LightAutoML

Sber AI Lab

Nov 12, 2021

PYTHON API

LightAutoML is open-source Python library aimed at automated machine learning. It is designed to be lightweight and efficient for various tasks with tabular, text data. LightAutoML provides easy-to-use pipeline creation, that enables:

- Automatic hyperparameter tuning, data processing.
- Automatic typing, feature selection.
- Automatic time utilization.
- Automatic report creation.
- Graphical profiling system.
- Easy-to-use modular scheme to create your own pipelines.

CHAPTER

ONE

LIGHTAUTOML.AUTOML

The main module, which includes the AutoML class, blenders and ready-made presets.

AutoML	Class for compile full pipeline of AutoML task.

1.1 AutoML

class lightautoml.automl.base.**AutoML**(*reader*, *levels*, *timer=None*, *blender=None*, *skip_conn=False*, *return_all_predictions=False*)

Bases: object

Class for compile full pipeline of AutoML task.

AutoML steps:

- Read, analyze data and get inner *LAMLDataset* from input dataset: performed by reader.
- Create validation scheme.
- Compute passed ml pipelines from levels. Each element of levels is list of `MLPipelines` prediction from current level are passed to next level pipelines as features.
- Time monitoring - check if we have enough time to calc new pipeline.
- Blend last level models and prune useless pipelines to speedup inference: performed by blender.
- Returns prediction on validation data. If crossvalidation scheme is used, out-of-fold prediction will returned. If validation data is passed it will return prediction on validation dataset. In case of cv scheme when some point of train data never was used as validation (ex. timeout exceeded or custom cv iterator like *TimeSeriesIterator* was used) NaN for this point will be returned.

Example

Common usecase - create custom pipelines or presets.

```
>>> reader = SomeReader()
>>> pipe = MLPipeline([SomeAlgo()])
>>> levels = [[pipe]]
>>> automl = AutoML(reader, levels, )
>>> automl.fit_predict(data, roles={'target': 'TARGET'})
```

__init__(*reader*, *levels*, *timer=None*, *blender=None*, *skip_conn=False*, *return_all_predictions=False*)

Parameters

- **reader** (*Reader*) – Instance of Reader class object that creates *LAMLDataset* from input data.
- **levels** (Sequence[Sequence[*MLPipeline*]]) – List of list of MLPipelines.
- **timer** (Optional[*PipelineTimer*]) – Timer instance of *PipelineTimer*. Default - unlimited timer.
- **blender** (Optional[*Blender*]) – Instance of Blender. Default - *BestModelSelector*.
- **skip_conn** (bool) – True if we should pass first level input features to next levels.

Note: There are several verbosity levels:

- *0*: No messages.
- *1*: Warnings.
- *2*: Info.
- *3*: Debug.

---

fit_predict(*train_data*, *roles*, *train_features=None*, *cv_iter=None*, *valid_data=None*, *valid_features=None*, *verbose=0*)

Fit on input data and make prediction on validation part.

Parameters

- **train_data** (Any) – Dataset to train.
- **roles** (dict) – Roles dict.
- **train_features** (Optional[Sequence[str]]) – Optional features names, if cannot be inferred from train_data.
- **cv_iter** (Optional[Iterable]) – Custom cv iterator. For example, *TimeSeriesIterator*.
- **valid_data** (Optional[Any]) – Optional validation dataset.
- **valid_features** (Optional[Sequence[str]]) – Optional validation dataset features if can't be inferred from *valid_data*.

Return type *LAMLDataset*

Returns Predicted values.

predict(*data*, *features_names=None*, *return_all_predictions=None*)

Predict with automl on new dataset.

Parameters

- **data** (Any) – Dataset to perform inference.
- **features_names** (Optional[Sequence[str]]) – Optional features names, if cannot be inferred from *train_data*.
- **return_all_predictions** (Optional[bool]) – if True, returns all model predictions from last level

Return type *LAMLDataset*

Returns Dataset with predictions.

collect_used_feats()
Get feats that automl uses on inference.

Return type List[str]

Returns Features names list.

collect_model_stats()
Collect info about models in automl.

Return type Dict[str, int]

Returns Dict with models and its runtime numbers.

1.2 Presets

Presets for end-to-end model training for special tasks.

base.AutoMLPreset	Basic class for automl preset.
whitebox_presets.WhiteBoxPreset	Preset for AutoWoE - logistic regression over binned features (scorecard).

1.2.1 AutoMLPreset

class lightautoml.automl.presets.base.**AutoMLPreset**(*task*, *timeout=3600*, *memory_limit=16*, *cpu_limit=4*, *gpu_ids='all'*, *timing_params=None*, *config_path=None*, ***kwargs*)

Bases: *lightautoml.automl.base.AutoML*

Basic class for automl preset.

It's almost like AutoML, but with delayed initialization. Initialization starts on fit, some params are inferred from data. Preset should be defined via `.create_automl` method. Params should be set via yaml config. Most usefull case - end-to-end model development.

Example

```
>>> automl = SomePreset(Task('binary'), timeout=3600)
>>> automl.fit_predict(data, roles={'target': 'TARGET'})
```

__init__(*task*, *timeout=3600*, *memory_limit=16*, *cpu_limit=4*, *gpu_ids='all'*, *timing_params=None*, *config_path=None*, ***kwargs*)
Commonly _params kwargs (ex. timing_params) set via config file (config_path argument). If you need to change just few params, it's possible to pass it as dict of dicts, like json. To get available params please look on default config template. Also you can find there param description. To generate config template call `SomePreset.get_config('config_path.yml')`.

Parameters

- **task** (*Task*) – Task to solve.
- **timeout** (int) – Timeout in seconds.
- **memory_limit** (int) – Memory limit that are passed to each automl.

- **cpu_limit** (int) – CPU limit that that are passed to each automl.
- **gpu_ids** (Optional[str]) – GPU IDs that are passed to each automl.
- **verbose** – Controls the verbosity: the higher, the more messages. <1 : messages are not displayed; >=1 : the computation process for layers is displayed; >=2 : the information about folds processing is also displayed; >=3 : the hyperparameters optimization process is also displayed; >=4 : the training process for every algorithm is displayed;
- **timing_params** (Optional[dict]) – Timing param dict.
- **config_path** (Optional[str]) – Path to config file.
- ****kwargs** – Not used.

classmethod get_config(*path=None*)

Create new config template.

Parameters **path** (Optional[str]) – Path to config.

Return type Optional[dict]

Returns Config.

create_automl(***fit_args*)

Abstract method - how to build automl.

Here you should create all automl components, like readers, levels, timers, blenders. Method `._initialize` should be called in the end to create automl.

Parameters ****fit_args** – params that are passed to `.fit_predict` method.

fit_predict(*train_data*, *roles*, *train_features=None*, *cv_iter=None*, *valid_data=None*, *valid_features=None*, *verbose=0*)

Fit on input data and make prediction on validation part.

Parameters

- **train_data** (Any) – Dataset to train.
- **roles** (dict) – Roles dict.
- **train_features** (Optional[Sequence[str]]) – Features names, if can't be inferred from *train_data*.
- **cv_iter** (Optional[Iterable]) – Custom cv-iterator. For example, *TimeSeriesIterator*.
- **valid_data** (Optional[Any]) – Optional validation dataset.
- **valid_features** (Optional[Sequence[str]]) – Optional validation dataset features if can't be inferred from *valid_data*.
- **verbose** (int) – Verbosity level that are passed to each automl.

Return type *LAMLDataset*

Returns Dataset with predictions. Call `.data` to get predictions array.

static set_verbosity_level(*verbose*)

Verbosity level setter.

Parameters **verbose** (int) – Controls the verbosity: the higher, the more messages. <1 : messages are not displayed; >=1 : the computation process for layers is displayed; >=2 : the information about folds processing is also displayed; >=3 : the hyperparameters optimization process is also displayed; >=4 : the training process for every algorithm is displayed;

1.2.2 WhiteBoxPreset

class lightautoml.automl.presets.whitebox_presets.**WhiteBoxPreset**(*task*, *timeout=3600*, *memory_limit=16*, *cpu_limit=4*, *gpu_ids=None*, *timing_params=None*, *config_path=None*, *general_params=None*, *reader_params=None*, *read_csv_params=None*, *whitebox_params=None*)

Bases: *lightautoml.automl.presets.base.AutoMLPreset*

Preset for AutoWoE - logistic regression over binned features (scorecard).

Supported data roles - numbers, dates, categories.

Limitations:

- Simple time management.
- No memory management.
- Working only with `pandas.DataFrame`.
- No batch inference.
- No text support.
- No parallel execution.
- No batch inference.
- No GPU usage.
- No cross-validation scheme. Supports only holdout validation (cv is created inside AutoWoE, but no oof pred returned).

Common usecase - fit lightweight interpretable model for binary classification task.

property whitebox

Get wrapped AutoWoE object.

Returns Model.

__init__(*task*, *timeout=3600*, *memory_limit=16*, *cpu_limit=4*, *gpu_ids=None*, *timing_params=None*, *config_path=None*, *general_params=None*, *reader_params=None*, *read_csv_params=None*, *whitebox_params=None*)

Commonly _params kwargs (ex. timing_params) set via config file (config_path argument). If you need to change just few params, it's possible to pass it as dict of dicts, like json. To get available params please look on default config template. Also you can find there param description To generate config template call `WhiteBoxPreset.get_config('config_path.yml')`.

Parameters

- **task** (*Task*) – Task to solve.
- **timeout** (`int`) – Timeout in seconds.
- **memory_limit** (`int`) – Memory limit that are passed to each automl.
- **cpu_limit** (`int`) – CPU limit that that are passed to each automl.
- **gpu_ids** (`Optional[str]`) – GPU IDs that are passed to each automl.

- **timing_params** (Optional[dict]) – Timing param dict.
- **config_path** (Optional[str]) – Path to config file.
- **general_params** (Optional[dict]) – General param dict.
- **reader_params** (Optional[dict]) – Reader param dict.
- **read_csv_params** (Optional[dict]) – Params to pass pandas.read_csv (case of train/predict from file).
- **whitebox_params** (Optional[dict]) – Params of WhiteBox algo (look at config file).

create_automl(**args*, ***kwargs*)

Create basic *WhiteBoxPreset* instance from data.

Parameters

- ***args** – Not used.
- ****kwargs** – everything passed to .fit_predict.

fit_predict(*train_data*, *roles*, *train_features=None*, *cv_iter=None*, *valid_data=None*, *valid_features=None*, *verbose=0*, ***fit_params*)

Fit and get prediction on validation dataset.

Almost same as *lightautoml.automl.base.AutoML.fit_predict*.

Additional features - working with different data formats. Supported now:

- Path to .csv, .parquet, .feather files.
- ndarray, or dict of ndarray. For example, {'data': X...}. In this case, roles are optional, but *train_features* and *valid_features* required.
- pandas.DataFrame.

Parameters

- **train_data** (Any) – Dataset to train.
- **roles** (dict) – Roles dict.
- **train_features** (Optional[Sequence[str]]) – Optional features names, if can't be inferred from *train_data*.
- **cv_iter** (Optional[Iterable]) – Custom cv-iterator. For example, *TimeSeriesIterator*.
- **valid_data** (Optional[Any]) – Optional validation dataset.
- **valid_features** (Optional[Sequence[str]]) – Optional validation dataset features if cannot be inferred from *valid_data*.
- **verbose** (int) – Controls the verbosity: the higher, the more messages. <1 : messages are not displayed; >=1 : the computation process for layers is displayed; >=2 : the information about folds processing is also displayed; >=3 : the hyperparameters optimization process is also displayed; >=4 : the training process for every algorithm is displayed;

Return type *NumpyDataset*

Returns Dataset with predictions. Call .data to get predictions array.

predict(*data*, *features_names=None*, *report=False*)

Almost same as AutoML .predict with additional features.

Additional features - generate extended WhiteBox `report=True` passed to args.

Parameters

- **data** (Any) – Dataset to perform inference.
- **features_names** (Optional[Sequence[str]]) – Optional features names, if can't be inferred from *train_data*.
- **report** (bool) – Flag if we need inner WhiteBox report update (`True` is slow). Only if `general_params['report'] = True`.

Return type *NumpyDataset*

Returns Dataset with predictions.

1.3 Blenders

Blender	Basic class for blending.
BestModelSelector	Select best single model from level.
MeanBlender	Simple average level predictions.
WeightedBlender	Weighted Blender based on coord descent, optimize task metric directly.

1.3.1 Blender

class lightautoml.automl.blend.**Blender**

Bases: object

Basic class for blending.

Blender learns how to make blend on sequence of prediction datasets and prune pipes, that are not used in final blend.

fit_predict(*predictions*, *pipes*)

Wraps custom `._fit_predict` methods of blenders.

Method wraps individual `._fit_predict` method of blenders. If input is single model - take it, else `._fit_predict` Note - some pipelines may have more than 1 model. So corresponding prediction dataset have multiple prediction cols.

Parameters

- **predictions** (Sequence[*LAMLDataset*]) – Sequence of datasets with predictions.
- **pipes** (Sequence[*MLPipeline*]) – Sequence of pipelines.

Return type Tuple[*LAMLDataset*, Sequence[*MLPipeline*]]

Returns Single prediction dataset and sequence of pruned pipelines.

predict(*predictions*)

Wraps custom `._fit_predict` methods of blenders.

Parameters **predictions** (Sequence[*LAMLDataset*]) – Sequence of predictions from pruned datasets.

Return type *LAMLDataset*

Returns Dataset with predictions.

split_models(*predictions*)
Split predictions by single model prediction datasets.

Parameters **predictions** (Sequence[*LAMLDataset*]) – Sequence of datasets with predictions.

Return type Tuple[Sequence[*LAMLDataset*], List[int], List[int]]

Returns Split predictions, model indices, pipe indices.

score(*dataset*)
Score metric for blender.

Parameters **dataset** (*LAMLDataset*) – Blended predictions dataset.

Return type float

Returns Metric value.

1.3.2 BestModelSelector

class lightautoml.automl.blend.**BestModelSelector**
Bases: *lightautoml.automl.blend.Blender*

Select best single model from level.

Drops pipes that are not used in calc best model. Works in general case (even on some custom things) and most efficient on inference. Perform worse than other on tables, specially if some of models was terminated by timer.

1.3.3 MeanBlender

class lightautoml.automl.blend.**MeanBlender**
Bases: *lightautoml.automl.blend.Blender*

Simple average level predictions.

Works only with TabularDatasets. Doesn't require target to fit. No pruning.

1.3.4 WeightedBlender

class lightautoml.automl.blend.**WeightedBlender**(*max_iters=5*, *max_inner_iters=7*, *max_nonzero_coef=0.05*)
Bases: *lightautoml.automl.blend.Blender*

Weighted Blender based on coord descent, optimize task metric directly.

Weight sum eq. 1. Good blender for tabular data, even if some predictions are NaN (ex. timeout). Model with low weights will be pruned.

__init__(*max_iters=5*, *max_inner_iters=7*, *max_nonzero_coef=0.05*)

Parameters

- **max_iters** (int) – Max number of coord desc loops.
- **max_inner_iters** (int) – Max number of iters to solve inner scalar optimization task.
- **max_nonzero_coef** (float) – Maximum model weight value to stay in ensemble.

CHAPTER

TWO

LIGHTAUTOML.ADDONS

Extensions of core functionality.

2.1 Utilization

TimeUtilization	Class that helps to utilize given time to *AutoMLPreset*.

2.1.1 TimeUtilization

class lightautoml.addons.utilization.utilization.**TimeUtilization**(*automl_factory*, *task*, *timeout=3600*, *memory_limit=16*, *cpu_limit=4*, *gpu_ids=None*, *timing_params=None*, *configs_list=None*, *inner_blend=None*, *outer_blend=None*, *drop_last=True*, *return_all_predictions=False*, *max_runs_per_config=5*, *random_state_keys=None*, *random_state=42*, ***kwargs*)

Bases: `object`

Class that helps to utilize given time to *AutoMLPreset*.

Useful to calc benchmarks and compete It takes list of config files as input and run it white time limit exceeded. If time left - it can perform multistart on same configs with new random state. In best case - blend different configurations of single preset. In worst case - averaging multiple automl's with different states.

Note: Basic usage.

```
>>> ensembled_automl = TimeUtilization(TabularAutoML, Task('binary'),
>>>     timeout=3600, configs_list=['cfg0.yml', 'cfg1.yml'])
```

Then `.fit_predict` and predict can be called like usual *AutoML* class.

__init__(*automl_factory*, *task*, *timeout=3600*, *memory_limit=16*, *cpu_limit=4*, *gpu_ids=None*, *timing_params=None*, *configs_list=None*, *inner_blend=None*, *outer_blend=None*, *drop_last=True*, *return_all_predictions=False*, *max_runs_per_config=5*, *random_state_keys=None*, *random_state=42*, ***kwargs*)

Parameters

- **automl_factory** (Type[*AutoMLPreset*]) – One of presets.
- **task** (*Task*) – Task to solve.
- **timeout** (int) – Timeout in seconds.
- **memory_limit** (int) – Memory limit that are passed to each automl.
- **cpu_limit** (int) – Cpu limit that that are passed to each automl.
- **gpu_ids** (Optional[str]) – Gpu_ids that are passed to each automl.
- **verbose** – Controls the verbosity: the higher, the more messages. <1 : messages are not displayed; >=1 : the computation process for layers is displayed; >=2 : the information about folds processing is also displayed; >=3 : the hyperparameters optimization process is also displayed; >=4 : the training process for every algorithm is displayed;
- **timing_params** (Optional[dict]) – Timing_params level that are passed to each automl.
- **configs_list** (Optional[Sequence[str]]) – List of str path to configs files.
- **inner_blend** (Optional[*Blender*]) – Blender instance to blend automl's with same configs and different random state.
- **outer_blend** (Optional[*Blender*]) – Blender instance to blend averaged by random_state automl's with different configs.
- **drop_last** (bool) – Usually last automl will be stopped with timeout. Flag that defines if we should drop it from ensemble
- **return_all_predictions** (bool) – Skip blend and return all model predictions
- **max_runs_per_config** (int) – Maximum number of multistart loops.
- **random_state_keys** (Optional[dict]) – Params of config that used as random state with initial values. If None - search for *random_state* key in default config of preset. If not found - assume, that seeds are not fixed and each run is random by default. For example `{'reader_params': {'random_state': 42}, 'gbm_params': {'default_params': {'seed': 42}}}`
- **random_state** (int) – initial random seed, that will be set in case of search in config.
- ****kwargs** – Additional params.

fit_predict(*train_data*, *roles*, *train_features=None*, *cv_iter=None*, *valid_data=None*, *valid_features=None*, *verbose=0*, *log_file=None*)

Fit and get prediction on validation dataset.

Almost same as *lightautoml.automl.base.AutoML.fit_predict*.

Additional features - working with different data formats. Supported now:

- Path to `.csv`, `.parquet`, `.feather` files.
- ndarray, or dict of ndarray. For example, `{'data': X...}`. In this case, roles are optional, but *train_features* and *valid_features* required.
- pandas.DataFrame.

Parameters

- **train_data** (Any) – Dataset to train.
- **roles** (dict) – Roles dict.
- **train_features** (Optional[Sequence[str]]) – Optional features names, if can't be inferred from *train_data*.
- **cv_iter** (Optional[Iterable]) – Custom cv-iterator. For example, *TimeSeriesIterator*.
- **valid_data** (Optional[Any]) – Optional validation dataset.
- **valid_features** (Optional[Sequence[str]]) – Optional validation dataset features if cannot be inferred from *valid_data*.

Return type *LAMLDataset*

Returns Dataset with predictions. Call `.data` to get predictions array.

predict(*data*, *features_names=None*, *return_all_predictions=None*, ***kwargs*)

Get dataset with predictions.

Almost same as *lightautoml.automl.base.AutoML.predict* on new dataset, with additional features.

Additional features - working with different data formats. Supported now:

- Path to `.csv`, `.parquet`, `.feather` files.
- ndarray, or dict of ndarray. For example, `{'data': X...}`. In this case roles are optional, but *train_features* and *valid_features* required.
- pandas.DataFrame.

Parameters

- **data** (Any) – Dataset to perform inference.
- **features_names** (Optional[Sequence[str]]) – Optional features names, if cannot be inferred from *train_data*.
- **return_all_predictions** (Optional[bool]) – bool - skip blending phase

Return type *LAMLDataset*

Returns Dataset with predictions.

CHAPTER

THREE

LIGHTAUTOML.DATASET

Provides an internal interface for working with data.

3.1 Dataset Interfaces

base.LAMLColumn	Basic class for pair - column, role.
base.LAMLDataset	Basic class to create dataset.
np_pd_dataset.NumpyDataset	Dataset that contains info in np.ndarray format.
np_pd_dataset.PandasDataset	Dataset that contains *pd.DataFrame* features and *pd.Series* targets.
np_pd_dataset.CSRSparseDataset	Dataset that contains sparse features and np.ndarray targets.

3.1.1 LAMLColumn

class lightautoml.dataset.base.**LAMLColumn**(*data*, *role*)

Bases: object

Basic class for pair - column, role.

__init__(*data*, *role*)

Set a pair column/role.

Parameters

- **data** (Any) – 1d array like.
- **role** (*ColumnRole*) – Column role.

3.1.2 LAMLDataset

class lightautoml.dataset.base.**LAMLDataset**(*data*, *features*, *roles*, *task=None*, ***kwargs*)

Bases: object

Basic class to create dataset.

__init__(*data*, *features*, *roles*, *task=None*, ***kwargs*)

Create dataset with given data, features, roles and special attributes.

Parameters

- **data** (Any) – 2d array of data of special type for each dataset type.

- **features** (Optional[list]) – Feature names or None for empty data.
- **roles** (Optional[Dict[str, *ColumnRole*]]) – Features roles or None for empty data.
- **task** (Optional[*Task*]) – Task for dataset if train/valid.
- ****kwargs** – Special named array of attributes (target, group etc..).

property features
Define how to get features names list.

Return type list

Returns Features names.

property data
Get data attribute.

Return type Any

Returns Any, array like or None.

property roles
Get roles dict.

Return type Dict[str, *ColumnRole*]

Returns Dict of feature roles.

property inverse_roles
Get inverse dict of feature roles.

Return type Dict[*ColumnRole*, List[str]]

Returns dict, keys - roles, values - features names.

set_data(*data*, *features*, *roles*)
Inplace set data, features, roles for empty dataset.

Parameters

- **data** (Any) – 2d array like or None.
- **features** (Any) – List of features names.
- **roles** (Any) – Roles dict.

empty()
Get new dataset for same task and targets, groups, without features.

Return type *LAMLDataset*

Returns New empty dataset.

property shape
Get size of 2d feature matrix.

Return type Tuple[Optional[int], Optional[int]]

Returns Tuple of 2 elements.

classmethod concat(*datasets*)
Concat multiple dataset.

