schema.

Return type
JSON schema

`validate_schema`(*X: Any*, *y: Optional[Any] = None*)

Validate that X and y are valid with respect to the input schema of this operator.

Parameters

- **X** – Features.
- **y** – Target class labels or None for unsupervised operators.

Raises
`ValueError` – If X or y are invalid as inputs.

`class lale.operators.PlannedIndividualOp`(*_lale_name: str*, *_lale_impl*, *_lale_schemas*, *_lale_frozen_hyperparameters=None*, *_lale_trained=False*, ***hp*)

Bases: *IndividualOp*, *PlannedOperator*

This is a concrete class that returns a trainable individual operator through its __call__ method. A configure method can use an optimizer and return the best hyperparameter combination.

Create a new IndividualOp.

Parameters

- **name** (String) – Name of the operator.

- **impl** – An instance of operator implementation class. This is a class that contains fit, predict/transform methods implementing an underlying algorithm.
- **schemas** (*dict*) – This is a dictionary of json schemas for the operator.

auto_configure(*X: Any, y: Optional[Any] = None, optimizer=None, cv=None, scoring=None, **kwargs*) → *TrainedIndividualOp*

Perform combined algorithm selection and hyperparameter tuning on this planned operator.

Parameters

- **X** – Features that conform to the X property of input_schema_fit.
- **y** (*optional*) – Labels that conform to the y property of input_schema_fit. Default is None.
- **optimizer** – lale.lib.lale.Hyperopt or lale.lib.lale.GridSearchCV default is None.
- **cv** – cross-validation option that is valid for the optimizer. Default is None, which will use the optimizer's default value.
- **scoring** – scoring option that is valid for the optimizer. Default is None, which will use the optimizer's default value.
- **kwargs** – Other keyword arguments to be passed to the optimizer.

Returns

Best operator discovered by the optimizer.

Return type

TrainableOperator

Raises

ValueError – If an invalid optimizer is provided

customize_schema(*schemas: Optional[Schema] = None, relevantToOptimizer: Optional[List[str]] = None, constraint: Optional[Union[Schema, Dict[str, Any], List[Union[Schema, Dict[str, Any]]]]] = None, tags: Optional[Dict] = None, forwards: Optional[Union[bool, List[str]]] = None, set_as_available: bool = False, **kwargs: Optional[Union[Schema, Dict[str, Any]]]*) → *PlannedIndividualOp*

free_hyperparams()

freeze_trainable() → *TrainableIndividualOp*

is_frozen_trainable() → bool

Return true if all hyperparameters are bound, in other words, search spaces contain no free hyperparameters to be tuned.

class lale.operators.**PlannedOperator**

Bases: *Operator*

Abstract class for Lale operators in the planned lifecycle state.

step_1 >> step_2 -> PlannedPipeline

Pipe combinator, create two-step pipeline with edge from step_1 to step_2.

If step_1 is a pipeline, create edges from all of its sinks. If step_2 is a pipeline, create edges to all of its sources.

Parameters

- **step_1** (Operator) – The origin of the edge(s).
- **step_2** (Operator) – The destination of the edge(s).

Returns

Pipeline with edge from step_1 to step_2.

Return type

BasePipeline

`step_1 & step_2 -> PlannedPipeline`

And combinator, create two-step pipeline without an edge between step_1 and step_2.

Parameters

- **step_1** (Operator) – The first step.
- **step_2** (Operator) – The second step.

Returns

Pipeline without any additional edges beyond those already inside of step_1 or step_2.

Return type

BasePipeline

`step_1 | step_2 -> OperatorChoice`

Or combinator, create operator choice between step_1 and step_2.

Parameters

- **step_1** (Operator) – The first step.
- **step_2** (Operator) – The second step.

Returns

Algorithmic coice between step_1 or step_2.

Return type

OperatorChoice

auto_configure(*X: Any*, *y: Optional[Any] = None*, *optimizer: Optional[*PlannedIndividualOp*] = None*, *cv: Optional[Any] = None*, *scoring: Optional[Any] = None*, ***kwargs*) → *TrainedOperator*

Perform combined algorithm selection and hyperparameter tuning on this planned operator.

Parameters

- **X** – Features that conform to the X property of input_schema_fit.
- **y** (*optional*) – Labels that conform to the y property of input_schema_fit. Default is None.
- **optimizer** – lale.lib.lale.Hyperopt or lale.lib.lale.GridSearchCV default is None.
- **cv** – cross-validation option that is valid for the optimizer. Default is None, which will use the optimizer's default value.
- **scoring** – scoring option that is valid for the optimizer. Default is None, which will use the optimizer's default value.
- **kwargs** – Other keyword arguments to be passed to the optimizer.

Returns

Best operator discovered by the optimizer.

Return type

TrainableOperator

Raises

ValueError – If an invalid optimizer is provided

class lale.operators.**PlannedPipeline**(*steps: List[PlannedOpType_co]*, *edges: Optional[Iterable[Tuple[PlannedOpType_co, PlannedOpType_co]]] = None*, *_lale_preds: Optional[Dict[int, List[int]]] = None*, *ordered: bool = False*)

Bases: `BasePipeline`[PlannedOpType_co], `PlannedOperator`

auto_configure(*X: Any*, *y: Optional[Any] = None*, *optimizer=None*, *cv=None*, *scoring=None*, ***kwargs*) → *TrainedPipeline*

Perform combined algorithm selection and hyperparameter tuning on this planned operator.

Parameters

- **X** – Features that conform to the X property of input_schema_fit.
- **y** (*optional*) – Labels that conform to the y property of input_schema_fit. Default is None.

- **optimizer** – lale.lib.lale.Hyperopt or lale.lib.lale.GridSearchCV default is None.
- **cv** – cross-validation option that is valid for the optimizer. Default is None, which will use the optimizer's default value.
- **scoring** – scoring option that is valid for the optimizer. Default is None, which will use the optimizer's default value.
- **kwargs** – Other keyword arguments to be passed to the optimizer.

Returns
Best operator discovered by the optimizer.

Return type
TrainableOperator

Raises
`ValueError` – If an invalid optimizer is provided

is_frozen_trainable() → bool

Return true if all hyperparameters are bound, in other words, search spaces contain no free hyperparameters to be tuned.

is_frozen_trained() → bool

Return true if all learnable coefficients are bound, in other words, there are no free parameters to be learned by fit.

remove_last(*inplace: bool = False*) → *PlannedPipeline*[PlannedOpType_co]

class lale.operators.**TrainableIndividualOp**(*_lale_name*, *_lale_impl*, *_lale_schemas*, *_lale_frozen_hyperparameters=None*, ***hp*)

Bases: *PlannedIndividualOp*, *TrainableOperator*

Create a new IndividualOp.

