LightAutoML

AI Lab ML Tools

Dec 23, 2024

CONTENTS

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.
- Easy-to-use modular scheme to create your own pipelines.

CHAPTER

ONE

INSTALLATION GUIDE

1.1 Basic

You can install library *LightAutoML* from PyPI.

```
pip install lightautoml
```

1.2 Development

You can also clone repository and install with poetry. First, install poetry. Then,

```
git clone git@github.com:AILab-MLTools/LightAutoML.git
cd LightAutoML

# Create virtual environment inside your project directory
poetry config virtualenvs.in-project true

# If you want to update dependencies, run the command:
poetry lock

# Installation
poetry install
```

CHAPTER

TWO

TUTORIALS

This section contains tutorials for both **LightAutoML** and **HypEx**, covering a wide range of use cases from basic model training to advanced hypothesis testing.

2.1 LightAutoML Tutorials

2.1.1 Tutorial 1: Basics

Official LightAutoML github repository is here

In this tutorial you will learn how to: * run LightAutoML training on tabular data * obtain feature importances and reports * configure resource usage in LightAutoML

0. Prerequisites

0.0. install LightAutoML

[1]:
```
#!pip install -U lightautoml
```

0.1. Import libraries

Here we will import the libraries we use in this kernel: - Standard python libraries for timing, working with OS and HTTP requests etc. - Essential python DS libraries like numpy, pandas, scikit-learn and torch (the last we will use in the next cell) - LightAutoML modules: presets for AutoML, task and report generation module

[1]:
```
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)
```

[1]:
```
# Standard python libraries
import os
import requests

# Essential DS libraries
import numpy as np
```

(continues on next page)

(continued from previous page)

```
import pandas as pd
from sklearn.metrics import roc_auc_score
from sklearn.model_selection import train_test_split
import torch

# LightAutoML presets, task and report generation
from lightautoml.automl.presets.tabular_presets import TabularAutoML,␣
↪TabularUtilizedAutoML
from lightautoml.tasks import Task
from lightautoml.report.report_deco import ReportDeco, ReportDecoUtilized
from lightautoml.addons.tabular_interpretation import SSWARM
```

0.2. Constants

Here we setup some parameters to use in the kernel: - `N_THREADS` - number of vCPUs for LightAutoML model creation - `N_FOLDS` - number of folds in LightAutoML inner CV - `RANDOM_STATE` - random seed for better reproducibility - `TEST_SIZE` - houldout data part size - `TIMEOUT` - limit in seconds for model to train - `TARGET_NAME` - target column name in dataset

[2]:
```
N_THREADS = 4
N_FOLDS = 5
RANDOM_STATE = 42
TEST_SIZE = 0.2
TIMEOUT = 300
TARGET_NAME = 'TARGET'
```

[3]:
```
DATASET_DIR = '../data/'
DATASET_NAME = 'sampled_app_train.csv'
DATASET_FULLNAME = os.path.join(DATASET_DIR, DATASET_NAME)
DATASET_URL = 'https://raw.githubusercontent.com/AILab-MLTools/LightAutoML/master/
↪examples/data/sampled_app_train.csv'
```

0.3. Imported models setup

For better reproducibility fix numpy random seed with max number of threads for Torch (which usually try to use all the threads on server):

[4]:
```
np.random.seed(RANDOM_STATE)
torch.set_num_threads(N_THREADS)
```

0.4. Data loading

Let's check the data we have:

[5]:
```
if not os.path.exists(DATASET_FULLNAME):
    os.makedirs(DATASET_DIR, exist_ok=True)

    dataset = requests.get(DATASET_URL).text
    with open(DATASET_FULLNAME, 'w') as output:
        output.write(dataset)
```

[6]:
```
data = pd.read_csv(DATASET_DIR + DATASET_NAME)
data.head()
```

[6]:
```
   SK_ID_CURR  TARGET NAME_CONTRACT_TYPE CODE_GENDER FLAG_OWN_CAR  \
0      313802       0         Cash loans           M            N
1      319656       0         Cash loans           F            N
2      207678       0    Revolving loans           F            Y
3      381593       0         Cash loans           F            N
4      258153       0         Cash loans           F            Y

  FLAG_OWN_REALTY  CNT_CHILDREN  AMT_INCOME_TOTAL  AMT_CREDIT  AMT_ANNUITY  \
0               Y             0          270000.0    327024.0      15372.0
1               N             0          108000.0    675000.0      19737.0
2               Y             2          112500.0    270000.0      13500.0
3               N             1           67500.0    142200.0       9630.0
4               Y             0          337500.0   1483231.5      46570.5

   ...  FLAG_DOCUMENT_18 FLAG_DOCUMENT_19 FLAG_DOCUMENT_20 FLAG_DOCUMENT_21  \
0  ...                 0                0                0                0
1  ...                 0                0                0                0
2  ...                 0                0                0                0
3  ...                 0                0                0                0
4  ...                 0                0                0                0

  AMT_REQ_CREDIT_BUREAU_HOUR AMT_REQ_CREDIT_BUREAU_DAY  \
0                        0.0                       0.0
1                        0.0                       0.0
2                        0.0                       0.0
3                        0.0                       0.0
4                        0.0                       0.0

   AMT_REQ_CREDIT_BUREAU_WEEK  AMT_REQ_CREDIT_BUREAU_MON  \
0                         0.0                        0.0
1                         0.0                        0.0
2                         0.0                        0.0
3                         0.0                        0.0
4                         0.0                        2.0

   AMT_REQ_CREDIT_BUREAU_QRT  AMT_REQ_CREDIT_BUREAU_YEAR
0                        0.0                        1.0
1                        0.0                        0.0
2                        0.0                        1.0
3                        0.0                        4.0
4                        0.0                        0.0

[5 rows x 122 columns]
```

[7]:
```
data.shape
```

[7]:
```
(10000, 122)
```

0.5. Data splitting for train-holdout

As we have only one file with target values, we can split it into 80%-20% for holdout usage:

[8]:
```
train_data, test_data = train_test_split(
    data,
    test_size=TEST_SIZE,
    stratify=data[TARGET_NAME],
    random_state=RANDOM_STATE
)

print(f'Data is splitted. Parts sizes: train_data = {train_data.shape}, test_data =
↪{test_data.shape}')

train_data.head()
```

```
Data is splitted. Parts sizes: train_data = (8000, 122), test_data = (2000, 122)
```

[8]:

	SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	\
6444	112261	0	Cash loans	F	N	
3586	115058	0	Cash loans	F	N	
9349	326623	0	Cash loans	F	N	
7734	191976	0	Cash loans	M	Y	
2174	281519	0	Revolving loans	F	N	

	FLAG_OWN_REALTY	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANNUITY	\
6444	N	1	90000.0	640080.0	31261.5	
3586	Y	0	180000.0	239850.0	23850.0	
9349	Y	0	112500.0	337500.0	31086.0	
7734	Y	1	67500.0	135000.0	9018.0	
2174	Y	0	67500.0	202500.0	10125.0	

	...	FLAG_DOCUMENT_18	FLAG_DOCUMENT_19	FLAG_DOCUMENT_20	\
6444	...	0	0	0	
3586	...	0	0	0	
9349	...	0	0	0	
7734	...	0	0	0	
2174	...	0	0	0	

	FLAG_DOCUMENT_21	AMT_REQ_CREDIT_BUREAU_HOUR	AMT_REQ_CREDIT_BUREAU_DAY	\
6444	0	0.0	0.0	
3586	0	0.0	0.0	
9349	0	0.0	0.0	
7734	0	NaN	NaN	
2174	0	0.0	0.0	

	AMT_REQ_CREDIT_BUREAU_WEEK	AMT_REQ_CREDIT_BUREAU_MON	\
6444	0.0	0.0	
3586	0.0	0.0	
9349	0.0	0.0	
7734	NaN	NaN	
2174	0.0	0.0	

	AMT_REQ_CREDIT_BUREAU_QRT	AMT_REQ_CREDIT_BUREAU_YEAR
6444	1.0	0.0

(continues on next page)

(continued from previous page)

```
3586                        0.0                         3.0
9349                        0.0                         2.0
7734                        NaN                         NaN
2174                        0.0                         2.0

[5 rows x 122 columns]
```

Note: missing values (NaN and other) in the data should be left as is, unless the reason for their presence or their specific meaning are known. Otherwise, AutoML model will perceive the filled NaNs as a true pattern between the data and the target variable, without knowledge and assumptions about missing values, which can negatively affect the model quality. LighAutoML can deal with missing values and outliers automatically.

1. Task definition

1.1. Task type

First we need to create `Task` object - the class to setup what task LightAutoML model should solve with specific loss and metric if necessary (more info can be found here in our documentation).

The following task types are available:

- `'binary'` - for binary classification.
- `'reg'` - for regression.
- `'multiclass'` - for multiclass classification.
- `'multi:reg` - for multiple regression.
- `'multilabel'` - for multi-label classification.

In this example we will consider a binary classification:

[11]:
```
task = Task('binary')
```

Note: only logloss loss is available for binary task and it is the default loss. Default metric for binary classification is ROC-AUC. See more info about available and default losses and metrics here.

Depending on the task, you can and should choose exactly those metrics and losses that you want and need to optimize.

1.2. Feature roles setup

To solve the task, we need to setup columns roles. LightAutoML can automatically define types and roles of data columns, but it is possible to specify it directly through the dictionary parameter `roles` when training AutoML model (see next section "AutoML training"). Specific roles can be specified using a string with the name (any role can be set like this). So the key in dictionary must be the name of the role, the value must be a list of the names of the corresponding columns in dataset. The **only role you must setup is `'target'` role** (that is column with target variable obviously), everything else (`'drop'`, `'numeric'`, `'categorical'`, `'group'`, `'weights'` etc) is up to user:

[12]:
```
roles = {
    'target': TARGET_NAME,
    'drop': ['SK_ID_CURR']
}
```

You can also optionally specify the following roles:

- `'numeric'` - numerical feature

- 'category' - categorical feature
- 'text' - text data
- 'datetime' - features with date and time
- 'date' - features with date only
- 'group' - features by which the data can be divided into groups and which can be taken into account for group k-fold validation (so the same group is not represented in both testing and training sets)
- 'drop' - features to drop, they will not be used in model building
- 'weights' - object weights for the loss and metric
- 'path' - image file paths (for CV tasks)
- 'treatment' - object group in uplift modelling tasks: treatment or control

Note: role name can be written in any case. Also it is possible to pass individual objects of role classes with specific arguments instead of strings with role names for specific tasks and more optimal pipeline construction (more details).

For example, to set the date role, you can use the DatetimeRole class.

[13]:
```
#from lightautoml.dataset.roles import DatetimeRole
```

Different seasonality can be extracted from the data through the seasonality parameter: years ('y'), months ('m'), days ('d'), weekdays ('wd'), hours ('hour'), minutes ('min'), seconds ('sec'), milliseconds ('ms'), nanoseconds ('ns'). This features will be considered as categorical. Another important parameter is base_date. It allows to specify the base date and convert the feature to the distances to this date (set to False by default). Also for all roles classes there is a force_input parameter, and if it is True, then the corresponding features will pass all further feature selections and won't be excluded (equals False by default). Also it is always possible to specify data type for all roles using dtype argument.

Here is an example of such a role assignment through a class object for date feature (but there is no such feature in the considered dataset):

[14]:
```
# roles = {
#     DatetimeRole(base_date=False, seasonality=('d', 'wd', 'hour')): 'date_time'
# }
```

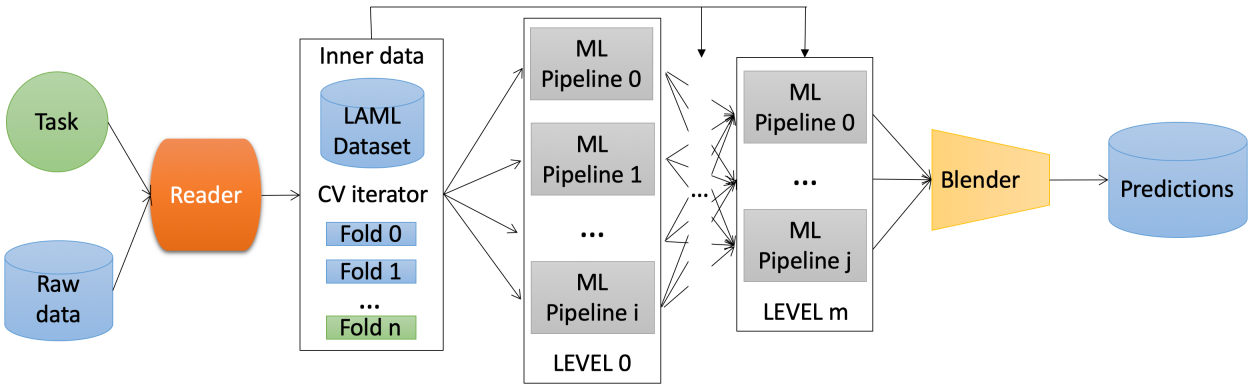
Any role can be set through a class object. Information about specific parameters of specific roles and other datailed information can be found here.

1.3. LightAutoML model creation - TabularAutoML preset

Next we are going to create LightAutoML model with TabularAutoML class - preset with default model structure in just several lines.

In general, the whole AutoML model consists of multiple levels, which can contain several pipelines with their own set of data processing methods and ML models. The outputs of one level are the inputs of the next, and on the last level predictions of previous level models are combined with blending procedure. All this can be combined into a model using the AutoML class and its various descendants (like TabularAutoML).

Let's look at how the LightAutoML model is arranged and what it consists in general.



1.3.1 Reader object

First the task and data go into `Reader` object. It analyzes the data and extracts various valuable information from them. Also it can detect and remove useless features, conduct feature selection, determine types and roles etc. Let's look at this steps in more detail.

Role and types guessing

Roles can be specified as a string or a specific class object, or defined automatically. For `TabularAutoML` preset `'numeric'`, `'datetime'` and `'category'` roles can be automatically defined. There are two ways of role defining. **First** is very simple: check if the value can be converted to a date (`'datetime'`), otherwise check if it can be converted to a number (`'numeric'`), otherwise declare it a category (`'categorical'`). But this method may not work well on large data or when encoding categories with integers. The **second** method is based on statistics: the distributions of numerical features are considered, and how similar they are to the distributions of real or categorical value. Also different ways of feature encoding (as a number or as a category) are compared and based on normalized Gini index it is decided which encoding is better. For this case a set of specific rules is created, and if at least one of them is fulfilled, then the feature will be assigned to numerical, otherwise to categorical. This check can be enabled or disabled using the `advanced_roles` parameter.

If roles are explicitly specified, automatic definition won't be applied to the specified dataset columns. In the case of specifying a role as an object of a certain class, through its arguments, it is possible to set the processing parameters in more detail.

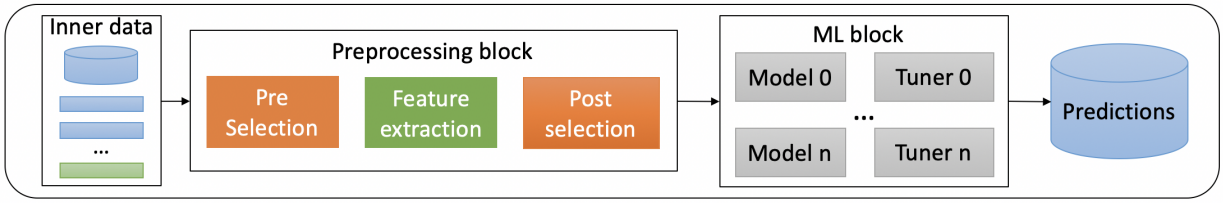
Feature selection

In general, the AutoML pipeline uses pre-selection, generation and post-selection of features. `TabularAutoML` has no post-selection stage. There are three feature selection methods: its absence, using features importances and more strict selection (forward selection). The GBM model is used to evaluate features importances. Importances can be calculated in 2 ways: based on splits (how many times a split was made for each feature in the entire ensemble) or using permutation feature importances (mixing feature values during validation and assessing quality change in this case). Second method is harder but it requires holdout data. Then features with importance above a certain threshold are selected. Faster and more strict feature selection method is forward selection. Features are sorted in descending order of importance, then in blocks (size of 1 by default) a model is built based on selected features, and its quality is measured. Then the next block of features is added, and they are saved if the quality has improved with them, and so on.

Also LightAutoML can merge some columns if it is adequate and leads to an improvement in the model quality (for example, an intersection between categorical variables). Different columns join options are considered, the best one is chosen by the normalized Gini index.

1.3.2 Machine learning pipelines architecture and training

As a result, after analyzing and processing the data, the `Reader` object forms and returns a `LAMA Dataset`. It contains the original data and markup with metainformation. In this dataset it is possible to see the roles defined by the `Reader` object, selected features etc. Then ML pipelines are trained on this data.



Each such pipeline is one or more machine learning algorithms that share one post-processing block and one validation scheme. Several such pipelines can be trained in parallel on one dataset, and they form a level. Number of levels can be unlimited as possible. List of all levels of AutoML pipeline is available via `.levels` attribute of `AutoML` instance. Level predictions can be inputs to other models or ML pipelines (i. e. stacking scheme). As inputs for subsequent levels, it is possible to use the original data by setting `skip_conn` argument in `True` when initializing preset instance. At the last level, if there are several pipelines, blending is used to build a prediction.

Different types of features are processed depending on the models. Numerical features are processed for linear model preprocessing: standardization, replacing missing values with median, discretization, log odds (if feature is probability - output of previous level). Categories are processed using label encoding (by default), one hot encoding, ordinal encoding, frequency encoding, out of fold target encoding.

The following algorithms are available in the LightAutoML: linear regression with L2 regularization, LightGBM, CatBoost, random forest.

By default KFold cross-validation is used during training at all levels (for hyperparameter optimization and building out-of-fold prediction during training), and for each algorithm a separate model is built for each validation fold, and their predictions are averaged. So the predictions at each level and the resulting prediction during training are out-of-fold predictions. But it is also possible to just pass a holdout data for validation or use custom cross-validation schemes, setting `cv_iter` iterator returning the indices of the objects for validation. LightAutoML has ready-made iterators, for example, `TimeSeriesIterator` for time series split. To further reduce the effect of overfitting, it is possible to use nested cross-validation (`nested_cv` parameter), but it is not used by default.

Prediction on new data is the averaging of models over all folds from validation and blending.

Hyperparameter tuning of machine learning algorithms can be performed during training (early stopping by the number of trees in gradient boosting or the number of training epochs of neural networks etc), based on expert rules (according to data characteristics and empirical recommendations, so-called expert parameters), by the sequential model-based optimization (SMBO, bayesian optimization: Optuna with TPESampler) or by grid search. LightGBM and CatBoost can be used with parameter tuning or with expert parameters, with no tuning. For linear regression parameters are always tuned using warm start model training technique.

At the last level blending is used to build a prediction. There are three available blending methods: choosing the best model based on a given metric (other models are just discarded), simple averaging of all models, or weighted averaging (weights are selected using coordinate descent algorithm with optimization of a given metric). `TabularAutoML` uses the latter strategy by default. It is worth noting that, unlike stacking, blending can exclude models from composition.

1.3.3 Timing

When creating AutoML object, a certain time limit is set, and it schedules a list of tasks that it can complete during this time, and it will initially allocate approximately equal time for each task. In the process of solving objectives, it understands how to adjust the time allocated to different subtasks. If AutoML finished working earlier than set timeout, it means that it completed the entire list of tasks. If AutoML worked to the limit and turned off, then most likely it sacrificed something, for example, reduced the number of algorithms for training, realized that it would not have time

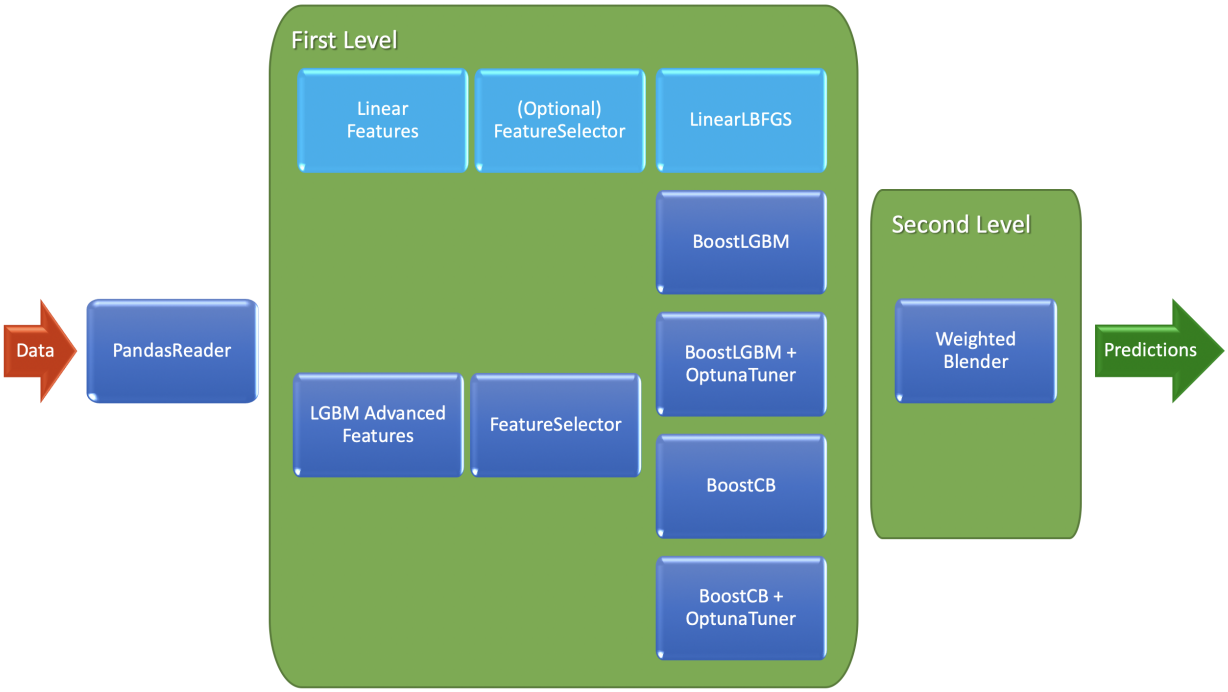
to train the next one, or it might not calculate the full cross-validation cycle for one of the models (then on folds, where the model has not trained, the predictiuons will be NaN, and the model related to this fold will not participate in the final averaging). The resulting quality is evaluated at the blending stage, and if necessary and possible, the composition will be corrected.

If you do not set the time for AutoML during initialization, then by default it will be equal to a very large number, that is, sooner or later AutoML will complete all tasks.

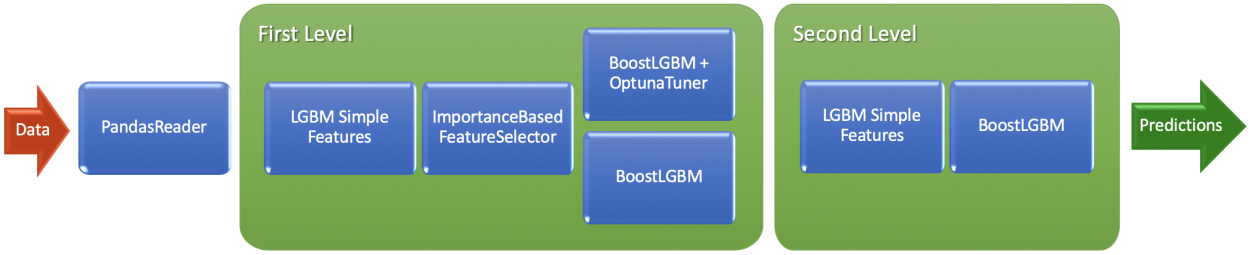
1.3.4 LightAutoML model creation

So the entire AutoML pipeline can be composed from various parts by user (see custom pipeline tutorial), but it is possible to use presets - in a certain sense, fixed strategies for dynamic pipeline building.

Here is a default AutoML pipeline for binary classification and regression tasks (`TabularAutoML` preset):



Another example:



Let's discuss some of the params we can setup: - `task` - the type of the ML task (the only **must have** parameter) - `timeout` - time limit in seconds for model to train - `cpu_limit` - vCPU count for model to use - `reader_params` - parameter change for `Reader` object inside preset, which works on the first step of data preparation: automatic feature typization, preliminary almost-constant features, correct CV setup etc. For example, we setup `n_jobs` threads for

typization algo, `cv` folds and `random_state` as inside CV seed. - `general_params` - general parameters dictionary, in which it is possible to specify a list of algorithms used (`'use_algos'`), nested CV using (`'nested_cv'`) etc.

Important note: `reader_params` key is one of the YAML config keys, which is used inside `TabularAutoML` preset. More details on its structure with explanation comments can be found on the link attached. Each key from this config can be modified with user settings during preset object initialization. To get more info about different parameters setting (for example, ML algos which can be used in `general_params->use_algos`) please take a look at our article on TowardsDataScience.

Moreover, to receive the automatic report for our model we will use `ReportDeco` decorator and work with the decorated version in the same way as we do with usual one.

[15]:
```
automl = TabularAutoML(
    task = task,
    timeout = TIMEOUT,
    cpu_limit = N_THREADS,
    reader_params = {'n_jobs': N_THREADS, 'cv': N_FOLDS, 'random_state': RANDOM_STATE},
)
```

2. AutoML training

To run autoML training use `fit_predict` method.

Main arguments:

- `train_data` - dataset to train.
- `roles` - column roles dict.
- `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;

Note: out-of-fold prediction is calculated during training and returned from the fit_predict method

[16]:
```
%%time
out_of_fold_predictions = automl.fit_predict(train_data, roles = roles, verbose = 1)
```

```
[08:50:50] Stdout logging level is INFO.
[08:50:50] Copying TaskTimer may affect the parent PipelineTimer, so copy will create␣
↪new unlimited TaskTimer
[08:50:50] Task: binary

[08:50:50] Start automl preset with listed constraints:
[08:50:50] - time: 300.00 seconds
[08:50:50] - CPU: 4 cores
[08:50:50] - memory: 16 GB

[08:50:50] Train data shape: (8000, 122)

[08:50:55] Layer 1 train process start. Time left 295.04 secs
[08:50:56] Start fitting Lvl_0_Pipe_0_Mod_0_LinearL2 ...
[08:51:00] Fitting Lvl_0_Pipe_0_Mod_0_LinearL2 finished. score = 0.7351718729105452
[08:51:00] Lvl_0_Pipe_0_Mod_0_LinearL2 fitting and predicting completed
[08:51:00] Time left 290.51 secs
```

(continues on next page)

(continued from previous page)

```
[08:51:02] Selector_LightGBM fitting and predicting completed
[08:51:03] Start fitting Lvl_0_Pipe_1_Mod_0_LightGBM ...
[08:51:15] Fitting Lvl_0_Pipe_1_Mod_0_LightGBM finished. score = 0.7331401653896874
[08:51:15] Lvl_0_Pipe_1_Mod_0_LightGBM fitting and predicting completed
[08:51:15] Start hyperparameters optimization for Lvl_0_Pipe_1_Mod_1_Tuned_LightGBM ... Time budget is 16.51 secs
[08:51:37] Hyperparameters optimization for Lvl_0_Pipe_1_Mod_1_Tuned_LightGBM completed
[08:51:37] Start fitting Lvl_0_Pipe_1_Mod_1_Tuned_LightGBM ...
[08:51:49] Fitting Lvl_0_Pipe_1_Mod_1_Tuned_LightGBM finished. score = 0.7089314555691407
[08:51:49] Lvl_0_Pipe_1_Mod_1_Tuned_LightGBM fitting and predicting completed
[08:51:49] Start fitting Lvl_0_Pipe_1_Mod_2_CatBoost ...
[08:51:54] Fitting Lvl_0_Pipe_1_Mod_2_CatBoost finished. score = 0.7176822653076455
[08:51:54] Lvl_0_Pipe_1_Mod_2_CatBoost fitting and predicting completed
[08:51:54] Start hyperparameters optimization for Lvl_0_Pipe_1_Mod_3_Tuned_CatBoost ... Time budget is 155.49 secs
[08:54:28] Hyperparameters optimization for Lvl_0_Pipe_1_Mod_3_Tuned_CatBoost completed
[08:54:28] Start fitting Lvl_0_Pipe_1_Mod_3_Tuned_CatBoost ...
[08:54:40] Fitting Lvl_0_Pipe_1_Mod_3_Tuned_CatBoost finished. score = 0.7376224015286756
[08:54:40] Lvl_0_Pipe_1_Mod_3_Tuned_CatBoost fitting and predicting completed
[08:54:40] Time left 70.52 secs

[08:54:40] Layer 1 training completed.

[08:54:40] Blending: optimization starts with equal weights and score 0.7485901992886845
[08:54:40] Blending: iteration 0: score = 0.7500463998499898, weights = [0.25878194 0.2476184  0.18001081 0.         0.3135889 ]
[08:54:40] Blending: iteration 1: score = 0.750143026341721, weights = [0.29799867 0.22685595 0.15482728 0.         0.3203181 ]
[08:54:40] Blending: iteration 2: score = 0.7501702390830668, weights = [0.3023924  0.22656146 0.1511439  0.         0.31990227]
[08:54:40] Blending: iteration 3: score = 0.7501702390830668, weights = [0.3023924  0.22656146 0.15114388 0.         0.31990227]
[08:54:40] Blending: no score update. Terminated

[08:54:40] Automl preset training completed in 230.04 seconds

[08:54:40] Model description:
Final prediction for new objects (level 0) =
         0.30239 * (5 averaged models Lvl_0_Pipe_0_Mod_0_LinearL2) +
         0.22656 * (5 averaged models Lvl_0_Pipe_1_Mod_0_LightGBM) +
         0.15114 * (5 averaged models Lvl_0_Pipe_1_Mod_1_Tuned_LightGBM) +
         0.31990 * (5 averaged models Lvl_0_Pipe_1_Mod_3_Tuned_CatBoost)

CPU times: user 13min 39s, sys: 1min 30s, total: 15min 10s
Wall time: 3min 50s
```

After training we can see logs with all the progress, final scores, weights assigned to the models in the final prediction etc.

Note that in this `fit_predict` you receive the model with only 3 out of 5 LightGBM models (you can see that from the log line in the end `0.25685 * (3 averaged models Lvl_0_Pipe_1_Mod_0_LightGBM)`) - to fix it you can set the bigger timeout to make LightAutoML train all the models.

3. Prediction on holdout and model evaluation

Now we can use trained AutoML model to build predictions on holdout and evaluate model quality. Note that in case of classification tasks LightAutoML model returns probabilities as predictions.