Default behavior - takes empty dataset from datasets[0] and concat all features from others.

Parameters **datasets** (Sequence[*LAMLDataset*]) – Sequence of datasets.

Return type *LAMLDataset*

Returns Concated dataset.

drop_features(*droplist*)
Inplace drop columns from dataset.

Parameters **droplist** (Sequence[str]) – Feature names.

Returns Dataset without columns.

static from_dataset(*dataset*)
Abstract method - how to create this type of dataset from others.

Parameters **dataset** (*LAMLDataset*) – Original type dataset.

Return type *LAMLDataset*

Returns Converted type dataset.

3.1.3 NumpyDataset

class lightautoml.dataset.np_pd_dataset.**NumpyDataset**(*data*, *features=()*, *roles=None*, *task=None*, ***kwargs*)

Bases: *lightautoml.dataset.base.LAMLDataset*

Dataset that contains info in np.ndarray format.

property features
Features list.

Return type List[str]

property roles
Roles dict.

Return type Dict[str, *ColumnRole*]

__init__(*data*, *features=()*, *roles=None*, *task=None*, ***kwargs*)
Create dataset from numpy arrays.

Parameters

- **data** (Union[ndarray, csr_matrix, None]) – 2d array of features.
- **features** (Union[Sequence[str], str, None]) – Features names.
- **roles** (Union[Sequence[*ColumnRole*], *ColumnRole*, Dict[str, *ColumnRole*], None]) – Roles specifier.
- **task** (Optional[*Task*]) – Task specifier.
- ****kwargs** – Named attributes like target, group etc ..

Note: For different type of parameter feature there is different behavior:

- list, should be same len as data.shape[1]
- None - automatic set names like feat_0, feat_1 . . .
- Prefix - automatic set names like Prefix_0, Prefix_1 . . .

For different type of parameter feature there is different behavior:

- list, should be same len as data.shape[1].
- None - automatic set NumericRole(np.float32).

- ColumnRole - single role.
- dict.

---

set_data(*data*, *features=()*, *roles=None*)

Inplace set data, features, roles for empty dataset.

Parameters

- **data** (Union[ndarray, csr_matrix]) – 2d np.ndarray of features.
- **features** (Union[Sequence[str], str, None]) – features names.
- **roles** (Union[Sequence[*ColumnRole*], *ColumnRole*, Dict[str, *ColumnRole*], None]) – Roles specifier.

Note: For different type of parameter feature there is different behavior:

- List, should be same len as data.shape[1]
- None - automatic set names like feat_0, feat_1 …
- Prefix - automatic set names like Prefix_0, Prefix_1 …

For different type of parameter feature there is different behavior:

- List, should be same len as data.shape[1].
- None - automatic set NumericRole(np.float32).
- ColumnRole - single role.
- dict.

---

to_numpy()

Empty method to convert to numpy.

Return type *NumpyDataset*

Returns Same NumpyDataset.

to_csr()

Convert to csr.

Return type *CSRSparseDataset*

Returns Same dataset in CSRSparseDatatset format.

to_pandas()

Convert to PandasDataset.

Return type *PandasDataset*

Returns Same dataset in PandasDataset format.

static from_dataset(*dataset*)

Convert random dataset to numpy.

Return type *NumpyDataset*

Returns numpy dataset.

3.1.4 PandasDataset

class lightautoml.dataset.np_pd_dataset.**PandasDataset**(*data=None*, *roles=None*, *task=None*, ***kwargs*)

Bases: *lightautoml.dataset.base.LAMLDataset*

Dataset that contains *pd.DataFrame* features and *pd.Series* targets.

property features

Get list of features.

Return type List[str]

Returns list of features.

__init__(*data=None*, *roles=None*, *task=None*, ***kwargs*)

Create dataset from *pd.DataFrame* and *pd.Series*.

Parameters

- **data** (Optional[DataFrame]) – Table with features.
- **features** – features names.
- **roles** (Optional[Dict[str, *ColumnRole*]]) – Roles specifier.
- **task** (Optional[*Task*]) – Task specifier.
- ****kwargs** – Series, array like attrs target, group etc…

set_data(*data*, *features*, *roles*)

Inplace set data, features, roles for empty dataset.

Parameters

- **data** (DataFrame) – Table with features.
- **features** (None) – *None*, just for same interface.
- **roles** (Dict[str, *ColumnRole*]) – Dict with roles.

to_numpy()

Convert to class:*NumpyDataset*.

Returns *NumpyDataset* format.

Return type Same dataset in class

to_pandas()

Empty method, return the same object.

Return type *PandasDataset*

Returns Self.

static from_dataset(*dataset*)

Convert random dataset to pandas dataset.

Return type *PandasDataset*

Returns Converted to pandas dataset.

nan_rate()

Counts overall number of nans in dataset.

Returns Number of nans.

3.1.5 CSRSparseDataset

class lightautoml.dataset.np_pd_dataset.**CSRSparseDataset**(*data*, *features=()*, *roles=None*, *task=None*, ***kwargs*)

Bases: *lightautoml.dataset.np_pd_dataset.NumpyDataset*

Dataset that contains sparse features and np.ndarray targets.

to_pandas()
Not implemented.

Return type Any

to_numpy()
Convert to NumpyDataset.

Return type *NumpyDataset*

Returns NumpyDataset.

property shape
Get size of 2d feature matrix.

Return type Tuple[Optional[int], Optional[int]]

Returns tuple of 2 elements.

__init__(*data*, *features=()*, *roles=None*, *task=None*, ***kwargs*)
Create dataset from csr_matrix.

Parameters

- **data** (Union[ndarray, csr_matrix, None]) – csr_matrix of features.
- **features** (Union[Sequence[str], str, None]) – Features names.
- **roles** (Union[Sequence[*ColumnRole*], *ColumnRole*, Dict[str, *ColumnRole*], None]) – Roles specifier.
- **task** (Optional[*Task*]) – Task specifier.
- ****kwargs** – Named attributes like target, group etc ..

Note: For different type of parameter feature there is different behavior:

- list, should be same len as data.shape[1]
- None - automatic set names like feat_0, feat_1 . . .
- Prefix - automatic set names like Prefix_0, Prefix_1 . . .

For different type of parameter feature there is different behavior:

- list, should be same len as data.shape[1].
- None - automatic set NumericRole(np.float32).
- ColumnRole - single role.
- dict.

---

set_data(*data*, *features=()*, *roles=None*)
Inplace set data, features, roles for empty dataset.

Parameters

- **data** (Union[ndarray, csr_matrix]) – csr_matrix of features.
- **features** (Union[Sequence[str], str, None]) – features names.
- **roles** (Union[Sequence[*ColumnRole*], *ColumnRole*, Dict[str, *ColumnRole*], None]) – Roles specifier.

Note: For different type of parameter feature there is different behavior:

- list, should be same len as data.shape[1]
- None - automatic set names like feat_0, feat_1 ...
- Prefix - automatic set names like Prefix_0, Prefix_1 ...

For different type of parameter feature there is different behavior:

- list, should be same len as data.shape[1].
- None - automatic set NumericRole(np.float32).
- ColumnRole - single role.
- dict.

---

static from_dataset(*dataset*)

Convert dataset to sparse dataset.

Return type *CSRSparseDataset*

Returns Dataset in sparse form.

3.2 Roles

Role contains information about the column, which determines how it is processed.

ColumnRole	Abstract class for column role.
NumericRole	Numeric role.
CategoryRole	Category role.
TextRole	Text role.
DatetimeRole	Datetime role.
TargetRole	Target role.
GroupRole	Group role.
DropRole	Drop role.
WeightsRole	Weights role.
FoldsRole	Folds role.
PathRole	Path role.

3.2.1 ColumnRole

class lightautoml.dataset.roles.**ColumnRole**

Bases: object

Abstract class for column role.

Role type defines column dtype, place of column in dataset and transformers and set additional attributes which impacts on the way how it's handled.

dtype

alias of object

property name

Get str role name.

Return type str

Returns str role name.

static from_string(*name*, ***kwargs*)

Create default params role from string.

Parameters **name** (str) – Role name.

Return type *ColumnRole*

Returns Corresponding role object.

3.2.2 NumericRole

class lightautoml.dataset.roles.**NumericRole**(*dtype=numpy.float32*, *force_input=False*, *prob=False*, *discretization=False*)

Bases: *lightautoml.dataset.roles.ColumnRole*

Numeric role.

__init__(*dtype=numpy.float32*, *force_input=False*, *prob=False*, *discretization=False*)

Create numeric role with specific numeric dtype.

Parameters

- **dtype** (Union[Callable, str]) – Variable type.
- **force_input** (bool) – Select a feature for training, regardless of the selector results.
- **prob** (bool) – If input number is probability.

3.2.3 CategoryRole

class lightautoml.dataset.roles.**CategoryRole**(*dtype=<class 'object'>*, *encoding_type='auto'*, *unknown=5*, *force_input=False*, *label_encoded=False*, *ordinal=False*)

Bases: *lightautoml.dataset.roles.ColumnRole*

Category role.

__init__(*dtype=<class 'object'>*, *encoding_type='auto'*, *unknown=5*, *force_input=False*, *label_encoded=False*, *ordinal=False*)

Create category role with specific dtype and attrs.

Parameters

- **dtype** (Union[Callable, str]) – Variable type.
- **encoding_type** (str) – Encoding type.
- **unknown** (int) – Cut-off freq to process rare categories as unseen.
- **force_input** (bool) – Select a feature for training, regardless of the selector results.

Note: Valid encoding_type:

- *'auto'* - default processing
- *'int'* - encode with int
- *'oof'* - out-of-fold target encoding
- *'freq'* - frequency encoding
- *'ohe'* - one hot encoding

3.2.4 TextRole

class lightautoml.dataset.roles.**TextRole**(*dtype=<class 'str'>*, *force_input=True*)

Bases: *lightautoml.dataset.roles.ColumnRole*

Text role.

__init__(*dtype=<class 'str'>*, *force_input=True*)

Create text role with specific dtype and attrs.

Parameters

- **dtype** (Union[Callable, str]) – Variable type.
- **force_input** (bool) – Select a feature for training, regardless of the selector results.

3.2.5 DatetimeRole

class lightautoml.dataset.roles.**DatetimeRole**(*dtype=numpy.datetime64*, *seasonality=('y', 'm', 'wd')*, *base_date=False*, *date_format=None*, *unit=None*, *origin='unix'*, *force_input=False*, *base_feats=True*, *country=None*, *prov=None*, *state=None*)

Bases: *lightautoml.dataset.roles.ColumnRole*

Datetime role.

__init__(*dtype=numpy.datetime64*, *seasonality=('y', 'm', 'wd')*, *base_date=False*, *date_format=None*, *unit=None*, *origin='unix'*, *force_input=False*, *base_feats=True*, *country=None*, *prov=None*, *state=None*)

Create datetime role with specific dtype and attrs.

Parameters

- **dtype** (Union[Callable, str]) – Variable type.
- **seasonality** (Optional[Sequence[str]]) – Seasons to extract from date. Valid are: 'y', 'm', 'd', 'wd', 'hour', 'min', 'sec', 'ms', 'ns'.
- **base_date** (bool) – Base date is used to calculate difference with other dates, like *age = report_dt - birth_dt*.

- **date_format** (Optional[str]) – Format to parse date.
- **unit** (Optional[str]) – The unit of the arg denote the unit, pandas like, see more: https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.to_datetime.html.
- **origin** (Union[str, datetime]) – Define the reference date, pandas like, see more: https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.to_datetime.html.
- **force_input** (bool) – Select a feature for training, regardless of the selector results.
- **base_feats** (bool) – To calculate feats on base date.
- **country** (Optional[str]) – Datetime metadata to extract holidays.
- **prov** (Optional[str]) – Datetime metadata to extract holidays.
- **state** (Optional[str]) – Datetime metadata to extract holidays.

3.2.6 TargetRole

class lightautoml.dataset.roles.**TargetRole**(*dtype=numpy.float32*)
Bases: *lightautoml.dataset.roles.ColumnRole*

Target role.

__init__(*dtype=numpy.float32*)
Create target role with specific numeric dtype.

Parameters **dtype** (Union[Callable, str]) – Dtype of target.

3.2.7 GroupRole

class lightautoml.dataset.roles.**GroupRole**
Bases: *lightautoml.dataset.roles.ColumnRole*

Group role.

3.2.8 DropRole

class lightautoml.dataset.roles.**DropRole**
Bases: *lightautoml.dataset.roles.ColumnRole*

Drop role.

3.2.9 WeightsRole

class lightautoml.dataset.roles.**WeightsRole**
Bases: *lightautoml.dataset.roles.ColumnRole*

Weights role.

3.2.10 FoldsRole

class lightautoml.dataset.roles.**FoldsRole**
Bases: *lightautoml.dataset.roles.ColumnRole*

Folds role.

3.2.11 PathRole

class lightautoml.dataset.roles.**PathRole**
Bases: *lightautoml.dataset.roles.ColumnRole*

Path role.

3.3 Utils

Utilities for working with the structure of a dataset.

roles_parser	Parser of roles.
get_common_concat	Get concatenation function for datasets of different types.
numpy_and_pandas_concat	Concat of numpy and pandas dataset.
concatenate	Dataset concatenation function.

3.3.1 roles_parser

lightautoml.dataset.utils.**roles_parser**(*init_roles*)
Parser of roles.

Parse roles from old format numeric: `[var1, var2, ...]` to `{var1:numeric, var2:numeric, ...}`.

Parameters **init_roles** (Dict[Union[*ColumnRole*, str], Union[str, Sequence[str]]]) – Mapping between roles and feature names.

Return type Dict[str, *ColumnRole*]

Returns Roles dict in format key - feature names, value - roles.

3.3.2 get_common_concat

lightautoml.dataset.utils.**get_common_concat**(*datasets*)
Get concatenation function for datasets of different types.

Takes multiple datasets as input and check, if is's ok to concatenate it and return function.

Parameters **datasets** (Sequence[*LAMLDataset*]) – Sequence of datasets.

Return type Tuple[Callable, Optional[type]]

Returns Function, that is able to concatenate datasets.

3.3.3 numpy_and_pandas_concat

lightautoml.dataset.utils.**numpy_and_pandas_concat**(*datasets*)

Concat of numpy and pandas dataset.

Parameters **datasets** (Sequence[Union[*NumpyDataset*, *PandasDataset*]]) – Sequence of datasets to concatenate.

Return type *PandasDataset*

Returns Concatenated dataset.

3.3.4 concatenate

lightautoml.dataset.utils.**concatenate**(*datasets*)

Dataset concatenation function.

Check if datasets have common concat function and then apply. Assume to take target/folds/weights etc from first one.

Parameters **datasets** (Sequence[*LAMLDataset*]) – Sequence of datasets.

Return type *LAMLDataset*

Returns Dataset with concatenated features.

CHAPTER

FOUR

LIGHTAUTOML.IMAGE

Provides an internal interface for working with image features.

4.1 Image Feature Extractors

Image feature extractors based on color histograms and CNN embeddings.

CreateImageFeatures	Class for parallel histogram computation.
EffNetImageEmbedder	Class to compute EfficientNet embeddings.

4.1.1 CreateImageFeatures

class lightautoml.image.image.**CreateImageFeatures**(*hist_size=30*, *is_hsv=True*, *n_jobs=4*, *loader=<function pil_loader>*)

Bases: object

Class for parallel histogram computation.

__init__(*hist_size=30*, *is_hsv=True*, *n_jobs=4*, *loader=<function pil_loader>*)

Create normalized color histogram for rgb or hsv image.

Parameters

- **hist_size** (int) – Number of bins for each channel.
- **is_hsv** (bool) – Convert image to hsv.
- **n_jobs** (int) – Number of threads for multiprocessing.
- **loader** (Callable) – Callable for reading image from path.

process(*im_path_i*)

Create normalized color histogram for input image by its path.

Parameters **im_path_i** (str) – Path to the image.

Return type List[Union[int, float]]

Returns List of histogram values.

transform(*samples*)

Transform input sequence with paths to histogram values.

Parameters **samples** (Sequence[str]) – Sequence with images paths.

Return type ndarray

Returns Array of histograms.

4.1.2 EffNetImageEmbedder

class lightautoml.image.image.**EffNetImageEmbedder**(*model_name='efficientnet-b0'*, *weights_path=None*, *is_advprop=True*, *device=torch.device*)

Bases: torch.nn.Module

Class to compute EfficientNet embeddings.

__init__(*model_name='efficientnet-b0'*, *weights_path=None*, *is_advprop=True*, *device=torch.device*)
Pytorch module for image embeddings based on efficient-net model.

Parameters

- **model_name** (str) – Name of effnet model.
- **weights_path** (Optional[str]) – Path to saved weights.
- **is_advprop** (bool) – Use adversarial training.
- **devices** – Device to use.

get_shape()
Calculate output embedding shape.

Return type int

Returns Shape of embedding.

4.2 PyTorch Image Datasets

ImageDataset	Image Dataset Class.
DeepImageEmbedder	Transformer for image embeddings.

4.2.1 ImageDataset

class lightautoml.image.image.**ImageDataset**(*data*, *is_advprop=True*, *loader=<function pil_loader>*)

Bases: object

Image Dataset Class.

__init__(*data*, *is_advprop=True*, *loader=<function pil_loader>*)
Pytorch Dataset for EffNetImageEmbedder.

Parameters

- **data** (Sequence[str]) – Sequence of paths.
- **is_advprop** (bool) – Use adversarial training.
- **loader** (Callable) – Callable for reading image from path.

4.2.2 DeepImageEmbedder

class lightautoml.image.image.**DeepImageEmbedder**(*device=torch.device*, *n_jobs=4*, *random_state=42*, *is_advprop=True*, *model_name='efficientnet-b0'*, *weights_path=None*, *batch_size=128*, *verbose=True*)

Bases: sklearn.base.TransformerMixin

Transformer for image embeddings.

__init__(*device=torch.device*, *n_jobs=4*, *random_state=42*, *is_advprop=True*, *model_name='efficientnet-b0'*, *weights_path=None*, *batch_size=128*, *verbose=True*)

Pytorch Dataset for EffNetImageEmbedder.

Parameters

- **device** (device) – Torch device.
- **n_jobs** – Number of threads for dataloader.
- **random_state** – Random seed.
- **is_advprop** – Use adversarial training.
- **model_name** – Name of effnet model.
- **weights_path** (Optional[str]) – Path to saved weights.
- **batch_size** (int) – Batch size.
- **verbose** (bool) – Verbose data processing.

transform(*data*)

Calculate image embeddings from pathes.

Parameters **data** (Sequence[str]) – Sequence of paths.

Return type ndarray

Returns Array of embeddings.

4.3 Utils

pil_loader	Load image from pathes.

4.3.1 pil_loader

lightautoml.image.utils.**pil_loader**(*path*)

Load image from pathes.

Parameters **path** (str) – Image path.

Return type <module 'PIL.Image' from '/home/docs/checkouts/readthedocs.org/user_builds/lightautoml/envs/latest/lib/pytl packages/PIL/Image.py'>

Returns Loaded PIL Image in rgb.

CHAPTER

FIVE

LIGHTAUTOML.ML_ALGO

Models used for machine learning pipelines.

5.1 Base Classes

MLAlgo	Abstract class for machine learning algorithm.
TabularMLAlgo	Machine learning algorithms that accepts numpy arrays as input.

5.1.1 MLAlgo

class lightautoml.ml_algo.base.**MLAlgo**(*default_params=None*, *freeze_defaults=True*, *timer=None*, *optimization_search_space={}*)

Bases: abc.ABC

Abstract class for machine learning algorithm. Assume that features are already selected, but parameters my be tuned and set before training.

property name
Get model name.

Return type str

property features
Get list of features.

Return type List[str]

property is_fitted
Get flag is the model fitted or not.

Return type bool

property params
Get model's params dict.

Return type dict

init_params_on_input(*train_valid_iterator*)
Init params depending on input data.

Parameters **train_valid_iterator** (*TrainValidIterator*) – Classic cv-iterator.

Return type dict

Returns Dict with model hyperparameters.

__init__(*default_params=None*, *freeze_defaults=True*, *timer=None*, *optimization_search_space={}*)

Parameters

- **default_params** (Optional[dict]) – Algo hyperparams.
- **freeze_defaults** (bool) –
  - True : params may be rewrited depending on dataset.
  - False: params may be changed only manually or with tuning.
- **timer** (Optional[*TaskTimer*]) – Timer for Algo.

abstract fit_predict(*train_valid_iterator*)

Abstract method.

Fit new algo on iterated datasets and predict on valid parts.

Parameters **train_valid_iterator** (*TrainValidIterator*) – Classic cv-iterator.

Return type *LAMLDataset*

abstract predict(*test*)

Predict target for input data.

Parameters **test** (*LAMLDataset*) – Dataset on test.

Return type *LAMLDataset*

Returns Dataset with predicted values.

score(*dataset*)

Score prediction on dataset with defined metric.

Parameters **dataset** (*LAMLDataset*) – Dataset with ground truth and predictions.

Return type float

Returns Metric value.

set_prefix(*prefix*)

Set prefix to separate models from different levels/pipelines.

Parameters **prefix** (str) – String with prefix.

set_timer(*timer*)

Set timer.

Return type *MLAlgo*

5.1.2 TabularMLAlgo

class lightautoml.ml_algo.base.**TabularMLAlgo**(*default_params=None*, *freeze_defaults=True*, *timer=None*, *optimization_search_space={}*)

Bases: *lightautoml.ml_algo.base.MLAlgo*

Machine learning algorithms that accepts numpy arrays as input.

fit_predict_single_fold(*train*, *valid*)

Train on train dataset and predict on holdout dataset.

Parameters

- **train** (Union[*NumpyDataset*, *PandasDataset*]) – Train Dataset.
- **valid** (Union[*NumpyDataset*, *PandasDataset*]) – Validation Dataset.

Return type Tuple[Any, ndarray]

Returns Target predictions for valid dataset.

fit_predict(*train_valid_iterator*)

Fit and then predict accordig the strategy that uses train_valid_iterator.

If item uses more then one time it will predict mean value of predictions. If the element is not used in training then the prediction will be `numpy.nan` for this item

Parameters **train_valid_iterator** (*TrainValidIterator*) – Classic cv-iterator.

Return type *NumpyDataset*

Returns Dataset with predicted values.

predict_single_fold(*model*, *dataset*)

Implements prediction on single fold.

Parameters

- **model** (Any) – Model uses to predict.
- **dataset** (Union[*NumpyDataset*, *PandasDataset*]) – Dataset used for prediction.

Return type ndarray

Returns Predictions for input dataset.

predict(*dataset*)

Mean prediction for all fitted models.

Parameters **dataset** (Union[*NumpyDataset*, *PandasDataset*]) – Dataset used for prediction.