Parameters
- **name** (`String`) – Name of the operator.
- **impl** – An instance of operator implementation class. This is a class that contains fit, predict/transform methods implementing an underlying algorithm.
- **schemas** (*dict*) – This is a dictionary of json schemas for the operator.

convert_to_trained() → *TrainedIndividualOp*

customize_schema(*schemas: Optional[Schema] = None*, *relevantToOptimizer: Optional[List[str]] = None*, *constraint: Optional[Union[Schema, Dict[str, Any], List[Union[Schema, Dict[str, Any]]]]] = None*, *tags: Optional[Dict] = None*, *forwards: Optional[Union[bool, List[str]]] = None*, *set_as_available: bool = False*, ***kwargs: Optional[Union[Schema, Dict[str, Any]]]*) → *TrainableIndividualOp*

decision_function(*X=None*)

Deprecated since version 0.0.0: The *decision_function* method is deprecated on a trainable operator, because the learned coefficients could be accidentally overwritten by retraining. Call *decision_function* on the trained operator returned by *fit* instead.

fit(*X: Any*, *y: Optional[Any] = None*, ***fit_params*) → *TrainedIndividualOp*

Train the learnable coefficients of this operator, if any.

Return a trained version of this operator. If this operator has free learnable coefficients, bind them to values that fit the data according to the operator's algorithm. Do nothing if the operator implementation lacks a *fit* method or if the operator has been marked as *is_frozen_trained*.

Parameters
- **X** – Features that conform to the X property of input_schema_fit.
- **y** (*optional*) – Labels that conform to the y property of input_schema_fit. Default is None.

- **fit_params** (*Dictionary, optional*) – A dictionary of keyword parameters to be used during training.

Returns
A new copy of this operators that is the same except that its learnable coefficients are bound to their trained values.

Return type
TrainedOperator

free_hyperparams() → Set[str]

freeze_trainable() → *TrainableIndividualOp*

Return a copy of the trainable parts of this operator that is the same except that all hyperparameters are bound and none are free to be tuned. If there is an operator choice, it is kept as is.

freeze_trained() → *TrainedIndividualOp*

Deprecated since version 0.0.0: The *freeze_trained* method is deprecated on a trainable operator, because the learned coefficients could be accidentally overwritten by retraining. Call *freeze_trained* on the trained operator returned by *fit* instead.

get_pipeline(*pipeline_name: Optional[str] = None*, *astype: Literal['lale', 'sklearn'] = 'lale'*) → Optional[*TrainableOperator*]

Deprecated since version 0.0.0: The *get_pipeline* method is deprecated on a trainable operator, because the learned coefficients could be accidentally overwritten by retraining. Call *get_pipeline* on the trained operator returned by *fit* instead.

input_schema_fit() → Dict[str, Any]

Input schema for the fit method.

partial_fit(*X: Any*, *y: Optional[Any] = None*, ***fit_params*) → *TrainedIndividualOp*

predict(*X=None*, ***predict_params*) → Any

Deprecated since version 0.0.0: The *predict* method is deprecated on a trainable operator, because the learned coefficients could be accidentally overwritten by retraining. Call *predict* on the trained operator returned by *fit* instead.

predict_log_proba(*X=None*)

Deprecated since version 0.0.0: The *predict_log_proba* method is deprecated on a trainable operator, because the learned coefficients could be accidentally overwritten by retraining. Call *predict_log_proba* on the trained operator returned by *fit* instead.

predict_proba(*X=None*)

Deprecated since version 0.0.0: The *predict_proba* method is deprecated on a trainable operator, because the learned coefficients could be accidentally overwritten by retraining. Call *predict_proba* on the trained operator returned by *fit* instead.

score(*X*, *y*, ***score_params*) → Any

Deprecated since version 0.0.0: The *score* method is deprecated on a trainable operator, because the learned coefficients could be accidentally overwritten by retraining. Call *score* on the trained operator returned by *fit* instead.

score_samples(*X=None*)

Deprecated since version 0.0.0: The *score_samples* method is deprecated on a trainable operator, because the learned coefficients could be accidentally overwritten by retraining. Call *score_samples* on the trained operator returned by *fit* instead.

set_params(***impl_params*)

This implements the set_params, as per the scikit-learn convention, extended as documented in the module docstring

summary() → DataFrame

Deprecated since version 0.0.0: The *summary* method is deprecated on a trainable operator, because the learned coefficients could be accidentally overwritten by retraining. Call *summary* on the trained operator returned by *fit* instead.

transform(*X: Any*, *y: Any = None*) → Any

Deprecated since version 0.0.0: The *transform* method is deprecated on a trainable operator, because the learned coefficients could be accidentally overwritten by retraining. Call *transform* on the trained operator returned by *fit* instead.

transform_schema(*s_X: Dict[str, Any]*)

Return the output schema given the input schema.

Parameters
: **s_X** – Input dataset or schema.

Returns
: Schema of the output data given the input data schema.

Return type
: JSON schema

class lale.operators.**TrainableOperator**

Bases: *PlannedOperator*

Abstract class for Lale operators in the trainable lifecycle state.

step_1 >> step_2 -> PlannedPipeline

Pipe combinator, create two-step pipeline with edge from step_1 to step_2.

If step_1 is a pipeline, create edges from all of its sinks. If step_2 is a pipeline, create edges to all of its sources.

Parameters
- **step_1** (Operator) – The origin of the edge(s).
- **step_2** (Operator) – The destination of the edge(s).

Returns
: Pipeline with edge from step_1 to step_2.

Return type
: *BasePipeline*

step_1 & step_2 -> PlannedPipeline

And combinator, create two-step pipeline without an edge between step_1 and step_2.

Parameters
- **step_1** (Operator) – The first step.
- **step_2** (Operator) – The second step.

Returns
: Pipeline without any additional edges beyond those already inside of step_1 or step_2.

Return type
: *BasePipeline*

step_1 | step_2 -> OperatorChoice

Or combinator, create operator choice between step_1 and step_2.

Parameters
- **step_1** (Operator) – The first step.
- **step_2** (Operator) – The second step.

Returns
Algorithmic coice between step_1 or step_2.

Return type
OperatorChoice

abstract fit(*X: Any*, *y: Optional[Any] = None*, ***fit_params*) → *TrainedOperator*

Train the learnable coefficients of this operator, if any.

Return a trained version of this operator. If this operator has free learnable coefficients, bind them to values that fit the data according to the operator's algorithm. Do nothing if the operator implementation lacks a *fit* method or if the operator has been marked as *is_frozen_trained*.

Parameters

- **X** – Features that conform to the X property of input_schema_fit.
- **y** (`optional`) – Labels that conform to the y property of input_schema_fit. Default is None.
- **fit_params** (`Dictionary, optional`) – A dictionary of keyword parameters to be used during training.

Returns
A new copy of this operators that is the same except that its learnable coefficients are bound to their trained values.

Return type
TrainedOperator

fit_transform(*X: Any*, *y: Optional[Any] = None*, ***fit_params*)

Fit to data, then transform it.

Fits transformer to *X* and *y* with optional parameters *fit_params* and returns a transformed version of *X*.

Parameters

- **X** – Features that conform to the X property of input_schema_fit.
- **y** (`optional`) – Labels that conform to the y property of input_schema_fit. Default is None.
- **fit_params** (`Dictionary, optional`) – A dictionary of keyword parameters to be used during training.

Returns
Transformed features; see output_transform schema of the operator.