[17]:
```
%%time

test_predictions = automl.predict(test_data)
print(f'Prediction for test_data:\n{test_predictions}\nShape = {test_predictions.shape}')
```

```
Prediction for test_data:
array([[0.06522178],
       [0.06859127],
       [0.0340898 ],
       ...,
       [0.0531715 ],
       [0.04148169],
       [0.18272203]], dtype=float32)
Shape = (2000, 1)
CPU times: user 1.67 s, sys: 90.5 ms, total: 1.76 s
Wall time: 678 ms
```

[18]:
```
print(f'OOF score: {roc_auc_score(train_data[TARGET_NAME].values, out_of_fold_
↪predictions.data[:, 0])}')
print(f'HOLDOUT score: {roc_auc_score(test_data[TARGET_NAME].values, test_predictions.
↪data[:, 0])}')
```

```
OOF score: 0.7501702390830668
HOLDOUT score: 0.7314741847826086
```

4. Model analysis

4.1. Reports

You can obtain the description of the resulting pipeline:

[19]:
```
print(automl.create_model_str_desc())
```

```
Final prediction for new objects (level 0) =
         0.30239 * (5 averaged models Lvl_0_Pipe_0_Mod_0_LinearL2) +
         0.22656 * (5 averaged models Lvl_0_Pipe_1_Mod_0_LightGBM) +
         0.15114 * (5 averaged models Lvl_0_Pipe_1_Mod_1_Tuned_LightGBM) +
         0.31990 * (5 averaged models Lvl_0_Pipe_1_Mod_3_Tuned_CatBoost)
```

Also for this purposes LightAutoML have `ReportDeco`, use it to build detailed reports:

[20]:
```
RD = ReportDeco(output_path = 'tabularAutoML_model_report')

automl_rd = RD(
    TabularAutoML(
        task = task,
        timeout = TIMEOUT,
        cpu_limit = N_THREADS,
        reader_params = {'n_jobs': N_THREADS, 'cv': N_FOLDS, 'random_state': RANDOM_
↪STATE}
```

(continues on next page)

(continued from previous page)

```
    )
)
```

[21]:
```
%%time
out_of_fold_predictions = automl_rd.fit_predict(train_data, roles = roles, verbose = 1)
```

```
[08:54:41] Stdout logging level is INFO.
[08:54:41] Task: binary

[08:54:41] Start automl preset with listed constraints:
[08:54:41] - time: 300.00 seconds
[08:54:41] - CPU: 4 cores
[08:54:41] - memory: 16 GB

[08:54:41] Train data shape: (8000, 122)

[08:54:42] Layer 1 train process start. Time left 298.57 secs
[08:54:43] Start fitting Lvl_0_Pipe_0_Mod_0_LinearL2 ...
[08:54:47] Fitting Lvl_0_Pipe_0_Mod_0_LinearL2 finished. score = 0.7351718729105452
[08:54:47] Lvl_0_Pipe_0_Mod_0_LinearL2 fitting and predicting completed
[08:54:47] Time left 294.28 secs

[08:54:49] Selector_LightGBM fitting and predicting completed
[08:54:50] Start fitting Lvl_0_Pipe_1_Mod_0_LightGBM ...
[08:55:01] Fitting Lvl_0_Pipe_1_Mod_0_LightGBM finished. score = 0.7331401653896874
[08:55:01] Lvl_0_Pipe_1_Mod_0_LightGBM fitting and predicting completed
[08:55:01] Start hyperparameters optimization for Lvl_0_Pipe_1_Mod_1_Tuned_LightGBM ...␣
↪Time budget is 20.80 secs
[08:55:24] Hyperparameters optimization for Lvl_0_Pipe_1_Mod_1_Tuned_LightGBM completed
[08:55:24] Start fitting Lvl_0_Pipe_1_Mod_1_Tuned_LightGBM ...
[08:55:37] Fitting Lvl_0_Pipe_1_Mod_1_Tuned_LightGBM finished. score = 0.7089314555691407
[08:55:37] Lvl_0_Pipe_1_Mod_1_Tuned_LightGBM fitting and predicting completed
[08:55:37] Start fitting Lvl_0_Pipe_1_Mod_2_CatBoost ...
[08:55:43] Fitting Lvl_0_Pipe_1_Mod_2_CatBoost finished. score = 0.7176822653076455
[08:55:43] Lvl_0_Pipe_1_Mod_2_CatBoost fitting and predicting completed
[08:55:43] Start hyperparameters optimization for Lvl_0_Pipe_1_Mod_3_Tuned_CatBoost ...␣
↪Time budget is 155.52 secs
[08:58:19] Hyperparameters optimization for Lvl_0_Pipe_1_Mod_3_Tuned_CatBoost completed
[08:58:19] Start fitting Lvl_0_Pipe_1_Mod_3_Tuned_CatBoost ...
[08:58:31] Fitting Lvl_0_Pipe_1_Mod_3_Tuned_CatBoost finished. score = 0.7376224015286756
[08:58:31] Lvl_0_Pipe_1_Mod_3_Tuned_CatBoost fitting and predicting completed
[08:58:31] Time left 69.91 secs

[08:58:31] Layer 1 training completed.

[08:58:31] Blending: optimization starts with equal weights and score 0.7485901992886845
[08:58:31] Blending: iteration 0: score = 0.7500463998499898, weights = [0.25878194 0.
↪2476184  0.18001081 0.         0.3135889 ]
[08:58:31] Blending: iteration 1: score = 0.750143026341721, weights = [0.29799867 0.
↪22685595 0.15482728 0.         0.3203181 ]
[08:58:31] Blending: iteration 2: score = 0.7501702390830668, weights = [0.3023924  0.
↪22656146 0.1511439  0.         0.31990227]
```

(continues on next page)

(continued from previous page)

```
[08:58:32] Blending: iteration 3: score = 0.7501702390830668, weights = [0.3023924  0.
↪22656146 0.15114388 0.         0.31990227]
[08:58:32] Blending: no score update. Terminated

[08:58:32] Automl preset training completed in 230.64 seconds

[08:58:32] Model description:
Final prediction for new objects (level 0) =
         0.30239 * (5 averaged models Lvl_0_Pipe_0_Mod_0_LinearL2) +
         0.22656 * (5 averaged models Lvl_0_Pipe_1_Mod_0_LightGBM) +
         0.15114 * (5 averaged models Lvl_0_Pipe_1_Mod_1_Tuned_LightGBM) +
         0.31990 * (5 averaged models Lvl_0_Pipe_1_Mod_3_Tuned_CatBoost)

CPU times: user 14min 16s, sys: 1min 42s, total: 15min 58s
Wall time: 3min 55s
```

The report will be available in the folder with specified name (`output_path` argument in `ReportDeco` initialization).

[22]:
```
!ls tabularAutoML_model_report
```

```
feature_importance.png           valid_pr_curve.png
lama_interactive_report.html     valid_preds_distribution_by_bins.png
valid_distribution_of_logits.png  valid_roc_curve.png
valid_pie_f1_metric.png
```

[23]:
```
%%time

test_predictions = automl_rd.predict(test_data)
print(f'Prediction for test_data:\n{test_predictions}\nShape = {test_predictions.shape}')
```

```
Prediction for test_data:
array([[0.06522178],
       [0.06859127],
       [0.0340898 ],
       ...,
       [0.0531715 ],
       [0.04148169],
       [0.18272203]], dtype=float32)
Shape = (2000, 1)
CPU times: user 14.1 s, sys: 11.2 s, total: 25.3 s
Wall time: 3.71 s
```

[24]:
```
print(f'OOF score: {roc_auc_score(train_data[TARGET_NAME].values, out_of_fold_
↪predictions.data[:, 0])}')
print(f'HOLDOUT score: {roc_auc_score(test_data[TARGET_NAME].values, test_predictions.
↪data[:, 0])}')
```

```
OOF score: 0.7501702390830668
HOLDOUT score: 0.7314741847826086
```

4.2 Feature importances calculation

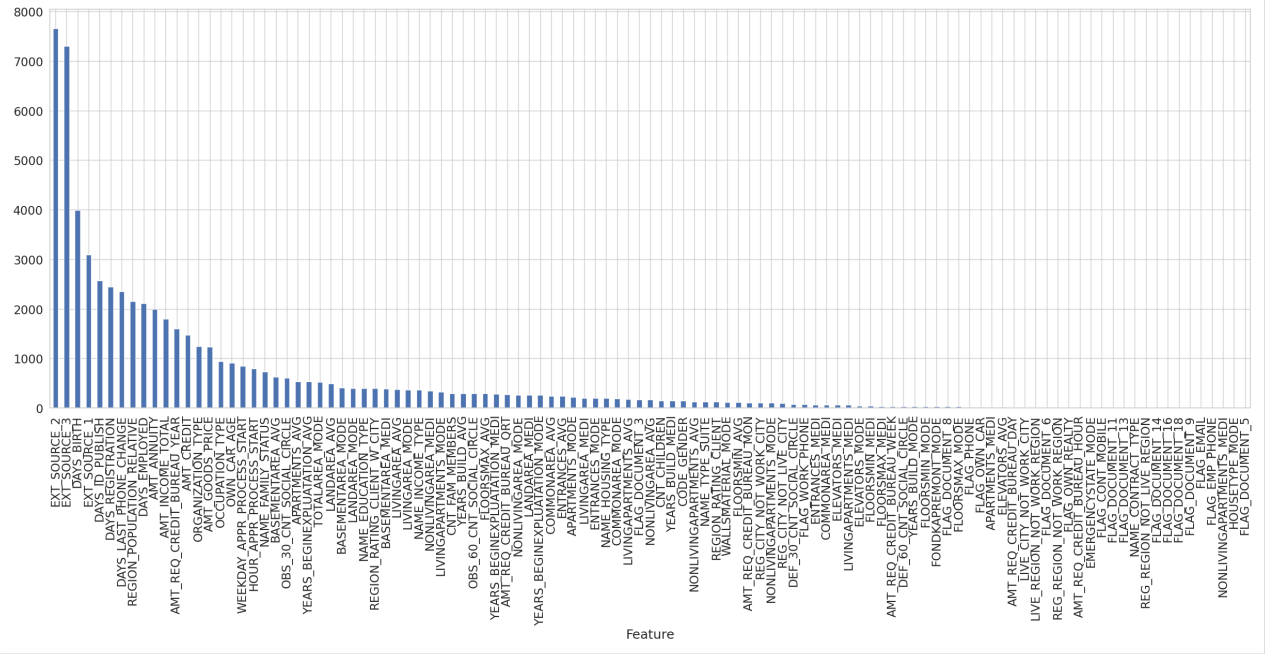
For feature importances calculation we have 2 different methods in LightAutoML: - Fast (`fast`) - this method uses feature importances from feature selector LGBM model inside LightAutoML. It works extremely fast and almost always (almost because of situations, when feature selection is turned off or selector was removed from the final models with all GBM models). There is no need to use new labelled data. - Accurate (`accurate`) - this method calculate *features permutation importances* for the whole LightAutoML model based on the **new labelled data**. It always works but can take a lot of time to finish (depending on the model structure, new labelled dataset size etc.).

In the cell below we will use `automl_rd.model` instead `automl_rd` because we want to take the importances from the model, not from the report. But **be careful** - everything, which is calculated using `automl_rd.model` will not go to the report.

[25]:
```
%%time

# Fast feature importances calculation
fast_fi = automl_rd.model.get_feature_scores('fast')
fast_fi.set_index('Feature')['Importance'].plot.bar(figsize = (30, 10), grid = True)
```

```
CPU times: user 397 ms, sys: 120 ms, total: 517 ms
Wall time: 284 ms
```

[25]:
```
<Axes: xlabel='Feature'>
```

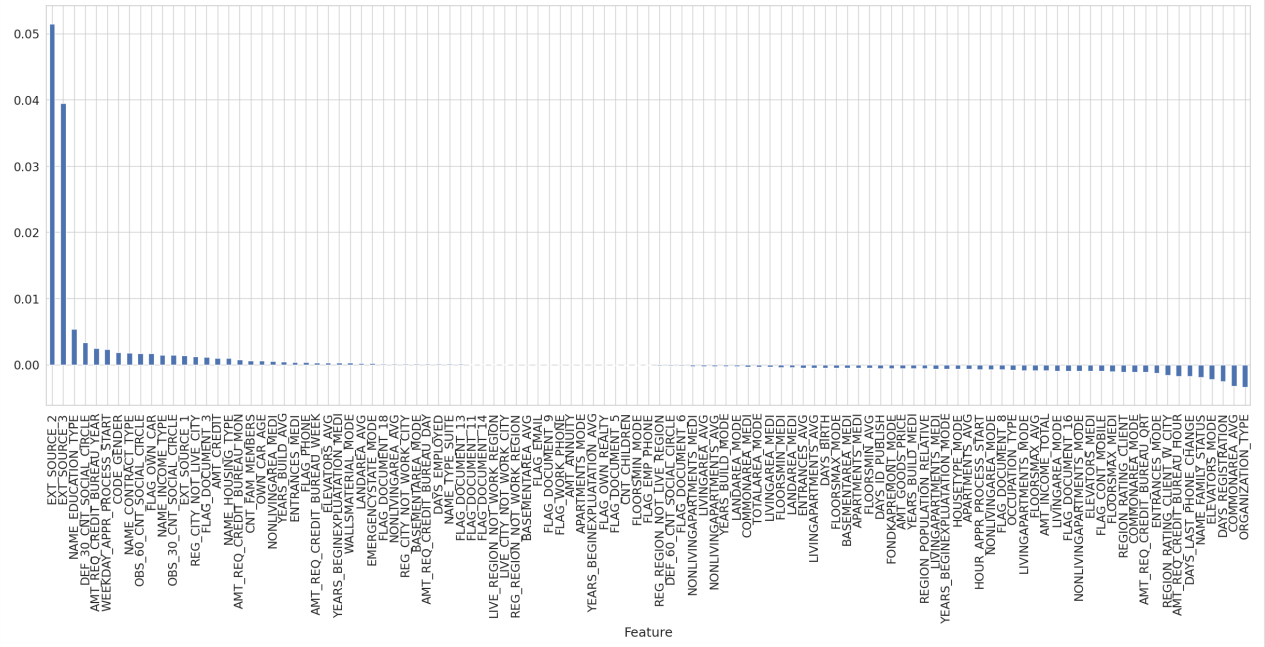


[26]:
```
%%time

# Accurate feature importances calculation with detailed info (Permutation importances) -
↪  can take long time to calculate
accurate_fi = automl_rd.model.get_feature_scores('accurate', test_data, silent = True)
```

```
CPU times: user 3min 18s, sys: 15.8 s, total: 3min 33s
Wall time: 1min 14s
```

[27]:
```
accurate_fi.set_index('Feature')['Importance'].plot.bar(figsize = (30, 10), grid = True)
```

[27]:
```
<Axes: xlabel='Feature'>
```



4.3 Shapley values

[28]:
```
explainer = SSWARM(automl_rd.model)
shap_values = explainer.shap_values(test_data, n_jobs=N_THREADS)
```

```
100%|| 3000/3000 [03:42<00:00, 13.46it/s]
```

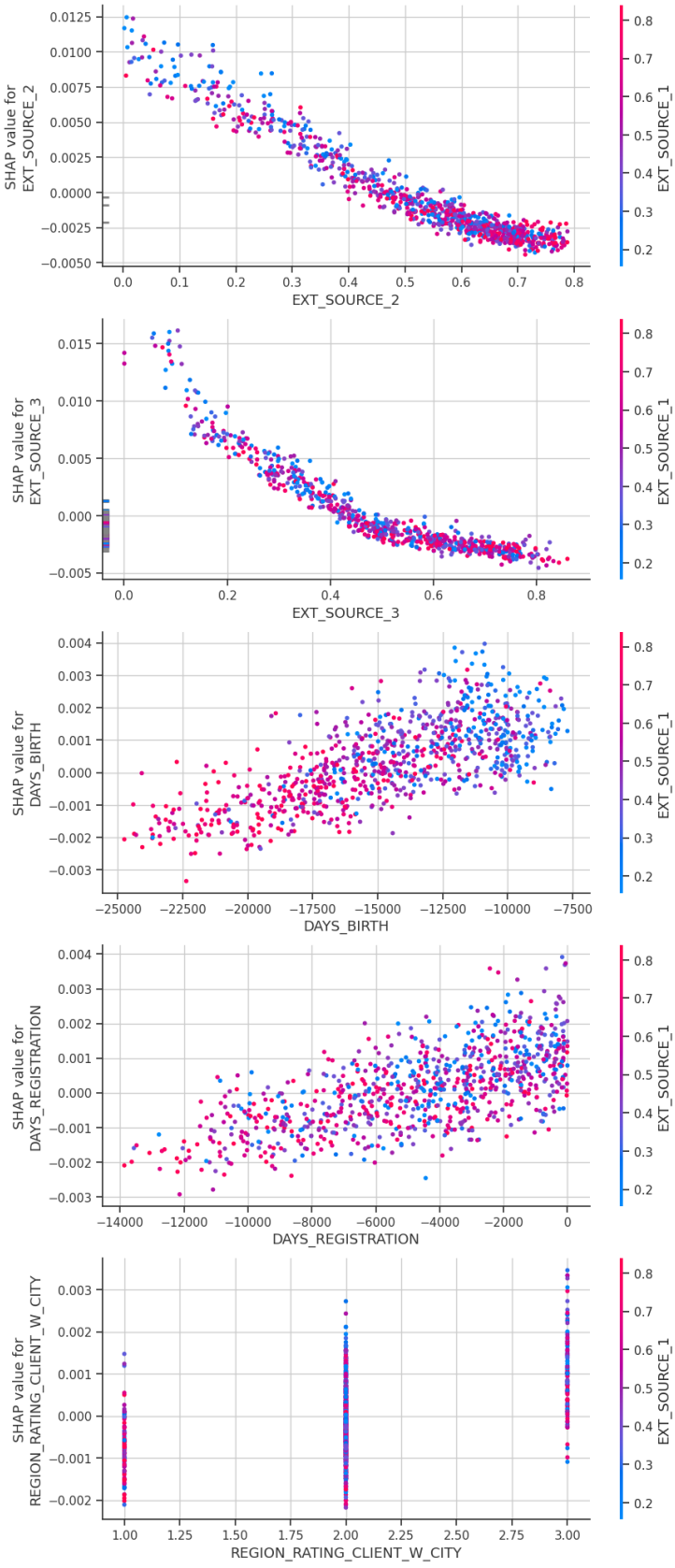
[29]:
```
# summary plot

# !pip install shap
import shap
shap.summary_plot(shap_values[1], test_data[explainer.used_feats])
```

[30]:
```
# dependence plots
import matplotlib.pyplot as plt

feats = ["EXT_SOURCE_2", "EXT_SOURCE_3", "DAYS_BIRTH", "DAYS_REGISTRATION", "REGION_
↪RATING_CLIENT_W_CITY"]
```

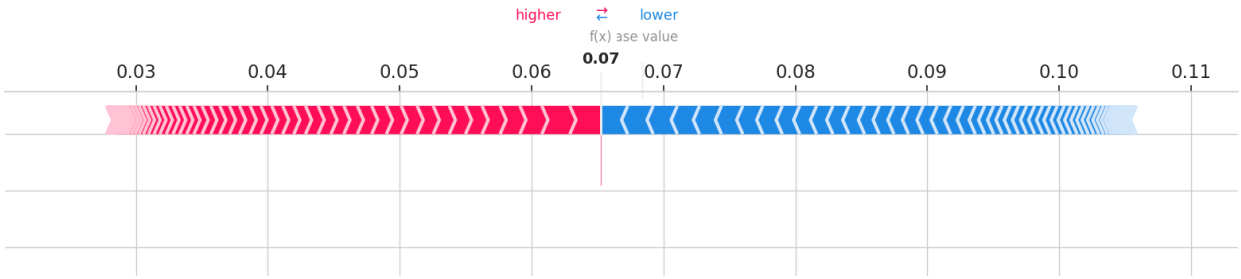
(continues on next page)

(continued from previous page)

```
fig, ax = plt.subplots(nrows=len(feats), ncols=1, figsize=(10, 5*len(feats)))

for i, feat in enumerate(feats):
    shap.dependence_plot(feat, shap_values[1], test_data[explainer.used_feats],
                         show=False, ax=ax[i], interaction_index="EXT_SOURCE_1")
```


SHAP value for EXT_SOURCE_2
EXT_SOURCE_2
EXT_SOURCE_1
SHAP value for EXT_SOURCE_3
EXT_SOURCE_3
EXT_SOURCE_1
SHAP value for DAYS_BIRTH
DAYS_BIRTH
EXT_SOURCE_1
SHAP value for DAYS_REGISTRATION
DAYS_REGISTRATION
EXT_SOURCE_1
SHAP value for REGION_RATING_CLIENT_W_CITY
REGION_RATING_CLIENT_W_CITY
EXT_SOURCE_1

[42]:
```
# individual force plot
shap.force_plot(explainer.expected_value[1], shap_values[1][0], test_data[explainer.used_
↪feats].iloc[0], matplotlib=True)
```



[43]:
```
# overall force plot
shap.initjs()
shap.force_plot(explainer.expected_value[1], shap_values[1][:500], test_data[explainer.
↪used_feats].iloc[:500])
```

```
<IPython.core.display.HTML object>
```

[43]:
```
<shap.plots._force.AdditiveForceArrayVisualizer at 0x7fd1db309280>
```

Bonus: where is the automatic report?

As we used `automl_rd` in our training and prediction cells, it is already ready in the folder we specified - you can check the output folder and find the `tabularAutoML_model_report` folder with `lama_interactive_report.html` report inside (or just click this link for short). It's interactive so you can click the black triangles on the left of the texts to go deeper in selected part.

5. Spending more from TIMEOUT - `TabularUtilizedAutoML` usage

To spend (almost) all the `TIMEOUT` time for building the model we can use `TabularUtilizedAutoML` preset instead of `TabularAutoML`, which has the same API. By default `TabularUtilizedAutoML` model trains with 7 different parameter configurations (see this for more details) sequentially, and if there is time left, then whole AutoML pipeline with these configs is run again with another cross-validation seed, and so on. Then results for each pipeline model are averaged over the considered validation seeds, and all averaged results at the end are also combined through blending. User can set his own set of configs by passing list of paths to according files in `configs_list` argument during `TabularUtilizedAutoML` instance initialization. Such configs allow the user to configure all pipeline parameters and can be used for any available preset.

[44]:
```
utilized_automl = TabularUtilizedAutoML(
    task = task,
    timeout = 900,
    cpu_limit = N_THREADS,
    reader_params = {'n_jobs': N_THREADS, 'cv': N_FOLDS, 'random_state': RANDOM_STATE},
)
```

[45]:
```
%%time

out_of_fold_predictions = utilized_automl.fit_predict(train_data, roles = roles, verbose␣
↪= 1)
```

```
[09:05:31] Start automl utilizator with listed constraints:
[09:05:31] - time: 900.00 seconds
[09:05:31] - CPU: 4 cores
[09:05:31] - memory: 16 GB

[09:05:31] If one preset completes earlier, next preset configuration will be started

[09:05:31] ==================================================
[09:05:31] Start 0 automl preset configuration:
[09:05:31] conf_0_sel_type_0.yml, random state: {'reader_params': {'random_state': 42},
↪'nn_params': {'random_state': 42}, 'general_params': {'return_all_predictions': False}}
[09:05:31] Stdout logging level is INFO.
[09:05:31] Task: binary

[09:05:31] Start automl preset with listed constraints:
[09:05:31] - time: 900.00 seconds
[09:05:31] - CPU: 4 cores
[09:05:31] - memory: 16 GB

[09:05:31] Train data shape: (8000, 122)

[09:05:37] Layer 1 train process start. Time left 894.60 secs
[09:05:38] Start fitting Lvl_0_Pipe_0_Mod_0_LinearL2 ...
[09:05:42] Fitting Lvl_0_Pipe_0_Mod_0_LinearL2 finished. score = 0.7351718729105452
[09:05:42] Lvl_0_Pipe_0_Mod_0_LinearL2 fitting and predicting completed
[09:05:42] Time left 889.74 secs

[09:05:43] Start fitting Lvl_0_Pipe_1_Mod_0_LightGBM ...
[09:06:01] Fitting Lvl_0_Pipe_1_Mod_0_LightGBM finished. score = 0.7208359669101568
[09:06:01] Lvl_0_Pipe_1_Mod_0_LightGBM fitting and predicting completed
[09:06:01] Start hyperparameters optimization for Lvl_0_Pipe_1_Mod_1_Tuned_LightGBM ...␣
↪Time budget is 92.70 secs
[09:07:35] Hyperparameters optimization for Lvl_0_Pipe_1_Mod_1_Tuned_LightGBM completed
[09:07:35] Start fitting Lvl_0_Pipe_1_Mod_1_Tuned_LightGBM ...
[09:07:39] Fitting Lvl_0_Pipe_1_Mod_1_Tuned_LightGBM finished. score = 0.7387079347889174
[09:07:39] Lvl_0_Pipe_1_Mod_1_Tuned_LightGBM fitting and predicting completed
[09:07:40] Start fitting Lvl_0_Pipe_1_Mod_2_CatBoost ...
[09:07:46] Fitting Lvl_0_Pipe_1_Mod_2_CatBoost finished. score = 0.7216183332238446
[09:07:46] Lvl_0_Pipe_1_Mod_2_CatBoost fitting and predicting completed
[09:07:46] Start hyperparameters optimization for Lvl_0_Pipe_1_Mod_3_Tuned_CatBoost ...␣
↪Time budget is 300.00 secs
[09:10:12] Hyperparameters optimization for Lvl_0_Pipe_1_Mod_3_Tuned_CatBoost completed
[09:10:12] Start fitting Lvl_0_Pipe_1_Mod_3_Tuned_CatBoost ...
[09:10:18] Fitting Lvl_0_Pipe_1_Mod_3_Tuned_CatBoost finished. score = 0.7400775010369544
[09:10:18] Lvl_0_Pipe_1_Mod_3_Tuned_CatBoost fitting and predicting completed
[09:10:18] Time left 613.68 secs

[09:10:18] Layer 1 training completed.

[09:10:18] Blending: optimization starts with equal weights and score 0.7511137558494106
[09:10:18] Blending: iteration 0: score = 0.7520976665286894, weights = [0.24006502 0.
↪11192521 0.24787743 0.09496131 0.305171  ]
[09:10:18] Blending: iteration 1: score = 0.7521652731829703, weights = [0.24298649 0.
```

(continues on next page)

(continued from previous page)

```
↪07254668 0.27720243 0.08237854 0.32488596]
[09:10:18] Blending: iteration 2: score = 0.7521720763683065, weights = [0.25121042 0.
↪07175855 0.274191   0.0814836  0.32135648]
[09:10:18] Blending: iteration 3: score = 0.7521720763683065, weights = [0.25121042 0.
↪07175854 0.274191   0.08148359 0.32135645]
[09:10:18] Blending: no score update. Terminated

[09:10:18] Automl preset training completed in 286.84 seconds

[09:10:18] Model description:
Final prediction for new objects (level 0) =
         0.25121 * (5 averaged models Lvl_0_Pipe_0_Mod_0_LinearL2) +
         0.07176 * (5 averaged models Lvl_0_Pipe_1_Mod_0_LightGBM) +
         0.27419 * (5 averaged models Lvl_0_Pipe_1_Mod_1_Tuned_LightGBM) +
         0.08148 * (5 averaged models Lvl_0_Pipe_1_Mod_2_CatBoost) +
         0.32136 * (5 averaged models Lvl_0_Pipe_1_Mod_3_Tuned_CatBoost)

[09:10:18] ==================================================
[09:10:18] Start 1 automl preset configuration:
[09:10:18] conf_1_sel_type_1.yml, random state: {'reader_params': {'random_state': 43},
↪'nn_params': {'random_state': 43}, 'general_params': {'return_all_predictions': False}}
[09:10:18] Stdout logging level is INFO.
[09:10:18] Task: binary

[09:10:18] Start automl preset with listed constraints:
[09:10:18] - time: 613.07 seconds
[09:10:18] - CPU: 4 cores
[09:10:18] - memory: 16 GB

[09:10:18] Train data shape: (8000, 122)

[09:10:20] Layer 1 train process start. Time left 611.60 secs
[09:10:21] Start fitting Lvl_0_Pipe_0_Mod_0_LinearL2 ...
[09:10:25] Fitting Lvl_0_Pipe_0_Mod_0_LinearL2 finished. score = 0.7343900380957119
[09:10:25] Lvl_0_Pipe_0_Mod_0_LinearL2 fitting and predicting completed
[09:10:25] Time left 606.69 secs

[09:10:28] Selector_LightGBM fitting and predicting completed
[09:10:29] Start fitting Lvl_0_Pipe_1_Mod_0_LightGBM ...
[09:10:49] Fitting Lvl_0_Pipe_1_Mod_0_LightGBM finished. score = 0.7440739472230141
[09:10:49] Lvl_0_Pipe_1_Mod_0_LightGBM fitting and predicting completed
[09:10:49] Start hyperparameters optimization for Lvl_0_Pipe_1_Mod_1_Tuned_LightGBM ...␣
↪Time budget is 38.55 secs
[09:11:32] Hyperparameters optimization for Lvl_0_Pipe_1_Mod_1_Tuned_LightGBM completed
[09:11:32] Start fitting Lvl_0_Pipe_1_Mod_1_Tuned_LightGBM ...
[09:11:49] Fitting Lvl_0_Pipe_1_Mod_1_Tuned_LightGBM finished. score = 0.7309214765718494
[09:11:49] Lvl_0_Pipe_1_Mod_1_Tuned_LightGBM fitting and predicting completed
[09:11:49] Start fitting Lvl_0_Pipe_1_Mod_2_CatBoost ...
[09:11:53] Fitting Lvl_0_Pipe_1_Mod_2_CatBoost finished. score = 0.7073296243217277
[09:11:53] Lvl_0_Pipe_1_Mod_2_CatBoost fitting and predicting completed
[09:11:53] Start hyperparameters optimization for Lvl_0_Pipe_1_Mod_3_Tuned_CatBoost ...␣
↪Time budget is 300.00 secs
```

(continues on next page)

(continued from previous page)

```
[09:14:42] Hyperparameters optimization for Lvl_0_Pipe_1_Mod_3_Tuned_CatBoost completed
[09:14:42] Start fitting Lvl_0_Pipe_1_Mod_3_Tuned_CatBoost ...
[09:14:51] Fitting Lvl_0_Pipe_1_Mod_3_Tuned_CatBoost finished. score = 0.7434170146389666
[09:14:51] Lvl_0_Pipe_1_Mod_3_Tuned_CatBoost fitting and predicting completed
[09:14:51] Time left 340.03 secs

[09:14:51] Layer 1 training completed.

[09:14:51] Blending: optimization starts with equal weights and score 0.751036369616209
[09:14:52] Blending: iteration 0: score = 0.7539285737823521, weights = [0.10462688 0.
↪3023047  0.40002117 0.05180185 0.14124545]
[09:14:52] Blending: iteration 1: score = 0.7542955205914349, weights = [0.1847324  0.
↪21803026 0.4247732  0.         0.17246413]
[09:14:52] Blending: iteration 2: score = 0.754351009071835, weights = [0.16244903 0.
↪24964364 0.41634628 0.         0.171561  ]
[09:14:52] Blending: iteration 3: score = 0.754351009071835, weights = [0.16244903 0.
↪24964364 0.41634628 0.         0.171561  ]
[09:14:52] Blending: no score update. Terminated

[09:14:52] Automl preset training completed in 273.55 seconds

[09:14:52] Model description:
Final prediction for new objects (level 0) =
         0.16245 * (5 averaged models Lvl_0_Pipe_0_Mod_0_LinearL2) +
         0.24964 * (5 averaged models Lvl_0_Pipe_1_Mod_0_LightGBM) +
         0.41635 * (5 averaged models Lvl_0_Pipe_1_Mod_1_Tuned_LightGBM) +
         0.17156 * (5 averaged models Lvl_0_Pipe_1_Mod_3_Tuned_CatBoost)

[09:14:52] ==================================================
[09:14:52] Start 2 automl preset configuration:
[09:14:52] conf_2_select_mode_1_no_typ.yml, random state: {'reader_params': {'random_
↪state': 44}, 'nn_params': {'random_state': 44}, 'general_params': {'return_all_
↪predictions': False}}
[09:14:52] Stdout logging level is INFO.
[09:14:52] Task: binary

[09:14:52] Start automl preset with listed constraints:
[09:14:52] - time: 339.45 seconds
[09:14:52] - CPU: 4 cores
[09:14:52] - memory: 16 GB

[09:14:52] Train data shape: (8000, 122)

[09:14:52] Layer 1 train process start. Time left 339.28 secs
[09:14:53] Start fitting Lvl_0_Pipe_0_Mod_0_LinearL2 ...
[09:14:58] Fitting Lvl_0_Pipe_0_Mod_0_LinearL2 finished. score = 0.7370140479399211
[09:14:58] Lvl_0_Pipe_0_Mod_0_LinearL2 fitting and predicting completed
[09:14:58] Time left 333.87 secs

[09:15:01] Selector_LightGBM fitting and predicting completed
[09:15:02] Start fitting Lvl_0_Pipe_1_Mod_0_LightGBM ...
[09:15:20] Fitting Lvl_0_Pipe_1_Mod_0_LightGBM finished. score = 0.7306162899296487
```

(continues on next page)

(continued from previous page)

```
[09:15:20] Lvl_0_Pipe_1_Mod_0_LightGBM fitting and predicting completed
[09:15:20] Start hyperparameters optimization for Lvl_0_Pipe_1_Mod_1_Tuned_LightGBM ... Time budget is 1.00 secs
[09:15:33] Hyperparameters optimization for Lvl_0_Pipe_1_Mod_1_Tuned_LightGBM completed
[09:15:33] Start fitting Lvl_0_Pipe_1_Mod_1_Tuned_LightGBM ...
[09:15:50] Fitting Lvl_0_Pipe_1_Mod_1_Tuned_LightGBM finished. score = 0.7192733602781992
[09:15:50] Lvl_0_Pipe_1_Mod_1_Tuned_LightGBM fitting and predicting completed
[09:15:51] Start fitting Lvl_0_Pipe_1_Mod_2_CatBoost ...
[09:15:57] Fitting Lvl_0_Pipe_1_Mod_2_CatBoost finished. score = 0.7293459013678443
[09:15:57] Lvl_0_Pipe_1_Mod_2_CatBoost fitting and predicting completed
[09:15:57] Start hyperparameters optimization for Lvl_0_Pipe_1_Mod_3_Tuned_CatBoost ... Time budget is 179.94 secs
[09:18:45] Hyperparameters optimization for Lvl_0_Pipe_1_Mod_3_Tuned_CatBoost completed
[09:18:45] Start fitting Lvl_0_Pipe_1_Mod_3_Tuned_CatBoost ...
[09:18:54] Fitting Lvl_0_Pipe_1_Mod_3_Tuned_CatBoost finished. score = 0.7519280120943626
[09:18:54] Lvl_0_Pipe_1_Mod_3_Tuned_CatBoost fitting and predicting completed
[09:18:54] Time left 97.01 secs

[09:18:54] Layer 1 training completed.

[09:18:54] Blending: optimization starts with equal weights and score 0.7553933846250989
[09:18:55] Blending: iteration 0: score = 0.7573044419060059, weights = [0.18904768 0. 
↪10002336 0.42633003 0.         0.28459892]
[09:18:55] Blending: iteration 1: score = 0.7573339932423109, weights = [0.19110394 0. 
↪09505802 0.4097812  0.         0.3040569 ]
[09:18:55] Blending: iteration 2: score = 0.7573339932423109, weights = [0.19110394 0. 
↪09505802 0.40978116 0.         0.3040569 ]
[09:18:55] Blending: no score update. Terminated

[09:18:55] Automl preset training completed in 242.82 seconds

[09:18:55] Model description:
Final prediction for new objects (level 0) =
         0.19110 * (5 averaged models Lvl_0_Pipe_0_Mod_0_LinearL2) +
         0.09506 * (5 averaged models Lvl_0_Pipe_1_Mod_0_LightGBM) +
         0.40978 * (5 averaged models Lvl_0_Pipe_1_Mod_1_Tuned_LightGBM) +
         0.30406 * (5 averaged models Lvl_0_Pipe_1_Mod_3_Tuned_CatBoost)

[09:18:55] ==================================================
[09:18:55] Blending: optimization starts with equal weights and score 0.7599013453086403
[09:18:55] Blending: iteration 0: score = 0.7607196409448859, weights = [0.10080308 0. 
↪3555537  0.5436432 ]
[09:18:55] Blending: iteration 1: score = 0.7607196409448859, weights = [0.10080308 0. 
↪3555537  0.5436432 ]
[09:18:55] Blending: no score update. Terminated

CPU times: user 48min 10s, sys: 4min 52s, total: 53min 2s
Wall time: 13min 23s
```