Return type *NumpyDataset*

Returns Dataset with predicted values.

5.2 Linear Models

LinearLBFGS	LBFGS L2 regression based on torch.
LinearL1CD	Coordinate descent based on sklearn implementation.

5.2.1 LinearLBFGS

class lightautoml.ml_algo.linear_sklearn.**LinearLBFGS**(*default_params=None*, *freeze_defaults=True*, *timer=None*, *optimization_search_space={}*)

Bases: *lightautoml.ml_algo.base.TabularMLAlgo*

LBFGS L2 regression based on torch.

default_params:

- cs: List of regularization coefficients.
- max_iter: Maximum iterations of L-BFGS.
- tol: The tolerance for the stopping criteria.

- early_stopping: Maximum rounds without improving.

freeze_defaults:

- `True` : params may be rewrited depending on dataset.
- `False`: params may be changed only manually or with tuning.

timer: *Timer* instance or `None`.

fit_predict_single_fold(*train*, *valid*)
Train on train dataset and predict on holdout dataset.

Parameters

- **train** (Union[*NumpyDataset*, *PandasDataset*]) – Train Dataset.
- **valid** (Union[*NumpyDataset*, *PandasDataset*]) – Validation Dataset.

Return type Tuple[TorchBasedLinearEstimator, ndarray]

Returns Target predictions for valid dataset.

predict_single_fold(*model*, *dataset*)
Implements prediction on single fold.

Parameters

- **model** (TorchBasedLinearEstimator) – Model uses to predict.
- **dataset** (Union[*NumpyDataset*, *PandasDataset*]) – NumpyDataset used for prediction.

Return type ndarray

Returns Predictions for input dataset.

5.2.2 LinearL1CD

class lightautoml.ml_algo.linear_sklearn.**LinearL1CD**(*default_params=None*, *freeze_defaults=True*, *timer=None*, *optimization_search_space={}*)

Bases: *lightautoml.ml_algo.base.TabularMLAlgo*

Coordinate descent based on sklearn implementation.

init_params_on_input(*train_valid_iterator*)
Get model parameters depending on dataset parameters.

Parameters **train_valid_iterator** (*TrainValidIterator*) – Classic cv-iterator.

Return type dict

Returns Parameters of model.

fit_predict_single_fold(*train*, *valid*)
Train on train dataset and predict on holdout dataset.

Parameters

- **train** (Union[*NumpyDataset*, *PandasDataset*]) – Train Dataset.
- **valid** (Union[*NumpyDataset*, *PandasDataset*]) – Validation Dataset.

Return type Tuple[Union[LogisticRegression, ElasticNet, Lasso], ndarray]

Returns Target predictions for valid dataset.

predict_single_fold(*model*, *dataset*)
Implements prediction on single fold.

Parameters

- **model** (Union[LogisticRegression, ElasticNet, Lasso]) – Model uses to predict.
- **dataset** (Union[*NumpyDataset*, *PandasDataset*]) – Dataset used for prediction.

Return type ndarray

Returns Predictions for input dataset.

5.3 Boosted Trees

BoostLGBM	Gradient boosting on decision trees from LightGBM library.
BoostCB	Gradient boosting on decision trees from catboost library.

5.3.1 BoostLGBM

class lightautoml.ml_algo.boost_lgbm.**BoostLGBM**(*default_params=None*, *freeze_defaults=True*, *timer=None*, *optimization_search_space={}*)

Bases: *lightautoml.ml_algo.base.TabularMLAlgo*, *lightautoml.pipelines.selection.base.ImportanceEstimator*

Gradient boosting on decision trees from LightGBM library.

default_params: All available parameters listed in lightgbm documentation:

- https://lightgbm.readthedocs.io/en/latest/Parameters.html

freeze_defaults:

- `True` : params may be rewritten depending on dataset.
- `False`: params may be changed only manually or with tuning.

timer: *Timer* instance or `None`.

init_params_on_input(*train_valid_iterator*)
Get model parameters depending on dataset parameters.

Parameters **train_valid_iterator** (*TrainValidIterator*) – Classic cv-iterator.

Return type dict

Returns Parameters of model.

fit_predict_single_fold(*train*, *valid*)
Implements training and prediction on single fold.

Parameters

- **train** (Union[*NumpyDataset*, *PandasDataset*]) – Train Dataset.
- **valid** (Union[*NumpyDataset*, *PandasDataset*]) – Validation Dataset.

Return type Tuple[Booster, ndarray]

Returns Tuple (model, predicted_values)

predict_single_fold(*model*, *dataset*)
Predict target values for dataset.

Parameters

- **model** (Booster) – Lightgbm object.
- **dataset** (Union[*NumpyDataset*, *PandasDataset*]) – Test Dataset.

Return type ndarray

Returns Predicted target values.

get_features_score()
Computes feature importance as mean values of feature importance provided by lightgbm per all models.

Return type Series

Returns Series with feature importances.

fit(*train_valid*)
Just to be compatible with *ImportanceEstimator*.

Parameters **train_valid** (*TrainValidIterator*) – Classic cv-iterator.

5.3.2 BoostCB

class lightautoml.ml_algo.boost_cb.**BoostCB**(*default_params=None*, *freeze_defaults=True*, *timer=None*, *optimization_search_space={}*)

Bases: *lightautoml.ml_algo.base.TabularMLAlgo*, *lightautoml.pipelines.selection.base.ImportanceEstimator*

Gradient boosting on decision trees from catboost library.

All available parameters listed in CatBoost documentation:

- https://catboost.ai/docs/concepts/python-reference_parameters-list.html#python-reference_parameters-list

freeze_defaults:

- True : params may be rewritten depending on dataset.
- False: params may be changed only manually or with tuning.

timer: *Timer* instance or None.

init_params_on_input(*train_valid_iterator*)
Get model parameters depending on input dataset parameters.

Parameters **train_valid_iterator** (*TrainValidIterator*) – Classic cv-iterator.

Return type dict

Returns Parameters of model.

fit_predict_single_fold(*train*, *valid*)
Implements training and prediction on single fold.

Parameters

- **train** (Union[*NumpyDataset*, *PandasDataset*]) – Train Dataset.
- **valid** (Union[*NumpyDataset*, *PandasDataset*]) – Validation Dataset.

Return type Tuple[CatBoost, ndarray]

Returns Tuple (model, predicted_values).

predict_single_fold(*model*, *dataset*)
Predict of target values for dataset.

Parameters

- **model** (CatBoost) – CatBoost object.
- **dataset** (Union[*NumpyDataset*, *PandasDataset*]) – Test dataset.

Return type ndarray

Returns Predicted target values.

get_features_score()
Computes feature importance.

Computes as mean values of feature importance, provided by CatBoost (PredictionValuesChange), per all models.

Return type Series

Returns Series with feature importances.

fit(*train_valid*)
Just to be compatible with *ImportanceEstimator*.

Parameters **train_valid** (*TrainValidIterator*) – Classic cv-iterator.

5.4 WhiteBox

WbMLAlgo	WhiteBox - scorecard model.

5.4.1 WbMLAlgo

class lightautoml.ml_algo.whitebox.**WbMLAlgo**(*default_params=None*, *freeze_defaults=True*, *timer=None*, *optimization_search_space={}*)

Bases: *lightautoml.ml_algo.base.TabularMLAlgo*

WhiteBox - scorecard model.

https://github.com/sberbank-ai-lab/AutoMLWhitebox

default_params:

- **monotonic: bool** Global condition for monotonic constraints. If True, then only monotonic binnings will be built. You can pass values to the .fit method that change this condition separately for each feature.
- **max_bin_count: int** Global limit for the number of bins. Can be specified for every feature in .fit
- **select_type: None or int** The type to specify the primary feature selection. If the type is an integer, then we select the number of features indicated by this number (with the best *feature_importance*). If the value is None, we leave only features with feature_importance greater than 0.
- **pearson_th: 0 < pearson_th < 1** Threshold for feature selection by correlation. All features with the absolute value of correlation coefficient greater then pearson_th will be discarded.

- **auc_th: .5 < auc_th < 1** Threshold for feature selection by one-dimensional AUC. WoE with AUC < auc_th will be discarded.
- **vif_th: vif_th > 0** Threshold for feature selection by VIF. Features with VIF > vif_th are iteratively discarded one by one, then VIF is recalculated until all VIFs are less than vif_th.
- **imp_th: real >= 0** Threshold for feature selection by feature importance
- **th_const:** Threshold, which determines that the feature is constant. If the number of valid values is greater than the threshold, then the column is not constant. For float, the number of valid values will be calculated as the sample size * th_const
- **force_single_split: bool** In the tree parameters, you can set the minimum number of observations in the leaf. Thus, for some features, splitting for 2 beans at least will be impossible. If you specify that `force_single_split = True`, it means that 1 split will be created for the feature, if the minimum bin size is greater than th_const.
- **th_nan: int >= 0** Threshold, which determines that WoE values are calculated to NaN.
- **th_cat: int >= 0** Threshold, which determines which categories are small.
- **woe_diff_th: float = 0.01** The option to merge NaNs and rare categories with another bin, if the difference in WoE is less than woe_diff_th.
- **min_bin_size: int > 1, 0 < float < 1** Minimum bin size when splitting.
- **min_bin_mults: list of floats > 1** If minimum bin size is specified, you can specify a list to check if large values work better, for example: [2, 4].
- **min_gains_to_split: list of floats >= 0** min_gain_to_split values that will be iterated to find the best split.
- **auc_tol: 1e-5 <= auc_tol <=1e-2** AUC tolerance. You can lower the auc_tol value from the maximum to make the model simpler.
- **cat_alpha: float > 0** Regularizer for category encoding.
- **cat_merge_to: str** The way of WoE values filling in the test sample for categories that are not in the training sample. Values - 'to_nan', 'to_woe_0', 'to_maxfreq', 'to_maxp', 'to_minp'
- **nan_merge_to: str** The way of WoE values filling on the test sample for real NaNs, if they are not included in their group. Values - 'to_woe_0', 'to_maxfreq', 'to_maxp', 'to_minp'
- **oof_woe: bool** Use OOF or standard encoding for WOE.
- **n_folds: int** Number of folds for feature selection / encoding, etc.
- **n_jobs: int > 0** Number of CPU cores to run in parallel.
- **l1_base_step: real > 0** Grid size in l1 regularization
- **l1_exp_step: real > 1** Grid scale in l1 regularization
- **population_size: None, int > 0** Feature selection type in the selector. If the value is `None` then L1 boost is used. If `int` is specified, then a standard step will be used for the number of random subsamples indicated by this value. Can be generalized to genetic algorithm.
- **feature_groups_count: int > 0** The number of groups in the genetic algorithm. Its effect is visible only when population_size > 0
- **imp_type: str** Feature importances type. Feature_imp and perm_imp are available. It is used to sort the features at the first and at the final stage of feature selection.
- **regularized_refit: bool** Use regularization at the time of model refit. Otherwise, we have a statistical model.

- **p_val: 0 < p_val <= 1** When training a statistical model, do backward selection until all p-values of the model's coefficient are
- **verbose: int 0-3** Verbosity level

freeze_defaults:

- `True` : params may be rewrited depending on dataset.
- `False`: params may be changed only manually or with tuning.

timer: *Timer* instance or `None`.

fit_predict_single_fold(*train*, *valid*)
Implements training and prediction on single fold.

Parameters

- **train** (*PandasDataset*) – Train Dataset.
- **valid** (*PandasDataset*) – Validation Dataset.

Return type Tuple[Union[AutoWoE, ReportDeco], ndarray]

Returns Tuple (model, predicted_values).

predict_single_fold(*model*, *dataset*)
Predict target values for dataset.

Parameters

- **model** (Union[AutoWoE, ReportDeco]) – WhiteBox model
- **dataset** (*PandasDataset*) – Test dataset.

Return type ndarray

Returns Predicted target values.

fit(*train_valid*)
Just to be compatible with ImportanceEstimator.

Parameters **train_valid** (*TrainValidIterator*) – classic cv iterator.

predict(*dataset*, *report=False*)
Predict on new dataset.

Parameters

- **dataset** (*PandasDataset*) – Dataset.
- **report** (bool) – Flag to generate report.

Return type *NumpyDataset*

Returns Dataset with predictions.

CHAPTER

SIX

LIGHTAUTOML.ML_ALGO.TUNING

Bunch of classes for hyperparameters tuning.

6.1 Base Classes

ParamsTuner	Base abstract class for hyperparameters tuners.
DefaultTuner	Default realization of ParamsTuner - just take algo's defaults.

6.1.1 ParamsTuner

class lightautoml.ml_algo.tuning.base.**ParamsTuner**

Bases: abc.ABC

Base abstract class for hyperparameters tuners.

property best_params

Get best params.

Return type dict

Returns Dict with best fitted params.

abstract fit(*ml_algo*, *train_valid_iterator=None*)

Tune model hyperparameters.

Parameters

- **ml_algo** (*MLAlgo*) – ML algorithm.
- **train_valid_iterator** (Optional[*TrainValidIterator*]) – Classic cv-iterator.

Return type Tuple[None, None]

Returns (None, None) if ml_algo is fitted or models are not fitted during training, (BestMLAlgo, BestPredictionsLAMLDataset) otherwise.

6.1.2 DefaultTuner

class lightautoml.ml_algo.tuning.base.**DefaultTuner**
Bases: *lightautoml.ml_algo.tuning.base.ParamsTuner*

Default realization of ParamsTuner - just take algo's defaults.

fit(*ml_algo*, *train_valid_iterator=None*)
Default fit method - just save defaults.

Parameters

- **ml_algo** (*MLAlgo*) – Algorithm that is tuned.
- **train_valid_iterator** (Optional[*TrainValidIterator*]) – Empty.

Returns:s Tuple (None, None).

Return type Tuple[None, None]

6.2 Tuning with Optuna

OptunaTuner	Wrapper for optuna tuner.

6.2.1 OptunaTuner

class lightautoml.ml_algo.tuning.optuna.**OptunaTuner**(*timeout=1000*, *n_trials=100*, *direction='maximize'*, *fit_on_holdout=True*, *random_state=42*)
Bases: *lightautoml.ml_algo.tuning.base.ParamsTuner*

Wrapper for optuna tuner.

__init__(*timeout=1000*, *n_trials=100*, *direction='maximize'*, *fit_on_holdout=True*, *random_state=42*)

Parameters

- **timeout** (Optional[int]) – Maximum learning time.
- **n_trials** (Optional[int]) – Maximum number of trials.
- **direction** (Optional[str]) – Direction of optimization. Set `minimize` for minimization and `maximize` for maximization.
- **fit_on_holdout** (bool) – Will be used holdout cv-iterator.
- **random_state** (int) – Seed for optuna sampler.

fit(*ml_algo*, *train_valid_iterator=None*)
Tune model.

Parameters

- **ml_algo** (*~TunableAlgo*) – Algo that is tuned.
- **train_valid_iterator** (Optional[*TrainValidIterator*]) – Classic cv-iterator.

Return type Tuple[Optional[~TunableAlgo], Optional[*LAMLDataset*]]

Returns Tuple (None, None) if an optuna exception raised or `fit_on_holdout=True` and `train_valid_iterator` is not *HoldoutIterator*. Tuple (MlALgo, preds_ds) otherwise.

plot()

Plot optimization history of all trials in a study.

CHAPTER

SEVEN

LIGHTAUTOML.PIPELINES

Pipelines for solving different tasks.

7.1 Utils

map_pipeline_names	Pipelines create name in the way 'prefix__feature_name'.
get_columns_by_role	Search for columns with specific role and attributes when building pipeline.

7.1.1 map_pipeline_names

lightautoml.pipelines.utils.**map_pipeline_names**(*input_names*, *output_names*)

Pipelines create name in the way 'prefix__feature_name'.

Multiple pipelines will create names in the way 'prefix1__prefix2__feature_name'. This function maps initial features names to outputs. Result may be not exact in some rare cases, but it's ok for real pipelines.

Parameters

- **input_names** (Sequence[str]) – Initial feature names.
- **output_names** (Sequence[str]) – Output feature names.

Return type List[Optional[str]]

Returns Mapping between feature names.

7.1.2 get_columns_by_role

lightautoml.pipelines.utils.**get_columns_by_role**(*dataset*, *role_name*, ***kwargs*)

Search for columns with specific role and attributes when building pipeline.

Parameters

- **dataset** (*LAMLDataset*) – Dataset to search.
- **role_name** (str) – Name of features role.
- ****kwargs** – Specific parameters values to search. Example: search for categories with OHE processing only.

Return type List[str]

Returns List of str features names.

CHAPTER

EIGHT

LIGHTAUTOML.PIPELINES.SELECTION

Feature selection module for ML pipelines.

8.1 Base Classes

ImportanceEstimator	Abstract class, that estimates feature importances.
SelectionPipeline	Abstract class, performing feature selection.

8.1.1 ImportanceEstimator

class lightautoml.pipelines.selection.base.**ImportanceEstimator**

Bases: object

Abstract class, that estimates feature importances.

get_features_score()

Get raw features importances.

Return type Series

Returns Pandas Series object with index - str features names and values - array of importances.

8.1.2 SelectionPipeline

class lightautoml.pipelines.selection.base.**SelectionPipeline**(*features_pipeline=None*, *ml_algo=None*, *imp_estimator=None*, *fit_on_holdout=False*, ***kwargs*)

Bases: object

Abstract class, performing feature selection. Instance should accept train/valid datasets and select features.

property is_fitted

Check if selection pipeline is already fitted.

Return type bool

Returns True for fitted pipeline and False for not fitted.

property selected_features

Get selected features.

Return type List[str]

Returns List of selected feature names.

property in_features
Input features to the selector.

Raises exception if not fitted beforehand.

Return type List[str]

Returns List of input features.

property dropped_features
Features that were dropped.

Return type List[str]

Returns list of dropped features.

__init__(*features_pipeline=None*, *ml_algo=None*, *imp_estimator=None*, *fit_on_holdout=False*, ***kwargs*)
Create features selection pipeline.

Parameters

- **features_pipeline** (Optional[*FeaturesPipeline*]) – Composition of feature transforms.
- **ml_algo** (Union[*MLAlgo*, Tuple[*MLAlgo*, *ParamsTuner*], None]) – Tuple (MlAlgo, ParamsTuner).
- **imp_estimator** (Optional[*ImportanceEstimator*]) – Feature importance estimator.
- **fit_on_holdout** (bool) – If use the holdout iterator.
- ****kwargs** – Not used.

perform_selection(*train_valid*)
Select features from train-valid iterator.

Method is used to perform selection based on features pipeline and ml model. Should save `_selected_features` attribute in the end of working.

Raises NotImplementedError. –

fit(*train_valid*)
Selection pipeline fit.

Find features selection for given dataset based on features pipeline and ml model.

Parameters train_valid (*TrainValidIterator*) – Dataset iterator.

select(*dataset*)
Takes only selected features from giving dataset and creates new dataset.

Parameters dataset (*LAMLDataset*) – Dataset for feature selection.

Return type *LAMLDataset*

Returns New dataset with selected features only.

map_raw_feature_importances(*raw_importances*)
Calculate input feature importances. Calculated as sum of importances on different levels of pipeline.

Parameters raw_importances (Series) – Importances of output features.

get_features_score()
Get input feature importances.

Returns Series with importances in not ascending order.

8.2 Importance Based Selectors

ModelBasedImportanceEstimator	Base class for performing feature selection using model feature importances.
ImportanceCutoffSelector	Selector based on importance threshold.
NpPermutationImportanceEstimator	Permutation importance based estimator.
NpIterativeFeatureSelector	Select features sequentially using chunks to find the best combination of chunks.

8.2.1 ModelBasedImportanceEstimator

class lightautoml.pipelines.selection.importance_based.**ModelBasedImportanceEstimator**

Bases: *lightautoml.pipelines.selection.base.ImportanceEstimator*

Base class for performing feature selection using model feature importances.

fit(*train_valid=None*, *ml_algo=None*, *preds=None*)

Find the importances of features.

Parameters

- **train_valid** (Optional[*TrainValidIterator*]) – dataset iterator.
- **ml_algo** (Optional[~ImportanceEstimatedAlgo]) – ML algorithm used for importance estimation.
- **preds** (Optional[*LAMLDataset*]) – predicted target values.

8.2.2 ImportanceCutoffSelector

class lightautoml.pipelines.selection.importance_based.**ImportanceCutoffSelector**(*feature_pipeline*, *ml_algo*, *imp_estimator*, *fit_on_holdout=True*, *cutoff=0.0*)

Bases: *lightautoml.pipelines.selection.base.SelectionPipeline*

Selector based on importance threshold.

It is important that data which passed to `.fit` should be ok to fit *ml_algo* or preprocessing pipeline should be defined.

__init__(*feature_pipeline*, *ml_algo*, *imp_estimator*, *fit_on_holdout=True*, *cutoff=0.0*)

Parameters

- **feature_pipeline** (Optional[*FeaturesPipeline*]) – Composition of feature transforms.
- **ml_algo** (*MLAlgo*) – Tuple (MlAlgo, ParamsTuner).
- **imp_estimator** (*ImportanceEstimator*) – Feature importance estimator.

- **fit_on_holdout** (bool) – If use the holdout iterator.
- **cutoff** (float) – Threshold to cut-off features.

perform_selection(*train_valid=None*)
Select features based on cutoff value.

Parameters **train_valid** (Optional[*TrainValidIterator*]) – Not used.

8.2.3 NpPermutationImportanceEstimator

class lightautoml.pipelines.selection.permutation_importance_based.**NpPermutationImportanceEstimator**(*rand*
Bases: *lightautoml.pipelines.selection.base.ImportanceEstimator*

Permutation importance based estimator.

Importance calculate, using random permutation of items in single column for each feature.

__init__(*random_state=42*)

Parameters **random_state** (int) – seed for random generation of features permutation.

fit(*train_valid=None*, *ml_algo=None*, *preds=None*)
Find importances for each feature in dataset.

Parameters

- **train_valid** (Optional[*TrainValidIterator*]) – Initial dataset iterator.
- **ml_algo** (Optional[*MLAlgo*]) – Algorithm.
- **preds** (Optional[*LAMLDataset*]) – Predicted target values for validation dataset.

8.2.4 NpIterativeFeatureSelector

class lightautoml.pipelines.selection.permutation_importance_based.**NpIterativeFeatureSelector**(*feature_pipe*
ml_algo=No
imp_estimat
fit_on_holdo
fea-
*ture_group_*
max_feature

Bases: *lightautoml.pipelines.selection.base.SelectionPipeline*

Select features sequentially using chunks to find the best combination of chunks.

The general idea of this algorithm is to sequentially check groups of features ordered by feature importances and if the quality of the model becomes better, we select such group, if not - ignore group.