Return type
result

abstract freeze_trainable() → *TrainableOperator*

Return a copy of the trainable parts of this operator that is the same except that all hyperparameters are bound and none are free to be tuned. If there is an operator choice, it is kept as is.

abstract is_transformer() → bool

Checks if the operator is a transformer

class lale.operators.**TrainablePipeline**(*steps: List[TrainableOpType_co]*, *edges: Optional[Iterable[Tuple[TrainableOpType_co, TrainableOpType_co]]] = None*, *_lale_preds: Optional[Dict[int, List[int]]] = None*, *ordered: bool = False*, *_lale_trained=False*)

Bases: `PlannedPipeline`[`TrainableOpType_co`], `TrainableOperator`

convert_to_trained() → *TrainedPipeline*[*TrainedIndividualOp*]

decision_function(*X*)

Deprecated since version 0.0.0: The *decision_function* method is deprecated on a trainable operator, because the learned coefficients could be accidentally overwritten by retraining. Call *decision_function* on the trained operator returned by *fit* instead.

fit(*X: Any*, *y: Optional[Any] = None*, ***fit_params*) → *TrainedPipeline[TrainedIndividualOp]*

Train the learnable coefficients of this operator, if any.

Return a trained version of this operator. If this operator has free learnable coefficients, bind them to values that fit the data according to the operator's algorithm. Do nothing if the operator implementation lacks a *fit* method or if the operator has been marked as *is_frozen_trained*.

Parameters

- **X** – Features that conform to the X property of input_schema_fit.
- **y** (`optional`) – Labels that conform to the y property of input_schema_fit. Default is None.
- **fit_params** (`Dictionary, optional`) – A dictionary of keyword parameters to be used during training.

Returns

A new copy of this operators that is the same except that its learnable coefficients are bound to their trained values.

Return type

TrainedOperator

freeze_trainable() → *TrainablePipeline*

Return a copy of the trainable parts of this operator that is the same except that all hyperparameters are bound and none are free to be tuned. If there is an operator choice, it is kept as is.

freeze_trained() → *TrainedPipeline*

is_transformer() → bool

Checks if the operator is a transformer

partial_fit(*X: Any*, *y: Optional[Any] = None*, *freeze_trained_prefix: bool = True*, *unsafe: bool = False*, ***fit_params*) → *TrainedPipeline[TrainedIndividualOp]*

partial_fit for a pipeline. This method assumes that all but the last node of a pipeline are frozen_trained and only the last node needs to be fit using its partial_fit method. If that is not the case, and *freeze_trained_prefix* is True, it freezes the prefix of the pipeline except the last node if they are trained.

Parameters

- **X** – Features; see partial_fit schema of the last node.
- **y** – Labels/target
- **freeze_trained_prefix** – If True, all but the last node are freeze_trained and only the last node is partial_fit.
- **unsafe** – boolean. This flag allows users to override the validation that throws an error when the the operators in the prefix of this pipeline are not tagged with *has_partial_transform*. Setting unsafe to True would perform the transform as if it was row-wise even in the case it may not be.
- **fit_params** – dict Additional keyword arguments to be passed to partial_fit of the estimator

Returns

A partially trained pipeline, which can be trained further by other calls to partial_fit

Return type

TrainedPipeline

Raises

`ValueError` – The piepline has a non-frozen prefix

predict(*X*, ***predict_params*) → Any

Deprecated since version 0.0.0: The *predict* method is deprecated on a trainable operator, because the learned coefficients could be accidentally overwritten by retraining. Call *predict* on the trained operator returned by *fit* instead.

predict_log_proba(*X*)

Deprecated since version 0.0.0: The *predict_log_proba* method is deprecated on a trainable operator, because the learned coefficients could be accidentally overwritten by retraining. Call *predict_log_proba* on the trained operator returned by *fit* instead.

predict_proba(*X*)

Deprecated since version 0.0.0: The *predict_proba* method is deprecated on a trainable operator, because the learned coefficients could be accidentally overwritten by retraining. Call *predict_proba* on the trained operator returned by *fit* instead.

remove_last(*inplace: bool = False*) → *TrainablePipeline*[TrainableOpType_co]

score(*X*, *y*, ***score_params*)

Deprecated since version 0.0.0: The *score* method is deprecated on a trainable operator, because the learned coefficients could be accidentally overwritten by retraining. Call *score* on the trained operator returned by *fit* instead.

score_samples(*X=None*)

Deprecated since version 0.0.0: The *score_samples* method is deprecated on a trainable operator, because the learned coefficients could be accidentally overwritten by retraining. Call *score_samples* on the trained operator returned by *fit* instead.

transform(*X: Any*, *y=None*) → Any

Deprecated since version 0.0.0: The *transform* method is deprecated on a trainable operator, because the learned coefficients could be accidentally overwritten by retraining. Call *transform* on the trained operator returned by *fit* instead.

class lale.operators.**TrainedIndividualOp**(**args*, *_lale_trained=False*, *_lale_impl=None*, ***kwargs*)

Bases: `TrainableIndividualOp`, `TrainedOperator`

Create a new IndividualOp.

Parameters

- **name** (`String`) – Name of the operator.
- **impl** – An instance of operator implementation class. This is a class that contains fit, predict/transform methods implementing an underlying algorithm.
- **schemas** (`dict`) – This is a dictionary of json schemas for the operator.

customize_schema(*schemas: Optional[Schema] = None*, *relevantToOptimizer: Optional[List[str]] = None*, *constraint: Optional[Union[Schema, Dict[str, Any], List[Union[Schema, Dict[str, Any]]]]] = None*, *tags: Optional[Dict] = None*, *forwards: Optional[Union[bool, List[str]]] = None*, *set_as_available: bool = False*, ***kwargs: Optional[Union[Schema, Dict[str, Any]]]*) → *TrainedIndividualOp*

decision_function(*X: Any = None*)

Confidence scores for all classes.

Parameters

X – Features; see input_decision_function schema of the operator.

Returns

Confidences; see output_decision_function schema of the operator.

Return type

result

fit(*X: Any*, *y: Optional[Any] = None*, ***fit_params*) → *TrainedIndividualOp*

Train the learnable coefficients of this operator, if any.

Return a trained version of this operator. If this operator has free learnable coefficients, bind them to values that fit the data according to the operator's algorithm. Do nothing if the operator implementation lacks a *fit* method or if the operator has been marked as *is_frozen_trained*.

Parameters

- **X** – Features that conform to the X property of input_schema_fit.
- **y** (*optional*) – Labels that conform to the y property of input_schema_fit. Default is None.
- **fit_params** (*Dictionary, optional*) – A dictionary of keyword parameters to be used during training.

Returns

A new copy of this operators that is the same except that its learnable coefficients are bound to their trained values.

Return type

TrainedOperator

freeze_trainable() → *TrainedIndividualOp*

Return a copy of the trainable parts of this operator that is the same except that all hyperparameters are bound and none are free to be tuned. If there is an operator choice, it is kept as is.

freeze_trained() → *TrainedIndividualOp*

Deprecated since version 0.0.0: The *freeze_trained* method is deprecated on a trainable operator, because the learned coefficients could be accidentally overwritten by retraining. Call *freeze_trained* on the trained operator returned by *fit* instead.

get_pipeline(*pipeline_name: None = None*, *astype: astype_type = 'lale'*) → Optional[*TrainedOperator*]

get_pipeline(*pipeline_name: str*, *astype: astype_type = 'lale'*) → Optional[*TrainableOperator*]

Deprecated since version 0.0.0: The *get_pipeline* method is deprecated on a trainable operator, because the learned coefficients could be accidentally overwritten by retraining. Call *get_pipeline* on the trained operator returned by *fit* instead.

is_frozen_trained() → bool

Return true if all learnable coefficients are bound, in other words, there are no free parameters to be learned by fit.

partial_fit(*X: Any*, *y: Optional[Any] = None*, ***fit_params*) → *TrainedIndividualOp*

predict(*X: Any = None*, ***predict_params*) → Any

Make predictions.