[46]:
```
print('out_of_fold_predictions:\n{}\nShape = {}'.format(out_of_fold_predictions, out_of_
↪fold_predictions.shape))
```

```
out_of_fold_predictions:
array([[0.02990189],
       [0.01378681],
       [0.02714774],
       ...,
       [0.02049532],
       [0.13665317],
       [0.07793609]], dtype=float32)
Shape = (8000, 1)
```

[47]:
```
print(utilized_automl.create_model_str_desc())
```

```
Final prediction for new objects =
        0.10080 * 1 averaged models with config = "conf_0_sel_type_0.yml" and different␣
↪CV random_states. Their structures:

            Model #0.
               ␣
↪================================================================================
                Final prediction for new objects (level 0) =
                         0.25121 * (5 averaged models Lvl_0_Pipe_0_Mod_0_LinearL2) +
                         0.07176 * (5 averaged models Lvl_0_Pipe_1_Mod_0_LightGBM) +
                         0.27419 * (5 averaged models Lvl_0_Pipe_1_Mod_1_Tuned_LightGBM)␣
↪+
                         0.08148 * (5 averaged models Lvl_0_Pipe_1_Mod_2_CatBoost) +
                         0.32136 * (5 averaged models Lvl_0_Pipe_1_Mod_3_Tuned_CatBoost)
               ␣
↪================================================================================


        + 0.35555 * 1 averaged models with config = "conf_1_sel_type_1.yml" and␣
↪different CV random_states. Their structures:

            Model #0.
               ␣
↪================================================================================
                Final prediction for new objects (level 0) =
                         0.16245 * (5 averaged models Lvl_0_Pipe_0_Mod_0_LinearL2) +
                         0.24964 * (5 averaged models Lvl_0_Pipe_1_Mod_0_LightGBM) +
                         0.41635 * (5 averaged models Lvl_0_Pipe_1_Mod_1_Tuned_LightGBM)␣
↪+
                         0.17156 * (5 averaged models Lvl_0_Pipe_1_Mod_3_Tuned_CatBoost)
               ␣
↪================================================================================


        + 0.54364 * 1 averaged models with config = "conf_2_select_mode_1_no_typ.yml"␣
↪and different CV random_states. Their structures:

            Model #0.
               ␣
↪================================================================================
                Final prediction for new objects (level 0) =
```

(continues on next page)

(continued from previous page)

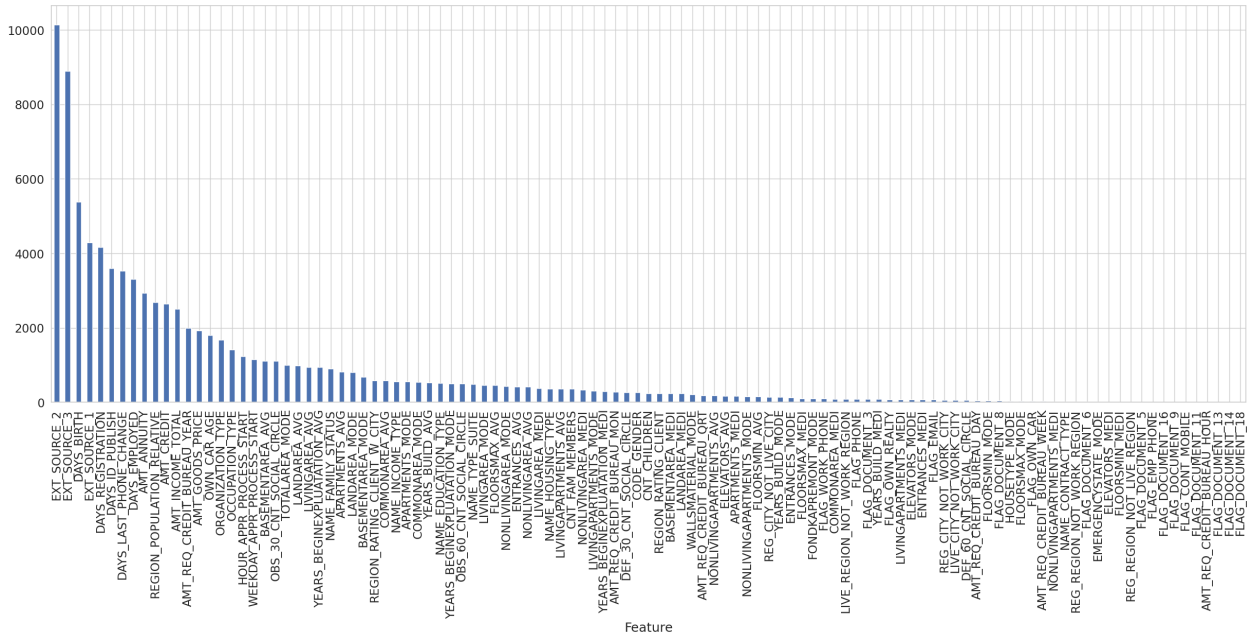
```
                        0.19110 * (5 averaged models Lvl_0_Pipe_0_Mod_0_LinearL2) +
                        0.09506 * (5 averaged models Lvl_0_Pipe_1_Mod_0_LightGBM) +
                        0.40978 * (5 averaged models Lvl_0_Pipe_1_Mod_1_Tuned_LightGBM)␣
↪+
                        0.30406 * (5 averaged models Lvl_0_Pipe_1_Mod_3_Tuned_CatBoost)

               ␣
↪==============================================================================
```

Feature importances calculation for `TabularUtilizedAutoML`:

[48]:
```
%%time

# Fast feature importances calculation
fast_fi = utilized_automl.get_feature_scores('fast', silent=False)
fast_fi.set_index('Feature')['Importance'].plot.bar(figsize = (30, 10), grid = True)
```

```
CPU times: user 4.27 s, sys: 3.49 s, total: 7.77 s
Wall time: 690 ms
```

[48]:
```
<Axes: xlabel='Feature'>
```

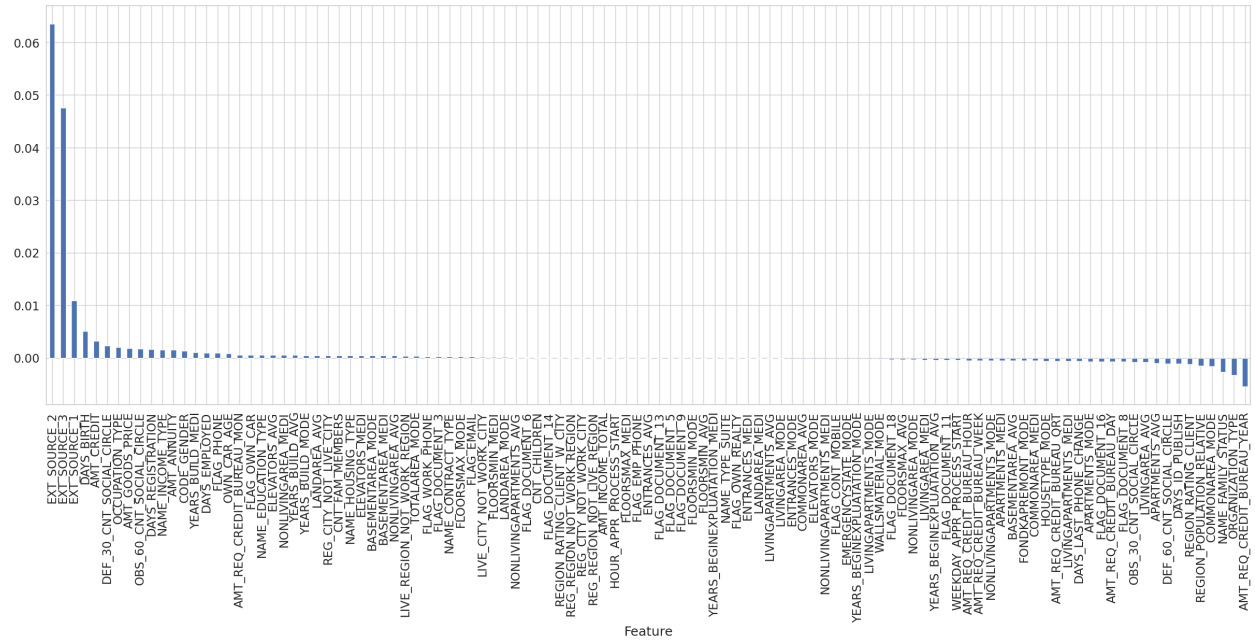


Note that in `TabularUtilizedAutoML` the first config doesn't have a LGBM feature selector (but second one already has it), so if there is enough time only for training with the first config, then `'fast'` feature importance calculation method won't work. `'accurate'` method will still work correctly.

[49]:
```
%%time

# Accurate feature importances calculation
fast_fi = utilized_automl.get_feature_scores('accurate', test_data, silent=True)
fast_fi.set_index('Feature')['Importance'].plot.bar(figsize = (30, 10), grid = True)
```

```
CPU times: user 13min 54s, sys: 1min 6s, total: 15min
Wall time: 4min 36s
```

[49]:
```
<Axes: xlabel='Feature'>
```



Prediction on holdout and metric calculation:

[50]:
```
%%time

test_predictions = utilized_automl.predict(test_data)
print(f'Prediction for test_data:\n{test_predictions}\nShape = {test_predictions.shape}')
```

```
Prediction for test_data:
array([[0.04470912],
       [0.05975739],
       [0.01950551],
       ...,
       [0.03931152],
       [0.02568556],
       [0.18183662]], dtype=float32)
Shape = (2000, 1)
CPU times: user 7.15 s, sys: 149 ms, total: 7.3 s
Wall time: 2.53 s
```

[51]:
```
print(f'OOF score: {roc_auc_score(train_data[TARGET_NAME].values, out_of_fold_
↪predictions.data[:, 0])}')
print(f'HOLDOUT score: {roc_auc_score(test_data[TARGET_NAME].values, test_predictions.
↪data[:, 0])}')
```

```
OOF score: 0.7607196409448859
HOLDOUT score: 0.7316508152173913
```

5.1 Report

[13]:
```
RDU = ReportDecoUtilized(output_path = 'tabularUtilizedAutoML_model_report')

automl_rdu = RDU(
    TabularUtilizedAutoML(
        task = task,
        timeout = 900,
        cpu_limit = N_THREADS,
        reader_params = {'n_jobs': N_THREADS, 'cv': N_FOLDS, 'random_state': RANDOM_
↪STATE},
    )
)
```

[14]:
```
%%time
out_of_fold_predictions = automl_rdu.fit_predict(train_data, roles = roles, verbose = 1)
```

```
[10:00:44] Start automl utilizator with listed constraints:
[10:00:44] - time: 900.00 seconds
[10:00:44] - CPU: 4 cores
[10:00:44] - memory: 16 GB

[10:00:44] If one preset completes earlier, next preset configuration will be started

[10:00:44] ================================================
[10:00:44] Start 0 automl preset configuration:
[10:00:44] conf_0_sel_type_0.yml, random state: {'reader_params': {'random_state': 42},
↪'nn_params': {'random_state': 42}, 'general_params': {'return_all_predictions': False}}
[10:00:44] Stdout logging level is INFO.
[10:00:44] Task: binary

[10:00:44] Start automl preset with listed constraints:
[10:00:44] - time: 900.00 seconds
[10:00:44] - CPU: 4 cores
[10:00:44] - memory: 16 GB

[10:00:44] Train data shape: (8000, 122)

[10:00:46] Layer 1 train process start. Time left 897.88 secs
[10:00:46] Start fitting Lvl_0_Pipe_0_Mod_0_LinearL2 ...
[10:00:47] Fitting Lvl_0_Pipe_0_Mod_0_LinearL2 finished. score = 0.735187286377323
[10:00:47] Lvl_0_Pipe_0_Mod_0_LinearL2 fitting and predicting completed
[10:00:47] Time left 896.30 secs

[10:00:48] Start fitting Lvl_0_Pipe_1_Mod_0_LightGBM ...
[10:00:57] Fitting Lvl_0_Pipe_1_Mod_0_LightGBM finished. score = 0.7208359669101568
[10:00:57] Lvl_0_Pipe_1_Mod_0_LightGBM fitting and predicting completed
[10:00:57] Start hyperparameters optimization for Lvl_0_Pipe_1_Mod_1_Tuned_LightGBM ...␣
↪Time budget is 154.80 secs
[10:00:57] Copying TaskTimer may affect the parent PipelineTimer, so copy will create␣
↪new unlimited TaskTimer
[10:03:33] Hyperparameters optimization for Lvl_0_Pipe_1_Mod_1_Tuned_LightGBM completed
[10:03:33] Start fitting Lvl_0_Pipe_1_Mod_1_Tuned_LightGBM ...
[10:03:38] Fitting Lvl_0_Pipe_1_Mod_1_Tuned_LightGBM finished. score = 0.6872182391698073
```

(continues on next page)

(continued from previous page)

```
[10:03:38] Lvl_0_Pipe_1_Mod_1_Tuned_LightGBM fitting and predicting completed
[10:03:38] Start fitting Lvl_0_Pipe_1_Mod_2_CatBoost ...
[10:03:44] Fitting Lvl_0_Pipe_1_Mod_2_CatBoost finished. score = 0.7203027672594154
[10:03:44] Lvl_0_Pipe_1_Mod_2_CatBoost fitting and predicting completed
[10:03:44] Start hyperparameters optimization for Lvl_0_Pipe_1_Mod_3_Tuned_CatBoost ...␣
↪Time budget is 300.00 secs
[10:05:50] Hyperparameters optimization for Lvl_0_Pipe_1_Mod_3_Tuned_CatBoost completed
[10:05:50] Start fitting Lvl_0_Pipe_1_Mod_3_Tuned_CatBoost ...
[10:05:58] Fitting Lvl_0_Pipe_1_Mod_3_Tuned_CatBoost finished. score = 0.7436991342308862
[10:05:58] Lvl_0_Pipe_1_Mod_3_Tuned_CatBoost fitting and predicting completed
[10:05:58] Time left 585.48 secs

[10:05:58] Layer 1 training completed.

[10:05:58] Blending: optimization starts with equal weights and score 0.7474317443856182
[10:05:58] Blending: iteration 0: score = 0.7499482851614662, weights = [0.28868666 0.
↪19095857 0.06477723 0.08807771 0.36749983]
[10:05:58] Blending: iteration 1: score = 0.7499771986991458, weights = [0.3108266  0.
↪17629829 0.05980414 0.0813158  0.37175515]
[10:05:58] Blending: iteration 2: score = 0.749987934976005, weights = [0.30671465 0.
↪1773502  0.06016096 0.08180097 0.37397328]
[10:05:58] Blending: iteration 3: score = 0.749987934976005, weights = [0.30671465 0.
↪1773502  0.06016096 0.08180097 0.37397328]
[10:05:58] Blending: no score update. Terminated

[10:05:58] Automl preset training completed in 314.89 seconds

[10:05:58] Model description:
Final prediction for new objects (level 0) =
         0.30671 * (5 averaged models Lvl_0_Pipe_0_Mod_0_LinearL2) +
         0.17735 * (5 averaged models Lvl_0_Pipe_1_Mod_0_LightGBM) +
         0.06016 * (5 averaged models Lvl_0_Pipe_1_Mod_1_Tuned_LightGBM) +
         0.08180 * (5 averaged models Lvl_0_Pipe_1_Mod_2_CatBoost) +
         0.37397 * (5 averaged models Lvl_0_Pipe_1_Mod_3_Tuned_CatBoost)

[10:05:58] ================================================
[10:05:58] Start 1 automl preset configuration:
[10:05:58] conf_1_sel_type_1.yml, random state: {'reader_params': {'random_state': 43},
↪'nn_params': {'random_state': 43}, 'general_params': {'return_all_predictions': False}}
[10:05:59] Stdout logging level is INFO.
[10:05:59] Task: binary

[10:05:59] Start automl preset with listed constraints:
[10:05:59] - time: 585.08 seconds
[10:05:59] - CPU: 4 cores
[10:05:59] - memory: 16 GB

[10:05:59] Train data shape: (8000, 122)

[10:06:01] Layer 1 train process start. Time left 582.60 secs
[10:06:01] Start fitting Lvl_0_Pipe_0_Mod_0_LinearL2 ...
[10:06:04] Fitting Lvl_0_Pipe_0_Mod_0_LinearL2 finished. score = 0.7343764317250391
```

(continues on next page)

(continued from previous page)

```
[10:06:04] Lvl_0_Pipe_0_Mod_0_LinearL2 fitting and predicting completed
[10:06:04] Time left 579.34 secs

[10:06:07] Selector_LightGBM fitting and predicting completed
[10:06:08] Start fitting Lvl_0_Pipe_1_Mod_0_LightGBM ...
[10:06:27] Fitting Lvl_0_Pipe_1_Mod_0_LightGBM finished. score = 0.7440739472230141
[10:06:27] Lvl_0_Pipe_1_Mod_0_LightGBM fitting and predicting completed
[10:06:27] Start hyperparameters optimization for Lvl_0_Pipe_1_Mod_1_Tuned_LightGBM ...␣
↪Time budget is 35.96 secs
[10:07:05] Hyperparameters optimization for Lvl_0_Pipe_1_Mod_1_Tuned_LightGBM completed
[10:07:05] Start fitting Lvl_0_Pipe_1_Mod_1_Tuned_LightGBM ...
[10:07:31] Fitting Lvl_0_Pipe_1_Mod_1_Tuned_LightGBM finished. score = 0.7309214765718494
[10:07:31] Lvl_0_Pipe_1_Mod_1_Tuned_LightGBM fitting and predicting completed
[10:07:31] Start fitting Lvl_0_Pipe_1_Mod_2_CatBoost ...
[10:07:35] Fitting Lvl_0_Pipe_1_Mod_2_CatBoost finished. score = 0.7081926721615145
[10:07:35] Lvl_0_Pipe_1_Mod_2_CatBoost fitting and predicting completed
[10:07:35] Start hyperparameters optimization for Lvl_0_Pipe_1_Mod_3_Tuned_CatBoost ...␣
↪Time budget is 300.00 secs
[10:10:49] Hyperparameters optimization for Lvl_0_Pipe_1_Mod_3_Tuned_CatBoost completed
[10:10:49] Start fitting Lvl_0_Pipe_1_Mod_3_Tuned_CatBoost ...
[10:10:57] Fitting Lvl_0_Pipe_1_Mod_3_Tuned_CatBoost finished. score = 0.7361510001001345
[10:10:57] Lvl_0_Pipe_1_Mod_3_Tuned_CatBoost fitting and predicting completed
[10:10:57] Time left 286.73 secs

[10:10:57] Layer 1 training completed.

[10:10:57] Blending: optimization starts with equal weights and score 0.7497769299308051
[10:10:57] Blending: iteration 0: score = 0.7534326853511901, weights = [0.08608949 0.
↪3215845  0.44514042 0.05545893 0.09172665]
[10:10:57] Blending: iteration 1: score = 0.7539372903635644, weights = [0.14486776 0.
↪31182623 0.40830123 0.05086922 0.08413548]
[10:10:57] Blending: iteration 2: score = 0.7539333572720418, weights = [0.14514707 0.
↪3117244  0.4081679  0.05085261 0.08410801]
[10:10:57] Blending: no score update. Terminated

[10:10:57] Automl preset training completed in 298.63 seconds

[10:10:57] Model description:
Final prediction for new objects (level 0) =
         0.14487 * (5 averaged models Lvl_0_Pipe_0_Mod_0_LinearL2) +
         0.31183 * (5 averaged models Lvl_0_Pipe_1_Mod_0_LightGBM) +
         0.40830 * (5 averaged models Lvl_0_Pipe_1_Mod_1_Tuned_LightGBM) +
         0.05087 * (5 averaged models Lvl_0_Pipe_1_Mod_2_CatBoost) +
         0.08414 * (5 averaged models Lvl_0_Pipe_1_Mod_3_Tuned_CatBoost)

[10:10:57] ==================================================
[10:10:57] Blending: optimization starts with equal weights and score 0.7558984148365566
[10:10:57] Blending: iteration 0: score = 0.7565288787776545, weights = [0.3279138 0.
↪6720862]
[10:10:57] Blending: iteration 1: score = 0.7565288787776545, weights = [0.3279138 0.
↪6720862]
[10:10:57] Blending: no score update. Terminated
```

(continues on next page)

(continued from previous page)

```
CPU times: user 24min 52s, sys: 4min 41s, total: 29min 33s
Wall time: 10min 16s
```

[15]:
```
!ls tabularUtilizedAutoML_model_report
```

```
feature_importance.png                 test_roc_curve_1.png
lama_interactive_report.html           valid_distribution_of_logits.png
test_distribution_of_logits_1.png      valid_pie_f1_metric.png
test_pie_f1_metric_1.png               valid_pr_curve.png
test_pr_curve_1.png                    valid_preds_distribution_by_bins.png
test_preds_distribution_by_bins_1.png  valid_roc_curve.png
```

[16]:
```
%%time

test_predictions = automl_rdu.predict(test_data)
print(f'Prediction for test_data:\n{test_predictions}\nShape = {test_predictions.shape}')
```

```
Prediction for test_data:
array([[0.05117375],
       [0.0546238 ],
       [0.0222431 ],
       ...,
       [0.04375046],
       [0.02917913],
       [0.19221795]], dtype=float32)
Shape = (2000, 1)
CPU times: user 7.91 s, sys: 2.22 s, total: 10.1 s
Wall time: 2.85 s
```

[17]:
```
print(f'OOF score: {roc_auc_score(train_data[TARGET_NAME].values, out_of_fold_
↪predictions.data[:, 0])}')
print(f'HOLDOUT score: {roc_auc_score(test_data[TARGET_NAME].values, test_predictions.
↪data[:, 0])}')
```

```
OOF score: 0.7565288787776545
HOLDOUT score: 0.7281521739130434
```

Bonus: another tasks examples

Regression task

Without big differences from the case of binary classification, LightAutoML can solve the regression problems.

Here you will use Ames Housing dataset. Load the data, split it into train and validation parts:

[52]:
```
data = pd.read_csv('https://raw.githubusercontent.com/reneemarama/aiming_high_in_aimes/
↪master/datasets/train.csv')
data.head()
```

[52]:

	Id	PID	MS SubClass	MS Zoning	Lot Frontage	Lot Area	Street	Alley	\
0	109	533352170	60	RL	NaN	13517	Pave	NaN	
1	544	531379050	60	RL	43.0	11492	Pave	NaN	
2	153	535304180	20	RL	68.0	7922	Pave	NaN	

(continues on next page)

(continued from previous page)

```
3  318  916386060           60        RL          73.0      9802   Pave   NaN
4  255  906425045           50        RL          82.0     14235   Pave   NaN

  Lot Shape Land Contour  ... Screen Porch Pool Area Pool QC Fence  \
0       IR1          Lvl  ...            0         0     NaN   NaN
1       IR1          Lvl  ...            0         0     NaN   NaN
2       Reg          Lvl  ...            0         0     NaN   NaN
3       Reg          Lvl  ...            0         0     NaN   NaN
4       IR1          Lvl  ...            0         0     NaN   NaN

  Misc Feature Misc Val Mo Sold Yr Sold  Sale Type  SalePrice
0          NaN        0       3    2010        WD      130500
1          NaN        0       4    2009        WD      220000
2          NaN        0       1    2010        WD      109000
3          NaN        0       4    2010        WD      174000
4          NaN        0       3    2010        WD      138500

[5 rows x 81 columns]
```

[53]:
```
data.shape
```

[53]:
```
(2051, 81)
```

[54]:
```
train_data, test_data = train_test_split(
    data,
    test_size=TEST_SIZE,
    random_state=RANDOM_STATE
)

print(f'Data is splitted. Parts sizes: train_data = {train_data.shape}, test_data =
↪{test_data.shape}')

train_data.head()
```

```
Data is splitted. Parts sizes: train_data = (1640, 81), test_data = (411, 81)
```

[54]:
```
        Id        PID  MS SubClass MS Zoning  Lot Frontage  Lot Area Street  \
1448   452  528174050          120        RL          47.0      6904   Pave
1771  1697  528110070           20        RL         110.0     14226   Pave
966   2294  923229100           80        RL           NaN     15957   Pave
1604  2449  528348010           60        RL          93.0     12090   Pave
1827  1859  533254100           80        RL          80.0      9600   Pave

     Alley Lot Shape Land Contour  ... Screen Porch Pool Area Pool QC  Fence  \
1448   NaN       IR1          Lvl  ...            0         0     NaN    NaN
1771   NaN       Reg          Lvl  ...            0         0     NaN    NaN
966    NaN       IR1          Low  ...            0         0     NaN  MnPrv
1604   NaN       Reg          Lvl  ...            0         0     NaN    NaN
1827   NaN       Reg          Lvl  ...            0         0     NaN    NaN

     Misc Feature Misc Val Mo Sold Yr Sold  Sale Type  SalePrice
1448          NaN        0       8    2009        WD      213000
1771          NaN        0       7    2007        New     395000
```

(continues on next page)

(continued from previous page)

```
966           NaN        0       9    2007       WD       188000
1604          NaN        0       7    2006       WD       258000
1827          NaN        0       8    2007       WD       187000

[5 rows x 81 columns]
```

Now we have a regression task, and it is necessary to specify it in `Task` object. Note that default loss and metric for regression task is MSE, but you can use any available functions, for example, MAE:

[55]:
```
task = Task('reg', loss='mae', metric='mae')
```

Specifying columns roles:

[56]:
```
roles = {
    'target': 'SalePrice',
    'drop': ['Id', 'PID']
}
```

Building AutoML model:

[57]:
```
automl = TabularAutoML(
    task = task,
    timeout = TIMEOUT,
    cpu_limit = N_THREADS,
    reader_params = {'n_jobs': N_THREADS, 'cv': N_FOLDS, 'random_state': RANDOM_STATE}
)
```

Training:

[58]:
```
%%time
out_of_fold_predictions = automl.fit_predict(train_data, roles = roles, verbose = 1)
```

```
[09:32:54] Stdout logging level is INFO.
[09:32:54] Task: reg

[09:32:54] Start automl preset with listed constraints:
[09:32:54] - time: 300.00 seconds
[09:32:54] - CPU: 4 cores
[09:32:54] - memory: 16 GB

[09:32:54] Train data shape: (1640, 81)

[09:32:59] Layer 1 train process start. Time left 295.37 secs
[09:32:59] Start fitting Lvl_0_Pipe_0_Mod_0_LinearL2 ...
[09:33:13] Fitting Lvl_0_Pipe_0_Mod_0_LinearL2 finished. score = -15890.449792778201
[09:33:13] Lvl_0_Pipe_0_Mod_0_LinearL2 fitting and predicting completed
[09:33:13] Time left 280.74 secs

[09:33:17] Selector_LightGBM fitting and predicting completed
[09:33:18] Start fitting Lvl_0_Pipe_1_Mod_0_LightGBM ...
[09:33:36] Fitting Lvl_0_Pipe_1_Mod_0_LightGBM finished. score = -15003.071067549543
[09:33:36] Lvl_0_Pipe_1_Mod_0_LightGBM fitting and predicting completed
[09:33:36] Start hyperparameters optimization for Lvl_0_Pipe_1_Mod_1_Tuned_LightGBM ...␣
```

(continues on next page)

(continued from previous page)

```
↪Time budget is 1.00 secs
[09:33:43] Hyperparameters optimization for Lvl_0_Pipe_1_Mod_1_Tuned_LightGBM completed
[09:33:43] Start fitting Lvl_0_Pipe_1_Mod_1_Tuned_LightGBM ...
[09:33:56] Fitting Lvl_0_Pipe_1_Mod_1_Tuned_LightGBM finished. score = -15126.
↪021036585365
[09:33:56] Lvl_0_Pipe_1_Mod_1_Tuned_LightGBM fitting and predicting completed
[09:33:56] Start fitting Lvl_0_Pipe_1_Mod_2_CatBoost ...
[09:34:07] Fitting Lvl_0_Pipe_1_Mod_2_CatBoost finished. score = -14772.536144721798
[09:34:07] Lvl_0_Pipe_1_Mod_2_CatBoost fitting and predicting completed
[09:34:07] Start hyperparameters optimization for Lvl_0_Pipe_1_Mod_3_Tuned_CatBoost ...␣
↪Time budget is 130.59 secs
[09:36:19] Hyperparameters optimization for Lvl_0_Pipe_1_Mod_3_Tuned_CatBoost completed
[09:36:19] Start fitting Lvl_0_Pipe_1_Mod_3_Tuned_CatBoost ...
[09:36:26] Fitting Lvl_0_Pipe_1_Mod_3_Tuned_CatBoost finished. score = -14591.
↪741982660062
[09:36:26] Lvl_0_Pipe_1_Mod_3_Tuned_CatBoost fitting and predicting completed
[09:36:26] Time left 88.35 secs

[09:36:26] Layer 1 training completed.

[09:36:26] Blending: optimization starts with equal weights and score -14078.349866615854
[09:36:26] Blending: iteration 0: score = -13998.232345655488, weights = [0.28806204 0. ␣
↪       0.22604953 0.2856552  0.20023331]
[09:36:26] Blending: iteration 1: score = -13997.699302115092, weights = [0.29516914 0. ␣
↪       0.21790323 0.28033847 0.20658915]
[09:36:26] Blending: iteration 2: score = -13997.699302115092, weights = [0.29516914 0. ␣
↪       0.21790323 0.28033847 0.20658915]
[09:36:26] Blending: no score update. Terminated

[09:36:26] Automl preset training completed in 211.78 seconds

[09:36:26] Model description:
Final prediction for new objects (level 0) =
         0.29517 * (5 averaged models Lvl_0_Pipe_0_Mod_0_LinearL2) +
         0.21790 * (5 averaged models Lvl_0_Pipe_1_Mod_1_Tuned_LightGBM) +
         0.28034 * (5 averaged models Lvl_0_Pipe_1_Mod_2_CatBoost) +
         0.20659 * (5 averaged models Lvl_0_Pipe_1_Mod_3_Tuned_CatBoost)

CPU times: user 13min 2s, sys: 1min 16s, total: 14min 18s
Wall time: 3min 31s
```

[59]:
```
%%time

test_predictions = automl.predict(test_data)
```

```
CPU times: user 1.74 s, sys: 164 ms, total: 1.9 s
Wall time: 575 ms
```

[60]:
```
print(f'Prediction for te_data:\n{test_predictions[:10]}\nShape = {test_predictions.
↪shape}')
```

```
Prediction for te_data:
```

(continues on next page)

(continued from previous page)

```
array([[134560.62 ],
       [209942.22 ],
       [278879.3  ],
       [125843.02 ],
       [200796.81 ],
       [390743.2  ],
       [158790.19 ],
       [300576.44 ],
       [161224.7  ],
       [ 81361.086]], dtype=float32)
Shape = (411, 1)
```

[61]:
```
from sklearn.metrics import mean_absolute_error
print(f'OOF score: {mean_absolute_error(train_data[roles["target"]].values, out_of_fold_
↪predictions.data[:, 0])}')
print(f'HOLDOUT score: {mean_absolute_error(test_data[roles["target"]].values, test_
↪predictions.data[:, 0])}')
```

```
OOF score: 13997.699302115092
HOLDOUT score: 12457.17400395377
```

In the same way as in the previous example with binary classification, you can build a detailed report using `ReportDeco`, calculate feature importances, use `TabularUtilizedAutoML` etc.