__init__(*feature_pipeline*, *ml_algo=None*, *imp_estimator=None*, *fit_on_holdout=True*, *feature_group_size=5*, *max_features_cnt_in_result=None*)

Parameters

- **feature_pipeline** (*FeaturesPipeline*) – Composition of feature transforms.
- **ml_algo** (Optional[*MLAlgo*]) – Tuple (MlAlgo, ParamsTuner).
- **imp_estimator** (Optional[*ImportanceEstimator*]) – Feature importance estimator.

- **fit_on_holdout** (bool) – If use the holdout iterator.
- **feature_group_size** (Optional[int]) – Chunk size.
- **max_features_cnt_in_result** (Optional[int]) – Lower bound of features after selection, if it is reached, it will stop.

perform_selection(*train_valid=None*)

Select features iteratively by checking model quality for current selected feats and new group.

Parameters **train_valid** (Optional[*TrainValidIterator*]) – Iterator for dataset.

8.3 Other Selectors

HighCorrRemoval	Selector to remove highly correlated features.

8.3.1 HighCorrRemoval

class lightautoml.pipelines.selection.linear_selector.**HighCorrRemoval**(*corr_co=0.98*, *subsample=100000*, *random_state=42*, ***kwargs*)

Bases: *lightautoml.pipelines.selection.base.SelectionPipeline*

Selector to remove highly correlated features.

Del totally correlated feats to speedup L1 regression models. For sparse data cosine will be used. It's not exact, but ok for remove very high correlations.

__init__(*corr_co=0.98*, *subsample=100000*, *random_state=42*, ***kwargs*)

Parameters

- **corr_co** (float) – Similarity threshold.
- **subsample** (Union[int, float]) – Number (int) of samples, or frac (float) from full dataset.
- **random_state** (int) – Random seed for subsample.
- ****kwargs** – Addtional parameters. Used for initialiation of parent class.

perform_selection(*train_valid*)

Select features to save in dataset during selection.

Method is used to perform selection based on features correlation. Should save `_selected_features` attribute in the end of working.

Parameters **train_valid** (Optional[*TrainValidIterator*]) – Classic cv-iterator.

CHAPTER

NINE

LIGHTAUTOML.PIPELINES.FEATURES

Pipelines for features generation.

9.1 Base Classes

FeaturesPipeline	Abstract class.
EmptyFeaturePipeline	Dummy feature pipeline - `.fit_transform` and transform do nothing.
TabularDataFeatures	Helper class contains basic features transformations for tabular data.

9.1.1 FeaturesPipeline

class lightautoml.pipelines.features.base.**FeaturesPipeline**(***kwargs*)

Bases: `object`

Abstract class.

Analyze train dataset and create composite transformer based on subset of features. Instance can be interpreted like Transformer (look for *LAMLTransformer*) with delayed initialization (based on dataset metadata) Main method, user should define in custom pipeline is `.create_pipeline`. For example, look at *LGBSimpleFeatures*. After FeaturePipeline instance is created, it is used like transformer with `.fit_transform` and `.transform` method.

property input_features

Names of input features of train data.

Return type `List[str]`

property output_features

List of feature names that produces _pipeline.

Return type `List[str]`

property used_features

List of feature names from original dataset that was used to produce output.

Return type `List[str]`

create_pipeline(*train*)

Analyse dataset and create composite transformer.

Parameters **train** (*LAMLDataset*) – Dataset with train data.

Return type *LAMLTransformer*

Returns Composite transformer (pipeline).

fit_transform(*train*)
Create pipeline and then fit on train data and then transform.

Parameters **train** (*LAMLDataset*) – Dataset with train data.

Return type *LAMLDataset*

Returns Dataset with new features.

transform(*test*)
Apply created pipeline to new data.

Parameters **test** (*LAMLDataset*) – Dataset with test data.

Return type *LAMLDataset*

Returns Dataset with new features.

9.1.2 EmptyFeaturePipeline

class lightautoml.pipelines.features.base.**EmptyFeaturePipeline**(***kwargs*)
Bases: *lightautoml.pipelines.features.base.FeaturesPipeline*

Dummy feature pipeline - `.fit_transform` and transform do nothing.

create_pipeline(*train*)
Create empty pipeline.

Parameters **train** (*LAMLDataset*) – Dataset with train data.

Return type *LAMLTransformer*

Returns Composite transformer (pipeline), that do nothing.

9.1.3 TabularDataFeatures

class lightautoml.pipelines.features.base.**TabularDataFeatures**(***kwargs*)
Bases: `object`

Helper class contains basic features transformations for tabular data.

This method can de shared by all tabular feature pipelines, to simplify `.create_automl` definition.

__init__(***kwargs*)
Set default parameters for tabular pipeline constructor.

Parameters ****kwargs** – Additional parameters.

static get_cols_for_datetime(*train*)
Get datetime columns to calculate features.

Parameters **train** (Union[*PandasDataset*, *NumpyDataset*]) – Dataset with train data.

Return type Tuple[List[str], List[str]]

Returns 2 list of features names - base dates and common dates.

get_datetime_diffs(*train*)
Difference for all datetimes with base date.

Parameters **train** (Union[*PandasDataset*, *NumpyDataset*]) – Dataset with train data.

Return type Optional[*LAMLTransformer*]

Returns Transformer or None if no required features.

get_datetime_seasons(*train*, *outp_role=None*)
Get season params from dates.

Parameters

- **train** (Union[*PandasDataset*, *NumpyDataset*]) – Dataset with train data.
- **outp_role** (Optional[*ColumnRole*]) – Role associated with output features.

Return type Optional[*LAMLTransformer*]

Returns Transformer or None if no required features.

static get_numeric_data(*train*, *feats_to_select=None*, *prob=None*)
Select numeric features.

Parameters

- **train** (Union[*PandasDataset*, *NumpyDataset*]) – Dataset with train data.
- **feats_to_select** (Optional[List[str]]) – Features to handle. If None - default filter.
- **prob** (Optional[bool]) – Probability flag.

Return type Optional[*LAMLTransformer*]

Returns Transformer.

static get_freq_encoding(*train*, *feats_to_select=None*)
Get frequency encoding part.

Parameters

- **train** (Union[*PandasDataset*, *NumpyDataset*]) – Dataset with train data.
- **feats_to_select** (Optional[List[str]]) – Features to handle. If None - default filter.

Return type Optional[*LAMLTransformer*]

Returns Transformer.

get_ordinal_encoding(*train*, *feats_to_select=None*)
Get order encoded part.

Parameters

- **train** (Union[*PandasDataset*, *NumpyDataset*]) – Dataset with train data.
- **feats_to_select** (Optional[List[str]]) – Features to handle. If None - default filter.

Return type Optional[*LAMLTransformer*]

Returns Transformer.

get_categorical_raw(*train*, *feats_to_select=None*)
Get label encoded categories data.

Parameters

- **train** (Union[*PandasDataset*, *NumpyDataset*]) – Dataset with train data.
- **feats_to_select** (Optional[List[str]]) – Features to handle. If None - default filter.

Return type Optional[*LAMLTransformer*]

Returns Transformer.

`get_target_encoder`(*train*)
Get target encoder func for dataset.

Parameters **train** (Union[*PandasDataset*, *NumpyDataset*]) – Dataset with train data.

Return type Optional[type]

Returns Class

`get_binned_data`(*train*, *feats_to_select=None*)
Get encoded quantiles of numeric features.

Parameters

- **train** (Union[*PandasDataset*, *NumpyDataset*]) – Dataset with train data.
- **feats_to_select** (Optional[List[str]]) – features to hanlde. If None - default filter.

Return type Optional[*LAMLTransformer*]

Returns Transformer.

`get_categorical_intersections`(*train*, *feats_to_select=None*)
Get transformer that implements categorical intersections.

Parameters

- **train** (Union[*PandasDataset*, *NumpyDataset*]) – Dataset with train data.
- **feats_to_select** (Optional[List[str]]) – features to handle. If None - default filter.

Return type Optional[*LAMLTransformer*]

Returns Transformer.

`get_uniques_cnt`(*train*, *feats*)
Get unique values cnt.

Parameters

- **train** (Union[*PandasDataset*, *NumpyDataset*]) – Dataset with train data.
- **feats** (List[str]) – Features names.

Return type Series

Returns Series.

`get_top_categories`(*train*, *top_n=5*)
Get top categories by importance.

If feature importance is not defined, or feats has same importance - sort it by unique values counts. In second case init param `ascending_by_cardinality` defines how - asc or desc.

Parameters

- **train** (Union[*PandasDataset*, *NumpyDataset*]) – Dataset with train data.
- **top_n** (int) – Number of top categories.

Return type List[str]

Returns List.

9.2 Feature Pipelines for Boosting Models

LGBSimpleFeatures	Creates simple pipeline for tree based models.
LGBAdvancedPipeline	Create advanced pipeline for trees based models.

9.2.1 LGBSimpleFeatures

class lightautoml.pipelines.features.lgb_pipeline.**LGBSimpleFeatures**(***kwargs*)

Bases: *lightautoml.pipelines.features.base.FeaturesPipeline*

Creates simple pipeline for tree based models.

Simple but is ok for select features. Numeric stay as is, Datetime transforms to numeric. Categorical label encoding. Maps input to output features exactly one-to-one.

create_pipeline(*train*)

Create tree pipeline.

Parameters **train** (Union[*PandasDataset*, *NumpyDataset*]) – Dataset with train features.

Return type *LAMLTransformer*

Returns Composite datetime, categorical, numeric transformer.

9.2.2 LGBAdvancedPipeline

class lightautoml.pipelines.features.lgb_pipeline.**LGBAdvancedPipeline**(*feats_imp=None*, *top_intersections=5*, *max_intersection_depth=3*, *subsample=None*, *multiclass_te_co=3*, *auto_unique_co=10*, *output_categories=False*, ***kwargs*)

Bases: *lightautoml.pipelines.features.base.FeaturesPipeline*, *lightautoml.pipelines.features.base.TabularDataFeatures*

Create advanced pipeline for trees based models.

Includes:

- Different cats and numbers handling according to role params.
- Dates handling - extracting seasons and create datediffs.
- Create categorical intersections.

__init__(*feats_imp=None*, *top_intersections=5*, *max_intersection_depth=3*, *subsample=None*, *multiclass_te_co=3*, *auto_unique_co=10*, *output_categories=False*, ***kwargs*)

Parameters

- **feats_imp** (Optional[*ImportanceEstimator*]) – Features importances mapping.
- **top_intersections** (int) – Max number of categories to generate intersections.
- **max_intersection_depth** (int) – Max depth of cat intersection.

- **subsample** (Union[float, int, None]) – Subsample to calc data statistics.
- **multiclass_te_co** (int) – Cutoff if use target encoding in cat handling on multiclass task if number of classes is high.
- **auto_unique_co** (int) – Switch to target encoding if high cardinality.

create_pipeline(*train*)

Create tree pipeline.

Parameters **train** (Union[*PandasDataset*, *NumpyDataset*]) – Dataset with train features.

Return type *LAMLTransformer*

Returns Transformer.

9.3 Feature Pipelines for Linear Models

LinearFeatures	Creates pipeline for linear models and nnets.

9.3.1 LinearFeatures

class lightautoml.pipelines.features.linear_pipeline.**LinearFeatures**(*feats_imp=None*, *top_intersections=5*, *max_bin_count=10*, *max_intersection_depth=3*, *subsample=None*, *sparse_ohe='auto'*, *auto_unique_co=50*, *output_categories=True*, *multiclass_te_co=3*, ***kwargs*)

Bases: *lightautoml.pipelines.features.base.FeaturesPipeline*, *lightautoml.pipelines.features.base.TabularDataFeatures*

Creates pipeline for linear models and nnets.

Includes:

- Create categorical intersections.
- OHE or embed idx encoding for categories.
- Other cats to numbers ways if defined in role params.
- Standartization and nan handling for numbers.
- Numbers discretization if needed.
- Dates handling.
- Handling probs (output of lower level models).

__init__(*feats_imp=None*, *top_intersections=5*, *max_bin_count=10*, *max_intersection_depth=3*, *subsample=None*, *sparse_ohe='auto'*, *auto_unique_co=50*, *output_categories=True*, *multiclass_te_co=3*, ***kwargs*)

Parameters

- **feats_imp** (Optional[*ImportanceEstimator*]) – Features importances mapping.
- **top_intersections** (int) – Max number of categories to generate intersections.
- **max_bin_count** (int) – Max number of bins to discretize numbers.
- **max_intersection_depth** (int) – Max depth of cat intersection.
- **subsample** (Union[float, int, None]) – Subsample to calc data statistics.
- **sparse_ohe** (Union[str, bool]) – Should we output sparse if ohe encoding was used during cat handling.
- **auto_unique_co** (int) – Switch to target encoding if high cardinality.
- **output_categories** (bool) – Output encoded categories or embed idxs.
- **multiclass_te_co** (int) – Cutoff if use target encoding in cat handling on multiclass task if number of classes is high.

create_pipeline(*train*)

Create linear pipeline.

Parameters **train** (Union[*PandasDataset*, *NumpyDataset*]) – Dataset with train features.

Return type *LAMLTransformer*

Returns Transformer.

9.4 Feature Pipelines for WhiteBox

WBFeatures	Simple WhiteBox pipeline.

9.4.1 WBFeatures

class lightautoml.pipelines.features.wb_pipeline.**WBFeatures**(***kwargs*)

Bases: *lightautoml.pipelines.features.base.FeaturesPipeline*, *lightautoml.pipelines.features.base.TabularDataFeatures*

Simple WhiteBox pipeline.

Just handles dates, other are handled inside WhiteBox.

create_pipeline(*train*)

Create pipeline for WhiteBox.

Parameters **train** (*PandasDataset*) – Dataset with train features.

Return type *LAMLTransformer*

Returns Transformer.

9.5 Image Feature Pipelines

ImageDataFeatures	Class contains basic features transformations for image data.
ImageSimpleFeatures	Class contains simple color histogram features for image data.
ImageAutoFeatures	Class contains efficient-net embeddings features for image data.

9.5.1 ImageDataFeatures

class lightautoml.pipelines.features.image_pipeline.**ImageDataFeatures**(***kwargs*)

Bases: object

Class contains basic features transformations for image data.

__init__(***kwargs*)

Set default parameters for image pipeline constructor.

Parameters ****kwargs** – Default parameters.

9.5.2 ImageSimpleFeatures

class lightautoml.pipelines.features.image_pipeline.**ImageSimpleFeatures**(***kwargs*)

Bases: *lightautoml.pipelines.features.base.FeaturesPipeline*, *lightautoml.pipelines.features.image_pipeline.ImageDataFeatures*

Class contains simple color histogram features for image data.

9.5.3 ImageAutoFeatures

class lightautoml.pipelines.features.image_pipeline.**ImageAutoFeatures**(***kwargs*)

Bases: *lightautoml.pipelines.features.base.FeaturesPipeline*, *lightautoml.pipelines.features.image_pipeline.ImageDataFeatures*

Class contains efficient-net embeddings features for image data.

9.6 Text Feature Pipelines

NLPDataFeatures	Class contains basic features transformations for text data.
TextAutoFeatures	Class contains embedding features for text data.
NLPTFiDFFeatures	Class contains tfidf features for text data.
TextBertFeatures	Features pipeline for BERT.

9.6.1 NLPDataFeatures

class lightautoml.pipelines.features.text_pipeline.**NLPDataFeatures**(***kwargs*)
Bases: object

Class contains basic features transformations for text data.

__init__(***kwargs*)
Set default parameters for nlp pipeline constructor.

Parameters ****kwargs** – default params.

9.6.2 TextAutoFeatures

class lightautoml.pipelines.features.text_pipeline.**TextAutoFeatures**(***kwargs*)
Bases: *lightautoml.pipelines.features.base.FeaturesPipeline*, *lightautoml.pipelines.features.text_pipeline.NLPDataFeatures*

Class contains embedding features for text data.

create_pipeline(*train*)
Create pipeline for textual data.

Parameters **train** (*LAMLDataset*) – Dataset with train features.

Return type *LAMLTransformer*

Returns Transformer.

9.6.3 NLPTFiDFFeatures

class lightautoml.pipelines.features.text_pipeline.**NLPTFiDFFeatures**(***kwargs*)
Bases: *lightautoml.pipelines.features.base.FeaturesPipeline*, *lightautoml.pipelines.features.text_pipeline.NLPDataFeatures*

Class contains tfidf features for text data.

create_pipeline(*train*)
Create pipeline for textual data.

Parameters **train** (*LAMLDataset*) – Dataset with train features.

Return type *LAMLTransformer*

Returns Transformer.

9.6.4 TextBertFeatures

class lightautoml.pipelines.features.text_pipeline.**TextBertFeatures**(***kwargs*)
Bases: *lightautoml.pipelines.features.base.FeaturesPipeline*, *lightautoml.pipelines.features.text_pipeline.NLPDataFeatures*

Features pipeline for BERT.

create_pipeline(*train*)
Create pipeline for BERT.

Parameters **train** (*LAMLDataset*) – Dataset with train data.

Return type *LAMLTransformer*

Returns Transformer.

CHAPTER

TEN

LIGHTAUTOML.PIPELINES.ML

Pipelines that merge together single model training steps.

10.1 Base Classes

MLPipeline	Single ML pipeline.

10.1.1 MLPipeline

class lightautoml.pipelines.ml.base.**MLPipeline**(*ml_algos*, *force_calc=True*, *pre_selection=None*, *features_pipeline=None*, *post_selection=None*)

Bases: object

Single ML pipeline.

Merge together stage of building ML model (every step, excluding model training, is optional):

- Pre selection: select features from input data. Performed by SelectionPipeline.
- Features generation: build new features from selected. Performed by FeaturesPipeline.
- Post selection: One more selection step - from created features. Performed by SelectionPipeline.
- Hyperparams optimization for one or multiple ML models. Performed by *ParamsTuner*.
- Train one or multiple ML models: Performed by *MLAlgo*. This step is the only required for at least 1 model.

__init__(*ml_algos*, *force_calc=True*, *pre_selection=None*, *features_pipeline=None*, *post_selection=None*)

Parameters

- **ml_algos** (Sequence[Union[*MLAlgo*, Tuple[*MLAlgo*, *ParamsTuner*]]]) – Sequence of MLAlgo's or Pair - (MlAlgo, ParamsTuner).
- **force_calc** (Union[bool, Sequence[bool]]) – Flag if single fold of ml_algo should be calculated anyway.
- **pre_selection** (Optional[*SelectionPipeline*]) – Initial feature selection. If None there is no initial selection.
- **features_pipeline** (Optional[*FeaturesPipeline*]) – Composition of feature transforms.

- **post_selection** (Optional[*SelectionPipeline*]) – Post feature selection. If None there is no post selection.

fit_predict(*train_valid*)
Fit on train/valid iterator and transform on validation part.

Parameters **train_valid** (*TrainValidIterator*) – Dataset iterator.

Return type *LAMLDataset*

Returns Dataset with predictions of all models.

predict(*dataset*)
Predict on new dataset.

Parameters **dataset** (*LAMLDataset*) – Dataset used for prediction.

Return type *LAMLDataset*

Returns Dataset with predictions of all trained models.

upd_model_names(*prefix*)
Update prefix pipeline models names.

Used to fit inside AutoML where multiple models with same names may be trained.

Parameters **prefix** (str) – New prefix name.

prune_algos(*idx*)
Prune model from pipeline.

Used to fit blender - some models may be excluded from final ensemble.

Parameters **idx** (Sequence[int]) – Selected algos.

10.2 Pipeline for Nested Cross-Validation

NestedTabularMLAlgo	Wrapper for MLAlgo to make it trainable over nested folds.
NestedTabularMLPipeline	Wrapper for MLPipeline to make it trainable over nested folds.

10.2.1 NestedTabularMLAlgo

class lightautoml.pipelines.ml.nested_ml_pipe.**NestedTabularMLAlgo**(*ml_algo*, *tuner=None*, *refit_tuner=False*, *cv=5*, *n_folds=None*)

Bases: *lightautoml.ml_algo.base.TabularMLAlgo*, *lightautoml.pipelines.selection.base.ImportanceEstimator*

Wrapper for MLAlgo to make it trainable over nested folds. Limitations - only for TabularMLAlgo.

property params
Parameters of ml_algo.

Return type dict

init_params_on_input(*train_valid_iterator*)
Init params depending on input data.

Return type dict

Returns dict with model hyperparameters.

fit_predict_single_fold(*train*, *valid*)
Implements training and prediction on single fold.

Parameters

- **train** (Union[*NumpyDataset*, *PandasDataset*]) – TabularDataset to train.
- **valid** (Union[*NumpyDataset*, *PandasDataset*]) – TabularDataset to validate.

Return type Tuple[Any, ndarray]

Returns Tuple (model, predicted_values).

fit(*train_valid*)
Just to be compatible with ImportanceEstimator.

Parameters **train_valid** (*TrainValidIterator*) – Classic cv iterator.

10.2.2 NestedTabularMLPipeline

class lightautoml.pipelines.ml.nested_ml_pipe.**NestedTabularMLPipeline**(*ml_algos*, *force_calc=True*, *pre_selection=None*, *features_pipeline=None*, *post_selection=None*, *cv=1*, *n_folds=None*, *inner_tune=False*, *refit_tuner=False*)

Bases: *lightautoml.pipelines.ml.base.MLPipeline*

Wrapper for MLPipeline to make it trainable over nested folds.

Limitations:

- Only for TabularMLAlgo
- Nested trained only MLAlgo. FeaturesPipelines and SelectionPipelines are trained as usual.

__init__(*ml_algos*, *force_calc=True*, *pre_selection=None*, *features_pipeline=None*, *post_selection=None*, *cv=1*, *n_folds=None*, *inner_tune=False*, *refit_tuner=False*)

Parameters

- **ml_algos** (Sequence[Union[*TabularMLAlgo*, Tuple[*TabularMLAlgo*, *ParamsTuner*]]]) – Sequence of MLAlgo's or Pair - (MlAlgo, ParamsTuner).
- **force_calc** (Union[bool, Sequence[bool]]) – Flag if single fold of MlAlgo should be calculated anyway.
- **pre_selection** (Optional[*SelectionPipeline*]) – Initial feature selection. If None there is no initial selection.
- **features_pipeline** (Optional[*FeaturesPipeline*]) – Composition of feature transforms.
- **post_selection** (Optional[*SelectionPipeline*]) – Post feature selection. If None there is no post selection.
- **cv** (int) – Nested folds cv split.

- **n_folds** (Optional[int]) – Limit of valid iterations from cv.
- **inner_tune** (bool) – Should we refit tuner each inner cv run or tune ones on outer cv.
- **refit_tuner** (bool) – Should we refit tuner each inner loop with inner_tune==True.

10.3 Pipeline for WhiteBox

WBPipeline	Special pipeline to handle WhiteBox model.