Parameters

- **X** – Features; see input_predict schema of the operator.
- **predict_params** – Additional parameters that should be passed to the predict method

Returns

Predictions; see output_predict schema of the operator.

Return type

result

predict_log_proba(*X: Any = None*)

Predicted class log-probabilities for X.

Parameters

X – Features.

Returns

Class log probabilities.

Return type

result

predict_proba(*X: Any = None*)

Probability estimates for all classes.

Parameters
X – Features; see input_predict_proba schema of the operator.
Returns
Probabilities; see output_predict_proba schema of the operator.
Return type
result

`score`(*X: Any*, *y: Any*, ***score_params*) → Any

Performance evaluation with a default metric.

Parameters
- **X** – Features.
- **y** – Ground truth labels.
- **score_params** – Any additional parameters expected by the score function of the underlying operator.

Returns
performance metric value
Return type
score

`score_samples`(*X: Any = None*)

Scores for each sample in X. The type of scores depends on the operator.

Parameters
X – Features.
Returns
scores per sample.
Return type
result

`summary`() → DataFrame

Deprecated since version 0.0.0: The *summary* method is deprecated on a trainable operator, because the learned coefficients could be accidentally overwritten by retraining. Call *summary* on the trained operator returned by *fit* instead.

`transform`(*X: Any*, *y: Any = None*) → Any

Transform the data.

Parameters
- **X** – Features; see input_transform schema of the operator.
- **y** (*None*) –

Returns
Transformed features; see output_transform schema of the operator.
Return type
result

`transform_X_y`(*X: Any*, *y: Any*) → Any

Transform the data and target.

Parameters
- **X** – Features; see input_transform schema of the operator.
- **y** – target; see input_transform schema of the operator.

Returns
Transformed features and target; see output_transform schema of the operator.
Return type
result

class `lale.operators.TrainedOperator`

Bases: *TrainableOperator*

Abstract class for Lale operators in the trained lifecycle state.

step_1 >> step_2 -> PlannedPipeline

Pipe combinator, create two-step pipeline with edge from step_1 to step_2.

If step_1 is a pipeline, create edges from all of its sinks. If step_2 is a pipeline, create edges to all of its sources.

Parameters

- **step_1** (Operator) – The origin of the edge(s).
- **step_2** (Operator) – The destination of the edge(s).

Returns

Pipeline with edge from step_1 to step_2.

Return type

BasePipeline

step_1 & step_2 -> PlannedPipeline

And combinator, create two-step pipeline without an edge between step_1 and step_2.

Parameters

- **step_1** (Operator) – The first step.
- **step_2** (Operator) – The second step.

Returns

Pipeline without any additional edges beyond those already inside of step_1 or step_2.

Return type

BasePipeline

step_1 | step_2 -> OperatorChoice

Or combinator, create operator choice between step_1 and step_2.

Parameters

- **step_1** (Operator) – The first step.
- **step_2** (Operator) – The second step.

Returns

Algorithmic coice between step_1 or step_2.

Return type

OperatorChoice

abstract decision_function(*X: Any*)

Confidence scores for all classes.

Parameters

X – Features; see input_decision_function schema of the operator.

Returns

Confidences; see output_decision_function schema of the operator.

Return type

result

abstract freeze_trained() → *TrainedOperator*

Return a copy of this trainable operator that is the same except that all learnable coefficients are bound and thus fit is a no-op.

abstract predict(*X: Any*, ***predict_params*) → Any

Make predictions.

Parameters

- **X** – Features; see input_predict schema of the operator.
- **predict_params** – Additional parameters that should be passed to the predict method

Returns

Predictions; see output_predict schema of the operator.

Return type
result

abstract predict_log_proba(*X: Any*)

Predicted class log-probabilities for X.

Parameters
X – Features.

Returns
Class log probabilities.

Return type
result

abstract predict_proba(*X: Any*)

Probability estimates for all classes.

Parameters
X – Features; see input_predict_proba schema of the operator.

Returns
Probabilities; see output_predict_proba schema of the operator.

Return type
result

abstract score(*X: Any*, *y: Any*, ***score_params*)

Performance evaluation with a default metric.

Parameters

- **X** – Features.
- **y** – Ground truth labels.
- **score_params** – Any additional parameters expected by the score function of the underlying operator.

Returns
performance metric value

Return type
score

abstract score_samples(*X: Any*)

Scores for each sample in X. The type of scores depends on the operator.

Parameters
X – Features.

Returns
scores per sample.

Return type
result

abstract transform(*X: Any*, *y: Optional[Any] = None*) → Any

Transform the data.

Parameters

- **X** – Features; see input_transform schema of the operator.
- **y** (*None*) –

Returns
Transformed features; see output_transform schema of the operator.

Return type
result

class lale.operators.**TrainedPipeline**(**args*, *_lale_trained=False*, ***kwargs*)

Bases: *TrainablePipeline*[TrainedOpType_co], *TrainedOperator*

decision_function(*X: Any*)

Confidence scores for all classes.

Parameters

X – Features; see input_decision_function schema of the operator.

Returns

Confidences; see output_decision_function schema of the operator.

Return type

result

freeze_trainable() → *TrainedPipeline*

Return a copy of the trainable parts of this operator that is the same except that all hyperparameters are bound and none are free to be tuned. If there is an operator choice, it is kept as is.

partial_fit(*X: Any*, *y: Optional[Any] = None*, *freeze_trained_prefix: bool = True*, *unsafe: bool = False*, *classes: Optional[Any] = None*, ***fit_params*) → *TrainedPipeline[TrainedIndividualOp]*

partial_fit for a pipeline. This method assumes that all but the last node of a pipeline are frozen_trained and only the last node needs to be fit using its partial_fit method. If that is not the case, and *freeze_trained_prefix* is True, it freezes the prefix of the pipeline except the last node if they are trained.

Parameters

- **X** – Features; see partial_fit schema of the last node.
- **y** – Labels/target
- **freeze_trained_prefix** – If True, all but the last node are freeze_trained and only the last node is partial_fit.
- **unsafe** – boolean. This flag allows users to override the validation that throws an error when the the operators in the prefix of this pipeline are not tagged with *has_partial_transform*. Setting unsafe to True would perform the transform as if it was row-wise even in the case it may not be.
- **fit_params** – dict Additional keyword arguments to be passed to partial_fit of the estimator
- **classes** (*Any*) –

Returns

A partially trained pipeline, which can be trained further by other calls to partial_fit

Return type

TrainedPipeline

Raises

ValueError – The piepline has a non-frozen prefix

predict(*X*, ***predict_params*) → Any

Deprecated since version 0.0.0: The *predict* method is deprecated on a trainable operator, because the learned coefficients could be accidentally overwritten by retraining. Call *predict* on the trained operator returned by *fit* instead.

predict_log_proba(*X: Any*)

Predicted class log-probabilities for X.

Parameters

X – Features.

Returns

Class log probabilities.