Shapley values example for regression

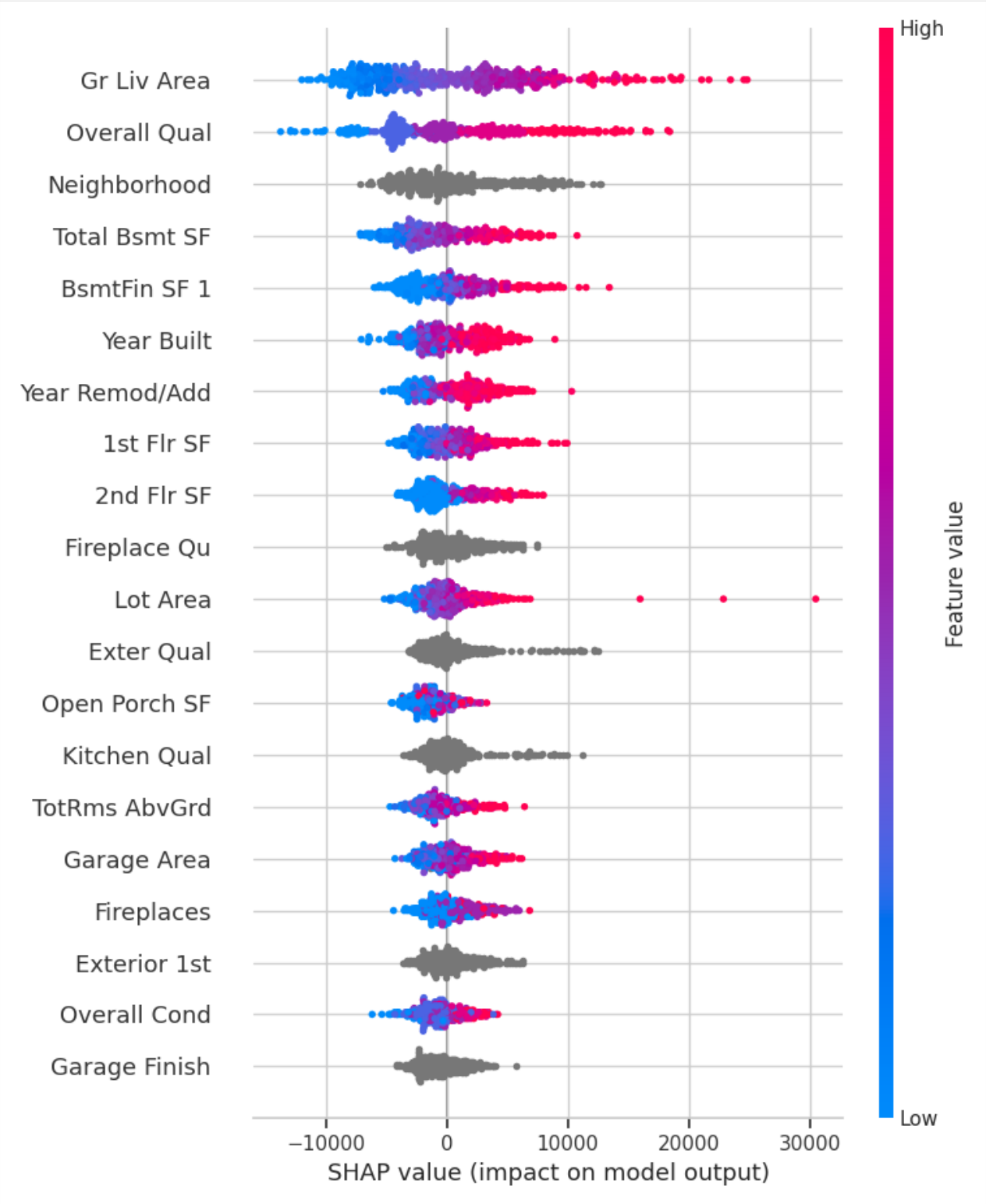
Obtaining shapley estimates

[62]:
```
explainer = SSWARM(automl)
shap_values = explainer.shap_values(test_data, n_jobs=N_THREADS)
```

```
100%|| 3000/3000 [00:48<00:00, 61.51it/s]
```

Interpretation plots

[63]:
```
# summary plot
shap.summary_plot(shap_values, test_data[explainer.used_feats])
```

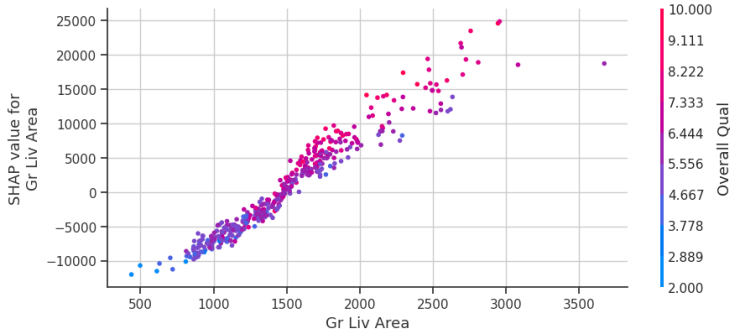


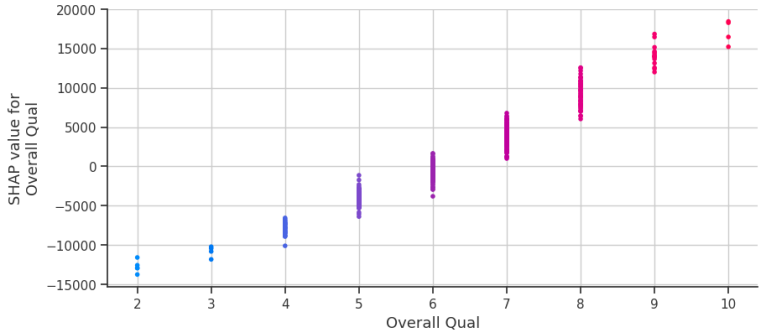
[67]:
```
# dependence plots
feats = ["Gr Liv Area", "Overall Qual", "Total Bsmt SF", "BsmtFin SF 1", "Year Built"]
fig, ax = plt.subplots(nrows=len(feats), ncols=1, figsize=(10, 5*len(feats)))
```

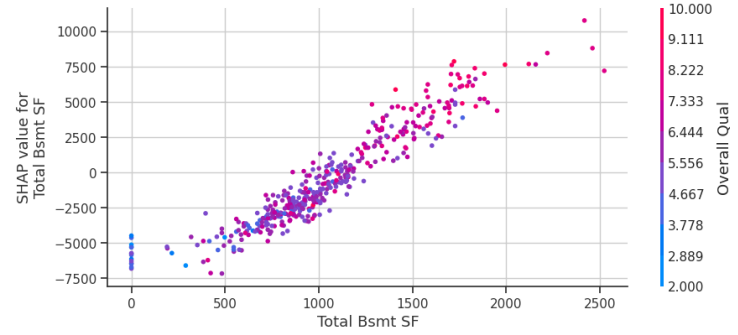
(continues on next page)

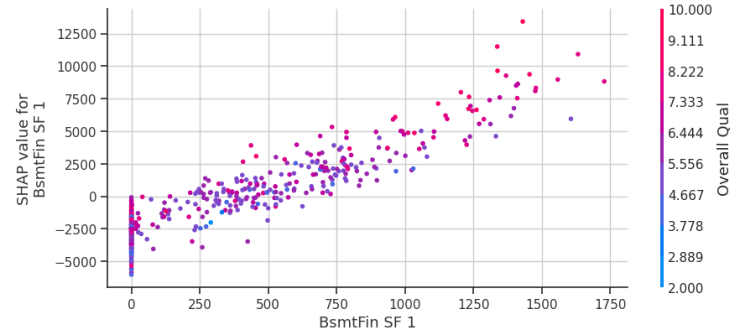
(continued from previous page)

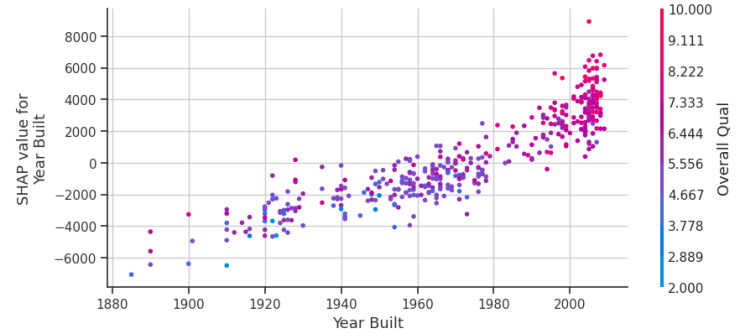
```
for i, feat in enumerate(feats):
    shap.dependence_plot(feat, shap_values, test_data[explainer.used_feats],
                         show=False, ax=ax[i], interaction_index="Overall Qual")
```


SHAP value for Gr Liv Area
Gr Liv Area
Overall Qual

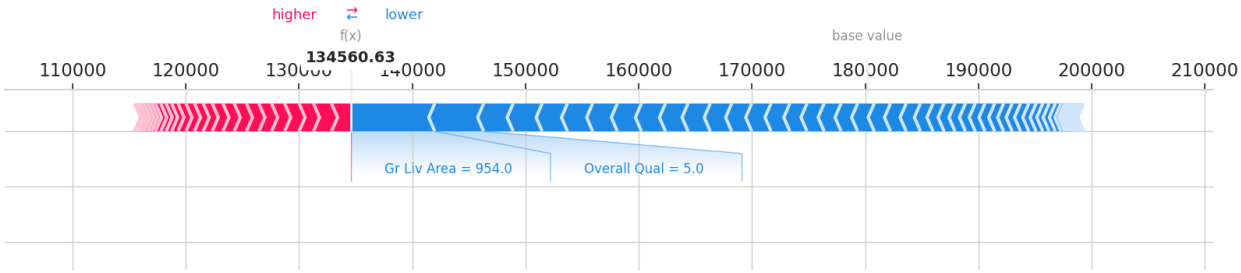

SHAP value for Overall Qual
Overall Qual


SHAP value for Total Bsmt SF
Total Bsmt SF
Overall Qual


SHAP value for BsmtFin SF 1
BsmtFin SF 1
Overall Qual


SHAP value for Year Built
Year Built
Overall Qual

[64]:
```
# individual force plot
shap.force_plot(explainer.expected_value, shap_values[0], test_data[explainer.used_
↪feats].iloc[0], matplotlib=True)
```



Multi-class classification

Now let's consider multi-class classification. Here you will use Anuran Calls (MFCCs) Data Set:

[68]:
```
from io import BytesIO
from zipfile import ZipFile
from urllib.request import urlopen

data = pd.read_csv(
    ZipFile(
        BytesIO(
            urlopen(
                "https://archive.ics.uci.edu/ml/machine-learning-databases/00406/Anuran
↪%20Calls%20(MFCCs).zip"
            ).read()
        )
    ).open('Frogs_MFCCs.csv')
)

data.head()
```

[68]:

	MFCCs_ 1	MFCCs_ 2	MFCCs_ 3	MFCCs_ 4	MFCCs_ 5	MFCCs_ 6	MFCCs_ 7	\
0	1.0	0.152936	-0.105586	0.200722	0.317201	0.260764	0.100945	
1	1.0	0.171534	-0.098975	0.268425	0.338672	0.268353	0.060835	
2	1.0	0.152317	-0.082973	0.287128	0.276014	0.189867	0.008714	
3	1.0	0.224392	0.118985	0.329432	0.372088	0.361005	0.015501	
4	1.0	0.087817	-0.068345	0.306967	0.330923	0.249144	0.006884	

	MFCCs_ 8	MFCCs_ 9	MFCCs_10	...	MFCCs_17	MFCCs_18	MFCCs_19	MFCCs_20	\
0	-0.150063	-0.171128	0.124676	...	-0.108351	-0.077623	-0.009568	0.057684	
1	-0.222475	-0.207693	0.170883	...	-0.090974	-0.056510	-0.035303	0.020140	
2	-0.242234	-0.219153	0.232538	...	-0.050691	-0.023590	-0.066722	-0.025083	
3	-0.194347	-0.098181	0.270375	...	-0.136009	-0.177037	-0.130498	-0.054766	
4	-0.265423	-0.172700	0.266434	...	-0.048885	-0.053074	-0.088550	-0.031346	

	MFCCs_21	MFCCs_22	Family	Genus	Species	RecordID
0	0.118680	0.014038	Leptodactylidae	Adenomera	AdenomeraAndre	1
1	0.082263	0.029056	Leptodactylidae	Adenomera	AdenomeraAndre	1
2	0.099108	0.077162	Leptodactylidae	Adenomera	AdenomeraAndre	1

(continues on next page)

(continued from previous page)

```
3 -0.018691  0.023954  Leptodactylidae  Adenomera  AdenomeraAndre          1
4  0.108610  0.079244  Leptodactylidae  Adenomera  AdenomeraAndre          1

[5 rows x 26 columns]
```

[69]:
```
train_data, test_data = train_test_split(
    data,
    test_size=TEST_SIZE,
    shuffle=True,
    random_state=RANDOM_STATE
)

print(f'Data is splitted. Parts sizes: train_data = {train_data.shape}, test_data =
↪{test_data.shape}')

train_data.head()
```

```
Data is splitted. Parts sizes: train_data = (5756, 26), test_data = (1439, 26)
```

[69]:
```
      MFCCs_ 1  MFCCs_ 2  MFCCs_ 3  MFCCs_ 4  MFCCs_ 5  MFCCs_ 6  MFCCs_ 7  \
3838       1.0  0.389057  0.283855  0.558597  0.142120  0.006777 -0.100356
293        1.0  0.339049 -0.001276  0.075088  0.298091  0.190639  0.022295
1593       1.0  0.211356  0.132368  0.530019  0.181015  0.047415 -0.142114
4669       1.0  0.069635  0.170713  0.583894  0.275507  0.086236 -0.152521
940        1.0  0.222777 -0.069955  0.299370  0.318585  0.094394 -0.019920

      MFCCs_ 8  MFCCs_ 9  MFCCs_10  ...  MFCCs_17  MFCCs_18  MFCCs_19  \
3838  0.015060  0.277700  0.062747  ...  0.221304  0.037511 -0.019166
293   0.049216  0.175380 -0.007751  ... -0.299407 -0.121592  0.108062
1593  0.000687  0.249328  0.032000  ...  0.207694  0.026302 -0.167216
4669 -0.032355  0.268403  0.054420  ...  0.256234 -0.116248 -0.230951
940   0.120537  0.192053  0.047852  ... -0.094684 -0.104692 -0.006204

      MFCCs_20  MFCCs_21  MFCCs_22           Family      Genus  \
3838 -0.042803  0.024793  0.177462  Leptodactylidae  Adenomera
293   0.124870 -0.004888 -0.040086  Leptodactylidae  Adenomera
1593 -0.160102  0.084770  0.276008  Leptodactylidae  Adenomera
4669 -0.058546  0.205891  0.211869  Leptodactylidae  Adenomera
940   0.023067 -0.044556  0.006679    Dendrobatidae   Ameerega

                    Species  RecordID
3838  AdenomeraHylaedactylus        22
293           AdenomeraAndre         7
1593  AdenomeraHylaedactylus        15
4669  AdenomeraHylaedactylus        24
940       Ameeregatrivittata        11

[5 rows x 26 columns]
```

Now we indicate that we have multi-class classification problem. Default metric and loss is cross-entropy function.

[70]:
```
task = Task('multiclass')
```

Set the column roles, build and train AutoML model:

[71]:
```
roles = {
    'target': 'Species',
    'drop': ['RecordID']

}
```

[72]:
```
automl = TabularAutoML(
    task = task,
    timeout = 900,
    cpu_limit = N_THREADS,
    reader_params = {'n_jobs': N_THREADS, 'cv': N_FOLDS, 'random_state': RANDOM_STATE}
)
```

Note that **in case of multi-class classification default pipeline architecture has a slightly different look**. First level is the same as level in default binary classification and regression, second level consists of linear regression and LightGBM model and the third is blending. It was decided to use two levels in default architecture based on the results of experiments on different tasks and data, where it gave an increase in model quality. Intuitively, this can be explained by the fact that only at the second and subsequent levels, the model that predicts the probability of belonging to one of the classes can see what the models that predict the rest of the classes see, that is, the models are able to exchange information about classes. Final prediction is blended from last pipeline level.

[73]:
```
%%time
out_of_fold_predictions = automl.fit_predict(train_data, roles = roles, verbose = 1)
```

```
[09:40:33] Stdout logging level is INFO.
[09:40:33] Task: multiclass

[09:40:33] Start automl preset with listed constraints:
[09:40:33] - time: 900.00 seconds
[09:40:33] - CPU: 4 cores
[09:40:33] - memory: 16 GB

[09:40:33] Train data shape: (5756, 26)

[09:40:44] Layer 1 train process start. Time left 889.93 secs
[09:40:44] Start fitting Lvl_0_Pipe_0_Mod_0_LinearL2 ...
[09:40:55] Fitting Lvl_0_Pipe_0_Mod_0_LinearL2 finished. score = -0.010852760572584014
[09:40:55] Lvl_0_Pipe_0_Mod_0_LinearL2 fitting and predicting completed
[09:40:55] Time left 878.58 secs

[09:41:05] Selector_LightGBM fitting and predicting completed
[09:41:05] Start fitting Lvl_0_Pipe_1_Mod_0_LightGBM ...
[09:41:40] Fitting Lvl_0_Pipe_1_Mod_0_LightGBM finished. score = -0.008332313774221245
[09:41:40] Lvl_0_Pipe_1_Mod_0_LightGBM fitting and predicting completed
[09:41:40] Start hyperparameters optimization for Lvl_0_Pipe_1_Mod_1_Tuned_LightGBM ...␣
↪Time budget is 22.16 secs
[09:42:03] Hyperparameters optimization for Lvl_0_Pipe_1_Mod_1_Tuned_LightGBM completed
[09:42:03] Start fitting Lvl_0_Pipe_1_Mod_1_Tuned_LightGBM ...
```

```
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
```

```
[09:42:09] Fitting Lvl_0_Pipe_1_Mod_1_Tuned_LightGBM finished. score = -0.
↪0061676167581894699
[09:42:09] Lvl_0_Pipe_1_Mod_1_Tuned_LightGBM fitting and predicting completed
```

(continues on next page)

(continued from previous page)

```
[09:42:09] Start fitting Lvl_0_Pipe_1_Mod_2_CatBoost ...
```

```
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
```

```
[09:42:39] Fitting Lvl_0_Pipe_1_Mod_2_CatBoost finished. score = -0.0052738622218159725
[09:42:39] Lvl_0_Pipe_1_Mod_2_CatBoost fitting and predicting completed
[09:42:39] Start hyperparameters optimization for Lvl_0_Pipe_1_Mod_3_Tuned_CatBoost ...␣
↪Time budget is 290.54 secs
```

```
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
```

```
[09:47:31] Hyperparameters optimization for Lvl_0_Pipe_1_Mod_3_Tuned_CatBoost completed
[09:47:31] Start fitting Lvl_0_Pipe_1_Mod_3_Tuned_CatBoost ...
[09:48:36] Fitting Lvl_0_Pipe_1_Mod_3_Tuned_CatBoost finished. score = -0.
↪004565362115455095
[09:48:36] Lvl_0_Pipe_1_Mod_3_Tuned_CatBoost fitting and predicting completed
[09:48:36] Time left 417.56 secs

[09:48:36] Layer 1 training completed.

[09:48:36] Layer 2 train process start. Time left 417.53 secs
```

```
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
```

```
[09:48:36] Start fitting Lvl_1_Pipe_0_Mod_0_LinearL2 ...
[09:48:45] Fitting Lvl_1_Pipe_0_Mod_0_LinearL2 finished. score = -0.006018851790978736
[09:48:45] Lvl_1_Pipe_0_Mod_0_LinearL2 fitting and predicting completed
[09:48:45] Time left 408.00 secs

[09:48:46] Start fitting Lvl_1_Pipe_1_Mod_0_LightGBM ...
[09:49:32] Fitting Lvl_1_Pipe_1_Mod_0_LightGBM finished. score = -0.005526817945493392
[09:49:32] Lvl_1_Pipe_1_Mod_0_LightGBM fitting and predicting completed
[09:49:32] Time left 361.18 secs

[09:49:32] Layer 2 training completed.

[09:49:32] Blending: optimization starts with equal weights and score -0.
↪005530615286885703
```

(continues on next page)

(continued from previous page)

```
[09:49:32] Blending: iteration 0: score = -0.005432028962983418, weights = [0.1987539 0.
↪8012461]
[09:49:32] Blending: iteration 1: score = -0.005432028962983418, weights = [0.1987539 0.
↪8012461]
[09:49:32] Blending: no score update. Terminated

[09:49:32] Automl preset training completed in 539.00 seconds

[09:49:32] Model description:
Models on level 0:
         5 averaged models Lvl_0_Pipe_0_Mod_0_LinearL2
         5 averaged models Lvl_0_Pipe_1_Mod_0_LightGBM
         5 averaged models Lvl_0_Pipe_1_Mod_1_Tuned_LightGBM
         5 averaged models Lvl_0_Pipe_1_Mod_2_CatBoost
         5 averaged models Lvl_0_Pipe_1_Mod_3_Tuned_CatBoost

Final prediction for new objects (level 1) =
         0.19875 * (5 averaged models Lvl_1_Pipe_0_Mod_0_LinearL2) +
         0.80125 * (5 averaged models Lvl_1_Pipe_1_Mod_0_LightGBM)

CPU times: user 32min 14s, sys: 1min 22s, total: 33min 36s
Wall time: 8min 59s
```

[74]:
```
%%time

test_predictions = automl.predict(test_data)
print(f'Prediction for test_data:\n{test_predictions}\nShape = {test_predictions.shape}')
```

```
Prediction for test_data:
array([[9.9961942e-01, 9.1690832e-05, 4.9272418e-05, ..., 2.9182707e-05,
        2.1924083e-05, 1.8040057e-05],
       [7.2180563e-05, 4.8093905e-05, 6.2713400e-05, ..., 3.6562527e-05,
        3.1231004e-05, 2.4088009e-05],
       [1.0252791e-04, 9.9920928e-01, 1.0877274e-04, ..., 6.9528498e-05,
        4.4692573e-05, 3.8138864e-05],
       ...,
       [2.8551594e-04, 1.0709166e-04, 9.9875408e-01, ..., 9.3088362e-05,
        7.5856973e-05, 6.2510102e-05],
       [2.2101802e-04, 7.7087658e-05, 9.9894643e-01, ..., 9.0580215e-05,
        7.7611970e-05, 6.3799351e-05],
       [1.2458056e-04, 9.9910909e-01, 9.0568836e-05, ..., 5.4615561e-05,
        4.9981187e-05, 3.5048710e-05]], dtype=float32)
Shape = (1439, 10)
CPU times: user 8.35 s, sys: 469 ms, total: 8.82 s
Wall time: 2.21 s
```

It is also important to note that the `Reader` object may re-label classes during training. To see the new labelling, you can call `.class_mapping` method of `Reader` object. If the output dict is empty, then the original order and class layout has been preserved.

[75]:
```
automl.reader.class_mapping
```

[75]:
```
{'AdenomeraHylaedactylus': 0,
 'HypsiboasCordobae': 1,
 'AdenomeraAndre': 2,
 'Ameeregatrivittata': 3,
 'HypsiboasCinerascens': 4,
 'HylaMinuta': 5,
 'LeptodactylusFuscus': 6,
 'ScinaxRuber': 7,
 'OsteocephalusOophagus': 8,
 'Rhinellagranulosa': 9}
```

Just in case, in order to avoid problems, it is better to relabel known class labels when calculating metrics, for example, in this or similar way:

[76]:
```
mapping = automl.reader.class_mapping
def map_class(x):
    return mapping[x]

mapped = np.vectorize(map_class)
mapped(train_data['Species'].values)
```

[76]:
```
array([0, 2, 0, ..., 4, 4, 3])
```

[77]:
```
from sklearn.metrics import log_loss
print(f'OOF score: {log_loss(mapped(train_data[roles["target"]].values), out_of_fold_
↪predictions.data)}')
print(f'HOLDOUT score: {log_loss(mapped(test_data[roles["target"]].values), test_
↪predictions.data)}')
```

```
OOF score: 0.005432028962983418
HOLDOUT score: 0.0014346529369004119
```

Feature importance calculation, building reports, time management etc are also available for multiclass classification.

Multi-label classification

Now let's consider multi-label classification task, here you will use the same dataset as in section above (Anuran Calls (MFCCs) Data Set). Let's pick labels in each column:

[78]:
```
data['AdenomeraHylaedactylus'] = (data['Species'] == 'AdenomeraHylaedactylus').
↪astype(int)
data['Adenomera'] = (data['Genus'] == 'Adenomera').astype(int)
data['Leptodactylidae'] = (data['Family'] == 'Leptodactylidae').astype(int)
```

[79]:
```
data.head()
```

[79]:

	MFCCs_ 1	MFCCs_ 2	MFCCs_ 3	MFCCs_ 4	MFCCs_ 5	MFCCs_ 6	MFCCs_ 7	\
0	1.0	0.152936	-0.105586	0.200722	0.317201	0.260764	0.100945	
1	1.0	0.171534	-0.098975	0.268425	0.338672	0.268353	0.060835	
2	1.0	0.152317	-0.082973	0.287128	0.276014	0.189867	0.008714	
3	1.0	0.224392	0.118985	0.329432	0.372088	0.361005	0.015501	
4	1.0	0.087817	-0.068345	0.306967	0.330923	0.249144	0.006884	

	MFCCs_ 8	MFCCs_ 9	MFCCs_10	...	MFCCs_20	MFCCs_21	MFCCs_22	\

(continues on next page)

(continued from previous page)

```
0 -0.150063 -0.171128  0.124676  ...  0.057684  0.118680  0.014038
1 -0.222475 -0.207693  0.170883  ...  0.020140  0.082263  0.029056
2 -0.242234 -0.219153  0.232538  ... -0.025083  0.099108  0.077162
3 -0.194347 -0.098181  0.270375  ... -0.054766 -0.018691  0.023954
4 -0.265423 -0.172700  0.266434  ... -0.031346  0.108610  0.079244

            Family      Genus         Species  RecordID  \
0  Leptodactylidae  Adenomera  AdenomeraAndre         1
1  Leptodactylidae  Adenomera  AdenomeraAndre         1
2  Leptodactylidae  Adenomera  AdenomeraAndre         1
3  Leptodactylidae  Adenomera  AdenomeraAndre         1
4  Leptodactylidae  Adenomera  AdenomeraAndre         1

   AdenomeraHylaedactylus  Adenomera  Leptodactylidae
0                       0          1                1
1                       0          1                1
2                       0          1                1
3                       0          1                1
4                       0          1                1

[5 rows x 29 columns]
```

[80]:
```
targets = ['Leptodactylidae', 'Adenomera', 'AdenomeraHylaedactylus']
```

Split it to train and validation:

[81]:
```
train_data, test_data = train_test_split(
    data,
    test_size=TEST_SIZE,
    random_state=RANDOM_STATE
)

print(f'Data is splitted. Parts sizes: train_data = {train_data.shape}, test_data =
↪{test_data.shape}')

train_data.head()
```

```
Data is splitted. Parts sizes: train_data = (5756, 29), test_data = (1439, 29)
```

[81]:
```
      MFCCs_ 1  MFCCs_ 2  MFCCs_ 3  MFCCs_ 4  MFCCs_ 5  MFCCs_ 6  MFCCs_ 7  \
3838       1.0  0.389057  0.283855  0.558597  0.142120  0.006777 -0.100356
293        1.0  0.339049 -0.001276  0.075088  0.298091  0.190639  0.022295
1593       1.0  0.211356  0.132368  0.530019  0.181015  0.047415 -0.142114
4669       1.0  0.069635  0.170713  0.583894  0.275507  0.086236 -0.152521
940        1.0  0.222777 -0.069955  0.299370  0.318585  0.094394 -0.019920

      MFCCs_ 8  MFCCs_ 9  MFCCs_10  ...  MFCCs_20  MFCCs_21  MFCCs_22  \
3838  0.015060  0.277700  0.062747  ... -0.042803  0.024793  0.177462
293   0.049216  0.175380 -0.007751  ...  0.124870 -0.004888 -0.040086
1593  0.000687  0.249328  0.032000  ... -0.160102  0.084770  0.276008
4669 -0.032355  0.268403  0.054420  ... -0.058546  0.205891  0.211869
940   0.120537  0.192053  0.047852  ...  0.023067 -0.044556  0.006679
```

(continues on next page)

(continued from previous page)

```
               Family      Genus                  Species  RecordID  \
3838  Leptodactylidae  Adenomera  AdenomeraHylaedactylus        22
293   Leptodactylidae  Adenomera          AdenomeraAndre         7
1593  Leptodactylidae  Adenomera  AdenomeraHylaedactylus        15
4669  Leptodactylidae  Adenomera  AdenomeraHylaedactylus        24
940     Dendrobatidae   Ameerega      Ameeregatrivittata        11

      AdenomeraHylaedactylus  Adenomera  Leptodactylidae
3838                       1          1                1
293                        0          1                1
1593                       1          1                1
4669                       1          1                1
940                        0          0                0

[5 rows x 29 columns]
```

Indicate that we are solving multi-label classification problem. Default metric and loss for this task is logloss.

[82]:
```
task = Task('multilabel')
```

```
multilabel isn`t supported in lgb
```

Specifying the roles. Now we have several columns with target variables, and it's necessary to specify them all.

[83]:
```
roles = {
    'target': ['Leptodactylidae', 'Adenomera', 'AdenomeraHylaedactylus'],
    'drop': ['RecordID', 'Species', 'Genus', 'Family']
}
```

Create `TabularAutoML` instance. One of the differences in this case is that by default, random forest algorithm will be used at the end before blending.

[84]:
```
automl = TabularAutoML(
    task = task,
    timeout = 3600,
    cpu_limit = N_THREADS,
    reader_params = {'n_jobs': N_THREADS, 'cv': N_FOLDS, 'random_state': RANDOM_STATE},
↪#TODO: N_THREADS
    general_params = {'use_algos': 'auto'}
)
```

Training:

[85]:
```
%%time
out_of_fold_predictions = automl.fit_predict(train_data, roles = roles, verbose = 1)
```

```
[09:49:35] Stdout logging level is INFO.
[09:49:35] Task: multilabel

[09:49:35] Start automl preset with listed constraints:
[09:49:35] - time: 3600.00 seconds
[09:49:35] - CPU: 4 cores
[09:49:35] - memory: 16 GB
```

(continues on next page)

(continued from previous page)

```
[09:49:35] Train data shape: (5756, 29)

[09:49:39] Layer 1 train process start. Time left 3595.80 secs
[09:49:39] Start fitting Lvl_0_Pipe_0_Mod_0_RFSklearn ...
```

```
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
```

```
[09:50:17] Fitting Lvl_0_Pipe_0_Mod_0_RFSklearn finished. score = -1.7417950747859665
[09:50:17] Lvl_0_Pipe_0_Mod_0_RFSklearn fitting and predicting completed
[09:50:17] Start hyperparameters optimization for Lvl_0_Pipe_0_Mod_1_Tuned_RFSklearn ...
↪Time budget is 300.00 secs
```

```
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
```

(continues on next page)

(continued from previous page)

```
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
```

```
[09:55:22] Hyperparameters optimization for Lvl_0_Pipe_0_Mod_1_Tuned_RFSklearn completed
[09:55:22] Start fitting Lvl_0_Pipe_0_Mod_1_Tuned_RFSklearn ...
[09:55:22] Time left 3253.57 secs

[09:55:22] Start fitting Lvl_0_Pipe_1_Mod_0_LinearL2 ...
```

```
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
```

(continues on next page)

(continued from previous page)

```
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
```

```
[09:55:27] Fitting Lvl_0_Pipe_1_Mod_0_LinearL2 finished. score = -1.7293712480945589
[09:55:27] Lvl_0_Pipe_1_Mod_0_LinearL2 fitting and predicting completed
[09:55:27] Time left 3248.38 secs
```

```
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
```

(continues on next page)

(continued from previous page)

```
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
```

```
[09:55:39] Selector_CatBoost fitting and predicting completed
[09:55:39] Start fitting Lvl_0_Pipe_2_Mod_0_CatBoost ...
[09:56:19] Fitting Lvl_0_Pipe_2_Mod_0_CatBoost finished. score = -1.7239793390118887
[09:56:19] Lvl_0_Pipe_2_Mod_0_CatBoost fitting and predicting completed
[09:56:19] Start hyperparameters optimization for Lvl_0_Pipe_2_Mod_1_Tuned_CatBoost ...␣
↪Time budget is 300.00 secs
```

```
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
```

```
[10:01:29] Hyperparameters optimization for Lvl_0_Pipe_2_Mod_1_Tuned_CatBoost completed
[10:01:29] Start fitting Lvl_0_Pipe_2_Mod_1_Tuned_CatBoost ...
```

```
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
```

```
[10:03:35] Fitting Lvl_0_Pipe_2_Mod_1_Tuned_CatBoost finished. score = -1.
↪7234197039272048
[10:03:35] Lvl_0_Pipe_2_Mod_1_Tuned_CatBoost fitting and predicting completed
[10:03:35] Time left 2759.75 secs

[10:03:35] Layer 1 training completed.

[10:03:35] Blending: optimization starts with equal weights and score -1.7268388233390646
```

```
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
```

(continues on next page)

(continued from previous page)

```
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
```

```
[10:03:36] Blending: iteration 0: score = -1.723183644831893, weights = [0.         0.  ␣
↪      0.4463267  0.55367327]
```

```
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
```

```
[10:03:36] Blending: iteration 1: score = -1.723183644831893, weights = [0.         0.  ␣
↪      0.4463267  0.55367327]
[10:03:36] Blending: no score update. Terminated

[10:03:36] Automl preset training completed in 840.67 seconds
```

(continues on next page)

(continued from previous page)

```
[10:03:36] Model description:
Final prediction for new objects (level 0) =
         0.44633 * (5 averaged models Lvl_0_Pipe_2_Mod_0_CatBoost) +
         0.55367 * (5 averaged models Lvl_0_Pipe_2_Mod_1_Tuned_CatBoost)

CPU times: user 40min 39s, sys: 3min 26s, total: 44min 5s
Wall time: 14min
```

Get a prediction on the test data:

[86]:
```
%%time

test_predictions = automl.predict(test_data)
print(f'Prediction for test_data:\n{test_predictions}\nShape = {test_predictions.shape}')
```

```
Prediction for test_data:
array([[9.9938047e-01, 9.9943602e-01, 9.9970806e-01],
       [2.3319555e-04, 1.4785888e-04, 3.1775104e-05],
       [3.7830789e-04, 4.8983089e-05, 5.8558126e-06],
       ...,
       [9.1259861e-01, 9.2034292e-01, 1.7304690e-05],
       [9.9985009e-01, 9.9985689e-01, 2.9136087e-05],
       [8.1000919e-04, 1.9849822e-04, 3.6050700e-05]], dtype=float32)
Shape = (1439, 3)
CPU times: user 1.15 s, sys: 178 ms, total: 1.33 s
Wall time: 185 ms
```

Note that in case of multi-label classification classes order always remains unchanged.

[87]:
```
automl.reader.class_mapping
```

[87]:
```
{'Leptodactylidae': None, 'Adenomera': None, 'AdenomeraHylaedactylus': None}
```

It is importatnt to note that in this case models taken in the final composition did not have time to learn on all cross-validation folds, so their predicts in them will be NaNs:

[88]:
```
np.unique(np.isnan(out_of_fold_predictions.data))
```

[88]:
```
array([False])
```

But for new data, normal numerical predictions are made:

[89]:
```
np.unique(np.isnan(test_predictions.data))
```

[89]:
```
array([False])
```

Therefore, we can calculate the logloss from sklearn only on the test set (because of NaNs):

[90]:
```
from sklearn.metrics import log_loss
print(f'HOLDOUT score: {log_loss(test_data[targets].values, test_predictions.data)}')
```

```
HOLDOUT score: 1.7310181043031334
```

```
The y_pred values do not sum to one. Starting from 1.5 thiswill result in an error.
```

Multi-output regression

For completeness, let's consider multi-output regression task. Here you will use Energy Efficiency dataset.