10.3.1 WBPipeline

class lightautoml.pipelines.ml.whitebox_ml_pipe.**WBPipeline**(*whitebox*)

Bases: *lightautoml.pipelines.ml.base.MLPipeline*

Special pipeline to handle WhiteBox model.

__init__(*whitebox*)

Create WhiteBox MLPipeline.

Parameters **whitebox** (Union[*WbMLAlgo*, Tuple[*WbMLAlgo*, *ParamsTuner*]]) – WhiteBox model.

fit_predict(*train_valid*)

Fit WhiteBox.

Parameters **train_valid** (*TrainValidIterator*) – Classic cv-iterator.

Return type *NumpyDataset*

Returns Dataset.

predict(*dataset*, *report=False*)

Predict WhiteBox.

Additional report param stands for WhiteBox report generation.

Parameters

- **dataset** (*PandasDataset*) – Dataset of text features.
- **report** (bool) – Flag if generate report.

Return type *NumpyDataset*

Returns Dataset.

CHAPTER

ELEVEN

LIGHTAUTOML.READER

Utils for reading, training and analysing data.

11.1 Readers

Reader	Abstract class for analyzing input data and creating inner *LAMLDataset* from raw data.
PandasToPandasReader	Reader to convert DataFrame to AutoML's *PandasDataset*.

11.1.1 Reader

class lightautoml.reader.base.**Reader**(*task*, **args*, ***kwargs*)

Bases: object

Abstract class for analyzing input data and creating inner *LAMLDataset* from raw data. Takes data in different formats as input, drop obviously useless features, estimates avaliable size and returns dataset.

__init__(*task*, **args*, ***kwargs*)

Parameters

- **task** (*Task*) – Task object
- ***args** – Not used.
- ****kwargs** – Not used.

property roles

Roles dict.

Return type Dict[str, ~RoleType]

property dropped_features

List of dropped features.

Return type List[str]

property used_features

List of used features.

Return type List[str]

property used_array_attrs
Dict of used array attributes.

Return type Dict[str, str]

fit_read(*train_data*, *features_names=None*, *roles=None*, ***kwargs*)
Abstract function to get dataset with initial feature selection.

read(*data*, *features_names*, ***kwargs*)
Abstract function to add validation columns.

upd_used_features(*add=None*, *remove=None*)
Updates the list of used features.

Parameters

- **add** (Optional[Sequence[str]]) – List of feature names to add or None.
- **remove** (Optional[Sequence[str]]) – List of feature names to remove or None.

classmethod from_reader(*reader*, ***kwargs*)
Create reader for new data type from existed.

Note - for now only Pandas reader exists, made for future plans.

Parameters

- **reader** (*Reader*) – Source reader.
- ****kwargs** – Ignored as in the class itself.

Return type *Reader*

Returns New reader.

cols_by_type(*col_type*)
Get roles names by it's type.

Parameters **col_type** (str) – Column type, for example 'Text'.

Return type List[str]

Returns Array with column names.

11.1.2 PandasToPandasReader

class lightautoml.reader.base.**PandasToPandasReader**(*task*, *samples=100000*, *max_nan_rate=0.999*, *max_constant_rate=0.999*, *cv=5*, *random_state=42*, *roles_params=None*, *n_jobs=4*, *advanced_roles=True*, *numeric_unique_rate=0.999*, *max_to_3rd_rate=1.1*, *binning_enc_rate=2*, *raw_decr_rate=1.1*, *max_score_rate=0.2*, *abs_score_val=0.04*, *drop_score_co=0.01*, ***kwargs*)

Bases: *lightautoml.reader.base.Reader*

Reader to convert DataFrame to AutoML's *PandasDataset*. Stages:

- Drop obviously useless features.
- Convert roles dict from user format to automl format.
- Simple role guess for features without input role.

- Create cv folds.
- Create initial PandasDataset.
- Optional: advanced guessing of role and handling types.

__init__(*task*, *samples=100000*, *max_nan_rate=0.999*, *max_constant_rate=0.999*, *cv=5*, *random_state=42*, *roles_params=None*, *n_jobs=4*, *advanced_roles=True*, *numeric_unique_rate=0.999*, *max_to_3rd_rate=1.1*, *binning_enc_rate=2*, *raw_decr_rate=1.1*, *max_score_rate=0.2*, *abs_score_val=0.04*, *drop_score_co=0.01*, ***kwargs*)

Parameters

- **task** (*Task*) – Task object.
- **samples** (Optional[int]) – Number of elements used when checking role type.
- **max_nan_rate** (float) – Maximum nan-rate.
- **max_constant_rate** (float) – Maximum constant rate.
- **cv** (int) – CV Folds.
- **random_state** (int) – Random seed.
- **roles_params** (Optional[dict]) – dict of params of features roles. Ex. {'numeric': {'dtype': np.float32}, 'datetime': {'date_format': '%Y-%m-%d'}} It's optional and commonly comes from config
- **n_jobs** (int) – Int number of processes.
- **advanced_roles** (bool) – Param of roles guess (experimental, do not change).
- **numeric_unqiue_rate** – Param of roles guess (experimental, do not change).
- **max_to_3rd_rate** (float) – Param of roles guess (experimental, do not change).
- **binning_enc_rate** (float) – Param of roles guess (experimental, do not change).
- **raw_decr_rate** (float) – Param of roles guess (experimental, do not change).
- **max_score_rate** (float) – Param of roles guess (experimental, do not change).
- **abs_score_val** (float) – Param of roles guess (experimental, do not change).
- **drop_score_co** (float) – Param of roles guess (experimental, do not change).
- ****kwargs** – For now not used.

fit_read(*train_data*, *features_names=None*, *roles=None*, ***kwargs*)

Get dataset with initial feature selection.

Parameters

- **train_data** (DataFrame) – Input data.
- **features_names** (Optional[Any]) – Ignored. Just to keep signature.
- **roles** (Optional[Dict[Union[str, ~RoleType, None], Sequence[str]]]) – Dict of features roles in format `{RoleX: ['feat0', 'feat1', ...], RoleY: 'TARGET', ...}`.
- ****kwargs** – Can be used for target/group/weights.

Return type *PandasDataset*

Returns Dataset with selected features.

read(*data*, *features_names=None*, *add_array_attrs=False*)
Read dataset with fitted metadata.

Parameters

- **data** (DataFrame) – Data.
- **features_names** (Optional[Any]) – Not used.
- **add_array_attrs** (bool) – Additional attributes, like target/group/weights/folds.

Return type *PandasDataset*

Returns Dataset with new columns.

advanced_roles_guess(*dataset*, *manual_roles=None*)
Advanced roles guess over user's definition and reader's simple guessing.

Strategy - compute feature's NormalizedGini for different encoding ways and calc stats over results. Role is inferred by comparing performance stats with manual rules. Rule params are params of roles guess in init. Defaults are ok in general case.

Parameters

- **dataset** (*PandasDataset*) – Input PandasDataset.
- **manual_roles** (Optional[Dict[str, ~RoleType]]) – Dict of user defined roles.

Return type Dict[str, ~RoleType]

Returns Dict.

11.2 Tabular Batch Generators

11.2.1 Batch Handler Classes

—

11.2.2 Data Read Functions

—

CHAPTER

TWELVE

LIGHTAUTOML.REPORT

Report generators and templates.

—

CHAPTER

THIRTEEN

LIGHTAUTOML.TASKS

13.1 Task Class

Task	Specify task (binary classification, multiclass classification, regression), metrics, losses.

13.1.1 Task

class lightautoml.tasks.base.**Task**(*name*, *loss=None*, *loss_params=None*, *metric=None*, *metric_params=None*, *greater_is_better=None*)

Bases: object

Specify task (binary classification, multiclass classification, regression), metrics, losses.

property name

Name of task.

Return type str

__init__(*name*, *loss=None*, *loss_params=None*, *metric=None*, *metric_params=None*, *greater_is_better=None*)

Parameters

- **name** (str) – Task name.
- **loss** (Union[dict, str, None]) – Objective function or dict of functions.
- **loss_params** (Optional[Dict]) – Additional loss parameters, if dict there is no presence check for loss_params.
- **metric** (Union[str, Callable, None]) – String name or callable.
- **metric_params** (Optional[Dict]) – Additional metric parameters.
- **greater_is_better** (Optional[bool]) – Whether or not higher value is better.

Note: There is 3 different task types:

- *'binary'* - for binary classification.
- *'reg'* - for regression.
- *'multiclass'* - for multiclass classification.

Avaliable losses for binary task:

- *'logloss'* - (uses by default) Standard logistic loss.

Avaliable losses for regression task:

- *'mse'* - (uses by default) Mean Squared Error.
- *'mae'* - Mean Absolute Error.
- *'mape'* - Mean Absolute Percentage Error.
- *'rmsle'* - Root Mean Squared Log Error.
- *'huber'* - Huber loss, reqired params: `a` - threshold between MAE and MSE losses.
- *'fair'* - Fair loss, required params: `c` - sets smoothness.
- *'quantile'* - Quantile loss, required params: `q` - sets quantile.

Avaliable losses for multi-classification task:

- *'crossentropy'* - (uses by default) Standard crossentropy function.
- *'f1'* - Optimizes F1-Macro Score, now avaliable for LightGBM and NN models. Here we implicitly assume that the prediction lies not in the set `{0, 1}`, but in the interval `[0, 1]`.

Available metrics for binary task:

- *'auc'* - (uses by default) ROC-AUC score.
- *'accuracy'* - Accuracy score (uses argmax prediction).
- *'logloss'* - Standard logistic loss.

Avaliable metrics for regression task:

- *'mse'* - (uses by default) Mean Squared Error.
- *'mae'* - Mean Absolute Error.
- *'mape'* - Mean Absolute Percentage Error.
- *'rmsle'* - Root Mean Squared Log Error.
- *'huber'* - Huber loss, reqired params: `a` - threshold between MAE and MSE losses.
- *'fair'* - Fair loss, required params: `c` - sets smoothness.
- *'quantile'* - Quantile loss, required params: `q` - sets quantile.

Avaliable metrics for multi-classification task:

- *'crossentropy'* - (uses by default) Standard cross-entropy loss.
- *'auc'* - ROC-AUC of each class against the rest.
- *'auc_mu'* - AUC-Mu. Multi-class extension of standard AUC for binary classification. In short, mean of n_classes * (n_classes - 1) / 2 binary AUCs. More info on http://proceedings.mlr.press/v97/kleiman19a/kleiman19a.pdf

---

Example

```
>>> task = Task('binary', metric='auc')
```

---

get_dataset_metric()
Create metric for dataset.

Get metric that is called on dataset.

Return type LAMLMetric

Returns Metric in scikit-learn compatible format.

13.2 Common Metrics

13.2.1 Classes

F1Factory	Wrapper for f1_score function.
BestClassBinaryWrapper	Metric wrapper to get best class prediction instead of probs.
BestClassMulticlassWrapper	Metric wrapper to get best class prediction instead of probs for multiclass.

F1Factory

class lightautoml.tasks.common_metric.**F1Factory**(*average='micro'*)
Bases: object

Wrapper for f1_score function.

__init__(*average='micro'*)

Parameters **average** (str) – Averaging type ('micro', 'macro', 'weighted').

BestClassBinaryWrapper

class lightautoml.tasks.common_metric.**BestClassBinaryWrapper**(*func*)
Bases: object

Metric wrapper to get best class prediction instead of probs.

There is cut-off for prediction by 0.5.

__init__(*func*)

Parameters **func** (Callable) – Metric function. Function format: func(y_pred, y_true, weights, **kwargs).

BestClassMulticlassWrapper

class lightautoml.tasks.common_metric.**BestClassMulticlassWrapper**(*func*)
Bases: object

Metric wrapper to get best class prediction instead of probs for multiclass.

Prediction provides by argmax.

__init__(*func*)

Parameters **func** – Metric function. Function format: func(y_pred, y_true, weights, **kwargs)

13.2.2 Functions

mean_quantile_error	Computes Mean Quantile Error.
mean_huber_error	Computes Mean Huber Error.
mean_fair_error	Computes Mean Fair Error.
mean_absolute_percentage_error	Computes Mean Absolute Percentage error.
roc_auc_ovr	ROC-AUC One-Versus-Rest.
rmsle	Root mean squared log error.
auc_mu	Compute multi-class metric AUC-Mu.

mean_quantile_error

lightautoml.tasks.common_metric.**mean_quantile_error**(*y_true*, *y_pred*, *sample_weight=None*, *q=0.9*)
Computes Mean Quantile Error.

Parameters

- **y_true** (ndarray) – True target values.
- **y_pred** (ndarray) – Predicted target values.
- **sample_weight** (Optional[ndarray]) – Specify weighted mean.
- **q** (float) – Metric coefficient.

Return type float

Returns metric value.

mean_huber_error

lightautoml.tasks.common_metric.**mean_huber_error**(*y_true*, *y_pred*, *sample_weight=None*, *a=0.9*)
Computes Mean Huber Error.

Parameters

- **y_true** (ndarray) – True target values.
- **y_pred** (ndarray) – Predicted target values.
- **sample_weight** (Optional[ndarray]) – Specify weighted mean.
- **a** (float) – Metric coefficient.

Return type float

Returns Metric value.

mean_fair_error

lightautoml.tasks.common_metric.**mean_fair_error**(*y_true*, *y_pred*, *sample_weight=None*, *c=0.9*)
Computes Mean Fair Error.

Parameters

- **y_true** (ndarray) – True target values.
- **y_pred** (ndarray) – Predicted target values.
- **sample_weight** (Optional[ndarray]) – Specify weighted mean.
- **c** (float) – Metric coefficient.

Return type float

Returns Metric value.

mean_absolute_percentage_error

lightautoml.tasks.common_metric.**mean_absolute_percentage_error**(*y_true*, *y_pred*, *sample_weight=None*)
Computes Mean Absolute Percentage error.

Parameters

- **y_true** (ndarray) – True target values.
- **y_pred** (ndarray) – Predicted target values.
- **sample_weight** (Optional[ndarray]) – Specify weighted mean.

Return type float

Returns Metric value.

roc_auc_ovr

lightautoml.tasks.common_metric.**roc_auc_ovr**(*y_true*, *y_pred*, *sample_weight=None*)
ROC-AUC One-Versus-Rest.

Parameters

- **y_true** (ndarray) – True target values.
- **y_pred** (ndarray) – Predicted target values.
- **sample_weight** (Optional[ndarray]) – Weights of samples.

Returns Metric values.

rmsle

lightautoml.tasks.common_metric.**rmsle**(*y_true*, *y_pred*, *sample_weight=None*)
Root mean squared log error.

Parameters

- **y_true** (ndarray) – True target values.
- **y_pred** (ndarray) – Predicted target values.
- **sample_weight** (Optional[ndarray]) – Weights of samples.

Returns Metric values.

auc_mu

lightautoml.tasks.common_metric.**auc_mu**(*y_true*, *y_pred*, *sample_weight=None*, *class_weights=None*)
Compute multi-class metric AUC-Mu.

We assume that confusion matrix full of ones, except diagonal elements. All diagonal elements are zeroes. By default, for averaging between classes scores we use simple mean.

Parameters

- **y_true** (ndarray) – True target values.
- **y_pred** (ndarray) – Predicted target values.
- **sample_weight** (Optional[ndarray]) – Not used.
- **class_weights** (Optional[ndarray]) – The between classes weight matrix. If None, the standard mean will be used. It is expected to be a lower triangular matrix (diagonal is also full of zeroes). In position (i, j), i > j, there is a partial positive score between i-th and j-th classes. All elements must sum up to 1.

Return type float

Returns Metric value.

Note: Code was refactored from https://github.com/kleimanr/auc_mu/blob/master/auc_mu.py

---

CHAPTER

FOURTEEN

LIGHTAUTOML.TASKS.LOSSES

Wrappers of loss and metric functions for different machine learning algorithms.

14.1 Base Classes

MetricFunc	Wrapper for metric.
Loss	Loss function with target transformation.

14.1.1 MetricFunc

class lightautoml.tasks.losses.base.**MetricFunc**(*metric_func*, *m*, *bw_func*)

Bases: object

Wrapper for metric.

__init__(*metric_func*, *m*, *bw_func*)

Parameters

- **metric_func** – Callable metric function.
- **m** – Multiplier for metric value.
- **bw_func** – Backward function.

14.1.2 Loss

class lightautoml.tasks.losses.base.**Loss**

Bases: object

Loss function with target transformation.

property fw_func

Forward transformation for target values and item weights.

Returns Callable transformation.

property bw_func

Backward transformation for predicted values.

Returns Callable transformation.

metric_wrapper(*metric_func*, *greater_is_better*, *metric_params=None*)
Customize metric.

Parameters

- **metric_func** (Callable) – Callable metric.
- **greater_is_better** (Optional[bool]) – Whether or not higher value is better.
- **metric_params** (Optional[Dict]) – Additional metric parameters.

Return type Callable

Returns Callable metric.

set_callback_metric(*metric*, *greater_is_better=None*, *metric_params=None*, *task_name=None*)
Callback metric setter.

Parameters

- **metric** (Union[str, Callable]) – Callback metric
- **greater_is_better** (Optional[bool]) – Whether or not higher value is better.
- **metric_params** (Optional[Dict]) – Additional metric parameters.
- **task_name** (Optional[Dict]) – Name of task.

Note: Value of task_name should be one of following options:

- *'binary'*
- *'reg'*
- *'multiclass'*

14.2 Wrappers for LightGBM

14.2.1 Classes

LGBFunc	Wrapper of metric function for LightGBM.
LGBLoss	Loss used for LightGBM.

LGBFunc

class lightautoml.tasks.losses.lgb.**LGBFunc**(*metric_func*, *greater_is_better*, *bw_func*)
Bases: object

Wrapper of metric function for LightGBM.

LGBLoss

class lightautoml.tasks.losses.lgb.**LGBLoss**(*loss*, *loss_params=None*, *fw_func=None*, *bw_func=None*)
Bases: *lightautoml.tasks.losses.base.Loss*

Loss used for LightGBM.

__init__(*loss*, *loss_params=None*, *fw_func=None*, *bw_func=None*)

Parameters

- **loss** (Union[str, Callable]) – Objective to optimize.
- **loss_params** (Optional[Dict]) – additional loss parameters. Format like in lightautoml.tasks.custom_metrics.
- **fw_func** (Optional[Callable]) – forward transformation. Used for transformation of target and item weights.
- **bw_func** (Optional[Callable]) – backward transformation. Used for predict values transformation.

Note: Loss can be one of the types:

- Str: one of default losses ('auc', 'mse', 'mae', 'logloss', 'accuray', 'r2', 'rmsle', 'mape', 'quantile', 'huber', 'fair') or another lightgbm objective.
- Callable: custom lightgbm style objective.

---

metric_wrapper(*metric_func*, *greater_is_better*, *metric_params=None*)
Customize metric.

Parameters

- **metric_func** (Callable) – Callable metric.
- **greater_is_better** (Optional[bool]) – Whether or not higher value is better.
- **metric_params** (Optional[Dict]) – Additional metric parameters.

Return type Callable

Returns Callable metric, that returns ('Opt metric', value, greater_is_better).

set_callback_metric(*metric*, *greater_is_better=None*, *metric_params=None*, *task_name=None*)
Callback metric setter.

Parameters

- **metric** (Union[str, Callable]) – Callback metric.
- **greater_is_better** (Optional[bool]) – Whether or not higher value is better.
- **metric_params** (Optional[Dict]) – Additional metric parameters.
- **task_name** (Optional[str]) – Name of task.

Note: Value of task_name should be one of following options:

- *'binary'*
- *'reg'*

- *'multiclass'*

14.2.2 Functions

softmax_ax1	Softmax columnwise.
lgb_f1_loss_multiclass	Custom loss for optimizing f1.

softmax_ax1

lightautoml.tasks.losses.lgb_custom.**softmax_ax1**(*x*)
Softmax columnwise.

Parameters **x** (ndarray) – input.

Return type ndarray

Returns softmax values.

lgb_f1_loss_multiclass

lightautoml.tasks.losses.lgb_custom.**lgb_f1_loss_multiclass**(*preds*, *train_data*, *clip=1e-05*)
Custom loss for optimizing f1.

Parameters

- **preds** (ndarray) – Predctions.
- **train_data** (Dataset) – Dataset in LightGBM format.
- **clip** (float) – Clump constant.

Return type Tuple[ndarray, ndarray]

Returns Gradient, hessian.

14.3 Wrappers for CatBoost

14.3.1 Classes

CBLoss	Loss used for CatBoost.
CBCustomMetric	Metric wrapper class for CatBoost.
CBRegressionMetric	Regression metric wrapper for CatBoost.
CBClassificationMetric	Classification metric wrapper for CatBoost.
CBMulticlassMetric	Multiclassification metric wrapper for CatBoost.

CBLoss

class lightautoml.tasks.losses.cb.**CBLoss**(*loss*, *loss_params=None*, *fw_func=None*, *bw_func=None*)
Bases: *lightautoml.tasks.losses.base.Loss*

Loss used for CatBoost.

__init__(*loss*, *loss_params=None*, *fw_func=None*, *bw_func=None*)

Parameters

- **loss** (Union[str, Callable]) – String with one of default losses.
- **loss_params** (Optional[Dict]) – additional loss parameters. Format like in lightautoml.tasks.custom_metrics.
- **fw_func** (Optional[Callable]) – Forward transformation. Used for transformation of target and item weights.
- **bw_func** (Optional[Callable]) – Backward transformation. Used for predict values transformation.

set_callback_metric(*metric*, *greater_is_better=None*, *metric_params=None*, *task_name=None*)
Callback metric setter.

Parameters

- **metric** (Union[str, Callable]) – Callback metric.
- **greater_is_better** (Optional[bool]) – Whether or not higher value is better.
- **metric_params** (Optional[Dict]) – Additional metric parameters.
- **task_name** (Optional[str]) – Name of task. For now it omitted.

CBCustomMetric

class lightautoml.tasks.losses.cb_custom.**CBCustomMetric**(*metric*, *greater_is_better=True*, *bw_func=None*)
Bases: object

Metric wrapper class for CatBoost.

__init__(*metric*, *greater_is_better=True*, *bw_func=None*)

Parameters

- **metric** (Callable) – Callable metric.
- **greater_is_better** (bool) – Bool with metric direction.

CBRegressionMetric

class lightautoml.tasks.losses.cb_custom.**CBRegressionMetric**(*metric*, *greater_is_better=True*, *bw_func=None*)

Bases: *lightautoml.tasks.losses.cb_custom.CBCustomMetric*

Regression metric wrapper for CatBoost.

CBClassificationMetric

class lightautoml.tasks.losses.cb_custom.**CBClassificationMetric**(*metric*, *greater_is_better*, *bw_func=None*, *use_proba=True*)

Bases: *lightautoml.tasks.losses.cb_custom.CBCustomMetric*

Classification metric wrapper for CatBoost.