Return type

result

predict_proba(*X: Any*)

Probability estimates for all classes.

Parameters

X – Features; see input_predict_proba schema of the operator.

Returns
Probabilities; see output_predict_proba schema of the operator.

Return type
result

`remove_last`(*inplace: bool = False*) → *TrainedPipeline*[TrainedOpType_co]

`score`(*X: Any*, *y: Any*, ***score_params*)

Performance evaluation with a default metric based on the final estimator.

Parameters
- **X** – Features.
- **y** – Ground truth labels.
- **score_params** – Any additional parameters expected by the score function of the final estimator. These will be ignored for now.

Returns
Performance metric value.

Return type
score

`score_samples`(*X: Any = None*)

Scores for each sample in X. There type of scores is based on the last operator in the pipeline.

Parameters
X – Features.

Returns
Scores per sample.

Return type
result

`transform`(*X: Any*, *y: Any = None*) → Any

Deprecated since version 0.0.0: The *transform* method is deprecated on a trainable operator, because the learned coefficients could be accidentally overwritten by retraining. Call *transform* on the trained operator returned by *fit* instead.

`transform_X_y`(*X: Any*, *y: Any = None*) → Any

`transform_with_batches`(*X: Any*, *y: Optional[Any] = None*, *serialize: bool = True*)

[summary]

Parameters
- **X** (*Any*) – [description]
- **y** (*[type], optional*) – by default None
- **serialize** (*boolean*) – should data be serialized if needed

Returns
[description]

Return type
[type]

`lale.operators.clone_op`(*op: CloneOpType*, *name: Optional[str] = None*) → CloneOpType

Clone any operator.

`lale.operators.customize_schema`(*op: CustomizeOpType*, *schemas: Optional[Schema] = None*, *relevantToOptimizer: Optional[List[str]] = None*, *constraint: Optional[Union[Schema, Dict[str, Any], List[Union[Schema, Dict[str, Any]]]]] = None*, *tags: Optional[Dict] = None*, *forwards: Optional[Union[bool, List[str]]] = None*, *set_as_available: bool = False*, ***kwargs: Optional[Union[Schema, Dict[str, Any]]]*) → CustomizeOpType

Return a new operator with a customized schema

Parameters

- **op** (Operator) – The base operator to customize
- **schemas** (Schema) – A dictionary of json schemas for the operator. Override the entire schema and ignore other arguments
- **input** (Schema) – (or *input_**) override the input schema for method *. *input_** must be an existing method (already defined in the schema for lale operators, existing method for external operators)
- **output** (Schema) – (or *output_**) override the output schema for method *. *output_** must be an existing method (already defined in the schema for lale operators, existing method for external operators)
- **relevantToOptimizer** (*String list*) – update the set parameters that will be optimized.
- **constraint** (Schema) – Add a constraint in JSON schema format.
- **tags** (*Dict*) – Override the tags of the operator.
- **forwards** (*boolean or a list of strings*) – Which methods/properties to forward to the underlying impl. (False for none, True for all).
- **set_as_available** (*bool*) – Override the list of available operators so *get_available_operators* returns this customized operator.
- **kwargs** (Schema) – Override the schema of the hyperparameter. *param* must be an existing parameter (already defined in the schema for lale operators, __init__ parameter for external operators)

Returns

Copy of the operator with a customized schema

Return type

PlannedIndividualOp

`lale.operators.`**`get_available_estimators`**(*tags: Optional[AbstractSet[str]] = None*) → List[*PlannedOperator*]

`lale.operators.`**`get_available_operators`**(*tag: str*, *more_tags: Optional[AbstractSet[str]] = None*) → List[*PlannedOperator*]

`lale.operators.`**`get_available_transformers`**(*tags: Optional[AbstractSet[str]] = None*) → List[*PlannedOperator*]

`lale.operators.`**`get_lib_schemas`**(*impl_class*) → Optional[Dict[str, Any]]

`lale.operators.`**`get_op_from_lale_lib`**(*impl_class*, *wrapper_modules=None*) → Optional[*IndividualOp*]

`lale.operators.`**`make_choice`**(**orig_steps: Union[Operator, Any]*, *name: Optional[str] = None*) → *OperatorChoice*

`lale.operators.`**`make_operator`**(*impl*, *schemas=None*, *name: Optional[str] = None*, *set_as_available: bool = True*) → *PlannedIndividualOp*

`lale.operators.`**`make_pipeline`**(**orig_steps: TrainedOperator*) → *TrainedPipeline*

`lale.operators.`**`make_pipeline`**(**orig_steps: TrainableOperator*) → *TrainablePipeline*

`lale.operators.`**`make_pipeline`**(**orig_steps: Union[Operator, Any]*) → *PlannedPipeline*

`lale.operators.`**`make_pipeline_graph`**(*steps: List[TrainedOperator]*, *edges: List[Tuple[Operator, Operator]]*, *ordered: bool = False*) → *TrainedPipeline*

`lale.operators.`**`make_pipeline_graph`**(*steps: List[TrainableOperator]*, *edges: List[Tuple[Operator, Operator]]*, *ordered: bool = False*) → *TrainablePipeline* lale.operators.**make_pipeline_graph**(*steps: List[Operator], edges: List[Tuple[Operator, Operator]], ordered: bool = False*) → *PlannedPipeline*

Based on the state of the steps, it is important to decide an appropriate type for a new Pipeline. This method will decide the type, create a new Pipeline of that type and return it. #TODO: If multiple independently trained components are composed together in a pipeline, should it be of type TrainedPipeline? Currently, it will be TrainablePipeline, i.e. it will be forced to train it again.

lale.operators.**make_pretrained_operator**(*impl, schemas=None, name: Optional[str] = None*) → *TrainedIndividualOp*

lale.operators.**make_union**(**orig_steps: TrainedOperator*) → *TrainedPipeline*

lale.operators.**make_union**(**orig_steps: TrainableOperator*) → *TrainablePipeline*

lale.operators.**make_union**(**orig_steps: Union[Operator, Any]*) → *PlannedPipeline*

lale.operators.**make_union_no_concat**(**orig_steps: TrainedOperator*) → *TrainedPipeline*

lale.operators.**make_union_no_concat**(**orig_steps: TrainableOperator*) → *TrainablePipeline*

lale.operators.**make_union_no_concat**(**orig_steps: Union[Operator, Any]*) → *PlannedPipeline*

lale.operators.**with_structured_params**(*try_mutate: bool, k, params: Dict[str, Any], hyper_parent*) → None

lale.operators.**wrap_operator**(*impl*) → *Operator*

lale.pretty_print module

lale.pretty_print.**hyperparams_to_string**(*hps: Dict[str, Any], steps: Optional[Dict[str, str]] = None, gen: Optional[_CodeGenState] = None*) → str

lale.pretty_print.**ipython_display**(*arg: Union[Dict[str, Any], Operator], *, show_imports: bool = True, combinators: bool = True, assign_nested: bool = True*)

lale.pretty_print.**json_to_string**(*jsn: Dict[str, Any]*) → str

lale.pretty_print.**to_string**(*arg: Union[Dict[str, Any], Operator], *, show_imports: bool = True, combinators: bool = True, assign_nested: bool = True, customize_schema: bool = False, astype: str = 'lale', call_depth: int = 1*) → str

lale.schema2enums module

class lale.schema2enums.**DiscoveredEnums**(*enums: Optional[Set[Any]] = None, children: Optional[Dict[str, DiscoveredEnums]] = None*)