Data loading and splitting:

[94]:
```
columns = [
    'relative_compactness', 'surface_area', 'wall_area', 'roof_area',
    'overall_height', 'orientation', 'glazing_area',
    'glazing_area_distribution', 'heating_load', 'cooling_load'
]
data = pd.read_excel("https://archive.ics.uci.edu/ml/machine-learning-databases/00242/
↪ENB2012_data.xlsx", names=columns, header=None)
data = data.drop(index=0, inplace=False)
data
```

[94]:
```
    relative_compactness surface_area wall_area roof_area overall_height  \
1                   0.98        514.5       294    110.25              7
2                   0.98        514.5       294    110.25              7
3                   0.98        514.5       294    110.25              7
4                   0.98        514.5       294    110.25              7
5                    0.9        563.5     318.5     122.5              7
..                   ...          ...       ...       ...            ...
764                 0.64          784       343     220.5            3.5
765                 0.62        808.5     367.5     220.5            3.5
766                 0.62        808.5     367.5     220.5            3.5
767                 0.62        808.5     367.5     220.5            3.5
768                 0.62        808.5     367.5     220.5            3.5

    orientation glazing_area glazing_area_distribution heating_load  \
1             2            0                         0        15.55
2             3            0                         0        15.55
3             4            0                         0        15.55
4             5            0                         0        15.55
5             2            0                         0        20.84
..          ...          ...                       ...          ...
764           5          0.4                         5        17.88
765           2          0.4                         5        16.54
766           3          0.4                         5        16.44
767           4          0.4                         5        16.48
768           5          0.4                         5        16.64

    cooling_load
1          21.33
2          21.33
3          21.33
4          21.33
5          28.28
..           ...
764         21.4
765        16.88
766        17.11
767        16.61
768        16.03
```

(continues on next page)

(continued from previous page)

```
[768 rows x 10 columns]
```

[95]:
```
train_data, test_data = train_test_split(
    data,
    test_size=TEST_SIZE,
    random_state=RANDOM_STATE
)

print(f'Data is splitted. Parts sizes: train_data = {train_data.shape}, test_data =
↪{test_data.shape}')

train_data.head()
```

```
Data is splitted. Parts sizes: train_data = (614, 10), test_data = (154, 10)
```

[95]:
```
    relative_compactness surface_area wall_area roof_area overall_height  \
61                  0.82        612.5     318.5       147              7
619                 0.64          784       343     220.5            3.5
347                 0.86          588       294       147              7
295                  0.9        563.5     318.5     122.5              7
232                 0.66        759.5     318.5     220.5            3.5

    orientation glazing_area glazing_area_distribution heating_load  \
61            2          0.1                          1        23.53
619           4          0.4                          2         18.9
347           4         0.25                          2        29.27
295           4         0.25                          1        32.84
232           5          0.1                          4        11.43

    cooling_load
61         27.31
619        22.09
347         29.9
295        32.71
232        14.83
```

[96]:
```
train_data = train_data.astype('float')
test_data = test_data.astype('float')
```

Specifying the `Task` object. Default loss and metric for multi-output regression is MAE.

[97]:
```
task = Task('multi:reg')
```

```
multi:reg isn`t supported in lgb
```

Roles setting:

[98]:
```
roles = {
    'target': ['heating_load', 'cooling_load'],
}
```

Create `TabularAutoML` instance:

[99]:
```
automl = TabularAutoML(
    task = task,
    timeout = 600,
    cpu_limit = N_THREADS,
    reader_params = {'n_jobs': N_THREADS, 'cv': N_FOLDS, 'random_state': RANDOM_STATE},
    general_params = {'use_algos': 'auto'}
)
```

By default, random forest algorithm will be used at the end before blending.

Training and getting out-of-fold prediction:

[100]:
```
%%time
out_of_fold_predictions = automl.fit_predict(train_data, roles = roles, verbose = 1)
```

```
[10:04:05] Stdout logging level is INFO.
[10:04:05] Task: multi:reg

[10:04:05] Start automl preset with listed constraints:
[10:04:05] - time: 600.00 seconds
[10:04:05] - CPU: 4 cores
[10:04:05] - memory: 16 GB

[10:04:05] Train data shape: (614, 10)

[10:04:08] Layer 1 train process start. Time left 596.07 secs
[10:04:09] Start fitting Lvl_0_Pipe_0_Mod_0_RFSklearn ...
[10:04:23] Fitting Lvl_0_Pipe_0_Mod_0_RFSklearn finished. score = -1.9434915645736046
[10:04:23] Lvl_0_Pipe_0_Mod_0_RFSklearn fitting and predicting completed
[10:04:23] Start hyperparameters optimization for Lvl_0_Pipe_0_Mod_1_Tuned_RFSklearn ...␣
↪Time budget is 197.38 secs
[10:07:42] Hyperparameters optimization for Lvl_0_Pipe_0_Mod_1_Tuned_RFSklearn completed
[10:07:42] Start fitting Lvl_0_Pipe_0_Mod_1_Tuned_RFSklearn ...
[10:07:42] Time left 382.33 secs

[10:07:42] Start fitting Lvl_0_Pipe_1_Mod_0_LinearL2 ...
[10:07:53] Fitting Lvl_0_Pipe_1_Mod_0_LinearL2 finished. score = -1.0981723066416937
[10:07:53] Lvl_0_Pipe_1_Mod_0_LinearL2 fitting and predicting completed
[10:07:53] Time left 371.20 secs

[10:07:54] Selector_CatBoost fitting and predicting completed
[10:07:54] Start fitting Lvl_0_Pipe_2_Mod_0_CatBoost ...
[10:07:58] Fitting Lvl_0_Pipe_2_Mod_0_CatBoost finished. score = -0.5153592339866712
[10:07:58] Lvl_0_Pipe_2_Mod_0_CatBoost fitting and predicting completed
[10:07:58] Start hyperparameters optimization for Lvl_0_Pipe_2_Mod_1_Tuned_CatBoost ...␣
↪Time budget is 210.14 secs
[10:10:00] Hyperparameters optimization for Lvl_0_Pipe_2_Mod_1_Tuned_CatBoost completed
[10:10:00] Start fitting Lvl_0_Pipe_2_Mod_1_Tuned_CatBoost ...
[10:10:05] Fitting Lvl_0_Pipe_2_Mod_1_Tuned_CatBoost finished. score = -0.
↪5340599466224449
[10:10:05] Lvl_0_Pipe_2_Mod_1_Tuned_CatBoost fitting and predicting completed
[10:10:05] Time left 239.55 secs

[10:10:05] Layer 1 training completed.
```

(continues on next page)

(continued from previous page)

```
[10:10:05] Blending: optimization starts with equal weights and score -0.867237476774458
[10:10:05] Blending: iteration 0: score = -0.5153592339866712, weights = [0. 0. 1. 0.]
[10:10:05] Blending: iteration 1: score = -0.5153592339866712, weights = [0. 0. 1. 0.]
[10:10:05] Blending: no score update. Terminated

[10:10:05] Automl preset training completed in 360.54 seconds

[10:10:05] Model description:
Final prediction for new objects (level 0) =
         1.00000 * (5 averaged models Lvl_0_Pipe_2_Mod_0_CatBoost)

CPU times: user 12min 33s, sys: 1min 48s, total: 14min 22s
Wall time: 6min
```

Make prediction on test data:

[101]:
```
%%time

test_predictions = automl.predict(test_data)
```

```
CPU times: user 412 ms, sys: 83.5 ms, total: 496 ms
Wall time: 54.1 ms
```

Evaluate regression quality:

[102]:
```
from sklearn.metrics import mean_absolute_error
mae_h_train = mean_absolute_error(train_data["heating_load"].values, out_of_fold_
↪predictions.data[:, 0])
mae_c_train = mean_absolute_error(train_data["cooling_load"].values, out_of_fold_
↪predictions.data[:, 1])
mae_h_test = mean_absolute_error(test_data["heating_load"].values, test_predictions.
↪data[:, 0])
mae_c_test = mean_absolute_error(test_data["cooling_load"].values, test_predictions.
↪data[:, 1])
print(f'OOF score, heating_load: {mae_h_train}')
print(f'OOF score, cooling_load: {mae_c_train}')
print(f'HOLDOUT score, heating_load: {mae_h_test}')
print(f'HOLDOUT score, cooling_load: {mae_c_test}')
print(f'OOF score, general: {(mae_h_train + mae_c_train) / 2}')
print(f'HOLDOUT score, general: {(mae_h_test + mae_c_test) / 2}')
```

```
OOF score, heating_load: 0.34296714751650537
OOF score, cooling_load: 0.6877513204568373
HOLDOUT score, heating_load: 0.3168395171475101
HOLDOUT score, cooling_load: 0.5896234752605487
OOF score, general: 0.5153592339866713
HOLDOUT score, general: 0.45323149620402936
```

Additional materials

- Official LightAutoML github repo
- LightAutoML documentation
- LightAutoML tutorials
- LightAutoML course:
  - Part 1 - general overview
  - Part 2 - LightAutoML specific applications
  - Part 3 - LightAutoML customization
- OpenDataScience AutoML benchmark leaderboard

2.1.2 Tutorial 2: AutoWoE (WhiteBox model for binary classification on tabular data)

Official LightAutoML github repository is here

Scorecard

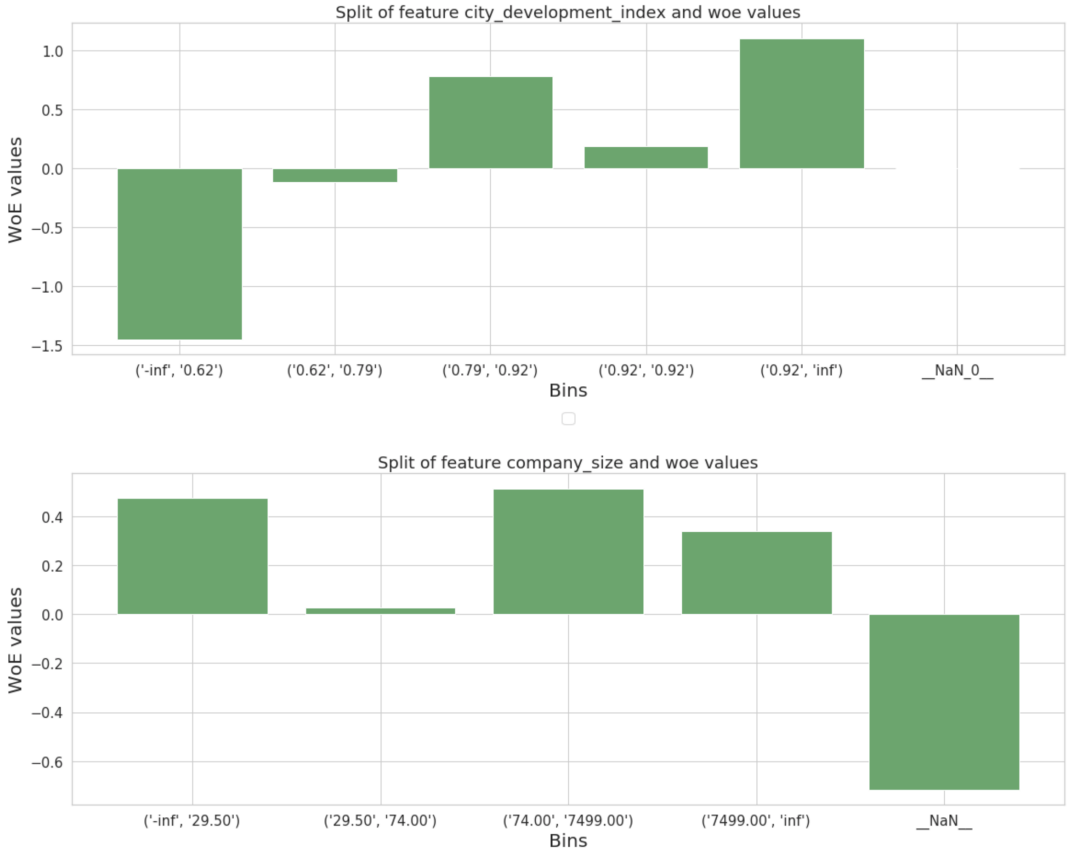
Variable	Value	WOE	COEF	POINTS
Intercept	None	None	-1.11	-1.11
city_development_index	city_development_index <= 0.62	-1.45	-0.97	1.42
city_development_index	0.62 < city_development_index <= 0.79	-0.12	-0.97	0.12
city_development_index	0.79 < city_development_index <= 0.92	0.78	-0.97	-0.76
city_development_index	0.92 < city_development_index <= 0.92	0.19	-0.97	-0.18
city_development_index	city_development_index > 0.92	1.1	-0.97	-1.07
city_development_index	__NaN_0__	0	-0.97	0.0
company_size	company_size <= 29.5	0.47	-0.8	-0.38
company_size	29.5 < company_size <= 74.0	0.03	-0.8	-0.02
company_size	74.0 < company_size <= 7499.0	0.51	-0.8	-0.41
company_size	company_size > 7499.0	0.34	-0.8	-0.27
company_size	__NaN__	-0.72	-0.8	0.57
company_type	Funded Startup	0.74	-0.4	-0.3
company_type	Pvt Ltd	0.4	-0.4	-0.16
company_type	Early Stage Startup, NGO, Other, Public Sector	0.16	-0.4	-0.07
company_type	__NaN__	-0.64	-0.4	0.26
company_type	__Small_0__	0	-0.4	0.0

Linear model

Table 4. Coefficients of the regression model

Feature name	Regression coefficient
education_level	-1.188926
city_development_index	-0.974107
company_size	-0.795953
company_type	-0.400146
enrolled_university	-0.251287
experience	-0.184238

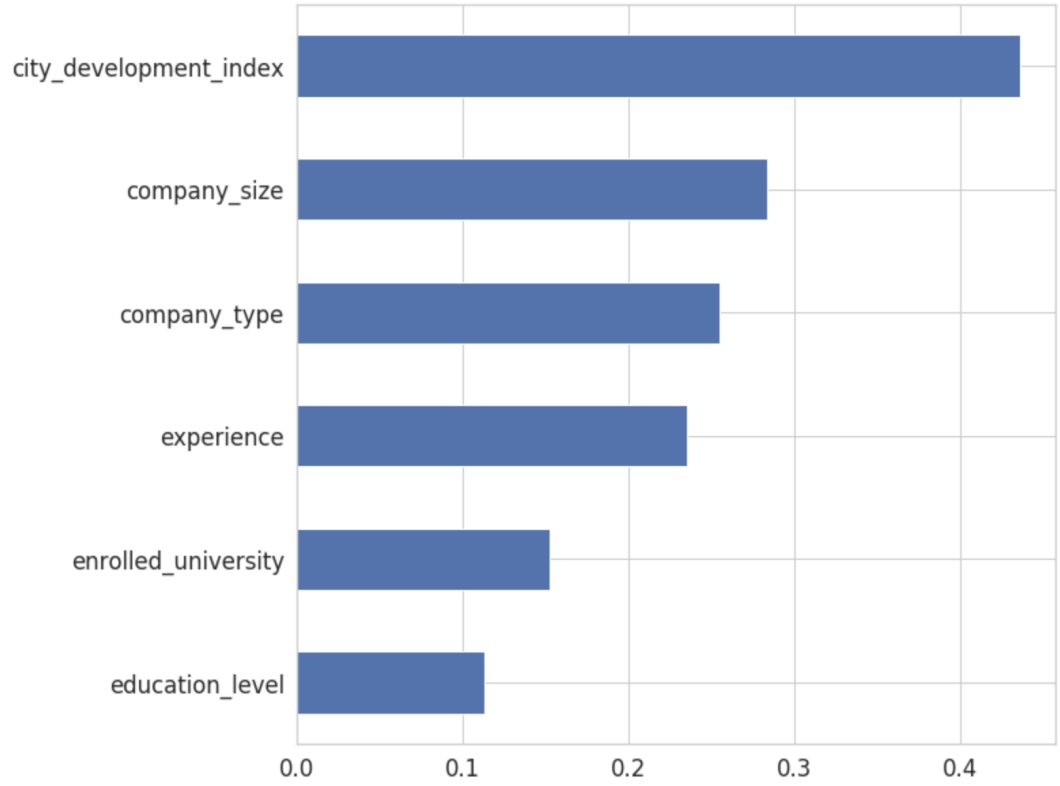
Discretization



Selection and One-dimensional analysis

▼ 3.5. One-dimensional Analysis

Gini for the training set



Whitebox pipeline:

General parameters

0. Technical
   - n_jobs
   - debug
1. Simple features typing and initial cleaning

   1.1. Remove trash features

```
Medium:
    - th_nan
    - th_const
```

   1.2. Typling (auto or user defined)

```
Critical:
    - features_type (dict) {'age': 'real', 'education': 'cat', 'birth_date': (None,␣
↪("d", "wd"), ...}
```

   1.3. Categories and datetimes encoding

```
Critical:
    - features_type (for datetimes)

Optional:
    - cat_alpha (int) - greater means more conservative encoding
```

2. Pre selection (based on BlackBox model importances)
    - Critical:
        - select_type (None or int)
        - imp_type (if type(select_type) is int 'perm_imt'/'feature_imp')
    - Optional:
        - imt_th (float) - threshold for select_type is None
3. Binning (discretization)
    - Critical:
        - monotonic / features_monotone_constraints
        - max_bin_count / max_bin_count
        - min_bin_size
        - cat_merge_to
        - nan_merge_to
    - Medium:
        - force_single_split
    - Optional:
        - min_bin_mults
        - min_gains_to_split
4. WoE estimation WoE = LN(((% 0 in bin) / (% 0 in sample)) / ((% 1 in bin) / (% 1 in sample))):
    - Critical:
        - oof_woe
    - Optional:
        - woe_diff_th
        - n_folds (if oof_woe)
5. 2nd selection stage:

    5.1. One-dimentional importance

```
Critical:
    - auc_th
```

    5.2. VIF

```
Critical:
    - vif_th
```

5.3. Partial correlations

```
Critical:
    - pearson_th
```

6. 3rd selection stage (model based)
   - Optional:
     - n_folds
     - l1_base_step
     - l1_exp_step
   - Do not touch:
     - population_size
     - feature_groups_count
7. Fitting the final model
   - Critical:
     - regularized_refit
     - p_val (if not regularized_refit)
     - validation (if not regularized_refit)
   - Optional:
     - interpreted_model
     - l1_base_step (if regularized_refit)
     - l1_exp_step (if regularized_refit)
8. Report generation
   - report_params

Imports

[25]:
```
import pandas as pd
from pandas import Series, DataFrame

import numpy as np

import os
import requests
import joblib

from sklearn.model_selection import train_test_split
from sklearn.metrics import roc_auc_score

from autowoe import AutoWoE, ReportDeco
```

Reading the data and train/test split

[26]:
```
DATASET_DIR = '../data/'
DATASET_NAME = 'jobs_train.csv'
DATASET_FULLNAME = os.path.join(DATASET_DIR, DATASET_NAME)
DATASET_URL = 'https://raw.githubusercontent.com/AILab-MLTools/LightAutoML/master/
↪examples/data/jobs_train.csv'
```

[27]:
```
%%time

if not os.path.exists(DATASET_FULLNAME):
    os.makedirs(DATASET_DIR, exist_ok=True)

    dataset = requests.get(DATASET_URL).text
    with open(DATASET_FULLNAME, 'w') as output:
        output.write(dataset)
```

```
CPU times: user 322 s, sys: 0 ns, total: 322 s
Wall time: 210 s
```

[28]:
```
data = pd.read_csv(DATASET_FULLNAME)
```

[29]:
```
data
```

[29]:
```
       enrollee_id      city  city_development_index gender  \
0             8949  city_103                   0.920   Male
1            29725   city_40                   0.776   Male
2            11561   city_21                   0.624    NaN
3            33241  city_115                   0.789    NaN
4              666  city_162                   0.767   Male
...            ...       ...                     ...    ...
19153         7386  city_173                   0.878   Male
19154        31398  city_103                   0.920   Male
19155        24576  city_103                   0.920   Male
19156         5756   city_65                   0.802   Male
19157        23834   city_67                   0.855    NaN

           relevant_experience enrolled_university education_level  \
0      Has relevant experience       no_enrollment        Graduate
1       No relevant experience       no_enrollment        Graduate
2       No relevant experience    Full time course        Graduate
3       No relevant experience                 NaN        Graduate
4      Has relevant experience       no_enrollment         Masters
...                        ...                 ...             ...
19153   No relevant experience       no_enrollment        Graduate
19154  Has relevant experience       no_enrollment        Graduate
19155  Has relevant experience       no_enrollment        Graduate
19156  Has relevant experience       no_enrollment     High School
19157   No relevant experience       no_enrollment  Primary School

      major_discipline  experience  company_size    company_type  \
0                 STEM        21.0           NaN             NaN
1                 STEM        15.0          99.0         Pvt Ltd
```

(continues on next page)

(continued from previous page)

```
2                  STEM          5.0            NaN              NaN
3       Business Degree          0.0            NaN          Pvt Ltd
4                  STEM         21.0           99.0   Funded Startup
...                 ...          ...            ...              ...
19153         Humanities         14.0            NaN              NaN
19154              STEM          14.0            NaN              NaN
19155              STEM          21.0           99.0          Pvt Ltd
19156               NaN           0.0          999.0          Pvt Ltd
19157               NaN           2.0            NaN              NaN

       last_new_job  training_hours  target
0               1.0              36     1.0
1               5.0              47     0.0
2               0.0              83     0.0
3               0.0              52     1.0
4               4.0               8     0.0
...             ...             ...     ...
19153           1.0              42     1.0
19154           4.0              52     1.0
19155           4.0              44     0.0
19156           2.0              97     0.0
19157           1.0             127     0.0

[19158 rows x 14 columns]
```

[30]:
```
train, test = train_test_split(data.drop('enrollee_id', axis=1), test_size=0.2,␣
↪stratify=data['target'])
```

AutoWoe: default settings

[31]:
```
auto_woe_0 = AutoWoE(interpreted_model=True,
                     monotonic=False,
                     max_bin_count=5,
                     select_type=None,
                     pearson_th=0.9,
                     metric_th=.505,
                     vif_th=10.,
                     imp_th=0,
                     th_const=32,
                     force_single_split=True,
                     th_nan=0.01,
                     th_cat=0.005,
                     metric_tol=1e-4,
                     cat_alpha=100,
                     cat_merge_to="to_woe_0",
                     nan_merge_to="to_woe_0",
                     imp_type="feature_imp",
                     regularized_refit=False,
                     p_val=0.05,
                     verbose=2
        )
```

(continues on next page)

(continued from previous page)

```
auto_woe_0 = ReportDeco(auto_woe_0, )
```

[32]:
```
auto_woe_0.fit(train,
               target_name="target",
              )
```

```
[LightGBM] [Info] Number of positive: 3027, number of negative: 9233
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing was 0.
↪000615 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 512
[LightGBM] [Info] Number of data points in the train set: 12260, number of used features:
↪ 12
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.246900 -> initscore=-1.115212
[LightGBM] [Info] Start training from score -1.115212
Training until validation scores don't improve for 10 rounds
Early stopping, best iteration is:
[10]    val_test's auc: 0.798546
city processing...
city_development_index processing...
gender processing...
relevant_experience processing...
enrolled_university processing...
education_level processing...
experience processing...
company_size processing...
company_type processing...
last_new_job processing...
training_hours processing...
dict_keys(['city', 'city_development_index', 'gender', 'relevant_experience', 'enrolled_
↪university', 'education_level', 'experience', 'company_size', 'company_type', 'last_
↪new_job', 'training_hours']) to selector !!!!!
Feature selection...
city_development_index   -0.986582
company_size             -0.816660
company_type             -0.400413
experience               -0.189662
enrolled_university      -0.209603
education_level          -1.169475
dtype: float64
```

[33]:
```
test_prediction = auto_woe_0.predict_proba(test)
test_prediction
```

[33]:
```
array([0.07859696, 0.26590075, 0.0693864 , ..., 0.12662937, 0.14364888,
       0.47728299])
```

[34]:
```
roc_auc_score(test['target'].values, test_prediction)
```

[34]:
```
0.7882607500905356
```

[35]:
```
report_params = {"output_path": "HR_REPORT_1", # folder for report generation
                 "report_name": "WHITEBOX REPORT",
                 "report_version_id": 1,
                 "city": "Moscow",
                 "model_aim": "Predict if candidate will work for the company",
                 "model_name": "HR model",
                 "zakazchik": "Kaggle",
                 "high_level_department": "Ai Lab",
                 "ds_name": "Btbpanda",
                 "target_descr": "Candidate will work for the company",
                 "non_target_descr": "Candidate will work for the company"}

auto_woe_0.generate_report(report_params, )
```

AutoWoE - simpler model

[36]:
```
auto_woe_1 = AutoWoE(interpreted_model=True,
                     monotonic=True,
                     max_bin_count=4,
                     select_type=None,
                     pearson_th=0.9,
                     metric_th=.505,
                     vif_th=10.,
                     imp_th=0,
                     th_const=32,
                     force_single_split=True,
                     th_nan=0.01,
                     th_cat=0.005,
                     metric_tol=1e-4,
                     cat_alpha=100,
                     cat_merge_to="to_woe_0",
                     nan_merge_to="to_woe_0",
                     imp_type="feature_imp",
                     regularized_refit=False,
                     p_val=0.05,
                     verbose=2
        )

auto_woe_1 = ReportDeco(auto_woe_1, )
```

[37]:
```
auto_woe_1.fit(train,
               target_name="target",
              )
```

```
[LightGBM] [Info] Number of positive: 3027, number of negative: 9233
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing was 0.
↪000498 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 512
[LightGBM] [Info] Number of data points in the train set: 12260, number of used features:
↪ 12
```

(continues on next page)

(continued from previous page)

```
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.246900 -> initscore=-1.115212
[LightGBM] [Info] Start training from score -1.115212
Training until validation scores don't improve for 10 rounds
Early stopping, best iteration is:
[10]    val_test's auc: 0.798546
city processing...
city_development_index processing...
gender processing...
relevant_experience processing...
enrolled_university processing...
education_level processing...
experience processing...
company_size processing...
company_type processing...
last_new_job processing...
training_hours processing...
dict_keys(['city', 'city_development_index', 'gender', 'relevant_experience', 'enrolled_
↪university', 'education_level', 'experience', 'company_size', 'company_type', 'last_
↪new_job', 'training_hours']) to selector !!!!!
Feature selection...
city                      -0.537704
city_development_index    -0.501377
company_size              -0.820035
company_type              -0.424950
experience                -0.193515
enrolled_university       -0.162144
education_level           -1.230515
dtype: float64
```

[38]:
```
test_prediction = auto_woe_1.predict_proba(test)
test_prediction
```

[38]:
```
array([0.08448153, 0.23365578, 0.06406811, ..., 0.10791522, 0.13339279,
       0.45904027])
```

[39]:
```
roc_auc_score(test['target'].values, test_prediction)
```

[39]:
```
0.7859235642130127
```

[40]:
```
report_params = {"output_path": "HR_REPORT_2", # folder for report generation
                 "report_name": "WHITEBOX REPORT",
                 "report_version_id": 2,
                 "city": "Moscow",
                 "model_aim": "Predict if candidate will work for the company",
                 "model_name": "HR model",
                 "zakazchik": "Kaggle",
                 "high_level_department": "Ai Lab",
                 "ds_name": "Btbpanda",
                 "target_descr": "Candidate will work for the company",
                 "non_target_descr": "Candidate will work for the company"}

auto_woe_1.generate_report(report_params, )
```

WhiteBox preset - like TabularAutoML

[41]:
```
from lightautoml.automl.presets.whitebox_presets import WhiteBoxPreset
from lightautoml.task import Task
```

```
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
Cell In[41], line 2
      1 from lightautoml.automl.presets.whitebox_presets import WhiteBoxPreset
----> 2 from lightautoml.task import Task

ModuleNotFoundError: No module named 'lightautoml.task'
```

[ ]:
```
task = Task('binary')
automl = WhiteBoxPreset(task)
```

[ ]:
```
train_pred = automl.fit_predict(train.reset_index(drop=True), roles={'target': 'target'})
```

```
Validation data is not set. Train will be used as valid in report and valid prediction
Start automl preset with listed constraints:
- time: 3600 seconds
- cpus: 4 cores
- memory: 16 gb

Train data shape: (15326, 13)
Feats was rejected during automatic roles guess: []


Layer 1 ...
Train process start. Time left 3595.0072581768036 secs
Start fitting Lvl_0_Pipe_0_Mod_0_WhiteBox ...

===== Start working with fold 0 for Lvl_0_Pipe_0_Mod_0_WhiteBox =====

 features [] contain too many nans or identical values
 features [] have low importance
city processing...
city_development_index processing...company_type processing...education_level␣
↪processing...


enrolled_university processing...
gender processing...
major_discipline processing...
relevent_experience processing...
company_size processing...
experience processing...
last_new_job processing...
training_hours processing...
dict_keys(['city', 'city_development_index', 'company_type', 'education_level',
↪'enrolled_university', 'gender', 'major_discipline', 'relevent_experience', 'company_
↪size', 'experience', 'last_new_job', 'training_hours']) to selector !!!!!
```

(continues on next page)

(continued from previous page)

```
Feature selection...
Feature training_hours removed due to low AUC value 0.5031265374717342
Feature city_development_index removed due to high VIF value = 40.56438648184099
C parameter range in [0.0002603488674824265:260.3488674824265], 20 values
Result(score=0.7856775296767177, reg_alpha=0.020431136952654548, is_neg=True, min_
↪weights=city                  -0.980620
company_size          -0.800535
company_type          -0.340185
experience            -0.198176
enrolled_university   -0.101047
relevent_experience    0.000000
education_level       -0.624324
last_new_job           0.000000
gender                 0.000000
major_discipline      -0.317699
dtype: float64)
Iter 0 of final refit starts with 7 features
Validation data checks
city                  -0.956550
company_size          -0.866063
company_type          -0.402941
experience            -0.329493
enrolled_university   -0.230776
education_level       -0.641994
major_discipline      -1.596907
dtype: float64
Lvl_0_Pipe_0_Mod_0_WhiteBox fitting and predicting completed
Time left 3587.2280378341675

Automl preset training completed in 12.77 seconds.
```

[ ]:
```
test_prediction = automl.predict(test).data[:, 0]
```

[ ]:
```
roc_auc_score(test['target'].values, test_prediction)
```

```
0.7966826628232216
```

Serialization

Important note: `auto_woe_1` is the `ReportDeco` object (the report generator object), not `AutoWoE` itself. To receive the `AutoWoE` object you can use the `auto_woe_1.model`.