CBMulticlassMetric

class lightautoml.tasks.losses.cb_custom.**CBMulticlassMetric**(*metric*, *greater_is_better*, *bw_func=None*, *use_proba=True*)

Bases: *lightautoml.tasks.losses.cb_custom.CBCustomMetric*

Multiclassification metric wrapper for CatBoost.

14.3.2 Functions

cb_str_loss_wrapper	CatBoost loss name wrapper, if it has keyword args.

cb_str_loss_wrapper

lightautoml.tasks.losses.cb.**cb_str_loss_wrapper**(*name*, ***params*)

CatBoost loss name wrapper, if it has keyword args.

Parameters

- **name** (str) – One of CatBoost loss names.
- ****params** – Additional parameters.

Returns Wrapped CatBoost loss name.

14.4 Wrappers for Sklearn

14.4.1 Classes

SKLoss	Loss used for scikit-learn.

SKLoss

class lightautoml.tasks.losses.sklearn.**SKLoss**(*loss*, *loss_params=None*, *fw_func=None*, *bw_func=None*)

Bases: *lightautoml.tasks.losses.base.Loss*

Loss used for scikit-learn.

__init__(*loss*, *loss_params=None*, *fw_func=None*, *bw_func=None*)

Parameters

- **loss** (str) – One of default loss function. Valid are: 'logloss', 'mse', 'crossentropy', 'rmsle'.
- **loss_params** (Optional[Dict]) – Addtional loss parameters.
- **fw_func** (Optional[Callable]) – Forward transformation. Used for transformation of target and item weights.
- **bw_func** (Optional[Callable]) – backward transformation. Used for predict values transformation.

set_callback_metric(*metric*, *greater_is_better=None*, *metric_params=None*, *task_name=None*)

Callback metric setter.

Uses default callback of parent class *Loss*.

Parameters

- **metric** (Union[str, Callable]) – Callback metric.
- **greater_is_better** (Optional[bool]) – Whether or not higher value is better.
- **metric_params** (Optional[Dict]) – Additional metric parameters.
- **task_name** (Optional[str]) – Name of task.

14.5 Wrappers for Torch

14.5.1 Classes

TorchLossWrapper	Customize PyTorch-based loss.
TORCHLoss	Loss used for PyTorch.

TorchLossWrapper

class lightautoml.tasks.losses.torch.**TorchLossWrapper**(*func*, *flatten=False*, *log=False*, ***kwargs*)

Bases: torch.nn.Module

Customize PyTorch-based loss.

Parameters

- **func** (Callable) – loss to customize. Example: *torch.nn.MSELoss*.
- ****kwargs** – additional parameters.

Returns callable loss, uses format (y_true, y_pred, sample_weight).

TORCHLoss

class lightautoml.tasks.losses.torch.**TORCHLoss**(*loss*, *loss_params=None*)
Bases: *lightautoml.tasks.losses.base.Loss*

Loss used for PyTorch.

__init__(*loss*, *loss_params=None*)

Parameters

- **loss** (Union[str, Callable]) – name or callable objective function.
- **loss_params** (Optional[Dict]) – additional loss parameters.

14.5.2 Functions

torch_rmsle	Computes Root Mean Squared Logarithmic Error.
torch_quantile	Computes Mean Quantile Error.
torch_fair	Computes Mean Fair Error.
torch_huber	Computes Mean Huber Error.
torch_f1	Computes F1 macro.
torch_mape	Computes Mean Absolute Percentage Error.

torch_rmsle

lightautoml.tasks.losses.torch.**torch_rmsle**(*y_true*, *y_pred*, *sample_weight=None*)
Computes Root Mean Squared Logarithmic Error.

Parameters

- **y_true** (Tensor) – true target values.
- **y_pred** (Tensor) – predicted target values.
- **sample_weight** (Optional[Tensor]) – specify weighted mean.

Returns metric value.

torch_quantile

lightautoml.tasks.losses.torch.**torch_quantile**(*y_true*, *y_pred*, *sample_weight=None*, *q=0.9*)
Computes Mean Quantile Error.

Parameters

- **y_true** (Tensor) – true target values.
- **y_pred** (Tensor) – predicted target values.
- **sample_weight** (Optional[Tensor]) – specify weighted mean.
- **q** (float) – metric coefficient.

Returns metric value.

torch_fair

lightautoml.tasks.losses.torch.**torch_fair**(*y_true*, *y_pred*, *sample_weight=None*, *c=0.9*)
Computes Mean Fair Error.

Parameters

- **y_true** (Tensor) – true target values.
- **y_pred** (Tensor) – predicted target values.
- **sample_weight** (Optional[Tensor]) – specify weighted mean.
- **c** (float) – metric coefficient.

Returns metric value.

torch_huber

lightautoml.tasks.losses.torch.**torch_huber**(*y_true*, *y_pred*, *sample_weight=None*, *a=0.9*)
Computes Mean Huber Error.

Parameters

- **y_true** (Tensor) – true target values.
- **y_pred** (Tensor) – predicted target values.
- **sample_weight** (Optional[Tensor]) – specify weighted mean.
- **a** (float) – metric coefficient.

Returns metric value.

torch_f1

lightautoml.tasks.losses.torch.**torch_f1**(*y_true*, *y_pred*, *sample_weight=None*)
Computes F1 macro.

Parameters

- **y_true** (Tensor) – true target values.
- **y_pred** (Tensor) – predicted target values.
- **sample_weight** (Optional[Tensor]) – specify weighted mean.

Returns metric value.

torch_mape

lightautoml.tasks.losses.torch.**torch_mape**(*y_true*, *y_pred*, *sample_weight=None*)
Computes Mean Absolute Percentage Error.

Parameters

- **y_true** (Tensor) – true target values.
- **y_pred** (Tensor) – predicted target values.
- **sample_weight** (Optional[Tensor]) – specify weighted mean.

Returns metric value.

CHAPTER

FIFTEEN

LIGHTAUTOML.TEXT

Provides an internal interface for working with text features.

15.1 Sentence Embedders

DLTransformer	Deep Learning based sentence embeddings.
BOREP	Class to compute Bag of Random Embedding Projections sentence embeddings from words embeddings.
RandomLSTM	Class to compute Random LSTM sentence embeddings from words embeddings.
BertEmbedder	Class to compute HuggingFace transformers words or sentence embeddings.
WeightedAverageTransformer	Weighted average of word embeddings.

15.1.1 DLTransformer

class lightautoml.text.dl_transformers.**DLTransformer**(*model*, *model_params*, *dataset*, *dataset_params*, *loader_params*, *device='cuda'*, *random_state=42*, *embedding_model=None*, *embedding_model_params=None*, *multigpu=False*, *verbose=False*)

Bases: sklearn.base.TransformerMixin

Deep Learning based sentence embeddings.

__init__(*model*, *model_params*, *dataset*, *dataset_params*, *loader_params*, *device='cuda'*, *random_state=42*, *embedding_model=None*, *embedding_model_params=None*, *multigpu=False*, *verbose=False*)

Class to compute sentence embeddings from words embeddings.

Parameters

- **model** – Torch model for aggregation word embeddings into sentence embedding.
- **model_params** – Dict with model parameters.
- **dataset** – Torch dataset.
- **dataset_params** – Dict with dataset params.
- **loader_params** – Dict with params for torch dataloader.
- **device** – String with torch device type or device ids. I.e: '0,2'.

- **random_state** – Determines random number generation.
- **embedding_model** – Torch word embedding model, if dataset do not return embeddings.
- **embedding_model_params** – Dict with embedding model params.
- **multigpu** – Use data parallel for multiple GPU.
- **verbose** – Show tqdm progress bar.

get_name()
Module name.

Return type str

Returns String with module name.

get_out_shape()
Output shape.

Return type int

Returns Int with module output shape.

15.1.2 BOREP

class lightautoml.text.dl_transformers.**BOREP**(*embed_size=300*, *proj_size=300*, *pooling='mean'*, *max_length=200*, *init='orthogonal'*, *pos_encoding=False*, ***kwargs*)

Bases: torch.nn.Module

Class to compute Bag of Random Embedding Projections sentence embeddings from words embeddings.

__init__(*embed_size=300*, *proj_size=300*, *pooling='mean'*, *max_length=200*, *init='orthogonal'*, *pos_encoding=False*, ***kwargs*)
Bag of Random Embedding Projections sentence embeddings.

Parameters

- **embed_size** (int) – Size of word embeddings.
- **proj_size** (int) – Size of output sentence embedding.
- **pooling** (str) – Pooling type.
- **max_length** (int) – Maximum length of sentence.
- **init** (str) – Type of weight initialization.
- **pos_encoding** (bool) – Add positional embedding.
- ****kwargs** – Ignored params.

Note: There are several pooling types:

- *'max'*: Maximum on seq_len dimension for non masked inputs.
- *'mean'*: Mean on seq_len dimension for non masked inputs.
- *'sum'*: Sum on seq_len dimension for non masked inputs.

For init parameter there are several options:

- *'orthogonal'*: Orthogonal init.
- *'normal'*: Normal with std 0.1.

- *'uniform'*: Uniform from -0.1 to 0.1.
- *'kaiming'*: Uniform kaiming init.
- *'xavier'*: Uniform xavier init.

---

get_out_shape()
Output shape.

Return type int

Returns Int with module output shape.

get_name()
Module name.

Return type str

Returns String with module name.

15.1.3 RandomLSTM

class lightautoml.text.dl_transformers.**RandomLSTM**(*embed_size=300*, *hidden_size=256*, *pooling='mean'*, *num_layers=1*, ***kwargs*)

Bases: torch.nn.Module

Class to compute Random LSTM sentence embeddings from words embeddings.

__init__(*embed_size=300*, *hidden_size=256*, *pooling='mean'*, *num_layers=1*, ***kwargs*)
Random LSTM sentence embeddings.

Parameters

- **embed_size** (int) – Size of word embeddings.
- **hidden_size** (int) – Size of hidden dimensions of LSTM.
- **pooling** (str) – Pooling type.
- **num_layers** (int) – Number of lstm layers.
- ****kwargs** – Ignored params.

Note: There are several pooling types:

- *'max'*: Maximum on seq_len dimension for non masked inputs.
- *'mean'*: Mean on seq_len dimension for non masked inputs.
- *'sum'*: Sum on seq_len dimension for non masked inputs.

---

get_out_shape()
Output shape.

Return type int

Returns Int with module output shape.

get_name()
Module name.

Return type str

Returns String with module name.

15.1.4 BertEmbedder

class lightautoml.text.dl_transformers.**BertEmbedder**(*model_name*, *pooling='none'*, ***kwargs*)

Bases: torch.nn.Module

Class to compute HuggingFace transformers words or sentence embeddings.

__init__(*model_name*, *pooling='none'*, ***kwargs*)

Bert sentence or word embeddings.

Parameters

- **model_name** (str) – Name of transformers model.
- **pooling** (str) – Pooling type.
- ****kwargs** – Ignored params.

Note: There are several pooling types:

- *'cls'*: Use CLS token for sentence embedding from last hidden state.
- *'max'*: Maximum on seq_len dimension for non masked inputs from last hidden state.
- *'mean'*: Mean on seq_len dimension for non masked inputs from last hidden state.
- *'sum'*: Sum on seq_len dimension for non masked inputs from last hidden state.
- *'none'*: Don't use pooling (for RandomLSTM pooling strategy).

---

freeze()

Freeze module parameters.

get_name()

Module name.

Return type str

Returns String with module name.

get_out_shape()

Output shape.

Return type int

Returns Int with module output shape.

15.1.5 WeightedAverageTransformer

class lightautoml.text.weighted_average_transformer.**WeightedAverageTransformer**(*embedding_model*, *embed_size*, *weight_type='idf'*, *use_svd=True*, *alpha=0.001*, *verbose=False*, ***kwargs*)

Bases: sklearn.base.TransformerMixin

Weighted average of word embeddings.

__init__(*embedding_model*, *embed_size*, *weight_type='idf'*, *use_svd=True*, *alpha=0.001*, *verbose=False*, ***kwargs*)

Calculate sentence embedding as weighted average of word embeddings.

Parameters

- **embedding_model** (Dict) – word2vec, fasttext, etc. Should have dict interface {<word>: <embedding>}.
- **embed_size** (int) – Size of embedding.
- **weight_type** (str) – 'idf' for idf weights, 'sif' for smoothed inverse frequency weights, '1' for all weights are equal.
- **use_svd** (bool) – Substract projection onto first singular vector.
- **alpha** (int) – Param for sif weights.
- **verbose** (bool) – Add prints.
- ****kwargs** – Unused arguments.

get_name()

Module name.

Return type str

Returns string with module name.

get_out_shape()

Output shape.

Return type int

Returns Int with module output shape.

reset_statistic()

Reset module statistics.

get_statistic()

Get module statistics.

15.2 Torch Datasets for Text

BertDataset	Dataset class with transformers tokenization.
EmbedDataset	Dataset class for extracting word embeddings.

15.2.1 BertDataset

class lightautoml.text.embed_dataset.**BertDataset**(*sentences*, *max_length*, *model_name*, ***kwargs*)

Bases: object

Dataset class with transformers tokenization.

__init__(*sentences*, *max_length*, *model_name*, ***kwargs*)

Class for preparing transformers input.

Parameters

- **sentences** (Sequence[str]) – List of tokenized sentences.
- **max_length** (int) – Max sentence length.
- **model_name** (str) – Name of transformer model.

15.2.2 EmbedDataset

class lightautoml.text.embed_dataset.**EmbedDataset**(*sentences*, *embedding_model*, *max_length*, *embed_size*, ***kwargs*)

Bases: object

Dataset class for extracting word embeddings.

__init__(*sentences*, *embedding_model*, *max_length*, *embed_size*, ***kwargs*)

Class for transforming list of tokens to dict of embeddings and sentence length.

Parameters

- **sentences** (Sequence[str]) – List of tokenized sentences.
- **embedding_model** (Dict) – word2vec, fasstext, etc. Should have dict interface {<word>: <embedding>}.
- **max_length** (int) – Max sentence length.
- **embed_size** (int) – Size of embedding.
- ****kwargs** – Not used.

15.3 Tokenizers

BaseTokenizer	Base class for tokenizer method.
SimpleRuTokenizer	Russian tokenizer.
SimpleEnTokenizer	English tokenizer.

15.3.1 BaseTokenizer

class lightautoml.text.tokenizer.**BaseTokenizer**(*n_jobs=4*, *to_string=True*, ***kwargs*)

Bases: object

Base class for tokenizer method.

__init__(*n_jobs=4*, *to_string=True*, ***kwargs*)

Tokenization with simple text cleaning and preprocessing.

Parameters

- **n_jobs** (int) – Number of threads for multiprocessing.
- **to_string** (bool) – Return string or list of tokens.

preprocess_sentence(*snt*)

Preprocess sentence string (lowercase, etc.).

Parameters **snt** (str) – Sentence string.

Return type str

Returns Resulting string.

tokenize_sentence(*snt*)
Convert sentence string to a list of tokens.

Parameters **snt** (str) – Sentence string.

Return type List[str]

Returns Resulting list of tokens.

filter_tokens(*snt*)
Clean list of sentence tokens.

Parameters **snt** (List[str]) – List of tokens.

Return type List[str]

Returns Resulting list of filtered tokens

postprocess_tokens(*snt*)
Additional processing steps: lemmatization, pos tagging, etc.

Parameters **snt** (List[str]) – List of tokens.

Return type List[str]

Returns Resulting list of processed tokens.

postprocess_sentence(*snt*)
Postprocess sentence string (merge words).

Parameters **snt** (str) – Sentence string.

Return type str

Returns Resulting string.

tokenize(*text*)
Tokenize list of texts.

Parameters **text** (List[str]) – List of texts.

Return type Union[List[List[str]], List[str]]

Returns Resulting tokenized list.

15.3.2 SimpleRuTokenizer

class lightautoml.text.tokenizer.**SimpleRuTokenizer**(*n_jobs=4*, *to_string=True*, *stopwords=False*, *is_stemmer=True*, ***kwargs*)

Bases: *lightautoml.text.tokenizer.BaseTokenizer*

Russian tokenizer.

__init__(*n_jobs=4*, *to_string=True*, *stopwords=False*, *is_stemmer=True*, ***kwargs*)
Tokenizer for Russian language.

Include numeric, punctuation and short word filtering. Use stemmer by default and do lowercase.

Parameters

- **n_jobs** (int) – Number of threads for multiprocessing.
- **to_string** (bool) – Return string or list of tokens.
- **stopwords** (Union[bool, Sequence[str], None]) – Use stopwords or not.

- **is_stemmer** (bool) – Use stemmer.

preprocess_sentence(*snt*)
Preprocess sentence string (lowercase, etc.).

Parameters **snt** (str) – Sentence string.

Return type str

Returns Resulting string.

tokenize_sentence(*snt*)
Convert sentence string to a list of tokens.

Parameters **snt** (str) – Sentence string.

Return type List[str]

Returns Resulting list of tokens.

filter_tokens(*snt*)
Clean list of sentence tokens.

Parameters **snt** (List[str]) – List of tokens.

Return type List[str]

Returns Resulting list of filtered tokens.

postprocess_tokens(*snt*)
Additional processing steps: lemmatization, pos tagging, etc.

Parameters **snt** (List[str]) – List of tokens.

Return type List[str]

Returns Resulting list of processed tokens.

postprocess_sentence(*snt*)
Postprocess sentence string (merge words).

Parameters **snt** (str) – Sentence string.

Return type str

Returns Resulting string.

15.3.3 SimpleEnTokenizer

class lightautoml.text.tokenizer.**SimpleEnTokenizer**(*n_jobs=4*, *to_string=True*, *stopwords=False*, *is_stemmer=True*, ***kwargs*)

Bases: *lightautoml.text.tokenizer.BaseTokenizer*

English tokenizer.

__init__(*n_jobs=4*, *to_string=True*, *stopwords=False*, *is_stemmer=True*, ***kwargs*)
Tokenizer for English language.

Parameters

- **n_jobs** (int) – Number of threads for multiprocessing.
- **to_string** (bool) – Return string or list of tokens.
- **stopwords** (Union[bool, Sequence[str], None]) – Use stopwords or not.
- **is_stemmer** (bool) – Use stemmer.

preprocess_sentence(*snt*)
Preprocess sentence string (lowercase, etc.).

Parameters **snt** (str) – Sentence string.

Return type str

Returns Resulting string.

tokenize_sentence(*snt*)
Convert sentence string to a list of tokens.

Parameters **snt** (str) – Sentence string.

Return type List[str]

Returns Resulting list of tokens.

filter_tokens(*snt*)
Clean list of sentence tokens.

Parameters **snt** (List[str]) – List of tokens.

Return type List[str]

Returns Resulting list of filtered tokens.

postprocess_tokens(*snt*)
Additional processing steps: lemmatization, pos tagging, etc.

Parameters **snt** (List[str]) – List of tokens.

Return type List[str]

Returns Resulting list of processed tokens.

postprocess_sentence(*snt*)
Postprocess sentence string (merge words).

Parameters **snt** (str) – Sentence string.

Return type str

Returns Resulting string.

15.4 Pooling Strategies

SequenceAbstractPooler	Abstract pooling class.
SequenceClsPooler	CLS token pooling.
SequenceMaxPooler	Max value pooling.
SequenceSumPooler	Sum value pooling.
SequenceAvgPooler	Mean value pooling.
SequenceIndentityPooler	Identity pooling.

15.4.1 SequenceAbstractPooler

class lightautoml.text.sentence_pooling.**SequenceAbstractPooler**(**args*, ***kwargs*)
Bases: torch.nn.Module

Abstract pooling class.

15.4.2 SequenceClsPooler

class lightautoml.text.sentence_pooling.**SequenceClsPooler**(**args*, ***kwargs*)
Bases: *lightautoml.text.sentence_pooling.SequenceAbstractPooler*

CLS token pooling.

15.4.3 SequenceMaxPooler

class lightautoml.text.sentence_pooling.**SequenceMaxPooler**(**args*, ***kwargs*)
Bases: *lightautoml.text.sentence_pooling.SequenceAbstractPooler*

Max value pooling.

15.4.4 SequenceSumPooler

class lightautoml.text.sentence_pooling.**SequenceSumPooler**(**args*, ***kwargs*)
Bases: *lightautoml.text.sentence_pooling.SequenceAbstractPooler*

Sum value pooling.

15.4.5 SequenceAvgPooler

class lightautoml.text.sentence_pooling.**SequenceAvgPooler**(**args*, ***kwargs*)
Bases: *lightautoml.text.sentence_pooling.SequenceAbstractPooler*

Mean value pooling.

15.4.6 SequenceIndentityPooler

class lightautoml.text.sentence_pooling.**SequenceIndentityPooler**(**args*, ***kwargs*)
Bases: *lightautoml.text.sentence_pooling.SequenceAbstractPooler*

Identity pooling.

15.5 Utils

seed_everything	Set random seed and cudnn params.
parse_devices	Parse devices and convert first to the torch device.
custom_collate	Puts each data field into a tensor with outer dimension batch size.
single_text_hash	Get text hash.
get_textarr_hash	Get hash of array with texts.

15.5.1 seed_everything

lightautoml.text.utils.**seed_everything**(*seed=42*, *deterministic=True*)

Set random seed and cudnn params.

Parameters

- **seed** (int) – Random state.
- **deterministic** (bool) – cudnn backend.

15.5.2 parse_devices

lightautoml.text.utils.**parse_devices**(*dvs*, *is_dp=False*)

Parse devices and convert first to the torch device.

Parameters

- **dvs** – List, string with device ids or torch.device.
- **is_dp** (bool) – Use data parallel - additionally returns device ids.

Return type tuple

Returns First torch device and list of gpu ids.

15.5.3 custom_collate

lightautoml.text.utils.**custom_collate**(*batch*)

Puts each data field into a tensor with outer dimension batch size.

Return type Tensor

15.5.4 single_text_hash

lightautoml.text.utils.**single_text_hash**(*x*)

Get text hash.

Parameters **x** (str) – Text.

Return type str

Returns String text hash.

15.5.5 get_textarr_hash

lightautoml.text.utils.**get_textarr_hash**(*x*)

Get hash of array with texts.

Parameters **x** (Sequence[str]) – Text array.

Return type str

Returns Hash of array.

CHAPTER

SIXTEEN

LIGHTAUTOML.TRANSFORMERS

Basic feature generation steps and helper utils.

16.1 Base Classes

LAMLTransformer	Base class for transformer method (like sklearn, but works with datasets).
SequentialTransformer	Transformer that contains the list of transformers and apply one by one sequentially.
UnionTransformer	Transformer that apply the sequence on transformers in parallel on dataset and concatenate the result.
ColumnsSelector	Select columns to pass to another transformers (or feature selection).
ColumnwiseUnion	Apply 1 columns transformer to all columns.
BestOfTransformers	Apply multiple transformers and select best.
ConvertDataset	Convert dataset to given type.
ChangeRoles	Change data roles (include dtypes etc).