Bases: `object`

lale.schema2enums.**accumulateDiscoveredEnumsToPythonEnums**(*de: Optional[DiscoveredEnums], path: List[str], acc: Dict[str, Enum]*) → None

lale.schema2enums.**addDictAsFields**(*obj: Any, d: Dict[str, Any], force=False*) → None

lale.schema2enums.**addSchemaEnumsAsFields**(*obj: Any, schema: Dict[str, Any], force=False*) → None

lale.schema2enums.**discoveredEnumsToPythonEnums**(*de: Optional[DiscoveredEnums]*) → Dict[str, Enum] lale.schema2enums.**schemaToDiscoveredEnums**(*schema: Dict[str, Any]*) → Optional[*DiscoveredEnums*]

Given a schema, returns a positive enumeration set. This is very conservative, and even includes negated enum constants (since the assumption is that they may, in some contexts, be valid)

lale.schema2enums.**schemaToPythonEnums**(*schema: Dict[str, Any]*) → Dict[str, Enum]

lale.schema_ranges module

class lale.schema_ranges.**SchemaRange**(*minimum=None*, *maximum=None*, *exclusive_minimum=False*, *exclusive_maximum=False*, *is_integer: bool = False*, *distribution: Optional[str] = None*)

Bases: `object`

diff(*other: SchemaRange*) → Optional[bool]

Returns None if the resulting region is impossible. Returns True if the other constraint was completely subtracted from self. If it could not be, then it returns False (and the caller should probably keep the other constraint as a negated constraint)

classmethod fromSchema(*schema: Any*) → *SchemaRange*

classmethod fromSchemaForOptimizer(*schema: Any*) → *SchemaRange*

is_empty() → bool

Determines if the range is empty (contains nothing)

classmethod is_empty2(*lower: SchemaRange*, *upper: SchemaRange*) → bool

Determines if the range given by taking lower bounds from lower and upper bound from upper is empty (contains nothing) is_integer is assumed to be their disjunction

classmethod point(*pt: Union[int, float]*)

remove_point(*other: Union[int, float]*) → Optional[bool]

Returns None if the resulting region is impossible. Returns True if the other constraint was completely subtracted from self. If it could not be, then it returns False (and the caller should probably keep the other constraint as a negated constraint)

classmethod to_schema_with_optimizer(*actual_range: SchemaRange*, *optimizer_range: SchemaRange*) → Dict[str, Any]

lale.schema_simplifier module

lale.schema_simplifier.**enumValues**(*es: set_with_str_for_keys[Any]*, *s: Dict[str, Any]*) → *set_with_str_for_keys*[Any]

Given an enumeration set and a schema, return all the consistent values of the enumeration.

lale.schema_simplifier.**filterForOptimizer**(*schema: Dict[str, Any]*) → Optional[Dict[str, Any]]

lale.schema_simplifier.**findRelevantFields**(*schema: Dict[str, Any]*) → Optional[Set[str]]

Either returns the relevant fields for the schema, or None if there was none specified

lale.schema_simplifier.**hasAllOperatorSchemas**(*schemas: List[Dict[str, Any]]*) → bool

lale.schema_simplifier.**hasAnyOperatorSchemas**(*schemas: List[Dict[str, Any]]*) → bool `lale.schema_simplifier.impossible()` → Dict[str, Any]

`lale.schema_simplifier.liftAllOf`(*schemas: List[Dict[str, Any]]*) → Iterable[Dict[str, Any]]

Given a list of schemas, if any of them are allOf schemas, lift them out to the top level

`lale.schema_simplifier.liftAnyOf`(*schemas: List[Dict[str, Any]]*) → Iterable[Dict[str, Any]]

Given a list of schemas, if any of them are anyOf schemas, lift them out to the top level

`lale.schema_simplifier.narrowSimplifyAndFilter`(*schema: Dict[str, Any], floatAny: bool*) → Optional[Dict[str, Any]]

`lale.schema_simplifier.narrowToGivenRelevantFields`(*schema: Dict[str, Any], relevantFields: Set[str]*) → Dict[str, Any]

`lale.schema_simplifier.narrowToRelevantConstraints`(*schema: Dict[str, Any]*) → Dict[str, Any]

`lale.schema_simplifier.narrowToRelevantFields`(*schema: Dict[str, Any]*) → Dict[str, Any]

class `lale.schema_simplifier.set_with_str_for_keys`(*elems: Union[Dict[str, VV], Iterable[VV]]*)

Bases: `Generic[VV]`

This mimicks a set, but uses the string representation of the elements for comparison tests. It can be used for unhashable elements, as long as the str function is injective

difference(**others*)

intersection(**others: set_with_str_for_keys[VV]*)

union(**others*)

`lale.schema_simplifier.simplify`(*schema: Dict[str, Any], floatAny: bool*) → Dict[str, Any]

Tries to simplify a schema into an equivalent but more compact/simpler one. If floatAny if true, then the only anyOf in the return value will be at the top level. Using this option may cause a combinatorial blowup in the size of the schema

`lale.schema_simplifier.simplifyAll`(*schemas: List[Dict[str, Any]], floatAny: bool*) → Dict[str, Any]

`lale.schema_simplifier.simplifyAny`(*schema: List[Dict[str, Any]], floatAny: bool*) → Dict[str, Any]

`lale.schema_simplifier.simplifyNot`(*schema: Dict[str, Any], floatAny: bool*) → Dict[str, Any]

`lale.schema_simplifier.simplifyNot_`(*schema: Dict[str, Any], floatAny: bool, alreadySimplified: bool = False*) → Dict[str, Any]

alreadySimplified=true implies that schema has already been simplified

`lale.schema_simplifier.toAllOfList`(*schema: Dict[str, Any]*) → List[Dict[str, Any]]

`lale.schema_simplifier.toAnyOfList`(*schema: Dict[str, Any]*) → List[Dict[str, Any]]

lale.schema_utils module

lale.schema_utils.**atomize_schema_enumerations**(*schema: Union[None, Dict[str, Any], List[Dict[str, Any]]]*) → None

Given a schema, converts structured enumeration values (records, arrays) into schemas where the structured part is specified as a schema, with the primitive as the enum.

lale.schema_utils.**check_operators_schema**(*schema: Optional[Union[List[Dict[str, Any]], Dict[str, Any]]]*, *warnings: List[str]*) → None

Given a schema, collect warnings if there are any enumeration with all Operator values that are not marked as *'laleType':'operator'*. This should be called after simplification.