`ReportDeco` object usage for inference is **not** recommended for several reasons: - The report object needs to have the target column because of model quality metrics calculation - Model inference using `ReportDeco` object is slower than the usual one because of the report update procedure

[ ]:
```
joblib.dump(auto_woe_1.model, 'model.pkl')
model = joblib.load('model.pkl')
```

SQL inference query

[21]:
```
sql_query = model.get_sql_inference_query('global_temp.TABLE_1')
print(sql_query)
```

```
SELECT
  1 / (1 + EXP(-(
    -1.111
    -0.516*WOE_TAB.city
    -0.513*WOE_TAB.city_development_index
    -0.815*WOE_TAB.company_size
    -0.398*WOE_TAB.company_type
    -0.175*WOE_TAB.experience
    -0.22*WOE_TAB.enrolled_university
    -1.24*WOE_TAB.education_level
  ))) as PROB,
  WOE_TAB.*
FROM
    (SELECT
    CASE
      WHEN (city IS NULL OR LOWER(CAST(city AS VARCHAR(50))) = 'nan') THEN 0
      WHEN city IN ('city_100', 'city_102', 'city_103', 'city_116', 'city_149', 'city_159
↪', 'city_160', 'city_45', 'city_46', 'city_64', 'city_71', 'city_73', 'city_83', 'city_
↪99') THEN 0.213
      WHEN city IN ('city_104', 'city_114', 'city_136', 'city_138', 'city_16', 'city_173
↪', 'city_23', 'city_28', 'city_36', 'city_50', 'city_57', 'city_61', 'city_65', 'city_
↪67', 'city_75', 'city_97') THEN 1.017
      WHEN city IN ('city_11', 'city_21', 'city_74') THEN -1.455
      ELSE -0.209
    END AS city,
    CASE
      WHEN (city_development_index IS NULL OR city_development_index = 'NaN') THEN 0
      WHEN city_development_index <= 0.6245 THEN -1.454
      WHEN city_development_index <= 0.7915 THEN -0.121
      WHEN city_development_index <= 0.9235 THEN 0.461
      ELSE 1.101
    END AS city_development_index,
    CASE
      WHEN (company_size IS NULL OR company_size = 'NaN') THEN -0.717
      WHEN company_size <= 74.0 THEN 0.221
      ELSE 0.467
    END AS company_size,
    CASE
      WHEN (company_type IS NULL OR LOWER(CAST(company_type AS VARCHAR(50))) = 'nan')␣
↪THEN -0.64
      WHEN company_type IN ('Early Stage Startup', 'NGO', 'Other', 'Public Sector') THEN␣
↪0.164
      WHEN company_type = 'Funded Startup' THEN 0.737
      WHEN company_type = 'Pvt Ltd' THEN 0.398
      ELSE 0
    END AS company_type,
    CASE
      WHEN (experience IS NULL OR experience = 'NaN') THEN 0
      WHEN experience <= 1.5 THEN -0.811
```

(continues on next page)

(continued from previous page)

```
      WHEN experience <= 7.5 THEN -0.319
      WHEN experience <= 11.5 THEN 0.119
      ELSE 0.533
    END AS experience,
    CASE
      WHEN (enrolled_university IS NULL OR LOWER(CAST(enrolled_university AS␣
↪VARCHAR(50))) = 'nan') THEN -0.327
      WHEN enrolled_university = 'Full time course' THEN -0.614
      WHEN enrolled_university = 'Part time course' THEN 0.026
      WHEN enrolled_university = 'no_enrollment' THEN 0.208
      ELSE 0
    END AS enrolled_university,
    CASE
      WHEN (education_level IS NULL OR LOWER(CAST(education_level AS VARCHAR(50))) = 'nan
↪') THEN 0.21
      WHEN education_level = 'Graduate' THEN -0.166
      WHEN education_level = 'High School' THEN 0.34
      WHEN education_level = 'Masters' THEN 0.21
      WHEN education_level IN ('Phd', 'Primary School') THEN 0.704
      ELSE 0
    END AS education_level
  FROM global_temp.TABLE_1) as WOE_TAB
```

Check the SQL query by PySpark

[23]:
```
from pyspark.sql import SparkSession
```

[ ]:
```
spark = SparkSession.builder \
                    .master("local[2]") \
                    .appName("spark-course") \
                    .config("spark.driver.memory", "512m") \
                    .getOrCreate()
sc = spark.sparkContext
```

[24]:
```
spark_df = spark.read.csv("jobs_train.csv", header=True)
spark_df.createGlobalTempView("TABLE_1")
```

[25]:
```
res = spark.sql(sql_query).toPandas()
```

[26]:
```
res
```

[26]:

	PROB	city	city_development_index	company_size	company_type	\
0	0.365512	0.213	0.461	-0.717	-0.640	
1	0.195716	-0.209	-0.121	0.467	0.398	
2	0.835002	-1.455	-1.454	-0.717	-0.640	
3	0.476161	-0.209	-0.121	-0.717	0.398	
4	0.117694	-0.209	-0.121	0.467	0.737	
...	...	...	...	...	...	
19153	0.275602	1.017	0.461	-0.717	-0.640	
19154	0.365512	0.213	0.461	-0.717	-0.640	
19155	0.126794	0.213	0.461	0.467	0.398	

(continues on next page)

(continued from previous page)

```
19156  0.060842   1.017                 0.461        0.467        0.398
19157  0.130552   1.017                 0.461       -0.717       -0.640

      experience enrolled_university education_level
0          0.533               0.208          -0.166
1          0.533               0.208          -0.166
2         -0.319              -0.614          -0.166
3         -0.811              -0.327          -0.166
4          0.533               0.208           0.210
...          ...                 ...             ...
19153      0.533               0.208          -0.166
19154      0.533               0.208          -0.166
19155      0.533               0.208          -0.166
19156     -0.811               0.208           0.340
19157     -0.319               0.208           0.704

[19158 rows x 8 columns]
```

[27]:
```
sc.stop()
```

[28]:
```
full_prediction = model.predict_proba(data)
full_prediction
```

[28]:
```
array([0.36557352, 0.19577798, 0.83497665, ..., 0.12678668, 0.06083813,
       0.13061427])
```

[29]:
```
(res['PROB'] - full_prediction).abs().max()
```

[29]:
```
0.0002878641803194526
```

2.1.3 Tutorial 3: SQL data source

Official LightAutoML github repository is here

Preparing

Step 1. Install LightAutoML

Uncomment if doesn't clone repository by git. (ex.: colab, kaggle version)

[1]:
```
#! pip install -U lightautoml
```

Step 2. Import necessary libraries

[2]:
```
# Standard python libraries
import os
import time
import requests
# Installed libraries
import numpy as np
import pandas as pd
from sklearn.metrics import roc_auc_score
from sklearn.model_selection import train_test_split
import torch

# Imports from our package
import gensim
from lightautoml.automl.presets.tabular_presets import TabularAutoML,␣
↪TabularUtilizedAutoML
from lightautoml.dataset.roles import DatetimeRole
from lightautoml.tasks import Task
```

Step 3. Parameters

[3]:
```
N_THREADS = 8 # threads cnt for lgbm and linear models
N_FOLDS = 5 # folds cnt for AutoML
RANDOM_STATE = 42 # fixed random state for various reasons
TEST_SIZE = 0.2 # Test size for metric check
TIMEOUT = 300 # Time in seconds for automl run
TARGET_NAME = 'TARGET' # Target column name
```

Step 4. Fix torch number of threads and numpy seed

[4]:
```
np.random.seed(RANDOM_STATE)
torch.set_num_threads(N_THREADS)
```

Step 5. Example data load

Load a dataset from the repository if doesn't clone repository by git.

[5]:
```
DATASET_DIR = '../data/'
DATASET_NAME = 'sampled_app_train.csv'
DATASET_FULLNAME = os.path.join(DATASET_DIR, DATASET_NAME)
DATASET_URL = 'https://raw.githubusercontent.com/AILab-MLTools/LightAutoML/master/
↪examples/data/sampled_app_train.csv'
```

[6]:
```
%%time

if not os.path.exists(DATASET_FULLNAME):
    os.makedirs(DATASET_DIR, exist_ok=True)

    dataset = requests.get(DATASET_URL).text
    with open(DATASET_FULLNAME, 'w') as output:
```

(continues on next page)

(continued from previous page)

```
        output.write(dataset)
```

```
CPU times: user 29 µs, sys: 20 µs, total: 49 µs
Wall time: 68.4 µs
```

[7]:
```
%%time

data = pd.read_csv(DATASET_FULLNAME)
data.head()
```

```
CPU times: user 104 ms, sys: 19.8 ms, total: 123 ms
Wall time: 122 ms
```

[7]:

	SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANNUITY	...	FLAG_DOCUMENT_18	FLAG_DOCUMENT_19	FLAG_DOCUMENT_20	FLAG_DOCUMENT_21	AMT_REQ_CREDIT_BUREAU_HOUR	AMT_REQ_CREDIT_BUREAU_DAY	AMT_REQ_CREDIT_BUREAU_WEEK	AMT_REQ_CREDIT_BUREAU_MON	AMT_REQ_CREDIT_BUREAU_QRT	AMT_REQ_CREDIT_BUREAU_YEAR
0	313802	0	Cash loans	M	N	Y	0	270000.0	327024.0	15372.0	...	0	0	0	0	0.0	0.0	0.0	0.0	0.0	1.0
1	319656	0	Cash loans	F	N	N	0	108000.0	675000.0	19737.0	...	0	0	0	0	0.0	0.0	0.0	0.0	0.0	0.0
2	207678	0	Revolving loans	F	Y	Y	2	112500.0	270000.0	13500.0	...	0	0	0	0	0.0	0.0	0.0	0.0	0.0	1.0
3	381593	0	Cash loans	F	N	N	1	67500.0	142200.0	9630.0	...	0	0	0	0	0.0	0.0	0.0	0.0	0.0	4.0
4	258153	0	Cash loans	F	Y	Y	0	337500.0	1483231.5	46570.5	...	0	0	0	0	0.0	0.0	0.0	2.0	0.0	0.0

(continues on next page)

(continued from previous page)

```
4                          0.0                          0.0

[5 rows x 122 columns]
```

Step 6. (Optional) Some user feature preparation

Cell below shows some user feature preparations to create task more difficult (this block can be omitted if you don't want to change the initial data):

[8]:
```
%%time

data['BIRTH_DATE'] = (np.datetime64('2018-01-01') + data['DAYS_BIRTH'].astype(np.dtype(
↪'timedelta64[D]'))).astype(str)
data['EMP_DATE'] = (np.datetime64('2018-01-01') + np.clip(data['DAYS_EMPLOYED'], None,␣
↪0).astype(np.dtype('timedelta64[D]'))
                    ).astype(str)

data['constant'] = 1
data['allnan'] = np.nan

data['report_dt'] = np.datetime64('2018-01-01')

data.drop(['DAYS_BIRTH', 'DAYS_EMPLOYED'], axis=1, inplace=True)
```

```
CPU times: user 105 ms, sys: 8.82 ms, total: 114 ms
Wall time: 112 ms
```

Step 7. (Optional) Data splitting for train-test

Block below can be omitted if you are going to train model only or you have specific train and test files:

[9]:
```
%%time

train_data, test_data = train_test_split(data,
                                         test_size=TEST_SIZE,
                                         stratify=data[TARGET_NAME],
                                         random_state=RANDOM_STATE)
print('Data splitted. Parts sizes: train_data = {}, test_data = {}'
              .format(train_data.shape, test_data.shape))
```

```
Data splitted. Parts sizes: train_data = (8000, 125), test_data = (2000, 125)
CPU times: user 11.2 ms, sys: 0 ns, total: 11.2 ms
Wall time: 9.95 ms
```

[10]:
```
train_data.head()
```

[10]:

	SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	\
6444	112261	0	Cash loans	F	N	
3586	115058	0	Cash loans	F	N	
9349	326623	0	Cash loans	F	N	
7734	191976	0	Cash loans	M	Y	
2174	281519	0	Revolving loans	F	N	

(continues on next page)

(continued from previous page)

```
     FLAG_OWN_REALTY  CNT_CHILDREN  AMT_INCOME_TOTAL  AMT_CREDIT  AMT_ANNUITY  \
6444               N             1           90000.0    640080.0      31261.5
3586               Y             0          180000.0    239850.0      23850.0
9349               Y             0          112500.0    337500.0      31086.0
7734               Y             1           67500.0    135000.0       9018.0
2174               Y             0           67500.0    202500.0      10125.0

      ...  AMT_REQ_CREDIT_BUREAU_DAY AMT_REQ_CREDIT_BUREAU_WEEK  \
6444  ...                        0.0                        0.0
3586  ...                        0.0                        0.0
9349  ...                        0.0                        0.0
7734  ...                        NaN                        NaN
2174  ...                        0.0                        0.0

     AMT_REQ_CREDIT_BUREAU_MON AMT_REQ_CREDIT_BUREAU_QRT  \
6444                       0.0                       1.0
3586                       0.0                       0.0
9349                       0.0                       0.0
7734                       NaN                       NaN
2174                       0.0                       0.0

     AMT_REQ_CREDIT_BUREAU_YEAR  BIRTH_DATE    EMP_DATE  constant  allnan  \
6444                        0.0  1985-06-28  2012-06-21         1     NaN
3586                        3.0  1953-12-27  2018-01-01         1     NaN
9349                        2.0  1975-06-21  2016-06-17         1     NaN
7734                        NaN  1988-04-27  2009-06-05         1     NaN
2174                        2.0  1975-06-13  1997-01-22         1     NaN

      report_dt
6444 2018-01-01
3586 2018-01-01
9349 2018-01-01
7734 2018-01-01
2174 2018-01-01

[5 rows x 125 columns]
```

Step 8. (Optional) Reading data from SqlDataSource

Preparing datasets as SQLite data bases

[11]:
```
import sqlite3 as sql

for _fname in ('train.db', 'test.db'):
    if os.path.exists(_fname):
        os.remove(_fname)

train_db = sql.connect('train.db')
train_data.to_sql('data', train_db)

test_db = sql.connect('test.db')
test_data.to_sql('data', test_db)
```

Using dataset wrapper for a connection

[12]:
```
from lightautoml.reader.tabular_batch_generator import SqlDataSource

# train_data is replaced with a wrapper for an SQLAlchemy connection
# Wrapper requires SQLAlchemy connection string and query to obtain data from
train_data = SqlDataSource('sqlite:///train.db', 'select * from data', index='index')
test_data = SqlDataSource('sqlite:///test.db', 'select * from data', index='index')
```

AutoML preset usage

Step 1. Create Task

[13]:
```
%%time

task = Task('binary', )
```

```
CPU times: user 6.11 ms, sys: 1.41 ms, total: 7.52 ms
Wall time: 5.65 ms
```

Step 2. Setup columns roles

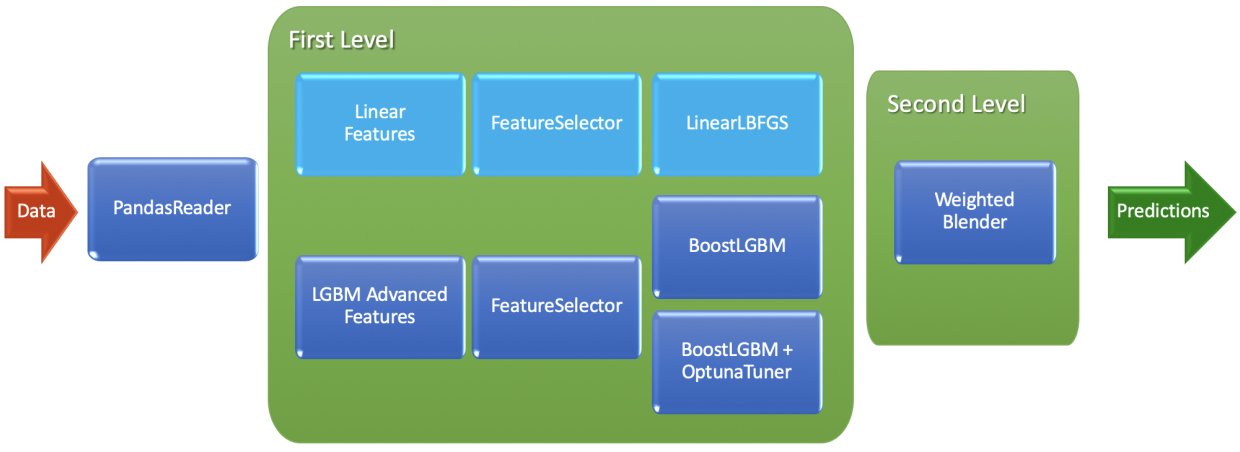
Roles setup here set target column and base date, which is used to calculate date differences:

[14]:
```
%%time

roles = {'target': TARGET_NAME,
         DatetimeRole(base_date=True, seasonality=(), base_feats=False): 'report_dt',
         }
```

```
CPU times: user 48 µs, sys: 32 µs, total: 80 µs
Wall time: 95.1 µs
```

Step 3. Create AutoML from preset

To create AutoML model here we use `TabularAutoML` preset, which looks like:



All params we set above can be send inside preset to change its configuration:

[15]:
```
%%time

automl = TabularAutoML(task = task,
                       timeout = TIMEOUT,
                       general_params = {'nested_cv': False, 'use_algos': [['linear_l2',
↪'lgb', 'lgb_tuned']]},
                       reader_params = {'cv': N_FOLDS, 'random_state': RANDOM_STATE},
                       tuning_params = {'max_tuning_iter': 20, 'max_tuning_time': 30},
                       lgb_params = {'default_params': {'num_threads': N_THREADS}})
oof_pred = automl.fit_predict(train_data, roles = roles)
print('oof_pred:\n{}\nShape = {}'.format(oof_pred, oof_pred.shape))
```

```
oof_pred:
array([[0.0226106 ],
       [0.02359573],
       [0.02438388],
       ...,
       [0.02287533],
       [0.15669319],
       [0.08664417]], dtype=float32)
Shape = (8000, 1)
CPU times: user 4min 19s, sys: 3.59 s, total: 4min 23s
Wall time: 1min 11s
```

Step 4. Predict to test data and check scores

[16]:
```
%%time

test_pred = automl.predict(test_data)
print('Prediction for test data:\n{}\nShape = {}'
              .format(test_pred, test_pred.shape))

print('Check scores...')
print('OOF score: {}'.format(roc_auc_score(train_data.data[TARGET_NAME].values, oof_pred.
↪data[:, 0])))
print('TEST score: {}'.format(roc_auc_score(test_data.data[TARGET_NAME].values, test_
↪pred.data[:, 0])))
```

```
Prediction for test data:
array([[0.05828221],
       [0.07749337],
       [0.02520473],
       ...,
       [0.05070161],
       [0.0373171 ],
       [0.23640296]], dtype=float32)
Shape = (2000, 1)
Check scores...
OOF score: 0.7500913646530726
TEST score: 0.7331657608695653
CPU times: user 1.05 s, sys: 4.05 ms, total: 1.06 s
Wall time: 449 ms
```

Step 5. Create AutoML with time utilization

Below we are going to create specific AutoML preset for TIMEOUT utilization (try to spend it as much as possible):

[20]:
```
%%time

automl = TabularUtilizedAutoML(task = task,
                        timeout = TIMEOUT,
                        general_params = {'nested_cv': False, 'use_algos': [['linear_l2',
↪'lgb', 'lgb_tuned']]},
                        reader_params = {'cv': N_FOLDS, 'random_state': RANDOM_STATE},
                        tuning_params = {'max_tuning_iter': 20, 'max_tuning_time': 30},
                        lgb_params = {'default_params': {'num_threads': N_THREADS}})
oof_pred = automl.fit_predict(train_data, roles = roles)
print('oof_pred:\n{}\nShape = {}'.format(oof_pred, oof_pred.shape))
```

```
oof_pred:
array([[0.0343032 ],
       [0.01933593],
       [0.02276292],
       ...,
       [0.02349434],
       [0.17084229],
       [0.09522362]], dtype=float32)
Shape = (8000, 1)
CPU times: user 16min 54s, sys: 12.3 s, total: 17min 6s
Wall time: 4min 29s
```

Step 6. Predict to test data and check scores for utilized automl

[21]:
```
%%time

test_pred = automl.predict(test_data)
print('Prediction for test data:\n{}\nShape = {}'
              .format(test_pred, test_pred.shape))

print('Check scores...')
print('OOF score: {}'.format(roc_auc_score(train_data.data[TARGET_NAME].values, oof_pred.
↪data[:, 0])))
print('TEST score: {}'.format(roc_auc_score(test_data.data[TARGET_NAME].values, test_
↪pred.data[:, 0])))
```

```
Prediction for test data:
array([[0.05981494],
       [0.07601136],
       [0.02678316],
       ...,
       [0.04721078],
       [0.03855655],
       [0.19377196]], dtype=float32)
Shape = (2000, 1)
Check scores...
OOF score: 0.7586795357421285
TEST score: 0.730679347826087
```

(continues on next page)

(continued from previous page)

```
CPU times: user 2.99 s, sys: 64.1 ms, total: 3.05 s
Wall time: 1.21 s
```

[ ]:

2.1.4 Tutorial 4: Interpretation Tutorial (requires GPU)

Official LightAutoML github repository is here

Some of HTML static content is not loading, to solve this problem you can use nbviewer. Link on tutorial on nbviewer here.

The last years deep neural networks / gradient boosting / ensembles of models allow to improve the soulution quality of many application task in field of natural language processing (NLP). The indicators of this improvement describe the partial behavior of the model and can hide errors, for example, errors in the construction of the model, errors in data collection. All this can be critical in tasks related to the processing of medical, forensic, banking data. In this tutorial we will check the NLP interpretation module of automl.

Download library and make some imports

[1]:
```
# !pip install lightautoml
```

[2]:
```
import shutil

import numpy as np
import pandas as pd

from sklearn.metrics import roc_auc_score, mean_squared_error
from sklearn.model_selection import train_test_split

from lightautoml.automl.presets.text_presets import TabularNLPAutoML
from lightautoml.tasks import Task

from lightautoml.addons.interpretation import LimeTextExplainer, L2XTextExplainer

import transformers
transformers.logging.set_verbosity(50)

import pickle
```

Download data

For this tutorial we will use train dataset (train.csv) from Jigsaw-Toxic-Comment-Classification-Challage. The dataset contains textual comments and 6 attributes of this text (toxic, serve_toxic, obscene, treat, insult, identity_hate). For now, we will use only toxic attribute.

[3]:
```
# train.csv file from
# https://www.kaggle.com/c/jigsaw-toxic-comment-classification-challenge/overview

data = pd.read_csv('train.csv')
data
```

[3]:
```
                     id                                       comment_text  \
0       0000997932d777bf  Explanation\nWhy the edits made under my usern...
1       000103f0d9cfb60f  D'aww! He matches this background colour I'm s...
2       000113f07ec002fd  Hey man, I'm really not trying to edit war. It...
3       0001b41b1c6bb37e  "\nMore\nI can't make any real suggestions on ...
4       0001d958c54c6e35  You, sir, are my hero. Any chance you remember...
...                  ...                                                ...
159566  ffe987279560d7ff  ":::::And for the second time of asking, when ...
159567  ffea4adeee384e90  You should be ashamed of yourself \n\nThat is ...
159568  ffee36eab5c267c9  Spitzer \n\nUmm, theres no actual article for ...
159569  fff125370e4aaaf3  And it looks like it was actually you who put ...
159570  fff46fc426af1f9a  "\nAnd ... I really don't think you understand...

        toxic  severe_toxic  obscene  threat  insult  identity_hate
0           0             0        0       0       0              0
1           0             0        0       0       0              0
2           0             0        0       0       0              0
3           0             0        0       0       0              0
4           0             0        0       0       0              0
...       ...           ...      ...     ...     ...            ...
159566      0             0        0       0       0              0
159567      0             0        0       0       0              0
159568      0             0        0       0       0              0
159569      0             0        0       0       0              0
159570      0             0        0       0       0              0

[159571 rows x 8 columns]
```

Usage of AutoML

We will use standard `lightautoml.automl.presets.text_presets.TabularNLPAutoML` preset with finetuned TinyBERT from Hugging Face.

[4]:
```
np.random.seed(42)

train, test = train_test_split(data, test_size=0.2, random_state=42)

roles = {
    'text': ['comment_text'],
    'drop': ['id', 'severe_toxic', 'obscene', 'threat', 'insult', 'identity_hate'],
    'target': 'toxic'
}

task = Task('binary')

automl = TabularNLPAutoML(
    task=task,
```

(continues on next page)

(continued from previous page)

```
    timeout=3600,
    cpu_limit=1,
    gpu_ids='0',
    general_params={
        'nested_cv': False,
        'use_algos': [['nn']]
    },
    autonlp_params={
        'sent_scaler': 'l2'
    },
    text_params={
        'lang': 'en',
        'bert_model': 'prajjwal1/bert-tiny'
    },
    nn_params={
        'opt_params': {'lr': 1e-5},
        'max_length': 128,
        'bs': 32,
        'n_epochs': 7,
    }

)
```

[5]:
```
%%time
oof_pred = automl.fit_predict(train, roles=roles, verbose = 10)
test_pred = automl.predict(test)
not_nan = np.any(~np.isnan(oof_pred.data), axis=1)

print('Check scores:')
print('OOF score: {}'.format(roc_auc_score(train[roles['target']].values[not_nan], oof_
↪pred.data[not_nan][:, 0])))
print('TEST score: {}'.format(roc_auc_score(test[roles['target']].values, test_pred.
↪data[:, 0])))
```

```
[11:22:30] Stdout logging level is DEBUG.
[11:22:30] Model language mode: en
[11:22:30] Task: binary

[11:22:30] Start automl preset with listed constraints:
[11:22:30] - time: 3600.00 seconds
[11:22:30] - CPU: 1 cores
[11:22:30] - memory: 16 GB

[11:22:30] Train data shape: (127656, 8)

[11:22:30] Layer 1 train process start. Time left 3599.85 secs
[11:22:31] Start fitting Lvl_0_Pipe_0_Mod_0_TorchNN ...
[11:22:31] Training params: {'bs': 32, 'num_workers': 1, 'max_length': 128, 'opt_params':
↪ {'lr': 1e-05}, 'scheduler_params': {'patience': 5, 'factor': 0.5, 'verbose': True},
↪'is_snap': False, 'snap_params': {'k': 1, 'early_stopping': True, 'patience': 1, 'swa':
↪ False}, 'init_bias': True, 'n_epochs': 7, 'input_bn': False, 'emb_dropout': 0.1, 'emb_
```

(continues on next page)

(continued from previous page)

```
↪ratio': 3, 'max_emb_size': 50, 'bert_name': 'prajjwal1/bert-tiny', 'pooling': 'cls',
↪'device': device(type='cuda', index=0), 'use_cont': True, 'use_cat': True, 'use_text':␣
↪True, 'lang': 'en', 'deterministic': False, 'multigpu': False, 'random_state': 42,
↪'path_to_save': None, 'verbose_inside': None, 'verbose': 1, 'device_ids': None, 'n_out
↪': 1, 'cat_features': [], 'cat_dims': [], 'cont_features': [], 'cont_dim': 0, 'text_
↪features': ['concated__comment_text'], 'bias': array([[-2.24401446]])}
[11:22:31] ===== Start working with fold 0 for Lvl_0_Pipe_0_Mod_0_TorchNN =====
[11:22:36] number of text features: 1
[11:22:36] number of categorical features: 0
[11:22:36] number of continuous features: 0
train (loss=0.257356): 100%|| 2660/2660 [02:12<00:00, 20.13it/s]
val: 100%|| 1330/1330 [01:07<00:00, 19.83it/s]
[11:25:59] Epoch: 0, train loss: 0.25735557079315186, val loss: 0.19599375128746033, val␣
↪metric: 0.9640350800072578

train (loss=0.168968): 100%|| 2660/2660 [02:09<00:00, 20.61it/s]
val: 100%|| 1330/1330 [01:04<00:00, 20.58it/s]
[11:29:13] Epoch: 1, train loss: 0.16896754503250122, val loss: 0.14401142299175262, val␣
↪metric: 0.9713461808486132

train (loss=0.131891): 100%|| 2660/2660 [02:09<00:00, 20.49it/s]
val: 100%|| 1330/1330 [01:03<00:00, 20.87it/s]
[11:32:26] Epoch: 2, train loss: 0.1318911910057068, val loss: 0.12361849099397659, val␣
↪metric: 0.9742718921629787

train (loss=0.114705): 100%|| 2660/2660 [02:07<00:00, 20.90it/s]
val: 100%|| 1330/1330 [01:04<00:00, 20.76it/s]
[11:35:38] Epoch: 3, train loss: 0.11470535397529602, val loss: 0.11394938081502914, val␣
↪metric: 0.9763582643756192

train (loss=0.103179): 100%|| 2660/2660 [02:09<00:00, 20.54it/s]
val: 100%|| 1330/1330 [01:05<00:00, 20.36it/s]
[11:38:53] Epoch: 4, train loss: 0.10317856818437576, val loss: 0.10656153410673141, val␣
↪metric: 0.9775081138714583

train (loss=0.0965996): 100%|| 2660/2660 [02:09<00:00, 20.49it/s]
val: 100%|| 1330/1330 [01:05<00:00, 20.24it/s]
[11:42:08] Epoch: 5, train loss: 0.09659960865974426, val loss: 0.10427780449390411, val␣
↪metric: 0.9783243683208365

train (loss=0.090561): 100%|| 2660/2660 [02:11<00:00, 20.24it/s]
val: 100%|| 1330/1330 [01:02<00:00, 21.23it/s]
[11:45:22] Epoch: 6, train loss: 0.09056100249290466, val loss: 0.10337436944246292, val␣
↪metric: 0.9788043902058639
[11:45:23] ===== Start working with fold 1 for Lvl_0_Pipe_0_Mod_0_TorchNN =====
```

(continues on next page)

(continued from previous page)

```
[11:45:28] number of text features: 1
[11:45:28] number of categorical features: 0
[11:45:28] number of continuous features: 0
```

```
train (loss=0.257485): 100%|| 2660/2660 [02:04<00:00, 21.30it/s]
val: 100%|| 1330/1330 [01:04<00:00, 20.67it/s]
```

```
[11:48:38] Epoch: 0, train loss: 0.2574850618839264, val loss: 0.19478833675384521, val␣
↪metric: 0.961936968917119
```

```
train (loss=0.170552): 100%|| 2660/2660 [02:08<00:00, 20.72it/s]
val: 100%|| 1330/1330 [01:06<00:00, 19.87it/s]
```

```
[11:51:53] Epoch: 1, train loss: 0.1705523431301117, val loss: 0.1437842845916748, val␣
↪metric: 0.970873732336761
```

```
train (loss=0.132485): 100%|| 2660/2660 [02:05<00:00, 21.15it/s]
val: 100%|| 1330/1330 [01:03<00:00, 20.97it/s]
```

```
[11:55:03] Epoch: 2, train loss: 0.13248467445373535, val loss: 0.12127983570098877, val␣
↪metric: 0.9751468710522353
```

```
train (loss=0.11448): 100%|| 2660/2660 [02:03<00:00, 21.51it/s]
val: 100%|| 1330/1330 [01:02<00:00, 21.11it/s]
```

```
[11:58:09] Epoch: 3, train loss: 0.11447965353727341, val loss: 0.11149459332227707, val␣
↪metric: 0.9768346789459879
```

```
train (loss=0.103458): 100%|| 2660/2660 [02:06<00:00, 20.99it/s]
val: 100%|| 1330/1330 [01:04<00:00, 20.65it/s]
```

```
[12:01:20] Epoch: 4, train loss: 0.10345754027366638, val loss: 0.10722416639328003, val␣
↪metric: 0.9782435623593337
```

```
train (loss=0.0963441): 100%|| 2660/2660 [02:05<00:00, 21.17it/s]
val: 100%|| 1330/1330 [01:03<00:00, 20.91it/s]
```

```
[12:04:30] Epoch: 5, train loss: 0.09634406119585037, val loss: 0.10441421717405319, val␣
↪metric: 0.978748563376753
```

```
train (loss=0.0900231): 100%|| 2660/2660 [02:05<00:00, 21.17it/s]
val: 100%|| 1330/1330 [01:03<00:00, 20.84it/s]
```

```
[12:07:39] Epoch: 6, train loss: 0.09002314507961273, val loss: 0.10312184691429138, val␣
↪metric: 0.9791290354336872
```

```
[12:07:40] ===== Start working with fold 2 for Lvl_0_Pipe_0_Mod_0_TorchNN =====
[12:07:44] number of text features: 1
[12:07:44] number of categorical features: 0
[12:07:44] number of continuous features: 0
```

```
train (loss=0.257448): 100%|| 2660/2660 [02:04<00:00, 21.45it/s]
val: 100%|| 1330/1330 [01:00<00:00, 21.91it/s]
```

```
[12:10:50] Epoch: 0, train loss: 0.2574479281902313, val loss: 0.19449889659881592, val␣
↪metric: 0.9648288318293945

train (loss=0.169502): 100%|| 2660/2660 [02:03<00:00, 21.52it/s]
val: 100%|| 1330/1330 [01:01<00:00, 21.79it/s]

[12:13:55] Epoch: 1, train loss: 0.1695016324520111, val loss: 0.14307956397533417, val␣
↪metric: 0.9706200035841146

train (loss=0.131626): 100%|| 2660/2660 [02:03<00:00, 21.54it/s]
val: 100%|| 1330/1330 [01:00<00:00, 21.84it/s]

[12:16:59] Epoch: 2, train loss: 0.13162554800510406, val loss: 0.12111066281795502, val␣
↪metric: 0.97454294780979

train (loss=0.114015): 100%|| 2660/2660 [02:03<00:00, 21.57it/s]
val: 100%|| 1330/1330 [01:00<00:00, 21.83it/s]

[12:20:04] Epoch: 3, train loss: 0.11401509493589401, val loss: 0.11131983995437622, val␣
↪metric: 0.9763178957078734

train (loss=0.104155): 100%|| 2660/2660 [02:03<00:00, 21.56it/s]
val: 100%|| 1330/1330 [01:00<00:00, 21.87it/s]

[12:23:08] Epoch: 4, train loss: 0.10415521264076233, val loss: 0.10691472887992859, val␣
↪metric: 0.9772204526836245

train (loss=0.0953203): 100%|| 2660/2660 [02:04<00:00, 21.41it/s]
val: 100%|| 1330/1330 [01:01<00:00, 21.67it/s]

[12:26:13] Epoch: 5, train loss: 0.09532025456428528, val loss: 0.10362745076417923, val␣
↪metric: 0.9780747656394276

train (loss=0.0899258): 100%|| 2660/2660 [02:04<00:00, 21.34it/s]
val: 100%|| 1330/1330 [01:01<00:00, 21.68it/s]

[12:29:20] Epoch: 6, train loss: 0.08992581069469452, val loss: 0.10427321493625641, val␣
↪metric: 0.9781931517871759


val: 100%|| 1330/1330 [01:01<00:00, 21.53it/s]

[12:30:21] Early stopping: val loss: 0.10362745076417923, val metric: 0.9780747656394276


[12:30:22] Fitting Lvl_0_Pipe_0_Mod_0_TorchNN finished. score = 0.9782371823652668
[12:30:22] Lvl_0_Pipe_0_Mod_0_TorchNN fitting and predicting completed
[12:30:22] Time left -472.15 secs

[12:30:22] Time limit exceeded. Last level models will be blended and unused pipelines␣
↪will be pruned.

[12:30:22] Layer 1 training completed.

[12:30:22] Automl preset training completed in 4072.15 seconds

[12:30:22] Model description:
Final prediction for new objects (level 0) =
         1.00000 * (3 averaged models Lvl_0_Pipe_0_Mod_0_TorchNN)
```

(continues on next page)

(continued from previous page)

```
[12:30:22] number of text features: 1
[12:30:22] number of categorical features: 0
[12:30:22] number of continuous features: 0
test: 100%|| 998/998 [00:47<00:00, 21.08it/s]
[12:31:15] number of text features: 1
[12:31:15] number of categorical features: 0
[12:31:15] number of continuous features: 0
test: 100%|| 998/998 [00:46<00:00, 21.51it/s]
[12:32:08] number of text features: 1
[12:32:08] number of categorical features: 0
[12:32:08] number of continuous features: 0
test: 100%|| 998/998 [00:46<00:00, 21.47it/s]
Check scores:
OOF score: 0.9782371823652668
TEST score: 0.9807740353486142
CPU times: user 18min 47s, sys: 1min 15s, total: 20min 3s
Wall time: 1h 10min 30s
```

[6]:
```
automl.set_verbosity_level(0) # refuse logging in automl
```

LIME

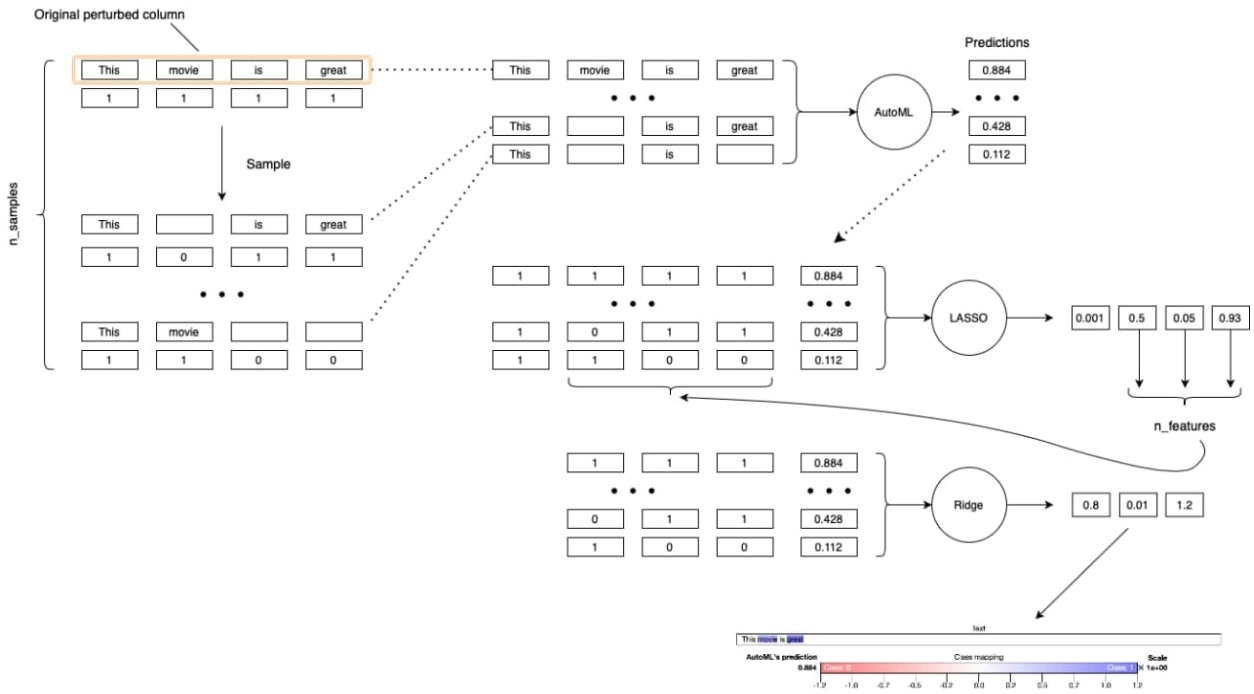
Linear approximation of model nearby selected object. The weights of this linear model is feature attribution for automl's prediction for this object.