16.1.1 LAMLTransformer

class lightautoml.transformers.base.**LAMLTransformer**

Bases: `object`

Base class for transformer method (like sklearn, but works with datasets).

property features

Get name of the features, that will be generated after transform.

Return type `List[str]`

Returns List of new names.

fit(*dataset*)

Fit transformer and return it's instance.

Parameters **dataset** (*LAMLDataset*) – Dataset to fit on.

Return type *LAMLTransformer*

Returns self.

transform(*dataset*)

Transform on dataset.

Parameters **dataset** (*LAMLDataset*) – Dataset to make transform.

Return type *LAMLDataset*

Returns LAMLDataset with new features.

fit_transform(*dataset*)
Default implementation of fit_transform - fit and then transform.

Parameters **dataset** (*LAMLDataset*) – Dataset to fit and then transform on it.

Return type *LAMLDataset*

Returns Dataset with new features.

16.1.2 SequentialTransformer

class lightautoml.transformers.base.**SequentialTransformer**(*transformer_list*)
Bases: *lightautoml.transformers.base.LAMLTransformer*

Transformer that contains the list of transformers and apply one by one sequentially.

__init__(*transformer_list*)

Parameters **transformer_list** (Sequence[*LAMLTransformer*]) – Sequence of transformers.

fit(*dataset*)
Fit not supported. Needs output to fit next transformer.

Parameters **dataset** (*LAMLDataset*) – Dataset to fit.

transform(*dataset*)
Apply the sequence of transformers to dataset one over output of previous.

Parameters **dataset** (*LAMLDataset*) – Dataset to transform.

Return type *LAMLDataset*

Returns Dataset with new features.

fit_transform(*dataset*)
Sequential `.fit_transform`.

Output features - features from last transformer with no prefix.

Parameters **dataset** (*LAMLDataset*) – Dataset to transform.

Return type *LAMLDataset*

Returns Dataset with new features.

16.1.3 UnionTransformer

class lightautoml.transformers.base.**UnionTransformer**(*transformer_list*, *n_jobs=1*)
Bases: *lightautoml.transformers.base.LAMLTransformer*

Transformer that apply the sequence on transformers in parallel on dataset and concatenate the result.

__init__(*transformer_list*, *n_jobs=1*)

Parameters

- **transformer_list** (Sequence[*LAMLTransformer*]) – Sequence of transformers.
- **n_jobs** (int) – Number of processes to run fit and transform.

fit(*dataset*)

Fit transformers in parallel.

Output names - concatenation of features names with no prefix.

Parameters **dataset** (*LAMLDataset*) – Dataset to fit on.

Return type *UnionTransformer*

Returns self.

fit_transform(*dataset*)

Fit and transform transformers in parallel. Output names - concatenation of features names with no prefix.

Parameters **dataset** (*LAMLDataset*) – Dataset to fit and transform on.

Return type *LAMLDataset*

Returns Dataset with new features.

transform(*dataset*)

Apply transformers in parallel. Output names - concatenation of features names with no prefix.

Parameters **dataset** (*LAMLDataset*) – Dataset to fit and transform on.

Return type *LAMLDataset*

Returns Dataset with new features.

16.1.4 ColumnsSelector

class lightautoml.transformers.base.**ColumnsSelector**(*keys*)

Bases: *lightautoml.transformers.base.LAMLTransformer*

Select columns to pass to another transformers (or feature selection).

__init__(*keys*)

Parameters **keys** (Sequence[str]) – Columns names.

fit(*dataset*)

Empty fit method - just set features.

Parameters **dataset** (*LAMLDataset*) – Dataset to fit on.

Return type *ColumnsSelector*

Returns self.

transform(*dataset*)

Select given keys from dataset.

Parameters **dataset** (*LAMLTransformer*) – Dataset to transform.

Return type *LAMLTransformer*

Returns Dataset with selected features.

16.1.5 ColumnwiseUnion

class lightautoml.transformers.base.**ColumnwiseUnion**(*transformer*, *n_jobs=1*)
Bases: *lightautoml.transformers.base.UnionTransformer*

Apply 1 columns transformer to all columns. Example: encode all categories with single category encoders.

__init__(*transformer*, *n_jobs=1*)
Create list of identical transformers from one.

Parameters **transformer** (*LAMLTransformer*) – Dataset - base transformer.

fit(*dataset*)
Create transformer list and then fit.

Parameters **dataset** (*LAMLDataset*) – Dataset with input features.

Returns self.

fit_transform(*dataset*)
Create transformer list and then fit and transform.

Parameters **dataset** (*LAMLDataset*) – Dataset with input features.

Return type *LAMLDataset*

Returns Dataset with new features.

16.1.6 BestOfTransformers

class lightautoml.transformers.base.**BestOfTransformers**(*transformer_list*, *criterion*)
Bases: *lightautoml.transformers.base.LAMLTransformer*

Apply multiple transformers and select best.

__init__(*transformer_list*, *criterion*)
Create selector from candidate list and selection criterion.

Parameters

- **transformer_list** (Sequence[*LAMLTransformer*]) – Sequence of transformers.
- **criterion** (Callable) – Score function (greater is better).

fit(*dataset*)

Empty method - raise error. This transformer supports only `fit_transform`.

Parameters **dataset** (*LAMLDataset*) – LAMLDataset to fit on.

Raises **NotImplementedError** – Always.

fit_transform(*dataset*)
Fit transform all transformers and then select best.

Parameters **dataset** (*LAMLDataset*) – Dataset to fit and then transform.

Return type *LAMLDataset*

Returns Dataset with new features.

transform(*dataset*)
Make transform by the best selected transformer.

Parameters **dataset** (*LAMLDataset*) – Dataset with input features.

Return type *LAMLDataset*

Returns Dataset with new features.

16.1.7 ConvertDataset

class lightautoml.transformers.base.**ConvertDataset**(*dataset_type*)
Bases: *lightautoml.transformers.base.LAMLTransformer*

Convert dataset to given type.

__init__(*dataset_type*)

Parameters **dataset_type** (ClassVar[*LAMLDataset*]) – Type to which to convert.

transform(*dataset*)
Dataset type should implement `from_dataset` method.

Parameters **dataset** (*LAMLDataset*) – Dataset to convert.

Return type *LAMLDataset*

Returns Converted dataset.

16.1.8 ChangeRoles

class lightautoml.transformers.base.**ChangeRoles**(*roles*)
Bases: *lightautoml.transformers.base.LAMLTransformer*

Change data roles (include dtypes etc).

__init__(*roles*)

Parameters **roles** (Union[Sequence[*ColumnRole*], *ColumnRole*, Dict[str, *ColumnRole*], None]) – New roles for dataset.

transform(*dataset*)
Paste new roles into dataset.

Parameters **dataset** (*LAMLDataset*) – Dataset to transform.

Return type *LAMLDataset*

Returns New dataset.

16.2 Numeric

NaNFlags	Create NaN flags.
FillnaMedian	Fillna with median.
FillInf	Fill inf with nan to handle as nan value.
LogOdds	Convert probs to logodds.
StandardScaler	Classic StandardScaler.
QuantileBinning	Discretization of numeric features by quantiles.

16.2.1 NaNFlags

class lightautoml.transformers.numeric.**NaNFlags**(*nan_rate=0.005*)

Bases: *lightautoml.transformers.base.LAMLTransformer*

Create NaN flags.

__init__(*nan_rate=0.005*)

Parameters **nan_rate** (float) – Nan rate cutoff.

fit(*dataset*)

Extract nan flags.

Parameters **dataset** (Union[*NumpyDataset*, *PandasDataset*]) – Pandas or Numpy dataset of categorical features.

Returns self.

transform(*dataset*)

Transform - extract null flags.

Parameters **dataset** (Union[*NumpyDataset*, *PandasDataset*]) – Pandas or Numpy dataset of categorical features.

Return type *NumpyDataset*

Returns Numpy dataset with encoded labels.

16.2.2 FillnaMedian

class lightautoml.transformers.numeric.**FillnaMedian**

Bases: *lightautoml.transformers.base.LAMLTransformer*

Fillna with median.

fit(*dataset*)

Estimate medians.

Parameters **dataset** (Union[*NumpyDataset*, *PandasDataset*]) – Pandas or Numpy dataset of categorical features.

Returns self.

transform(*dataset*)

Transform - fillna with medians.

Parameters **dataset** (Union[*NumpyDataset*, *PandasDataset*]) – Pandas or Numpy dataset of categorical features.

Return type *NumpyDataset*

Returns Numpy dataset with encoded labels.

16.2.3 FillInf

class lightautoml.transformers.numeric.**FillInf**

Bases: *lightautoml.transformers.base.LAMLTransformer*

Fill inf with nan to handle as nan value.

transform(*dataset*)

Replace inf to nan.

Parameters **dataset** (Union[*NumpyDataset*, *PandasDataset*]) – Pandas or Numpy dataset of categorical features.

Return type *NumpyDataset*

Returns Numpy dataset with encoded labels.

16.2.4 LogOdds

class lightautoml.transformers.numeric.**LogOdds**

Bases: *lightautoml.transformers.base.LAMLTransformer*

Convert probs to logodds.

transform(*dataset*)

Transform - convert num values to logodds.

Parameters **dataset** (Union[*NumpyDataset*, *PandasDataset*]) – Pandas or Numpy dataset of categorical features.

Return type *NumpyDataset*

Returns Numpy dataset with encoded labels.

16.2.5 StandardScaler

class lightautoml.transformers.numeric.**StandardScaler**

Bases: *lightautoml.transformers.base.LAMLTransformer*

Classic StandardScaler.

fit(*dataset*)

Estimate means and stds.

Parameters **dataset** (Union[*NumpyDataset*, *PandasDataset*]) – Pandas or Numpy dataset of categorical features.

Returns self.

transform(*dataset*)

Scale test data.

Parameters **dataset** (Union[*NumpyDataset*, *PandasDataset*]) – Pandas or Numpy dataset of numeric features.

Return type *NumpyDataset*

Returns Numpy dataset with encoded labels.

16.2.6 QuantileBinning

class lightautoml.transformers.numeric.**QuantileBinning**(*nbins=10*)

Bases: *lightautoml.transformers.base.LAMLTransformer*

Discretization of numeric features by quantiles.

__init__(*nbins=10*)

Parameters **nbins** (int) – maximum number of bins.

fit(*dataset*)

Estimate bins borders.

Parameters **dataset** (Union[*NumpyDataset*, *PandasDataset*]) – Pandas or Numpy dataset of numeric features.

Returns self.

transform(*dataset*)

Apply bin borders.

Parameters **dataset** (Union[*NumpyDataset*, *PandasDataset*]) – Pandas or Numpy dataset of numeric features.

Return type *NumpyDataset*

Returns Numpy dataset with encoded labels.

16.3 Categorical

LabelEncoder	Simple LabelEncoder in order of frequency.
OHEEncoder	Simple OneHotEncoder over label encoded categories.
FreqEncoder	Labels are encoded with frequency in train data.
OrdinalEncoder	Encoding ordinal categories into numbers.
TargetEncoder	Out-of-fold target encoding.
MultiClassTargetEncoder	Out-of-fold target encoding for multiclass task.
CatIntersectstions	Build label encoded intertsections of categorical variables.

16.3.1 LabelEncoder

class lightautoml.transformers.categorical.**LabelEncoder**(*subs=None*, *random_state=42*)

Bases: *lightautoml.transformers.base.LAMLTransformer*

Simple LabelEncoder in order of frequency.

Labels are integers from 1 to n. Unknown category encoded as 0. NaN is handled as a category value.

__init__(*subs=None*, *random_state=42*)

Parameters

- **subs** (Optional[int]) – Subsample to calculate freqs. If None - full data.
- **random_state** (int) – Random state to take subsample.

fit(*dataset*)

Estimate label frequencies and create encoding dicts.

Parameters **dataset** (Union[*NumpyDataset*, *PandasDataset*]) – Pandas or Numpy dataset of categorical features.

Returns self.

transform(*dataset*)

Transform categorical dataset to int labels.

Parameters **dataset** (Union[*NumpyDataset*, *PandasDataset*]) – Pandas or Numpy dataset of categorical features.

Return type *NumpyDataset*

Returns Numpy dataset with encoded labels.

16.3.2 OHEEncoder

class lightautoml.transformers.categorical.**OHEEncoder**(*make_sparse=None*, *total_feats_cnt=None*, *dtype=numpy.float32*)

Bases: *lightautoml.transformers.base.LAMLTransformer*

Simple OneHotEncoder over label encoded categories.

property features

Features list.

Return type List[str]

__init__(*make_sparse=None*, *total_feats_cnt=None*, *dtype=numpy.float32*)

Parameters

- **make_sparse** (Optional[bool]) – Create sparse matrix.
- **total_feats_cnt** (Optional[int]) – Initial features number.
- **dtype** (type) – Dtype of new features.

fit(*dataset*)

Calc output shapes.

Automatically do ohe in sparse form if approximate fill_rate < *0.2*.

Parameters **dataset** (Union[*NumpyDataset*, *PandasDataset*]) – Pandas or Numpy dataset of categorical features.

Returns self.

transform(*dataset*)
Transform categorical dataset to ohe.

Parameters **dataset** (Union[*NumpyDataset*, *PandasDataset*]) – Pandas or Numpy dataset of categorical features.

Return type *NumpyDataset*

Returns Numpy dataset with encoded labels.

16.3.3 FreqEncoder

class lightautoml.transformers.categorical.**FreqEncoder**(**args*, ***kwargs*)
Bases: *lightautoml.transformers.categorical.LabelEncoder*

Labels are encoded with frequency in train data.

Labels are integers from 1 to n. Unknown category encoded as 1.

fit(*dataset*)
Estimate label frequencies and create encoding dicts.

Parameters **dataset** (Union[*NumpyDataset*, *PandasDataset*]) – Pandas or Numpy dataset of categorical features

Returns self.

16.3.4 OrdinalEncoder

class lightautoml.transformers.categorical.**OrdinalEncoder**(**args*, ***kwargs*)
Bases: *lightautoml.transformers.categorical.LabelEncoder*

Encoding ordinal categories into numbers. Number type categories passed as is, object type sorted in ascending lexicographical order.

fit(*dataset*)
Estimate label frequencies and create encoding dicts.

Parameters **dataset** (Union[*NumpyDataset*, *PandasDataset*]) – Pandas or Numpy dataset of categorical features.

16.3.5 TargetEncoder

class lightautoml.transformers.categorical.**TargetEncoder**(*alphas=(0.5, 1.0, 2.0, 5.0, 10.0, 50.0, 250.0, 1000.0)*)
Bases: *lightautoml.transformers.base.LAMLTransformer*

Out-of-fold target encoding.

Limitation:

- Required .folds attribute in dataset - array of int from 0 to n_folds-1.
- Working only after label encoding.

__init__(*alphas=(0.5, 1.0, 2.0, 5.0, 10.0, 50.0, 250.0, 1000.0)*)

Parameters **alphas** (Sequence[float]) – Smooth coefficients.

static binary_score_func(*candidates*, *target*)
Score candidates alpha with logloss metric.

Parameters

- **candidates** (ndarray) – Candidate oof encoders.
- **target** (ndarray) – Target array.

Return type int

Returns Index of best encoder.

static reg_score_func(*candidates*, *target*)
Score candidates alpha with mse metric.

Parameters

- **candidates** (ndarray) – Candidate oof encoders.
- **target** (ndarray) – Target array.

Return type int

Returns Index of best encoder.

fit_transform(*dataset*)
Calc oof encoding and save encoding stats for new data.

Parameters **dataset** (Union[*NumpyDataset*, *PandasDataset*]) – Pandas or Numpy dataset of categorical label encoded features.

Return type *NumpyDataset*

Returns NumpyDataset - target encoded features.

transform(*dataset*)
Transform categorical dataset to target encoding.

Parameters **dataset** (Union[*NumpyDataset*, *PandasDataset*]) – Pandas or Numpy dataset of categorical features.

Return type *NumpyDataset*

Returns Numpy dataset with encoded labels.

16.3.6 MultiClassTargetEncoder

class lightautoml.transformers.categorical.**MultiClassTargetEncoder**(*alphas=(0.5, 1.0, 2.0, 5.0, 10.0, 50.0, 250.0, 1000.0)*)

Bases: *lightautoml.transformers.base.LAMLTransformer*

Out-of-fold target encoding for multiclass task.

Limitation:

- Required .folds attribute in dataset - array of int from 0 to n_folds-1.
- Working only after label encoding

static score_func(*candidates*, *target*)

Parameters

- **candidates** (ndarray) – np.ndarray.
- **target** (ndarray) – np.ndarray.

Return type int

Returns index of best encoder.

fit_transform(*dataset*)

Estimate label frequencies and create encoding dicts.

Parameters **dataset** (Union[*NumpyDataset*, *PandasDataset*]) – Pandas or Numpy dataset of categorical label encoded features.

Return type *NumpyDataset*

Returns NumpyDataset - target encoded features.

transform(*dataset*)

Transform categorical dataset to target encoding.

Parameters **dataset** (Union[*NumpyDataset*, *PandasDataset*]) – Pandas or Numpy dataset of categorical features.

Return type *NumpyDataset*

Returns Numpy dataset with encoded labels.

16.3.7 CatIntersectstions

class lightautoml.transformers.categorical.**CatIntersectstions**(*subs=None*, *random_state=42*, *intersections=None*, *max_depth=2*)

Bases: *lightautoml.transformers.categorical.LabelEncoder*

Build label encoded intertsections of categorical variables.

__init__(*subs=None*, *random_state=42*, *intersections=None*, *max_depth=2*)

Create label encoded intersection columns for categories.

Parameters

- **intersections** (Optional[Sequence[Sequence[str]]]) – Columns to create intersections. Default is None - all.
- **max_depth** (int) – Max intersection depth.

fit(*dataset*)

Create label encoded intersections and save mapping.

Parameters **dataset** (Union[*NumpyDataset*, *PandasDataset*]) – Pandas or Numpy dataset of categorical features.

Returns self.

transform(*dataset*)

Create label encoded intersections and apply mapping

Parameters **dataset** (Union[*NumpyDataset*, *PandasDataset*]) – Pandas or Numpy dataset of categorical features

Returns:

Return type *NumpyDataset*

16.4 Datetime

TimeToNum	Basic conversion strategy, used in selection one-to-one transformers.
BaseDiff	Basic conversion strategy, used in selection one-to-one transformers.
DateSeasons	Basic conversion strategy, used in selection one-to-one transformers.

16.4.1 TimeToNum

class lightautoml.transformers.datetime.**TimeToNum**

Bases: *lightautoml.transformers.base.LAMLTransformer*

Basic conversion strategy, used in selection one-to-one transformers. Datetime converted to difference with basic_date (`basic_date == '2020-01-01'`).

transform(*dataset*)

Transform dates to numeric differences with base date.

Parameters **dataset** (*PandasDataset*) – Numpy or Pandas dataset with datetime columns.

Return type *NumpyDataset*

Returns Numpy dataset of numeric features.

16.4.2 BaseDiff

class lightautoml.transformers.datetime.**BaseDiff**(*base_names*, *diff_names*, *basic_interval='D'*)

Bases: *lightautoml.transformers.base.LAMLTransformer*

Basic conversion strategy, used in selection one-to-one transformers. Datetime converted to difference with basic_date.

property features

List of features.

Return type List[str]

__init__(*base_names*, *diff_names*, *basic_interval='D'*)

Parameters

- **base_names** (Sequence[str]) – Base date names.
- **diff_names** (Sequence[str]) – Difference date names.
- **basic_interval** (Optional[str]) – Time unit.

fit(*dataset*)

Fit transformer and return it's instance.

Parameters **dataset** (*LAMLDataset*) – Dataset to fit on.

Return type *LAMLTransformer*

Returns self.

transform(*dataset*)
Transform dates to numeric differences with base date.

Parameters **dataset** (*PandasDataset*) – Numpy or Pandas dataset with datetime columns.

Return type *NumpyDataset*

Returns NumpyDataset of numeric features.

16.4.3 DateSeasons

class lightautoml.transformers.datetime.**DateSeasons**(*output_role=None*)
Bases: *lightautoml.transformers.base.LAMLTransformer*

Basic conversion strategy, used in selection one-to-one transformers. Datetime converted to difference with basic_date.

property features
List of features names.

Return type List[str]

__init__(*output_role=None*)

Parameters **output_role** (Optional[*ColumnRole*]) – Which role to assign for input features.

fit(*dataset*)
Fit transformer and return it's instance.

Parameters **dataset** (*LAMLDataset*) – LAMLDataset to fit on.

Return type *LAMLTransformer*

Returns self.

transform(*dataset*)
Transform dates to categories - seasons and holiday flag.

Parameters **dataset** (*PandasDataset*) – Numpy or Pandas dataset with datetime columns.

Return type *NumpyDataset*

Returns Numpy dataset of numeric features.

16.5 Decompositions

PCATransformer	PCA.
SVDTransformer	TruncatedSVD.

16.5.1 PCATransformer

class lightautoml.transformers.decomposition.**PCATransformer**(*subs=None*, *random_state=42*, *n_components=500*)

Bases: *lightautoml.transformers.base.LAMLTransformer*

PCA.

property features
Features list.

Return type List[str]

__init__(*subs=None*, *random_state=42*, *n_components=500*)

Parameters

- **subs** (Optional[int]) – Subsample to fit algorithm. If None - full data.
- **random_state** (int) – Random state to take subsample.
- **n_components** (int) – Number of PCA components

fit(*dataset*)
Fit algorithm on dataset.

Parameters **dataset** (Union[*NumpyDataset*, *PandasDataset*]) – Sparse or Numpy dataset of text features.

transform(*dataset*)
Transform input dataset to PCA representation.

Parameters **dataset** (Union[*NumpyDataset*, *PandasDataset*]) – Pandas or Numpy dataset of text features.

Return type *NumpyDataset*

Returns Numpy dataset with text embeddings.

16.5.2 SVDTransformer

class lightautoml.transformers.decomposition.**SVDTransformer**(*subs=None*, *random_state=42*, *n_components=100*)

Bases: *lightautoml.transformers.base.LAMLTransformer*

TruncatedSVD.

property features
Features list.

Return type List[str]

__init__(*subs=None*, *random_state=42*, *n_components=100*)

Parameters

- **subs** (Optional[int]) – Subsample to fit algorithm. If None - full data.
- **random_state** (int) – Random state to take subsample.
- **n_components** (int) – Number of SVD components.

fit(*dataset*)
Fit algorithm on dataset.

Parameters **dataset** (*NumpyDataset*) – Sparse or Numpy dataset of text features.

transform(*dataset*)
Transform input dataset to SVD representation.

Parameters **dataset** (*NumpyDataset*) – Sparse or Numpy dataset of text features.