lale.schema_utils.**forOptimizer**(*schema: Dict[str, Any]*) → Optional[Dict[str, Any]]

lale.schema_utils.**getExclusiveMaximum**(*obj*)

lale.schema_utils.**getExclusiveMinimum**(*obj*)

lale.schema_utils.**getForOptimizer**(*obj*, *prop: str*)

lale.schema_utils.**getMaximum**(*obj*)

lale.schema_utils.**getMinimum**(*obj*)

lale.schema_utils.**has_operator**(*schema: Dict[str, Any]*) → bool

lale.schema_utils.**isForOptimizer**(*s: Dict[str, Any]*) → bool

lale.schema_utils.**is_false_schema**(*s: Dict[str, Any]*) → bool

lale.schema_utils.**is_lale_any_schema**(*s: Dict[str, Any]*) → bool

lale.schema_utils.**is_true_schema**(*s: Dict[str, Any]*) → bool

lale.schema_utils.**makeAllOf**(*schemas: List[Dict[str, Any]]*) → Dict[str, Any]

lale.schema_utils.**makeAnyOf**(*schemas: List[Dict[str, Any]]*) → Dict[str, Any]

lale.schema_utils.**makeOneOf**(*schemas: List[Dict[str, Any]]*) → Dict[str, Any]

lale.schema_utils.**makeSingleton_**(*k: str*, *schemas: List[Dict[str, Any]]*) → Dict[str, Any]

lale.schemas module

class lale.schemas.**AllOf**(*types: ~typing.Optional[~typing.List[~lale.schemas.Schema]] = None*, *desc: ~typing.Union[~lale.schemas.Undefined, str] = <lale.schemas.Undefined object>*, *default: ~typing.Union[~lale.schemas.Undefined, ~typing.Any] = <lale.schemas.Undefined object>*)

Bases: `Schema`

schema: Dict[str, Any]

class lale.schemas.**AnyOf**(*types: ~typing.Optional[~typing.List[~lale.schemas.Schema]] = None*, *desc: ~typing.Union[~lale.schemas.Undefined, str] = <lale.schemas.Undefined object>*, *default: ~typing.Union[~lale.schemas.Undefined, ~typing.Any] = <lale.schemas.Undefined object>*, *forOptimizer: bool = True*)

Bases: `Schema`

schema: Dict[str, Any]

class lale.schemas.**Array**(*items: ~lale.schemas.Schema, desc: ~typing.Union[~lale.schemas.Undefined, str] = <lale.schemas.Undefined object>, default: ~typing.Union[~lale.schemas.Undefined, ~typing.List[~typing.Any]] = <lale.schemas.Undefined object>, forOptimizer: bool = True, minItems: ~typing.Union[~lale.schemas.Undefined, int] = <lale.schemas.Undefined object>, minItemsForOptimizer: ~typing.Union[~lale.schemas.Undefined, int] = <lale.schemas.Undefined object>, maxItems: ~typing.Union[~lale.schemas.Undefined, int] = <lale.schemas.Undefined object>, maxItemsForOptimizer: ~typing.Union[~lale.schemas.Undefined, int] = <lale.schemas.Undefined object>, laleType: ~typing.Union[~lale.schemas.Undefined, str] = <lale.schemas.Undefined object>*)

Bases: *Schema*

schema: Dict[str, Any]

class lale.schemas.**Bool**(*desc: ~typing.Union[~lale.schemas.Undefined, str] = <lale.schemas.Undefined object>, default: ~typing.Union[~lale.schemas.Undefined, bool] = <lale.schemas.Undefined object>, forOptimizer: bool = True*)

Bases: *Schema*

schema: Dict[str, Any]

class lale.schemas.**Enum**(*values: ~typing.Optional[~typing.List[~typing.Any]] = None, desc: ~typing.Union[~lale.schemas.Undefined, str] = <lale.schemas.Undefined object>, default: ~typing.Union[~lale.schemas.Undefined, ~typing.Any] = <lale.schemas.Undefined object>, forOptimizer: bool = True*)

Bases: *Schema*

schema: Dict[str, Any]

class lale.schemas.**Float**(*desc: ~typing.Union[~lale.schemas.Undefined, str] = <lale.schemas.Undefined object>, default: ~typing.Union[~lale.schemas.Undefined, float] = <lale.schemas.Undefined object>, forOptimizer: bool = True, minimum: ~typing.Union[~lale.schemas.Undefined, float] = <lale.schemas.Undefined object>, exclusiveMinimum: ~typing.Union[~lale.schemas.Undefined, bool] = <lale.schemas.Undefined object>, minimumForOptimizer: ~typing.Union[~lale.schemas.Undefined, float] = <lale.schemas.Undefined object>, exclusiveMinimumForOptimizer: ~typing.Union[~lale.schemas.Undefined, bool] = <lale.schemas.Undefined object>, maximum: ~typing.Union[~lale.schemas.Undefined, float] = <lale.schemas.Undefined object>, exclusiveMaximum: ~typing.Union[~lale.schemas.Undefined, bool] = <lale.schemas.Undefined object>, maximumForOptimizer: ~typing.Union[~lale.schemas.Undefined, float] = <lale.schemas.Undefined object>, exclusiveMaximumForOptimizer: ~typing.Union[~lale.schemas.Undefined, bool] = <lale.schemas.Undefined object>, distribution: ~typing.Union[~lale.schemas.Undefined, str] = <lale.schemas.Undefined object>*)

Bases: *Schema*

schema: Dict[str, Any]

class lale.schemas.**Int**(*desc: ~typing.Union[~lale.schemas.Undefined, str] = <lale.schemas.Undefined object>, default: ~typing.Union[~lale.schemas.Undefined, int] = <lale.schemas.Undefined object>, forOptimizer: bool = True, minimum: ~typing.Union[~lale.schemas.Undefined, int] = <lale.schemas.Undefined object>, exclusiveMinimum: ~typing.Union[~lale.schemas.Undefined, bool] = <lale.schemas.Undefined object>, minimumForOptimizer: ~typing.Union[~lale.schemas.Undefined, int] = <lale.schemas.Undefined object>, exclusiveMinimumForOptimizer: ~typing.Union[~lale.schemas.Undefined, bool] = <lale.schemas.Undefined object>, maximum: ~typing.Union[~lale.schemas.Undefined, int] = <lale.schemas.Undefined object>, exclusiveMaximum: ~typing.Union[~lale.schemas.Undefined, bool] = <lale.schemas.Undefined object>, maximumForOptimizer: ~typing.Union[~lale.schemas.Undefined, int] = <lale.schemas.Undefined object>, exclusiveMaximumForOptimizer: ~typing.Union[~lale.schemas.Undefined, bool] = <lale.schemas.Undefined object>, distribution: ~typing.Union[~lale.schemas.Undefined, str] = <lale.schemas.Undefined object>, laleMaximum: ~typing.Union[~lale.schemas.Undefined, str] = <lale.schemas.Undefined object>*)

Bases: *Schema*

schema: Dict[str, Any]

class lale.schemas.**JSON**(*body: Dict[str, Any]*)

Bases: *Schema*

schema: Dict[str, Any]

class lale.schemas.**Not**(*body:* Schema)

Bases: *Schema*

schema: Dict[str, Any]

class lale.schemas.**Null**(*desc: ~typing.Union[~lale.schemas.Undefined, str] = <lale.schemas.Undefined object>, forOptimizer: bool = True*)

Bases: *Schema*

schema: Dict[str, Any]

class lale.schemas.**Object**(*default: ~typing.Union[~lale.schemas.Undefined, ~typing.Any] = <lale.schemas.Undefined object>, desc: ~typing.Union[~lale.schemas.Undefined, str] = <lale.schemas.Undefined object>, forOptimizer: bool = True, required: ~typing.Union[~lale.schemas.Undefined, ~typing.List[str]] = <lale.schemas.Undefined object>, additionalProperties: ~typing.Union[~lale.schemas.Undefined, bool] = <lale.schemas.Undefined object>, **kwargs: ~lale.schemas.Schema*)