Algorithm:

0. Select object to interpret.
1. Select the input text column, that will be explained (`perturb_column`). All other columns of object will be fixed.
2. A dataset of size `n_sample` (by default `5000`) is created by randomly deleting tokens (in groups). Dataset is binary (there is a token if one and no token if zero).
3. Predict with AutoML module target values for created dataset.
4. Optionally, the selection of features (important tokens) is performed using LASSO (`feature_selection='lasso'`, you can also `'none'` to not select and get them all). The number of features used after feature selection is `n_feautres` (`= 10` by default).
5. We train the explained model on this (a linear model with weights, the method of calculating weights is the cosine distance by default, you can also use your own function or the name of the distance from `sklearn.metrics.pairwise_distances`).
6. The weights of the linear model are the interpretation.

P.S. Care about the sentence length. Detokenization works within $O(n^2)$, where n – sentence length.

Scheme of work:



[7]:
```
# LimeTextExplainer for NLP preset
lime = LimeTextExplainer(automl, feature_selection='lasso', force_order=False)
```

Let's try it on neutral text

[8]:
```
exp = lime.explain_instance(test.loc[34019], labels=(0, 1), perturb_column='comment_text
↪')
exp.visualize_in_notebook(1)
```

```
test: 100%|| 157/157 [00:02<00:00, 77.34it/s]
test: 100%|| 157/157 [00:01<00:00, 79.52it/s]
test: 100%|| 157/157 [00:01<00:00, 78.82it/s]
```

Toxic comments

[9]:
```
exp = lime.explain_instance(test.loc[78687], labels=(0, 1), perturb_column='comment_text
↪')
exp.visualize_in_notebook(1)
```

```
test: 100%|| 157/157 [00:01<00:00, 93.42it/s]
test: 100%|| 157/157 [00:01<00:00, 92.47it/s]
test: 100%|| 157/157 [00:01<00:00, 93.75it/s]
```

Let's see on uncertain expamples

[10]:
```
exp = lime.explain_instance(test.loc[4733], labels=(0, 1), perturb_column='comment_text',
↪ n_features=20)
exp.visualize_in_notebook(1)
```

```
test: 100%|| 157/157 [00:02<00:00, 71.46it/s]
test: 100%|| 157/157 [00:02<00:00, 71.48it/s]
test: 100%|| 157/157 [00:02<00:00, 71.15it/s]
```

Let's delete 'important' from this abstract. We can see that automl increase it's probability of toxicity of this abstract

[11]:
```
test.loc[4733, 'comment_text'] = 'Why are you still here ? Can you not find anything␣
↪more to do , like killing yourself ?'
```

[12]:
```
exp = lime.explain_instance(test.loc[4733], labels=(0, 1), perturb_column='comment_text',
↪ n_features=20)
exp.visualize_in_notebook(1)
```

```
test: 100%|| 157/157 [00:02<00:00, 73.97it/s]
test: 100%|| 157/157 [00:02<00:00, 73.07it/s]
test: 100%|| 157/157 [00:02<00:00, 73.22it/s]
```

If we add the 'relability' the AutoML decrease the toxicity probability.

[13]:
```
test.loc[4733, 'comment_text'] = 'Why are you still here ? Can you not find anything␣
↪more to do , like killing yourself ? relability'
```

[14]:
```
exp = lime.explain_instance(test.loc[4733], labels=(0, 1), perturb_column='comment_text',
↪ n_features=20)
exp.visualize_in_notebook(1)
```

```
test: 100%|| 157/157 [00:02<00:00, 68.66it/s]
test: 100%|| 157/157 [00:02<00:00, 64.12it/s]
test: 100%|| 157/157 [00:02<00:00, 68.18it/s]
```

Another example

[15]:
```
exp = lime.explain_instance(test.loc[40112], labels=(0, 1), perturb_column='comment_text
↪', n_features=20)
exp.visualize_in_notebook(1)
```

```
test: 100%|| 157/157 [00:02<00:00, 57.57it/s]
test: 100%|| 157/157 [00:02<00:00, 56.72it/s]
test: 100%|| 157/157 [00:02<00:00, 56.36it/s]
```

Let's delete the toxic words to 'good boy'

[16]:
```
test.loc[40112, 'comment_text'] = "stop editing this, you good boy. why do you have to␣
↪be such a good boy? the ghosts of bill maas' past will haunt you forever!!! MWAHAHHAHAA
↪"
```

[17]:
```
exp = lime.explain_instance(test.loc[40112], labels=(0, 1), perturb_column='comment_text
↪', n_features=20)
exp.visualize_in_notebook(1)
```

```
test: 100%|| 157/157 [00:02<00:00, 55.40it/s]
test: 100%|| 157/157 [00:02<00:00, 56.35it/s]
test: 100%|| 157/157 [00:02<00:00, 55.86it/s]
```

Let's try from neutral make toxic abstract.

[18]:
```
exp = lime.explain_instance(test.loc[18396], labels=(0, 1), perturb_column='comment_text
↪', n_features=20)
exp.visualize_in_notebook(1)
```

```
test: 100%|| 157/157 [00:01<00:00, 101.90it/s]
test: 100%|| 157/157 [00:01<00:00, 100.88it/s]
test: 100%|| 157/157 [00:01<00:00, 99.18it/s]
```

[19]:
```
test.loc[18396] = "Okay , thanks . I will do so . dumbass please"
```

[20]:
```
exp = lime.explain_instance(test.loc[18396], labels=(0, 1), perturb_column='comment_text
↪', n_features=20)
exp.visualize_in_notebook(1)
```

```
test: 100%|| 157/157 [00:01<00:00, 89.68it/s]
test: 100%|| 157/157 [00:01<00:00, 90.71it/s]
test: 100%|| 157/157 [00:01<00:00, 90.35it/s]
```

Adding some happy words

[21]:
```
test.loc[18396] = "Okay , thanks . I will do so . happy dumbass please"
```

[22]:
```
exp = lime.explain_instance(test.loc[18396], labels=(0, 1), perturb_column='comment_text
↪', n_features=20)
exp.visualize_in_notebook(1)
```

```
test: 100%|| 157/157 [00:01<00:00, 86.02it/s]
test: 100%|| 157/157 [00:01<00:00, 87.59it/s]
test: 100%|| 157/157 [00:01<00:00, 85.69it/s]
```

More happy words.

[23]:
```
test.loc[18396] = "Okay , thanks . I will do so . happy cheerful joyfull glorious elated␣
↪dumbass please"
```

[24]:
```
exp = lime.explain_instance(test.loc[18396], labels=(0, 1), perturb_column='comment_text
↪', n_features=20)
exp.visualize_in_notebook(1)
```

```
test: 100%|| 157/157 [00:02<00:00, 75.00it/s]
test: 100%|| 157/157 [00:02<00:00, 74.62it/s]
test: 100%|| 157/157 [00:02<00:00, 74.52it/s]
```

L2X for Regression

For this part the BeerAdvocate we will use. The dataset contains the reviews on alcoholic drinks (texutal comment + 5 attributes: overview, taste, plate, aroma, appearance). For this experiment we will use only `appearance` attribute.

[25]:
```
def download_from_gdrive(file_id, file_name, chunk_size=2**15):
    import requests

    def handle_warning(res):
        for k, v in res.cookies.items():
            if k.startswith("download_warning"):
                return v

    template_url = "https://docs.google.com/uc?export=download"
    session = requests.Session()
    res = session.get(template_url, params={"id": file_id}, stream=True)
    print('GET: {} CODE'.format(res.status_code))
    token = handle_warning(res)
    if token:
        res = session.get(template_url, params={"id": file_id, "confirm": token},␣
↪stream=True)
    print('Started downloading...')
    with open(file_name, 'wb') as f:
        for chunk in res.iter_content(chunk_size):
            if chunk:
                f.write(chunk)
    print('Downloaded.')


download_from_gdrive('1s8PG13Y0BvYM67nNL0EQpdgB5S4gJK9r', 'beeradvocate.tar.gz')
shutil.unpack_archive('beeradvocate.tar.gz', '.')
```

```
GET: 200 CODE
Started downloading...
Downloaded.
```

[26]:
```
train_data = pd.read_csv('./datasets/reviews.aspect0.train.csv')
valid_data = pd.read_csv('./datasets/reviews.aspect0.heldout.csv')

train_data.head()
```

[26]:

	Appearance	Aroma	Palate	Taste	Overall	Review	tokens_number
0	1.5	1.5	2.5	1.5	1.5	the main problem with this beer is that it has...	62
1	2.0	2.0	3.0	2.0	3.0	it is very unfortunate this situation we have ...	179
2	4.0	2.5	3.0	1.5	2.0	appearance is a light golden yellow with a thi...	79
3	4.5	3.5	2.0	3.5	3.0	it has a great color to the body . this beer p...	87
4	4.0	4.5	1.0	1.5	1.0	though this beer is , or course , not carbonat...	246

Train AutoML

In this part we use BERT-Base model.

[27]:
```
roles = {
    'text': ['Review'],
    'drop': ['tokens_number', 'Aroma', 'Palete', 'Taste', 'Overall'],
    'target': 'Appearance'
}

task = Task('reg')

automl = TabularNLPAutoML(
    task=task,
    timeout=3600,
    cpu_limit=1,
    gpu_ids='1',
    general_params={
        'nested_cv': False,
        'use_algos': [['nn']],
        'n_folds': 3
    },
    reader_params={
        'cv': 3
    },
    autonlp_params={
        'sent_scaler': 'l2'
    },
    text_params={
        'lang': 'en',
        'bert_model': 'bert-base-uncased'
    },
    nn_params={
        'opt_params': {'lr': 1e-5},
        'max_length': 128,
        'bs': 32,
        'n_epochs': 7,
    },
)

oof_pred = automl.fit_predict(train_data, roles=roles, verbose=2)
test_pred = automl.predict(valid_data)
not_nan = np.any(~np.isnan(oof_pred.data), axis=1)

print('Check scores:')
print('OOF score: {}'.format(mean_squared_error(train_data[roles['target']].values[not_
↪nan], oof_pred.data[not_nan][:, 0])))
print('TEST score: {}'.format(mean_squared_error(valid_data[roles['target']].values,␣
↪test_pred.data[:, 0])))
```

```
[12:38:00] Stdout logging level is INFO2.
[12:38:00] Task: reg

[12:38:00] Start automl preset with listed constraints:
```

(continues on next page)

(continued from previous page)

```
[12:38:00] - time: 3600.00 seconds
[12:38:00] - CPU: 1 cores
[12:38:00] - memory: 16 GB

[12:38:00] Train data shape: (80000, 7)

[12:38:01] Layer 1 train process start. Time left 3599.63 secs
[12:38:01] Start fitting Lvl_0_Pipe_0_Mod_0_TorchNN ...
[12:38:01] ===== Start working with fold 0 for Lvl_0_Pipe_0_Mod_0_TorchNN =====
```

```
train (loss=0.755747): 100%|| 1667/1667 [06:45<00:00,  4.11it/s]
val: 100%|| 834/834 [02:04<00:00,  6.68it/s]
train (loss=0.442306): 100%|| 1667/1667 [06:48<00:00,  4.08it/s]
val: 100%|| 834/834 [02:05<00:00,  6.66it/s]
train (loss=0.344638): 100%|| 1667/1667 [06:52<00:00,  4.04it/s]
val: 100%|| 834/834 [02:06<00:00,  6.61it/s]
val: 100%|| 834/834 [02:05<00:00,  6.64it/s]
```

```
[13:07:23] ===== Start working with fold 1 for Lvl_0_Pipe_0_Mod_0_TorchNN =====
```

```
train (loss=0.760973): 100%|| 1667/1667 [06:51<00:00,  4.05it/s]
val: 100%|| 834/834 [02:06<00:00,  6.62it/s]
train (loss=0.44357): 100%|| 1667/1667 [06:50<00:00,  4.06it/s]
val: 100%|| 834/834 [02:06<00:00,  6.61it/s]
train (loss=0.343338): 100%|| 1667/1667 [06:49<00:00,  4.07it/s]
val: 100%|| 834/834 [02:05<00:00,  6.66it/s]
val: 100%|| 834/834 [02:05<00:00,  6.66it/s]
```

```
[13:36:29] Time limit exceeded after calculating fold 1

[13:36:29] Fitting Lvl_0_Pipe_0_Mod_0_TorchNN finished. score = -0.46728458911890136
[13:36:29] Lvl_0_Pipe_0_Mod_0_TorchNN fitting and predicting completed
[13:36:29] Time left 91.29 secs

[13:36:29] Time limit exceeded in one of the tasks. AutoML will blend level 1 models.

[13:36:29] Layer 1 training completed.

[13:36:29] Automl preset training completed in 3508.71 seconds

[13:36:29] Model description:
Final prediction for new objects (level 0) =
         1.00000 * (2 averaged models Lvl_0_Pipe_0_Mod_0_TorchNN)
```

```
test: 100%|| 313/313 [00:47<00:00,  6.63it/s]
test: 100%|| 313/313 [00:47<00:00,  6.64it/s]
```

```
Check scores:
OOF score: 0.46728458911890136
TEST score: 0.43322843977913716
```

[28]:
```
# >>> about 2gb
with open('apperance_model.pkl', 'wb') as f:
    pickle.dump(automl, f)
```

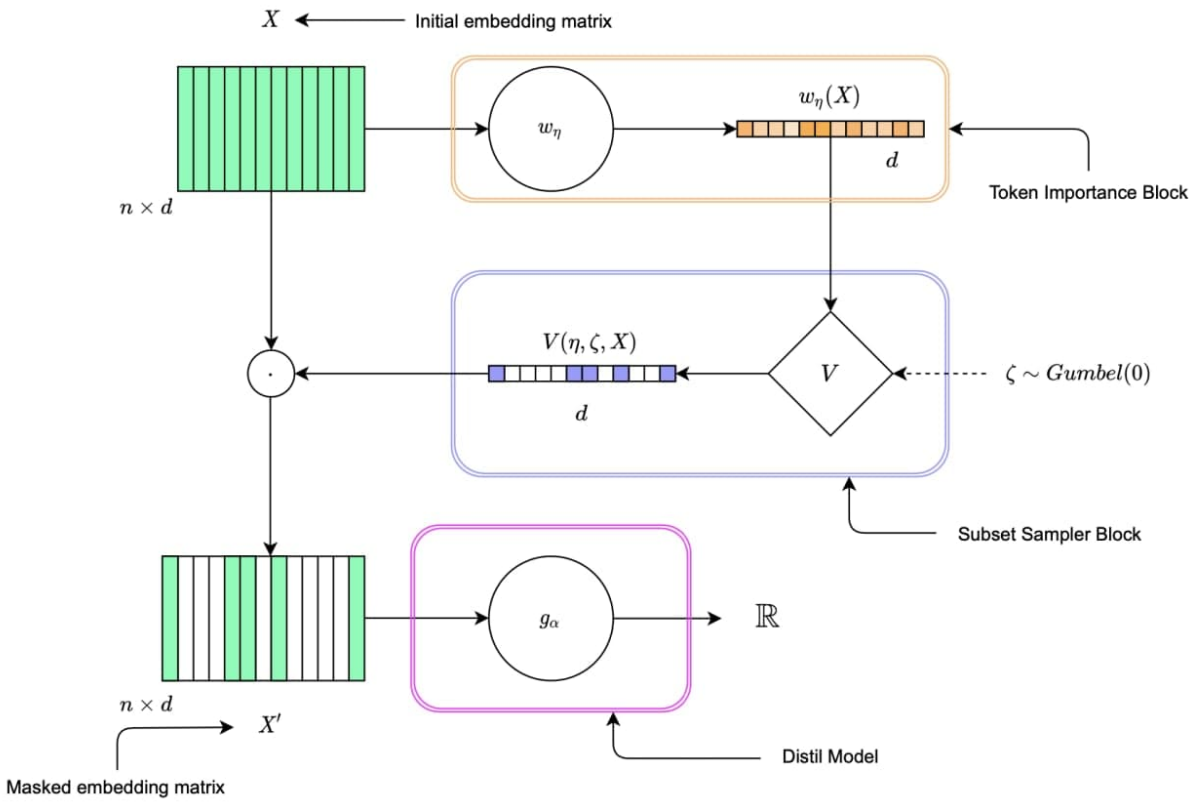
[29]:
```
with open('apperance_model.pkl', 'rb') as f:
    automl = pickle.load(f)
automl.set_verbosity_level(2)
```

```
[13:38:29] Stdout logging level is INFO2.
```

L2X

Algorithm.

0. The general idea of method is find the most informative subset of tokens with respect to target using Mutual Information. The number of tokens in this subset is fixed and equals `n_important`.
1. There is may be some misunderstanding with tokenization that used inside models in automl and tokenization in this method. L2X has its own tokenization, so they are different. If it isn't set we infer it from default tokenization for language in `text_params` of `TabularNLPAutoML`. Else you can set it with language: `'ru'` or `'en'` for russian and english languages, respectively. Also it can be scepcified as callable function that from string produces list of tokens.
2. After tokenization sentence was presented as the matrix of embedding vectors (you can specify `embedder` or randomly initialized embeddings will be used). Not important vectors of this matrix will be masked (important tokens selected with Token Importance + Subset Sampler blocks), and the other use for model (Distil model), that tries to imitate the original automl model (learns to predict the same outputs).
3. Scheme of L2X:



4. Some info about parameters:

- `n_important` - number of important tokens;
- `temperature` - initial temperature used in gumbel softmax trick;
- `train_device` - device used for training;
- `inference_device` - device used for inference;
- `verbose` - verbose mode;
- `binning_mode` - for training we use batch sampling by the length of sequence. So, batch formed only by the sequences from the respect bin. This parameter used for method of automatic binning border choosing. There are two of them: `'linear'` (min-max binning, like linspace), `'hist'` (histogram binning).
- `bins_number` - number of bins in batch sampling process;
- `n_epochs` - number of epochs of training of the L2X;
- `learning_rate` - learning rate of L2X model;
- `patience` - number of epoches before learning rate decreasing (`torch.optim.lr_scheduler.ReduceLROnPlateu`);
- `extreme_patience` - number of epoches before early stopping by the validation dataset;
- `train_batch_size` - size of batch for training process;
- `valid_batch_size` - size of batch for validation process;
- `temp_anneal_factor` - annealing factor for temperature. The temperature will be multiplied by this coefficient every epoch.
- `importance_sampler` - specifies method of sampling importance (there are two of them `'gumbeltopk'` - method from the original paper, `'softsub'` - another method);
- `max_vocab_length` - maximum length of vocabular (vocabular build up from `max_vocab_length` the most frequent tokens). If `max_vocab_length` is `-1` then include all in train set.
- `embedder` - embedding dictionary or path to fasttext/dict of embeddings.

5. Some links for more info about L2X:
    1. Gumbel Softmax Trick
    2. L2X Original Paper
    3. SoftSub Paper

[30]:
```
l2x = L2XTextExplainer(automl, train_device='cuda:1',
                       inference_device='cuda:1',
                       embedding_dim=300,
                       gamma=0.1, temperature=2, temp_anneal_factor=0.95,
                       n_epochs=200, importance_sampler='gumbeltopk',
                       n_important=20, patience=25,
                       extreme_patience=30, trainable_embeds=True)

l2x.fit(train_data, valid_data, cols_to_explain='Review')
```

```
test: 100%|| 2500/2500 [06:14<00:00,  6.67it/s]
test: 100%|| 2500/2500 [06:15<00:00,  6.66it/s]
test: 100%|| 313/313 [00:47<00:00,  6.66it/s]
test: 100%|| 313/313 [00:46<00:00,  6.66it/s]
train nll (loss=7.8830): 100%|| 1249/1251 [00:41<00:00, 30.12it/s]
train nll (loss=1.4016): 100%|| 1249/1251 [00:40<00:00, 30.63it/s]
```

(continues on next page)

(continued from previous page)

```
train nll (loss=1.3859): 100%|| 1249/1251 [00:40<00:00, 30.53it/s]
train nll (loss=1.3684): 100%|| 1249/1251 [00:40<00:00, 30.57it/s]
train nll (loss=1.0265): 100%|| 1249/1251 [00:41<00:00, 30.44it/s]
train nll (loss=0.7086): 100%|| 1249/1251 [00:41<00:00, 30.46it/s]
train nll (loss=0.6344): 100%|| 1249/1251 [00:40<00:00, 30.62it/s]
train nll (loss=0.5779): 100%|| 1249/1251 [00:40<00:00, 30.68it/s]
train nll (loss=0.5318): 100%|| 1249/1251 [00:41<00:00, 30.22it/s]
train nll (loss=0.4962): 100%|| 1249/1251 [00:40<00:00, 30.56it/s]
train nll (loss=0.4575): 100%|| 1249/1251 [00:40<00:00, 30.69it/s]
train nll (loss=0.4233): 100%|| 1249/1251 [00:40<00:00, 30.65it/s]
train nll (loss=0.3882): 100%|| 1249/1251 [00:40<00:00, 30.58it/s]
train nll (loss=0.3574): 100%|| 1249/1251 [00:40<00:00, 30.59it/s]
train nll (loss=0.3326): 100%|| 1249/1251 [00:41<00:00, 30.41it/s]
train nll (loss=0.3177): 100%|| 1249/1251 [00:40<00:00, 30.70it/s]
train nll (loss=0.2997): 100%|| 1249/1251 [00:40<00:00, 30.50it/s]
train nll (loss=0.2885): 100%|| 1249/1251 [00:40<00:00, 30.50it/s]
train nll (loss=0.2768): 100%|| 1249/1251 [00:41<00:00, 30.24it/s]
train nll (loss=0.2667): 100%|| 1249/1251 [00:41<00:00, 30.41it/s]
train nll (loss=0.2569): 100%|| 1249/1251 [00:42<00:00, 29.40it/s]
train nll (loss=0.2500): 100%|| 1249/1251 [00:42<00:00, 29.54it/s]
train nll (loss=0.2439): 100%|| 1249/1251 [00:41<00:00, 30.26it/s]
train nll (loss=0.2349): 100%|| 1249/1251 [00:40<00:00, 30.54it/s]
train nll (loss=0.2278): 100%|| 1249/1251 [00:41<00:00, 30.25it/s]
train nll (loss=0.2244): 100%|| 1249/1251 [00:41<00:00, 30.35it/s]
train nll (loss=0.2220): 100%|| 1249/1251 [00:41<00:00, 30.35it/s]
train nll (loss=0.2158): 100%|| 1249/1251 [00:41<00:00, 30.36it/s]
train nll (loss=0.2110): 100%|| 1249/1251 [00:41<00:00, 30.45it/s]
train nll (loss=0.2080): 100%|| 1249/1251 [00:41<00:00, 30.43it/s]
train nll (loss=0.2050): 100%|| 1249/1251 [00:41<00:00, 30.40it/s]
train nll (loss=0.2003): 100%|| 1249/1251 [00:41<00:00, 30.38it/s]
train nll (loss=0.1977): 100%|| 1249/1251 [00:41<00:00, 30.43it/s]
train nll (loss=0.1925): 100%|| 1249/1251 [00:41<00:00, 30.42it/s]
train nll (loss=0.1919): 100%|| 1249/1251 [00:40<00:00, 30.55it/s]
train nll (loss=0.1888): 100%|| 1249/1251 [00:40<00:00, 30.49it/s]
train nll (loss=0.1842): 100%|| 1249/1251 [00:41<00:00, 30.37it/s]
train nll (loss=0.1841): 100%|| 1249/1251 [00:41<00:00, 30.44it/s]
train nll (loss=0.1820): 100%|| 1249/1251 [00:40<00:00, 30.52it/s]
train nll (loss=0.1777): 100%|| 1249/1251 [00:41<00:00, 30.37it/s]
train nll (loss=0.1785): 100%|| 1249/1251 [00:41<00:00, 30.27it/s]
train nll (loss=0.1778): 100%|| 1249/1251 [00:41<00:00, 30.40it/s]
train nll (loss=0.1748): 100%|| 1249/1251 [00:41<00:00, 30.40it/s]
train nll (loss=0.1719): 100%|| 1249/1251 [00:41<00:00, 30.42it/s]
train nll (loss=0.1704): 100%|| 1249/1251 [00:40<00:00, 30.58it/s]
train nll (loss=0.1715): 100%|| 1249/1251 [00:40<00:00, 30.74it/s]
train nll (loss=0.1715): 100%|| 1249/1251 [00:40<00:00, 30.48it/s]
train nll (loss=0.1734): 100%|| 1249/1251 [00:40<00:00, 30.68it/s]
train nll (loss=0.1732): 100%|| 1249/1251 [00:40<00:00, 30.65it/s]
train nll (loss=0.1781): 100%|| 1249/1251 [00:40<00:00, 30.58it/s]
train nll (loss=0.1770): 100%|| 1249/1251 [00:40<00:00, 30.57it/s]
train nll (loss=0.1737): 100%|| 1249/1251 [00:40<00:00, 30.66it/s]
train nll (loss=0.1728): 100%|| 1249/1251 [00:40<00:00, 30.59it/s]
train nll (loss=0.1731): 100%|| 1249/1251 [00:42<00:00, 29.48it/s]
```

(continues on next page)

(continued from previous page)

```
train nll (loss=0.1708): 100%|| 1249/1251 [00:41<00:00, 30.26it/s]
train nll (loss=0.1696): 100%|| 1249/1251 [00:41<00:00, 30.35it/s]
train nll (loss=0.1699): 100%|| 1249/1251 [00:41<00:00, 30.01it/s]
train nll (loss=0.1699): 100%|| 1249/1251 [00:42<00:00, 29.39it/s]
train nll (loss=0.1681): 100%|| 1249/1251 [00:42<00:00, 29.37it/s]
train nll (loss=0.1682): 100%|| 1249/1251 [00:45<00:00, 27.39it/s]
train nll (loss=0.1684): 100%|| 1249/1251 [00:43<00:00, 28.57it/s]
train nll (loss=0.1666): 100%|| 1249/1251 [00:46<00:00, 26.91it/s]
train nll (loss=0.1659): 100%|| 1249/1251 [00:51<00:00, 24.20it/s]
train nll (loss=0.1656): 100%|| 1249/1251 [00:46<00:00, 27.09it/s]
train nll (loss=0.1665): 100%|| 1249/1251 [00:43<00:00, 28.92it/s]
train nll (loss=0.1676): 100%|| 1249/1251 [00:43<00:00, 28.65it/s]
train nll (loss=0.1657): 100%|| 1249/1251 [00:43<00:00, 29.02it/s]
train nll (loss=0.1651): 100%|| 1249/1251 [00:43<00:00, 28.92it/s]
train nll (loss=0.1631): 100%|| 1249/1251 [00:43<00:00, 28.87it/s]
train nll (loss=0.1634): 100%|| 1249/1251 [00:43<00:00, 28.79it/s]
train nll (loss=0.1634): 100%|| 1249/1251 [00:43<00:00, 28.77it/s]
train nll (loss=0.1626): 100%|| 1249/1251 [00:43<00:00, 28.40it/s]
train nll (loss=0.1631): 100%|| 1249/1251 [00:42<00:00, 29.35it/s]
train nll (loss=0.1613): 100%|| 1249/1251 [00:41<00:00, 30.44it/s]
train nll (loss=0.1614): 100%|| 1249/1251 [00:40<00:00, 30.63it/s]
train nll (loss=0.1638): 100%|| 1249/1251 [00:40<00:00, 30.51it/s]
train nll (loss=0.1633): 100%|| 1249/1251 [00:40<00:00, 30.52it/s]
train nll (loss=0.1618): 100%|| 1249/1251 [00:41<00:00, 30.39it/s]
train nll (loss=0.1612): 100%|| 1249/1251 [00:41<00:00, 30.43it/s]
train nll (loss=0.1628): 100%|| 1249/1251 [00:41<00:00, 30.34it/s]
train nll (loss=0.1616): 100%|| 1249/1251 [00:41<00:00, 29.99it/s]
train nll (loss=0.1618): 100%|| 1249/1251 [00:41<00:00, 29.77it/s]
train nll (loss=0.1594): 100%|| 1249/1251 [00:42<00:00, 29.29it/s]
train nll (loss=0.1617): 100%|| 1249/1251 [00:42<00:00, 29.55it/s]
train nll (loss=0.1617): 100%|| 1249/1251 [00:42<00:00, 29.27it/s]
train nll (loss=0.1610): 100%|| 1249/1251 [00:41<00:00, 30.35it/s]
train nll (loss=0.1590): 100%|| 1249/1251 [00:41<00:00, 30.35it/s]
train nll (loss=0.1602): 100%|| 1249/1251 [00:43<00:00, 28.49it/s]
train nll (loss=0.1602): 100%|| 1249/1251 [00:42<00:00, 29.07it/s]
train nll (loss=0.1613): 100%|| 1249/1251 [00:43<00:00, 28.80it/s]
train nll (loss=0.1620): 100%|| 1249/1251 [00:43<00:00, 28.86it/s]
train nll (loss=0.1593): 100%|| 1249/1251 [00:43<00:00, 28.95it/s]
train nll (loss=0.1612): 100%|| 1249/1251 [00:43<00:00, 28.89it/s]
train nll (loss=0.1620): 100%|| 1249/1251 [00:43<00:00, 28.93it/s]
train nll (loss=0.1614): 100%|| 1249/1251 [00:44<00:00, 28.26it/s]
train nll (loss=0.1630): 100%|| 1249/1251 [00:42<00:00, 29.19it/s]
train nll (loss=0.1665): 100%|| 1249/1251 [00:43<00:00, 28.75it/s]
train nll (loss=0.1605): 100%|| 1249/1251 [00:42<00:00, 29.48it/s]
train nll (loss=0.1605): 100%|| 1249/1251 [00:41<00:00, 30.01it/s]
train nll (loss=0.1636): 100%|| 1249/1251 [00:40<00:00, 30.58it/s]
train nll (loss=0.1617): 100%|| 1249/1251 [00:41<00:00, 30.08it/s]
train nll (loss=0.1635): 100%|| 1249/1251 [00:41<00:00, 30.44it/s]
train nll (loss=0.1606): 100%|| 1249/1251 [00:40<00:00, 30.65it/s]
train nll (loss=0.1631): 100%|| 1249/1251 [00:41<00:00, 30.45it/s]
train nll (loss=0.1645): 100%|| 1249/1251 [00:40<00:00, 30.60it/s]
train nll (loss=0.1652): 100%|| 1249/1251 [00:41<00:00, 30.37it/s]
```

(continues on next page)