Return type *NumpyDataset*

Returns Numpy dataset with text embeddings.

16.6 Text

TunableTransformer	Base class for ML transformers.
TfidfTextTransformer	Simple Tfidf vectorizer.
TokenizerTransformer	Simple tokenizer transformer.
OneToOneTransformer	Out-of-fold sgd model prediction to reduce dimension of encoded text data.
ConcatTextTransformer	Concat text features transformer.
AutoNLPWrap	Calculate text embeddings.

16.6.1 TunableTransformer

class lightautoml.transformers.text.**TunableTransformer**(*default_params=None*, *freeze_defaults=True*)

Bases: *lightautoml.transformers.base.LAMLTransformer*

Base class for ML transformers.

Assume that parameters my set before training.

property params
Parameters.

Return type dict

Returns Dict.

init_params_on_input(*dataset*)
Init params depending on input data.

Return type dict

Returns Dict with model hyperparameters.

__init__(*default_params=None*, *freeze_defaults=True*)

Parameters

- **default_params** (Optional[dict]) – algo hyperparams.
- **freeze_defaults** (bool) –
  - True : params may be rewritten depending on dataset.
  - False: params may be changed only manually or with tuning.

16.6.2 TfidfTextTransformer

class lightautoml.transformers.text.**TfidfTextTransformer**(*default_params=None*, *freeze_defaults=True*, *subs=None*, *random_state=42*)

Bases: *lightautoml.transformers.text.TunableTransformer*

Simple Tfidf vectorizer.

property features

Features list.

Return type List[str]

__init__(*default_params=None*, *freeze_defaults=True*, *subs=None*, *random_state=42*)

Parameters

- **default_params** (Optional[dict]) – algo hyperparams.
- **freeze_defaults** (bool) – Flag.
- **subs** (Optional[int]) – Subsample to calculate freqs. If None - full data.
- **random_state** (int) – Random state to take subsample.

Note: The behaviour of *freeze_defaults*:

- True : params may be rewritten depending on dataset.
- False: params may be changed only manually or with tuning.

---

init_params_on_input(*dataset*)

Get transformer parameters depending on dataset parameters.

Parameters **dataset** (Union[*NumpyDataset*, *PandasDataset*]) – Dataset used for model parmaeters initialization.

Return type dict

Returns Parameters of model.

fit(*dataset*)

Fit tfidf vectorizer.

Parameters **dataset** (Union[*NumpyDataset*, *PandasDataset*]) – Pandas or Numpy dataset of text features.

Returns self.

transform(*dataset*)

Transform text dataset to sparse tfidf representation.

Parameters **dataset** (Union[*NumpyDataset*, *PandasDataset*]) – Pandas or Numpy dataset of text features.

Return type *CSRSparseDataset*

Returns Sparse dataset with encoded text.

16.6.3 TokenizerTransformer

class lightautoml.transformers.text.**TokenizerTransformer**(*tokenizer=<lightautoml.text.tokenizer.SimpleEnTokenizer object>*)

Bases: *lightautoml.transformers.base.LAMLTransformer*

Simple tokenizer transformer.

__init__(*tokenizer=<lightautoml.text.tokenizer.SimpleEnTokenizer object>*)

Parameters **tokenizer** (*BaseTokenizer*) – text tokenizer.

transform(*dataset*)

Transform text dataset to tokenized text dataset.

Parameters **dataset** (Union[*NumpyDataset*, *PandasDataset*]) – Pandas or Numpy dataset of text features.

Return type *PandasDataset*

Returns Pandas dataset with tokenized text.

16.6.4 OneToOneTransformer

class lightautoml.transformers.text.**OneToOneTransformer**(*default_params=None*, *freeze_defaults=False*)

Bases: *lightautoml.transformers.text.TunableTransformer*

Out-of-fold sgd model prediction to reduce dimension of encoded text data.

property features

Features list.

Return type List[str]

init_params_on_input(*dataset*)

Get model parameters depending on dataset parameters.

Parameters **dataset** (Union[*NumpyDataset*, *PandasDataset*]) – NumpyOrPandas.

Return type dict

Returns Parameters of model.

fit(*dataset*)

Apply fit transform.

Parameters **dataset** (Union[*NumpyDataset*, *PandasDataset*]) – Pandas or Numpy dataset of encoded text features.

fit_transform(*dataset*)

Fit and predict out-of-fold sgd model.

Parameters **dataset** (Union[*NumpyDataset*, *PandasDataset*]) – Pandas or Numpy dataset of encoded text features.

Return type *NumpyDataset*

Returns Numpy dataset with out-of-fold model prediction.

transform(*dataset*)

Transform dataset to out-of-fold model-based encoding.

Parameters **dataset** (Union[*NumpyDataset*, *PandasDataset*]) – Pandas or Numpy dataset of encoded text features.

Return type *NumpyDataset*

Returns Numpy dataset with out-of-fold model prediction.

16.6.5 ConcatTextTransformer

class lightautoml.transformers.text.**ConcatTextTransformer**(*special_token=' [SEP] '*)

Bases: *lightautoml.transformers.base.LAMLTransformer*

Concat text features transformer.

__init__(*special_token=' [SEP] '*)

Parameters **special_token** (str) – Add special token between columns.

transform(*dataset*)

Transform text dataset to one text column.

Parameters **dataset** (Union[*NumpyDataset*, *PandasDataset*]) – Pandas or Numpy dataset of text features.

Return type *PandasDataset*

Returns Pandas dataset with one text column.

16.6.6 AutoNLPWrap

class lightautoml.transformers.text.**AutoNLPWrap**(*model_name*, *embedding_model=None*, *cache_dir='./cache_NLP'*, *bert_model=None*, *transformer_params=None*, *subs=None*, *multigpu=False*, *random_state=42*, *train_fasttext=False*, *fasttext_params=None*, *fasttext_epochs=2*, *sent_scaler=None*, *verbose=False*, *device='0'*, ***kwargs*)

Bases: *lightautoml.transformers.base.LAMLTransformer*

Calculate text embeddings.

property features

Features list.

Return type List[str]

__init__(*model_name*, *embedding_model=None*, *cache_dir='./cache_NLP'*, *bert_model=None*, *transformer_params=None*, *subs=None*, *multigpu=False*, *random_state=42*, *train_fasttext=False*, *fasttext_params=None*, *fasttext_epochs=2*, *sent_scaler=None*, *verbose=False*, *device='0'*, ***kwargs*)

Parameters

- **model_name** (str) – Method for aggregating word embeddings into sentence embedding.
- **transformer_params** (Optional[Dict]) – Aggregating model parameters.
- **embedding_model** (Optional[str]) – Word level embedding model with dict interface or path to gensim fasttext model.

- **cache_dir** (str) – If None - do not cache transformed datasets.
- **bert_model** (Optional[str]) – Name of HuggingFace transformer model.
- **subs** (Optional[int]) – Subsample to calculate freqs. If None - full data.
- **multigpu** (bool) – Use Data Parallel.
- **random_state** (int) – Random state to take subsample.
- **train_fasttext** (bool) – Train fasttext.
- **fasttext_params** (Optional[Dict]) – Fasttext init params.
- **fasttext_epochs** (int) – Number of epochs to train.
- **verbose** (bool) – Verbosity.
- **device** (Any) – Torch device or str.
- ****kwargs** – Unused params.

fit(*dataset*)
Fit chosen transformer and create feature names.

Parameters **dataset** (Union[*NumpyDataset*, *PandasDataset*]) – Pandas or Numpy dataset of text features.

transform(*dataset*)
Transform tokenized dataset to text embeddings.

Parameters **dataset** (Union[*NumpyDataset*, *PandasDataset*]) – Pandas or Numpy dataset of text features.

Return type Union[*NumpyDataset*, *PandasDataset*]

Returns Numpy dataset with text embeddings.

16.7 Image

ImageFeaturesTransformer	Simple image histogram.
AutoCVWrap	Calculate image embeddings.

16.7.1 ImageFeaturesTransformer

class lightautoml.transformers.image.**ImageFeaturesTransformer**(*hist_size=30*, *is_hsv=True*, *n_jobs=4*, *loader=<function pil_loader>*)

Bases: *lightautoml.transformers.base.LAMLTransformer*

Simple image histogram.

__init__(*hist_size=30*, *is_hsv=True*, *n_jobs=4*, *loader=<function pil_loader>*)
Create normalized color histogram for rgb or hsv image.

Parameters

- **hist_size** (int) – Number of bins for each channel.
- **is_hsv** (bool) – Convert image to hsv.

- **n_jobs** (int) – Number of threads for multiprocessing.
- **loader** (Callable) – Callable for reading image from path.

property features
Features list.

Return type List[str]

Returns List of features names.

fit(*dataset*)
Init hist class and create feature names.

Parameters **dataset** (Union[*NumpyDataset*, *PandasDataset*]) – Pandas or Numpy dataset of text features.

Returns self.

transform(*dataset*)
Transform image dataset to color histograms.

Parameters **dataset** (Union[*NumpyDataset*, *PandasDataset*]) – Pandas or Numpy dataset of image paths.

Return type *NumpyDataset*

Returns Dataset with encoded text.

16.7.2 AutoCVWrap

class lightautoml.transformers.image.**AutoCVWrap**(*model='efficientnet-b0'*, *weights_path=None*, *cache_dir='./cache_CV'*, *subs=None*, *device=torch.device*, *n_jobs=4*, *random_state=42*, *is_advprop=True*, *batch_size=128*, *verbose=True*)
Bases: *lightautoml.transformers.base.LAMLTransformer*

Calculate image embeddings.

property features
Features list.

Return type List[str]

Returns List of features names.

__init__(*model='efficientnet-b0'*, *weights_path=None*, *cache_dir='./cache_CV'*, *subs=None*, *device=torch.device*, *n_jobs=4*, *random_state=42*, *is_advprop=True*, *batch_size=128*, *verbose=True*)

Parameters

- **model** – Name of effnet model.
- **weights_path** – Path to saved weights.
- **cache_dir** – Path to cache directory or None.
- **subs** – Subsample to fit transformer. If None - full data.
- **device** – Torch device.
- **n_jobs** – Number of threads for dataloader.

- **random_state** – Random state to take subsample and set torch seed.
- **is_advprop** – Use adversarial training.
- **batch_size** – Batch size for embedding model.
- **verbose** – Verbose data processing.

fit(*dataset*)

Fit chosen transformer and create feature names.

Parameters **dataset** (Union[*NumpyDataset*, *PandasDataset*]) – Pandas or Numpy dataset of text features.

transform(*dataset*)

Transform dataset to image embeddings.

Parameters **dataset** (Union[*NumpyDataset*, *PandasDataset*]) – Pandas or Numpy dataset of image paths.

Return type *NumpyDataset*

Returns Numpy dataset with image embeddings.

CHAPTER

SEVENTEEN

LIGHTAUTOML.UTILS

Common util tools.

17.1 Timer

Timer	Timer to limit the duration tasks.
PipelineTimer	Timer is used to control time over full automl run.
TaskTimer	Timer is used to control time over single ML task run.

17.1.1 Timer

class lightautoml.utils.timer.**Timer**

Bases: object

Timer to limit the duration tasks.

17.1.2 PipelineTimer

class lightautoml.utils.timer.**PipelineTimer**(*timeout=None*, *overhead=0.1*, *mode=1*, *tuning_rate=0.7*)

Bases: *lightautoml.utils.timer.Timer*

Timer is used to control time over full automl run.

It decides how much time spend to each algo

__init__(*timeout=None*, *overhead=0.1*, *mode=1*, *tuning_rate=0.7*)

Create global automl timer.

Parameters

- **timeout** (Optional[float]) – Maximum amount of time that AutoML can run.
- **overhead** (float) – (0, 1) - Rate of time that will be used to early stop. Ex. if set to *0.1* and timing mode is set to 2, timer will finish tasks after *0.9* of all time spent.
- **mode** (int) – Timing mode. Can be 0, 1 or 2. Keep in mind - all time limitations will turn on after at least single model/single fold will be computed.
- **tuning_rate** (float) – Approximate fraction of all time will be used for tuning.

Note: Modes explanation:

- 0 - timer is used to estimate runtime, but if something goes out of time, keep it run (Real life mode).
- 1 - timer is used to terminate tasks, but do it after real timeout (Trade off mode).
- 2 - timer is used to terminate tasks with the goal to be exactly in time (Benchmarking/competitions mode).

17.1.3 TaskTimer

class lightautoml.utils.timer.**TaskTimer**(*pipe_timer*, *key=None*, *score=1.0*, *overhead=1*, *mode=1*, *default_tuner_time_rate=0.7*)

Bases: *lightautoml.utils.timer.Timer*

Timer is used to control time over single ML task run.

It decides how much time is ok to spend on tuner and if we have enough time to calc more folds.

property in_progress

Check if the task is running.

Return type bool

__init__(*pipe_timer*, *key=None*, *score=1.0*, *overhead=1*, *mode=1*, *default_tuner_time_rate=0.7*)

Parameters

- **pipe_timer** (*PipelineTimer*) – Global automl timer.
- **key** (Optional[str]) – String name that will be associated with this task.
- **score** (float) – Time score for current task. For ex. if you want to give more of total time to task set it > 1.
- **overhead** (Optional[float]) – See overhead of *PipelineTimer*.
- **mode** (int) – See mode for *PipelineTimer*.
- **default_tuner_time_rate** (float) – If no timing history for the moment of estimating tuning time, timer will use this rate of *time_left*.

start()

Starts counting down.

Returns self.

set_control_point()

Set control point.

Updates the countdown and time left parameters.

write_run_info()

Collect timer history.

get_run_results()

Get timer history.

Return type Optional[ndarray]

Returns None if there is no history, or array with history of runs.

get_run_scores()

Get timer scores.

Return type Optional[ndarray]

Returns None if there is no scores, or array with scores of runs.

estimate_folds_time(*n_folds=1*)
Estimate time for n_folds.

Parameters **n_folds** (int) – Number of folds.

Return type Optional[float]

Returns Estimated time needed to run all *n_folds*.

estimate_tuner_time(*n_folds=1*)
Estimates time that is ok to spend on tuner.

Return type float

Returns How much time timer will be able spend on tuner.

time_limit_exceeded()
Estimate time limit and send results to parent timer.

Return type bool

Returns True if time limit exceeded.

split_timer(*n_parts*)
Split the timer into equal-sized tasks.

Parameters **n_parts** (int) – Number of tasks.

Return type List[*TaskTimer*]

CHAPTER

EIGHTEEN

LIGHTAUTOML.VALIDATION

The module provide classes and functions for model validation.

18.1 Iterators

TrainValidIterator	Abstract class to train/validation iteration.
DummyIterator	Simple Iterator which use train data as validation.
HoldoutIterator	Iterator for classic holdout - just predefined train and valid samples.
CustomIterator	Iterator that uses function to create folds indexes.
FoldsIterator	Classic cv iterator.
TimeSeriesIterator	Time Series Iterator.

18.1.1 TrainValidIterator

class lightautoml.validation.base.**TrainValidIterator**(*train*, ***kwargs*)

Bases: object

Abstract class to train/validation iteration.

Train/valid iterator: should implement *__iter__* and *__next__* for using in ml_pipeline.

property features

Dataset features names.

Returns List of features names.

__init__(*train*, ***kwargs*)

Parameters

- **train** (*~Dataset*) – Train dataset.
- ****kwargs** – Key-word parameters.

get_validation_data()

Abstract method. Get validation sample.

Return type *LAMLDataset*

apply_feature_pipeline(*features_pipeline*)

Apply features pipeline on train data.

Parameters **features_pipeline** (*FeaturesPipeline*) – Composite transformation of features.

Return type *TrainValidIterator*

Returns Copy of object with transformed features.

apply_selector(*selector*)
Select features on train data.

Check if selector is fitted. If not - fit and then perform selection. If fitted, check if it's ok to apply.

Parameters **selector** – Uses for feature selection.

Return type *TrainValidIterator*

Returns Dataset with selected features.

convert_to_holdout_iterator()
Abstract method. Convert iterator to HoldoutIterator.

Return type *HoldoutIterator*

18.1.2 DummyIterator

class lightautoml.validation.base.**DummyIterator**(*train*)
Bases: *lightautoml.validation.base.TrainValidIterator*

Simple Iterator which use train data as validation.

__init__(*train*)
Create iterator. WARNING: validation on train.

Parameters **train** (*~Dataset*) – Train dataset.

get_validation_data()
Just get validation sample.

Return type ~Dataset

Returns Whole train dataset.

convert_to_holdout_iterator()
Convert iterator to hold-out-iterator.

Returns Holdout iterator with `'train == valid'`.

Return type iterator

18.1.3 HoldoutIterator

class lightautoml.validation.base.**HoldoutIterator**(*train*, *valid*)
Bases: *lightautoml.validation.base.TrainValidIterator*

Iterator for classic holdout - just predefined train and valid samples.

__init__(*train*, *valid*)
Create iterator.

Parameters

- **train** (*LAMLDataset*) – Dataset of train data.

- **valid** (*LAMLDataset*) – Dataset of valid data.

get_validation_data()
Just get validation sample.

Return type *LAMLDataset*

Returns Whole validation dataset.

apply_feature_pipeline(*features_pipeline*)
Inplace apply features pipeline to iterator components.

Parameters **features_pipeline** (*FeaturesPipeline*) – Features pipeline to apply.

Return type *HoldoutIterator*

Returns New iterator.

apply_selector(*selector*)
Same as for basic class, but also apply to validation.

Parameters **selector** – Uses for feature selection.

Return type *HoldoutIterator*

Returns New iterator.

convert_to_holdout_iterator()
Do nothing, just return itself.

Return type *HoldoutIterator*

Returns self.

18.1.4 CustomIterator

class lightautoml.validation.base.**CustomIterator**(*train*, *iterator*)
Bases: *lightautoml.validation.base.TrainValidIterator*

Iterator that uses function to create folds indexes.

Usefull for example - classic timeseries splits.

__init__(*train*, *iterator*)
Create iterator.

Parameters

- **train** (*LAMLDataset*) – Dataset of train data.
- **iterator** (Iterable[Tuple[Sequence, Sequence]]) – Callable(dataset) -> Iterator of train/valid indexes.

get_validation_data()
Simple return train dataset.

Return type *LAMLDataset*

Returns Dataset of train data.

convert_to_holdout_iterator()
Convert iterator to hold-out-iterator.

Use first train/valid split for *HoldoutIterator* creation.

Return type *HoldoutIterator*

Returns New hold out iterator.

18.1.5 FoldsIterator

class lightautoml.validation.np_iterators.**FoldsIterator**(*train*, *n_folds=None*)

Bases: *lightautoml.validation.base.TrainValidIterator*

Classic cv iterator.

Folds should be defined in Reader, based on cross validation method.

__init__(*train*, *n_folds=None*)

Creates iterator.

Parameters

- **train** (Union[*NumpyDataset*, *PandasDataset*]) – Dataset for folding.
- **n_folds** (Optional[int]) – Number of folds.

get_validation_data()

Just return train dataset.

Return type Union[*NumpyDataset*, *PandasDataset*]

Returns Whole train dataset.

convert_to_holdout_iterator()

Convert iterator to hold-out-iterator.

Fold 0 is used for validation, everything else is used for training.

Return type *HoldoutIterator*

Returns new hold-out-iterator.

18.1.6 TimeSeriesIterator

class lightautoml.validation.np_iterators.**TimeSeriesIterator**(*datetime_col*, *n_splits=5*, *date_splits=None*, *sorted_kfold=False*)

Bases: object

Time Series Iterator.

static split_by_dates(*datetime_col*, *splitter*)

Create indexes of folds splitted by thresholds.

Parameters

- **datetime_col** – Column with value which can be interpreted as time/ordinal value (ex: np.datetime64).
- **splitter** – List of thresholds (same value as).

Returns Array of folds' indexes.

Return type folds

static split_by_parts(*datetime_col*, *n_splits*)

Create indexes of folds splitted into equal parts.

Parameters

- **datetime_col** – Column with value which can be interpreted as time/ordinal value (ex: np.datetime64).
- **n_splits** (int) – Number of splits(folds).

Returns Array of folds' indexes.

Return type folds

__init__(*datetime_col*, *n_splits=5*, *date_splits=None*, *sorted_kfold=False*)

Generates time series data split. Sorter - include left, exclude right.

Parameters

- **datetime_col** – Column with value which can be interpreted as time/ordinal value (ex: np.datetime64).
- **n_splits** (Optional[int]) – Number of splits.
- **date_splits** (Optional[Sequence]) – List of thresholds.
- **sorted_kfold** (bool) – is sorted.

18.2 Iterators Getters and Utils

create_validation_iterator	Creates train-validation iterator.
get_numpy_iterator	Get iterator for np/sparse dataset.

18.2.1 create_validation_iterator

lightautoml.validation.utils.**create_validation_iterator**(*train*, *valid=None*, *n_folds=None*, *cv_iter=None*)

Creates train-validation iterator.

If train is one of common datasets types (PandasDataset, NumpyDataset, CSRSparseDataset) the *get_numpy_iterator* will be used. Else if validation dataset is defined, the holdout-iterator will be used. Else the dummy iterator will be used.

Parameters

- **train** (*LAMLDataset*) – Dataset to train.
- **valid** (Optional[*LAMLDataset*]) – Optional dataset for validate.
- **n_folds** (Optional[int]) – maximum number of folds to iterate. If None - iterate through all folds.
- **cv_iter** (Optional[Callable]) – Takes dataset as input and return an iterator of indexes of train/valid for train dataset.

Return type *TrainValidIterator*

Returns New iterator.

18.2.2 get_numpy_iterator

lightautoml.validation.np_iterators.**get_numpy_iterator**(*train*, *valid=None*, *n_folds=None*, *iterator=None*)

Get iterator for np/sparse dataset.

If valid is defined, other parameters are ignored. Else if iterator is defined n_folds is ignored.

Else if n_folds is defined iterator will be created by folds index. Else `DummyIterator` - (train, train) will be created.

Parameters

- **train** (Union[*NumpyDataset*, *PandasDataset*]) – `LAMLDataset` to train.
- **valid** (Union[*NumpyDataset*, *PandasDataset*, None]) – Optional `LAMLDataset` for validate.
- **n_folds** (Optional[int]) – maximum number of folds to iterate. If `None` - iterate through all folds.
- **iterator** (Optional[Iterable[Tuple[Sequence, Sequence]]]) – Takes dataset as input and return an iterator of indexes of train/valid for train dataset.

Return type Union[*FoldsIterator*, *HoldoutIterator*, *CustomIterator*, *DummyIterator*]

Returns new train-validation iterator.

CHAPTER

NINETEEN

INDICES AND TABLES

- genindex

INDEX

Symbols

A

B

C

D

E

F

G

H

I

L

M

N

O

P

Q

R

S

T

U

W