Bases: *Schema*

schema: Dict[str, Any]

class lale.schemas.**Schema**(*desc: ~typing.Union[~lale.schemas.Undefined, str] = <lale.schemas.Undefined object>, default: ~typing.Union[~lale.schemas.Undefined, ~typing.Any] = <lale.schemas.Undefined object>, forOptimizer: bool = True*)

Bases: object

schema: Dict[str, Any]

set(*prop: str, value: Union[*Undefined*, Any]*)

class lale.schemas.**String**(*desc: ~typing.Union[~lale.schemas.Undefined, str] = <lale.schemas.Undefined object>, default: ~typing.Union[~lale.schemas.Undefined, str] = <lale.schemas.Undefined object>, forOptimizer: bool = False*)

Bases: `Schema`

schema: `Dict[str, Any]`

class lale.schemas.**Undefined**

Bases: `object`

lale.settings module

lale.settings.**set_disable_data_schema_validation**(*flag: bool*)

Lale can validate the input and output data used for fit, predict, predict_proba etc. against the data schemas defined for an operator. This method allows users to control whether the data schema validation should be turned on or not.

Parameters

flag (*bool*) – A value of True will disable the data schema validation, and a value of False will enable it. It is True by default.

lale.settings.**set_disable_hyperparams_schema_validation**(*flag: bool*)

Lale can validate the hyperparameter values passed while creating an operator against the json schema defined for hyperparameters of an operator. This method allows users to control whether such validation should be turned on or not.

Parameters

flag (*bool*) – A value of True will disable the hyperparameter schema validation, and a value of False will enable it. It is False by default.

lale.sklearn_compat module

lale.sklearn_compat.**make_sklearn_compat**(*op*)

This is a deprecated method for backward compatibility and will be removed soon

lale.sklearn_compat.**sklearn_compat_clone**(*impl: Any*) → Any

This is a deprecated method for backward compatibility and will be removed soon. call lale.operators.clone (or scikit-learn clone) instead

lale.type_checking module

Lale uses JSON Schema to check machine-learning pipelines for correct types.

In general, there are two kinds of checks. The first is an instance check ($v: s$), which checks whether a JSON value v is valid for a schema s. The second is a subschema check ($s <: t$), which checks whether one schema s is a subchema of another schema t.

Besides regular JSON values, Lale also supports certain JSON-like values. For example, a `np.ndarray` of numbers is treated like a JSON array of arrays of numbers. Furthermore, Lale supports an 'Any' type for which all instance and subschema checks on the left as well as the right side succeed. This is specified using `{'laleType': 'Any'}`.

exception lale.type_checking.**SubschemaError**(*sub, sup, sub_name='sub', sup_name='super'*)

Bases: `Exception`

Raised when a subschema check (sub <: sup) failed.

lale.type_checking.**always_validate_schema**(*value: Any*, *schema: Dict[str, Any]*, *subsample_array: bool = True*)

Validate that the value is an instance of the schema.

Parameters

- **value** (JSON *(int, float, str, list, dict) or JSON-like (tuple, np.ndarray, pd.DataFrame ...).*) – Left-hand side of instance check.
- **schema** (*JSON schema*) – Right-hand side of instance check.
- **subsample_array** (*bool*) – Speed up checking by doing only partial conversion to JSON.

Raises

jsonschema.ValidationError – The value was invalid for the schema.

lale.type_checking.**get_default_schema**(*impl*)

Creates combined schemas for a bare operator implementation class.

Used when there were no explicit combined schemas provided when the operator was created. The default schema provides defaults by inspecting the signature of the `__init__` method, and uses 'Any' types for the inputs and outputs of other methods.

Returns

Combined schema with properties for hyperparams and all applicable method inputs and outputs.

Return type

JSON Schema

lale.type_checking.**get_hyperparam_defaults**(*impl*)

lale.type_checking.**get_hyperparam_names**(*op: IndividualOp*) → List[str]

Names of the arguments to the constructor of the impl.

Parameters

op (`lale.operators.IndividualOp`) – Operator whose hyperparameters to get.

Returns

List of hyperparameter names.

Return type

List[str]

lale.type_checking.**has_data_constraints**(*hyperparam_schema: Dict[str, Any]*) → bool

lale.type_checking.**is_schema**(*value*) → bool

lale.type_checking.**is_subschema**(*sub_schema: Dict[str, Any]*, *super_schema: Dict[str, Any]*) → bool

Is sub_schema a subschema of super_schema?

Parameters

- **sub_schema** (*JSON schema*) – Left-hand side of subschema check.
- **super_schema** (*JSON schema*) – Right-hand side of subschema check.

Returns

True if *sub_schema <: super_schema*, False otherwise.

Return type

bool

Raises

jsonschema.ValueError – An error occured while checking the subschema relation

lale.type_checking.**join_schemas**(**schemas: Dict[str, Any]*) → Dict[str, Any]

Compute the lattice join (union type, disjunction) of the arguments.

Parameters

***schemas** (*list of JSON schemas*) – Schemas to be joined.

Returns
The joined schema.

Return type
JSON schema

`lale.type_checking.`**`replace_data_constraints`**(*hyperparam_schema: Dict[str, Any]*, *data_schema: Dict[str, Any]*) → Dict[str, Any]

`lale.type_checking.`**`validate_is_schema`**(*value: Dict[str, Any]*)

`lale.type_checking.`**`validate_method`**(*op:* IndividualOp, *schema_name: str*)

Check whether the operator has the given method schema.

Parameters
- **op** (`lale.operators.IndividualOp`) – Operator whose methods to check.
- **schema_name** (*'input_fit' or 'input_predict' or 'input_predict_proba' or 'input_transform' 'output_predict' or 'output_predict_proba' or 'output_transform'*) – Name of schema to check.

Raises
`AssertionError` – The operator does not have the given schema.

`lale.type_checking.`**`validate_schema`**(*lhs: Any*, *super_schema: Dict[str, Any]*)

Validate that lhs is an instance of or a subschema of super_schema.

Parameters
- **lhs** (*value*) – Left-hand side of instance or subschema check.
- **super_schema** (*JSON schema*) – Right-hand side of instance or subschema check.

Raises
- **jsonschema.ValidationError** – The lhs was an invalid value for super_schema.
- ***SubschemaError*** – The lhs had a schema that was not a subschema of super_schema.

`lale.type_checking.`**`validate_schema_directly`**(*value: Any*, *schema: Dict[str, Any]*, *subsample_array: bool = True*)

Validate that the value is an instance of the schema.

Parameters
- **value** (JSON *(int, float, str, list, dict) or JSON-like (tuple, np.ndarray, pd.DataFrame ...).*) – Left-hand side of instance check.
- **schema** (*JSON schema*) – Right-hand side of instance check.
- **subsample_array** (*bool*) – Speed up checking by doing only partial conversion to JSON.

Raises
jsonschema.ValidationError – The value was invalid for the schema.

lale.visualize module

`lale.visualize.json_to_graphviz(jsn, ipython_display, dot_graph_attr)`

Module contents

PYTHON MODULE INDEX

l

INDEX

A

B

C

D

E

F

G

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X