(continued from previous page)

```
train nll (loss=0.1641): 100%|| 1249/1251 [00:41<00:00, 29.96it/s]
train nll (loss=0.1669): 100%|| 1249/1251 [00:43<00:00, 28.80it/s]
train nll (loss=0.1610): 100%|| 1249/1251 [00:42<00:00, 29.11it/s]
train nll (loss=0.1630): 100%|| 1249/1251 [00:43<00:00, 28.96it/s]
train nll (loss=0.1644): 100%|| 1249/1251 [00:42<00:00, 29.11it/s]
train nll (loss=0.1681): 100%|| 1249/1251 [00:42<00:00, 29.06it/s]
train nll (loss=0.1691): 100%|| 1249/1251 [00:42<00:00, 29.12it/s]
train nll (loss=0.1728): 100%|| 1249/1251 [00:42<00:00, 29.15it/s]
train nll (loss=0.1710): 100%|| 1249/1251 [00:42<00:00, 29.16it/s]
train nll (loss=0.1679): 100%|| 1249/1251 [00:42<00:00, 29.30it/s]
train nll (loss=0.1697): 100%|| 1249/1251 [00:42<00:00, 29.31it/s]
train nll (loss=0.1669): 100%|| 1249/1251 [00:41<00:00, 29.92it/s]
train nll (loss=0.1708): 100%|| 1249/1251 [00:41<00:00, 30.28it/s]
train nll (loss=0.1662): 100%|| 1249/1251 [00:40<00:00, 30.56it/s]
train nll (loss=0.1739): 100%|| 1249/1251 [00:41<00:00, 30.34it/s]
train nll (loss=0.1977): 100%|| 1249/1251 [00:40<00:00, 30.68it/s]
train nll (loss=0.1844): 100%|| 1249/1251 [00:40<00:00, 30.72it/s]
train nll (loss=0.1685): 100%|| 1249/1251 [00:41<00:00, 30.17it/s]
train nll (loss=0.1664): 100%|| 1249/1251 [00:40<00:00, 30.48it/s]
train nll (loss=0.1760): 100%|| 1249/1251 [00:40<00:00, 30.63it/s]
train nll (loss=0.1678): 100%|| 1249/1251 [00:40<00:00, 30.71it/s]
train nll (loss=0.1681): 100%|| 1249/1251 [00:41<00:00, 30.25it/s]
train nll (loss=0.1806): 100%|| 1249/1251 [00:40<00:00, 30.67it/s]
train nll (loss=0.1724): 100%|| 1249/1251 [00:40<00:00, 30.70it/s]
train nll (loss=0.1691): 100%|| 1249/1251 [00:40<00:00, 30.67it/s]
train nll (loss=0.1712): 100%|| 1249/1251 [00:40<00:00, 30.71it/s]
train nll (loss=0.1762): 100%|| 1249/1251 [00:40<00:00, 30.55it/s]
train nll (loss=0.1655): 100%|| 1249/1251 [00:41<00:00, 30.00it/s]
train nll (loss=0.1860): 100%|| 1249/1251 [00:42<00:00, 29.72it/s]
train nll (loss=0.1897): 100%|| 1249/1251 [00:41<00:00, 29.76it/s]
train nll (loss=0.1770): 100%|| 1249/1251 [00:41<00:00, 30.45it/s]
train nll (loss=0.1796): 100%|| 1249/1251 [00:42<00:00, 29.65it/s]
train nll (loss=0.1816): 100%|| 1249/1251 [00:40<00:00, 30.48it/s]
train nll (loss=0.1870): 100%|| 1249/1251 [00:40<00:00, 30.75it/s]
train nll (loss=0.1797): 100%|| 1249/1251 [00:40<00:00, 30.57it/s]
train nll (loss=0.1878): 100%|| 1249/1251 [00:40<00:00, 30.75it/s]
train nll (loss=0.1903): 100%|| 1249/1251 [00:40<00:00, 30.72it/s]
train nll (loss=0.1787): 100%|| 1249/1251 [00:40<00:00, 30.63it/s]
train nll (loss=0.1815): 100%|| 1249/1251 [00:40<00:00, 30.70it/s]
train nll (loss=0.1781): 100%|| 1249/1251 [00:40<00:00, 30.68it/s]
train nll (loss=0.1749): 100%|| 1249/1251 [00:40<00:00, 30.68it/s]
train nll (loss=0.1735): 100%|| 1249/1251 [00:40<00:00, 30.51it/s]
train nll (loss=0.1675): 100%|| 1249/1251 [00:40<00:00, 30.56it/s]
train nll (loss=0.1671): 100%|| 1249/1251 [00:40<00:00, 30.58it/s]
```

[31]:
```
expl_train = l2x['Review'].explain_instances(train_data)
```

[32]:
```
expl_valid = l2x['Review'].explain_instances(valid_data)
```

Examples

[33]:
```
expl_valid[66].visualize_in_notebook()
```

[34]:
```
expl_valid[55].visualize_in_notebook()
```

[35]:
```
expl_valid[77].visualize_in_notebook()
```

[36]:
```
expl_valid[88].visualize_in_notebook()
```

[37]:
```
expl_valid[121].visualize_in_notebook()
```

[38]:
```
expl_valid[888].visualize_in_notebook()
```

[39]:
```
expl_valid[999].visualize_in_notebook()
```

[40]:
```
expl_valid[333].visualize_in_notebook()
```

[41]:
```
expl_valid[111].visualize_in_notebook()
```

[42]:
```
expl_valid[100].visualize_in_notebook()
```

[43]:
```
expl_valid[1021].visualize_in_notebook()
```

[44]:
```
expl_valid[9999].visualize_in_notebook()
```

[45]:
```
expl_valid[7676].visualize_in_notebook()
```

[46]:
```
expl_valid[6767].visualize_in_notebook()
```

[47]:
```
expl_valid[3131].visualize_in_notebook()
```

2.1.5 Tutorial 5: Uplift modeling

Official LightAutoML github repository is here

[ ]:
```
%load_ext autoreload
%autoreload 2
```

Install LightAutoML

Uncomment if doesn't clone repository by git. (ex.: colab, kaggle version)

[ ]:
```
#! pip install -U lightautoml
```

Import necessary libraries

[ ]:
```
# Standard python libraries
from copy import deepcopy
import os
import requests

# Installed libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.metrics import roc_auc_score
from sklearn.model_selection import train_test_split
import torch

# Imports from our package
from lightautoml.automl.presets.tabular_presets import TabularAutoML
from lightautoml.dataset.roles import DatetimeRole
from lightautoml.tasks import Task

from lightautoml.addons.uplift.base import AutoUplift, BaseLearnerWrapper,␣
↪MetaLearnerWrapper
from lightautoml.addons.uplift import metalearners
from lightautoml.addons.uplift.metrics import (_available_uplift_modes,
                                               TUpliftMetric,
                                               calculate_graphic_uplift_curve,
                                               calculate_min_max_uplift_auc,
                                               calculate_uplift_at_top,
                                               calculate_uplift_auc,
                                               perfect_uplift_curve)
from lightautoml.addons.uplift.utils import create_linear_automl
from lightautoml.report.report_deco import ReportDecoUplift


%matplotlib inline
```

Parameters

Setting

[ ]:
```
N_THREADS = 8 # threads cnt for lgbm and linear models
N_FOLDS = 5 # folds cnt for AutoML
RANDOM_STATE = 42 # fixed random state for various reasons
```

(continues on next page)

(continued from previous page)

```
TEST_SIZE = 0.2 # Test size for metric check
TIMEOUT = 300 # Time in seconds for automl run
TARGET_NAME = 'TARGET' # Target column name
TREATMENT_NAME = 'CODE_GENDER'
```

Fix torch number of threads and numpy seed

[ ]:
```
np.random.seed(RANDOM_STATE)
torch.set_num_threads(N_THREADS)
```

Example data load

Load a dataset from the repository if doesn't clone repository by git.

[ ]:
```
DATASET_DIR = '../data/'
DATASET_NAME = 'sampled_app_train.csv'
DATASET_FULLNAME = os.path.join(DATASET_DIR, DATASET_NAME)
DATASET_URL = 'https://raw.githubusercontent.com/AILab-MLTools/LightAutoML/master/
↪example_data/test_data_files/sampled_app_train.csv'
```

[ ]:
```
%%time

if not os.path.exists(DATASET_FULLNAME):
    os.makedirs(DATASET_DIR, exist_ok=True)

    dataset = requests.get(DATASET_URL).text
    with open(DATASET_FULLNAME, 'w') as output:
        output.write(dataset)
```

[ ]:
```
%%time

data = pd.read_csv(DATASET_FULLNAME)
data.head()
```

(Optional) Some user feature preparation

[ ]:
```
%%time

data['BIRTH_DATE'] = (np.datetime64('2018-01-01') + data['DAYS_BIRTH'].astype(np.dtype(
↪'timedelta64[D]'))).astype(str)
data['EMP_DATE'] = (np.datetime64('2018-01-01') + np.clip(data['DAYS_EMPLOYED'], None,␣
↪0).astype(np.dtype('timedelta64[D]'))
                    ).astype(str)
data['report_dt'] = np.datetime64('2018-01-01')
data['constant'] = 1
data['allnan'] = np.nan
data.drop(['DAYS_BIRTH', 'DAYS_EMPLOYED'], axis=1, inplace=True)
data['CODE_GENDER'] = (data['CODE_GENDER'] == 'M').astype(int)
```

Data splitting for train-test

[ ]:
```
%%time


stratify_value = data[TARGET_NAME] + 10 * data[TREATMENT_NAME]
train, test = train_test_split(data, test_size=3000, stratify=stratify_value, random_
↪state=42)
test_target, test_treatment = test[TARGET_NAME].values.ravel(), test[TREATMENT_NAME].
↪values.ravel()
```

Setup columns roles

[ ]:
```
%%time

roles = {
    'target': TARGET_NAME,
    'treatment': TREATMENT_NAME,
    DatetimeRole(base_date=True, seasonality=(), base_feats=False): 'report_dt'
}
```

AutoUplift (use predefined uplift methods)

Fit autouplift

[ ]:
```
%%time

task = Task('binary')

autouplift = AutoUplift(task,
                        metric='adj_qini',
                        has_report=True,
                        test_size=0.2,
                        timeout=200,
                        # timeout_metalearner=5
)

autouplift.fit(train, roles, verbose=1)
```

Show rating of uplift methods (meta-learners)

[ ]:
```
%%time

rating_table = autouplift.get_metalearners_rating()
rating_table
```

Get best metalearner

[ ]:
```
%%time

best_metalearner = autouplift.create_best_metalearner(
```

(continues on next page)

(continued from previous page)

```
    update_metalearner_params={'timeout': None},
    update_baselearner_params={'timeout': 30}
)
best_metalearner.fit(train, roles)
_ = best_metalearner.predict(test);
```

Predict to test data and check metrics

[ ]:
```
%%time

uplift_pred, treatment_pred, control_pred = best_metalearner.predict(test)
uplift_pred = uplift_pred.ravel()

roc_auc_treatment = roc_auc_score(test_target[test_treatment == 1], treatment_pred[test_
↪treatment == 1])
roc_auc_control = roc_auc_score(test_target[test_treatment == 0], control_pred[test_
↪treatment == 0])

uplift_auc_algo = calculate_uplift_auc(test_target, uplift_pred, test_treatment,␣
↪normed=False)
uplift_auc_algo_normed = calculate_uplift_auc(test_target, uplift_pred, test_treatment,␣
↪normed=True)
auc_base, auc_perfect = calculate_min_max_uplift_auc(test_target, test_treatment)

print('--- Check scores ---')
print('OOF scores "ROC_AUC":')
print('\tTreatment = {:.5f}'.format(roc_auc_treatment))
print('\tControl   = {:.5f}'.format(roc_auc_control))
print('Uplift score of test group (default="adj_qini"):')
print('\tBaseline      = {:.5f}'.format(auc_base))
print('\tAlgo (Normed) = {:.5f} ({:.5f})'.format(uplift_auc_algo, uplift_auc_algo_
↪normed))
print('\tPerfect       = {:.5f}'.format(auc_perfect))
```

AutoUplift (custom uplift methods)

Fit autouplift

[ ]:
```
%%time

# Set uplift candidate for choosing best of them
# !!!ATTENTION!!!
#    This is a demonstration of the possibilities,
#    You may use default set of candidates

task = Task('binary')

uplift_candidates = [
    MetaLearnerWrapper(
        name='TLearner__Default',
        klass=metalearners.TLearner,
```

(continues on next page)

(continued from previous page)

```
        params={'base_task': task}
    ),
    MetaLearnerWrapper(
        name='TLearner__Custom',
        klass=metalearners.TLearner,
        params={
            'treatment_learner': BaseLearnerWrapper(
                name='__TabularAutoML__',
                klass=TabularAutoML,
                params={'task': task, 'timeout': 10}),
            'control_learner': BaseLearnerWrapper(
                name='__Linear__',
                klass=create_linear_automl,
                params={'task': Task('binary')})
        }
    ),
    MetaLearnerWrapper(
        name='XLearner__Custom',
        klass=metalearners.XLearner,
        params={
            'outcome_learners': [
                TabularAutoML(task=task, timeout=10), # [sec] , Only speed up example,␣
↪don't change it!
                create_linear_automl(task=Task('binary'))
            ],
            'effect_learners': [BaseLearnerWrapper(
                name='__TabularAutoML__',
                klass=TabularAutoML,
                params={'task': Task('reg'), 'timeout': 5})],
            'propensity_learner': create_linear_automl(task=Task('binary')),
        }
    )
]

autouplift = AutoUplift(task,
                        uplift_candidates=uplift_candidates,
                        metric='adj_qini',
                        test_size=0.2,
                        threshold_imbalance_treatment=0.0,    # Doesn't affect, see␣
↪warnings
                        timeout=600)                          # Doesn't affect, see␣
↪warnings

autouplift.fit(train, roles, verbose=1)
```

Show rating of uplift methods (meta-learners)

[ ]:
```
%%time

rating_table = autouplift.get_metalearners_rating()
rating_table
```

Predict to test data and check metrics

[ ]:
```
%%time

uplift_pred, treatment_pred, control_pred = autouplift.predict(test)
uplift_pred = uplift_pred.ravel()

roc_auc_treatment = roc_auc_score(test_target[test_treatment == 1], treatment_pred[test_
↪treatment == 1])
roc_auc_control = roc_auc_score(test_target[test_treatment == 0], control_pred[test_
↪treatment == 0])

uplift_auc_algo = calculate_uplift_auc(test_target, uplift_pred, test_treatment,␣
↪normed=False)
uplift_auc_algo_normed = calculate_uplift_auc(test_target, uplift_pred, test_treatment,␣
↪normed=True)
auc_base, auc_perfect = calculate_min_max_uplift_auc(test_target, test_treatment)

print('--- Check scores ---')
print('OOF scores "ROC_AUC":')
print('\tTreatment = {:.5f}'.format(roc_auc_treatment))
print('\tControl   = {:.5f}'.format(roc_auc_control))
print('Uplift score of test group (default="adj_qini"):')
print('\tBaseline      = {:.5f}'.format(auc_base))
print('\tAlgo (Normed) = {:.5f} ({:.5f})'.format(uplift_auc_algo, uplift_auc_algo_
↪normed))
print('\tPerfect       = {:.5f}'.format(auc_perfect))
```

AutoUplift with custom metric

Fit autouplift

[ ]:
```
%%time

# Using a custom metric
# How to determine custom metric, see below

task = Task('binary')


class CustomUpliftMetric(TUpliftMetric):
    def __call__(self, target: np.ndarray, uplift_pred: np.ndarray, treatment: np.
↪ndarray) -> float:
        up_10 = calculate_uplift_at_top(target, uplift_pred, treatment, 10)
        up_20 = calculate_uplift_at_top(target, uplift_pred, treatment, 20)

        return 0.5 * (up_10 + up_20)

autouplift = AutoUplift(task,
                        add_dd_candidates=True,
                        metric=CustomUpliftMetric(),
                        test_size=0.2,
                        threshold_imbalance_treatment=0.0,
```

(continues on next page)

(continued from previous page)

```
                        cpu_limit=10,
                        timeout=100)

autouplift.fit(train, roles)
```

Show rating of uplift methods (meta-learners)

[ ]:
```
%%time

rating_table = autouplift.get_metalearners_rating()
rating_table
```

MetaLearner

TLearner

Fit on train data

[ ]:
```
%%time

# Default setting
tlearner = metalearners.TLearner(base_task=Task('binary'), cpu_limit=5)
tlearner.fit(train, roles)
```

Predict to test data and check metrics

[ ]:
```
%%time

uplift_pred, treatment_pred, control_pred = tlearner.predict(test)
uplift_pred = uplift_pred.ravel()

roc_auc_treatment = roc_auc_score(test_target[test_treatment == 1], treatment_pred[test_
↪treatment == 1])
roc_auc_control = roc_auc_score(test_target[test_treatment == 0], control_pred[test_
↪treatment == 0])

uplift_auc_algo = calculate_uplift_auc(test_target, uplift_pred, test_treatment,␣
↪normed=False)
uplift_auc_algo_normed = calculate_uplift_auc(test_target, uplift_pred, test_treatment,␣
↪normed=True)
auc_base, auc_perfect = calculate_min_max_uplift_auc(test_target, test_treatment)

print('--- Check scores ---')
print('OOF scores "ROC_AUC":')
print('\tTreatment = {:.5f}'.format(roc_auc_treatment))
print('\tControl   = {:.5f}'.format(roc_auc_control))
print('Uplift score of test group (default="adj_qini"):')
print('\tBaseline      = {:.5f}'.format(auc_base))
print('\tAlgo (Normed) = {:.5f} ({:.5f})'.format(uplift_auc_algo, uplift_auc_algo_
↪normed))
print('\tPerfect       = {:.5f}'.format(auc_perfect))
```

XLearner

Fit on train data

[ ]:
```
%%time

# Custom base algorithm
xlearner = metalearners.XLearner(
    propensity_learner=TabularAutoML(task=Task('binary'), timeout=10),
    outcome_learners=[
        TabularAutoML(task=Task('binary'), timeout=10),
        TabularAutoML(task=Task('binary'), timeout=10)
    ],
    effect_learners=[
        TabularAutoML(task=Task('reg'), timeout=10),
        TabularAutoML(task=Task('reg'), timeout=10)
    ]
)
xlearner.fit(train, roles)
```

Predict to test data and check metrics

[ ]:
```
%%time

uplift_pred, treatment_pred, control_pred = xlearner.predict(test)
uplift_pred = uplift_pred.ravel()

roc_auc_treatment = roc_auc_score(test_target[test_treatment == 1], treatment_pred[test_
↪treatment == 1])
roc_auc_control = roc_auc_score(test_target[test_treatment == 0], control_pred[test_
↪treatment == 0])

uplift_auc_algo = calculate_uplift_auc(test_target, uplift_pred, test_treatment,␣
↪normed=False)
uplift_auc_algo_normed = calculate_uplift_auc(test_target, uplift_pred, test_treatment,␣
↪normed=True)
auc_base, auc_perfect = calculate_min_max_uplift_auc(test_target, test_treatment)

print('--- Check scores ---')
print('OOF scores "ROC_AUC":')
print('\tTreatment = {:.5f}'.format(roc_auc_treatment))
print('\tControl   = {:.5f}'.format(roc_auc_control))
print('Uplift score of test group (default="adj_qini"):')
print('\tBaseline      = {:.5f}'.format(auc_base))
print('\tAlgo (Normed) = {:.5f} ({:.5f})'.format(uplift_auc_algo, uplift_auc_algo_
↪normed))
print('\tPerfect       = {:.5f}'.format(auc_perfect))
```

Uplift metrics and graphics (using xlearner predictions)

[ ]:
```
%%time

UPLIFT_METRIC = 'adj_qini'

print("All available uplift metrics: {}".format(_available_uplift_modes))
```

Algorithm uplift curve

[ ]:
```
%%time

# Algorithm curve
xs_xlearner, ys_xlearner = calculate_graphic_uplift_curve(
    test_target, uplift_pred, test_treatment, mode=UPLIFT_METRIC
)
```

Baseline, perfect curve

[ ]:
```
# Baseline curve
xs_base, ys_base = xs_xlearner, xs_xlearner * ys_xlearner[-1]

# Perfect curver
perfect_uplift = perfect_uplift_curve(test_target, test_treatment)
xs_perfect, ys_perfect = calculate_graphic_uplift_curve(
    test_target, perfect_uplift, test_treatment, mode=UPLIFT_METRIC)
```

[ ]:
```
plt.figure(figsize=(10, 7))

plt.plot(xs_base, ys_base, 'black')
plt.plot(xs_xlearner, ys_xlearner, 'red')
plt.plot(xs_perfect, ys_perfect, 'green')

plt.fill_between(xs_xlearner, ys_base, ys_xlearner, alpha=0.5, color='orange')

plt.xlabel('Cumulative percentage of people in T/C groups')
plt.ylabel('Uplift metric (%s)'.format(UPLIFT_METRIC))
plt.grid()
plt.legend(['Baseline', 'XLearner', 'Perfect']);
```

Uplift TOP-K

[ ]:
```
tops = np.arange(5, 101, 5)

uplift_at_tops = []
for top in tops:
    uat = calculate_uplift_at_top(test_target, uplift_pred, test_treatment, top=top)
    uplift_at_tops.append(uat)

plt.figure(figsize=(10, 7))
```

(continues on next page)

(continued from previous page)

```
plt.plot(tops, uplift_at_tops, marker='.')

plt.legend(['Uplift_At_K'])
plt.xticks(np.arange(0, 101, 10))
plt.grid()
```

Custom metric

[ ]:
```
# Custom metric can be used in AutoUplift
# There msut be a function's signature:
# def custom_metric(target, uplift_pred, treatment) -> float:


class CustomUpliftMetric(TUpliftMetric):
    def __call__(self, target: np.ndarray, uplift_pred: np.ndarray, treatment: np.
↪ndarray) -> float:
        up_10 = calculate_uplift_at_top(target, uplift_pred, treatment, 10)
        up_20 = calculate_uplift_at_top(target, uplift_pred, treatment, 20)

        return 0.5 * (up_10 + up_20)


metric = CustomUpliftMetric()
metric_value = metric(test_target, uplift_pred, test_treatment)

print("Metric = {}".format(metric_value))
```

Report

[ ]:
```
%%time

RDU = ReportDecoUplift()
tlearner_deco = RDU(metalearners.TLearner(base_task=Task('binary')))
tlearner_deco.fit(train, roles)
_ = tlearner_deco.predict(test)

# Path to report: PATH_TO_CURRENT_NOTEBOOK/lama_report/lama_interactive_report.html
```

2.1.6 Tutorial 6: Custom pipeline tutorial

Official LightAutoML github repository is here

Preparing

Step 1. Install LightAutoML

Uncomment if doesn't clone repository by git. (ex.: colab, kaggle version)

[1]:
```
#! pip install -U lightautoml
```

Step 2. Import necessary libraries

[2]:
```
# Standard python libraries
import os
import time
import requests


# Installed libraries
import numpy as np
import pandas as pd
from sklearn.metrics import roc_auc_score
from sklearn.model_selection import train_test_split
import torch

# Imports from our package
from lightautoml.automl.base import AutoML
from lightautoml.ml_algo.boost_lgbm import BoostLGBM
from lightautoml.ml_algo.tuning.optuna import OptunaTuner
from lightautoml.pipelines.features.lgb_pipeline import LGBSimpleFeatures
from lightautoml.pipelines.ml.base import MLPipeline
from lightautoml.pipelines.selection.importance_based import ImportanceCutoffSelector,␣
↪ModelBasedImportanceEstimator
from lightautoml.reader.base import PandasToPandasReader
from lightautoml.tasks import Task
from lightautoml.automl.blend import WeightedBlender
```

Step 3. Parameters

[3]:
```
N_THREADS = 8 # threads cnt for lgbm and linear models
N_FOLDS = 5 # folds cnt for AutoML
RANDOM_STATE = 42 # fixed random state for various reasons
TEST_SIZE = 0.2 # Test size for metric check
TARGET_NAME = 'TARGET' # Target column name
```

Step 4. Fix torch number of threads and numpy seed

[4]:
```
np.random.seed(RANDOM_STATE)
torch.set_num_threads(N_THREADS)
```

Step 5. Example data load

Load a dataset from the repository if doesn't clone repository by git.

[5]:
```
DATASET_DIR = '../data/'
DATASET_NAME = 'sampled_app_train.csv'
DATASET_FULLNAME = os.path.join(DATASET_DIR, DATASET_NAME)
DATASET_URL = 'https://raw.githubusercontent.com/AILab-MLTools/LightAutoML/master/
↪examples/data/sampled_app_train.csv'
```

[6]:
```
%%time

if not os.path.exists(DATASET_FULLNAME):
    os.makedirs(DATASET_DIR, exist_ok=True)

    dataset = requests.get(DATASET_URL).text
    with open(DATASET_FULLNAME, 'w') as output:
        output.write(dataset)
```

```
CPU times: user 28 µs, sys: 20 µs, total: 48 µs
Wall time: 64.4 µs
```

[7]:
```
%%time

data = pd.read_csv(DATASET_FULLNAME)
data.head()
```

```
CPU times: user 105 ms, sys: 14.5 ms, total: 119 ms
Wall time: 118 ms
```

[7]:

	SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	\
0	313802	0	Cash loans	M	N	
1	319656	0	Cash loans	F	N	
2	207678	0	Revolving loans	F	Y	
3	381593	0	Cash loans	F	N	
4	258153	0	Cash loans	F	Y	

	FLAG_OWN_REALTY	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANNUITY	\
0	Y	0	270000.0	327024.0	15372.0	
1	N	0	108000.0	675000.0	19737.0	
2	Y	2	112500.0	270000.0	13500.0	
3	N	1	67500.0	142200.0	9630.0	
4	Y	0	337500.0	1483231.5	46570.5	

	...	FLAG_DOCUMENT_18	FLAG_DOCUMENT_19	FLAG_DOCUMENT_20	FLAG_DOCUMENT_21	\
0	...	0	0	0	0	
1	...	0	0	0	0	
2	...	0	0	0	0	
3	...	0	0	0	0	
4	...	0	0	0	0	

	AMT_REQ_CREDIT_BUREAU_HOUR	AMT_REQ_CREDIT_BUREAU_DAY	\
0	0.0	0.0	
1	0.0	0.0	
2	0.0	0.0	

(continues on next page)

(continued from previous page)

```
3                           0.0                           0.0
4                           0.0                           0.0

   AMT_REQ_CREDIT_BUREAU_WEEK  AMT_REQ_CREDIT_BUREAU_MON  \
0                         0.0                        0.0
1                         0.0                        0.0
2                         0.0                        0.0
3                         0.0                        0.0
4                         0.0                        2.0

   AMT_REQ_CREDIT_BUREAU_QRT  AMT_REQ_CREDIT_BUREAU_YEAR
0                        0.0                        1.0
1                        0.0                        0.0
2                        0.0                        1.0
3                        0.0                        4.0
4                        0.0                        0.0

[5 rows x 122 columns]
```

Step 6. (Optional) Some user feature preparation

Cell below shows some user feature preparations to create task more difficult (this block can be omitted if you don't want to change the initial data):

[8]:
```
%%time

data['BIRTH_DATE'] = (np.datetime64('2018-01-01') + data['DAYS_BIRTH'].astype(np.dtype(
↪'timedelta64[D]'))).astype(str)
data['EMP_DATE'] = (np.datetime64('2018-01-01') + np.clip(data['DAYS_EMPLOYED'], None,␣
↪0).astype(np.dtype('timedelta64[D]'))
                    ).astype(str)

data['constant'] = 1
data['allnan'] = np.nan

data['report_dt'] = np.datetime64('2018-01-01')

data.drop(['DAYS_BIRTH', 'DAYS_EMPLOYED'], axis=1, inplace=True)
```

```
CPU times: user 108 ms, sys: 4.5 ms, total: 113 ms
Wall time: 111 ms
```

Step 7. (Optional) Data splitting for train-test

Block below can be omitted if you are going to train model only or you have specific train and test files:

[9]:
```
%%time

train_data, test_data = train_test_split(data,
                                         test_size=TEST_SIZE,
                                         stratify=data[TARGET_NAME],
                                         random_state=RANDOM_STATE)
```

(continues on next page)

(continued from previous page)

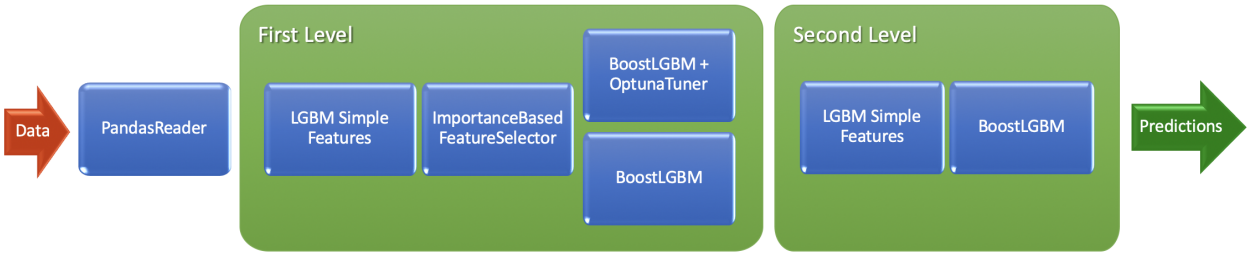
```
print('Data splitted. Parts sizes: train_data = {}, test_data = {}'
              .format(train_data.shape, test_data.shape))
```

```
Data splitted. Parts sizes: train_data = (8000, 125), test_data = (2000, 125)
CPU times: user 7.85 ms, sys: 3.89 ms, total: 11.7 ms
Wall time: 10.1 ms
```

[10]:
```
train_data.head()
```

[10]:
```
      SK_ID_CURR  TARGET NAME_CONTRACT_TYPE CODE_GENDER FLAG_OWN_CAR  \
6444      112261       0         Cash loans           F            N
3586      115058       0         Cash loans           F            N
9349      326623       0         Cash loans           F            N
7734      191976       0         Cash loans           M            Y
2174      281519       0    Revolving loans           F            N

     FLAG_OWN_REALTY  CNT_CHILDREN  AMT_INCOME_TOTAL  AMT_CREDIT  AMT_ANNUITY  \
6444               N             1           90000.0    640080.0      31261.5
3586               Y             0          180000.0    239850.0      23850.0
9349               Y             0          112500.0    337500.0      31086.0
7734               Y             1           67500.0    135000.0       9018.0
2174               Y             0           67500.0    202500.0      10125.0

      ...  AMT_REQ_CREDIT_BUREAU_DAY AMT_REQ_CREDIT_BUREAU_WEEK  \
6444  ...                        0.0                        0.0
3586  ...                        0.0                        0.0
9349  ...                        0.0                        0.0
7734  ...                        NaN                        NaN
2174  ...                        0.0                        0.0

     AMT_REQ_CREDIT_BUREAU_MON AMT_REQ_CREDIT_BUREAU_QRT  \
6444                       0.0                       1.0
3586                       0.0                       0.0
9349                       0.0                       0.0
7734                       NaN                       NaN
2174                       0.0                       0.0

     AMT_REQ_CREDIT_BUREAU_YEAR  BIRTH_DATE    EMP_DATE  constant  allnan  \
6444                        0.0  1985-06-28  2012-06-21         1     NaN
3586                        3.0  1953-12-27  2018-01-01         1     NaN
9349                        2.0  1975-06-21  2016-06-17         1     NaN
7734                        NaN  1988-04-27  2009-06-05         1     NaN
2174                        2.0  1975-06-13  1997-01-22         1     NaN

      report_dt
6444 2018-01-01
3586 2018-01-01
9349 2018-01-01
7734 2018-01-01
2174 2018-01-01

[5 rows x 125 columns]
```

AutoML creation



Step 1. Create Task and PandasReader

[11]:
```
%%time

task = Task('binary')
reader = PandasToPandasReader(task, cv=N_FOLDS, random_state=RANDOM_STATE)
```

```
CPU times: user 4.03 ms, sys: 25 µs, total: 4.05 ms
Wall time: 2.99 ms
```

Step 2. Create feature selector (if necessary)

[12]:
```
%%time

model0 = BoostLGBM(
    default_params={'learning_rate': 0.05, 'num_leaves': 64, 'seed': 42, 'num_threads':␣
↪N_THREADS}
)
pipe0 = LGBSimpleFeatures()
mbie = ModelBasedImportanceEstimator()
selector = ImportanceCutoffSelector(pipe0, model0, mbie, cutoff=0)
```

```
Copying TaskTimer may affect the parent PipelineTimer, so copy will create new unlimited␣
↪TaskTimer
CPU times: user 0 ns, sys: 1.91 ms, total: 1.91 ms
Wall time: 1.56 ms
```

Step 3.1. Create 1st level ML pipeline for AutoML

Our first level ML pipeline: - Simple features for gradient boosting built on selected features (using step 2) - 2 different models: * LightGBM with params tuning (using OptunaTuner) * LightGBM with heuristic params

[13]:
```
%%time

pipe = LGBSimpleFeatures()

params_tuner1 = OptunaTuner(n_trials=20, timeout=30) # stop after 20 iterations or after␣
↪30 seconds
model1 = BoostLGBM(
```

(continues on next page)

(continued from previous page)

```
    default_params={'learning_rate': 0.05, 'num_leaves': 128, 'seed': 1, 'num_threads':␣
↪N_THREADS}
)
model2 = BoostLGBM(
    default_params={'learning_rate': 0.025, 'num_leaves': 64, 'seed': 2, 'num_threads':␣
↪N_THREADS}
)

pipeline_lvl1 = MLPipeline([
    (model1, params_tuner1),
    model2
], pre_selection=selector, features_pipeline=pipe, post_selection=None)
```

```
CPU times: user 51 µs, sys: 37 µs, total: 88 µs
Wall time: 96.8 µs
```

Step 3.2. Create 2nd level ML pipeline for AutoML

Our second level ML pipeline: - Using simple features as well, but now it will be Out-Of-Fold (OOF) predictions of algos from 1st level - Only one LGBM model without params tuning - Without feature selection on this stage because we want to use all OOFs here

[14]:
```
%%time

pipe1 = LGBSimpleFeatures()

model = BoostLGBM(
    default_params={'learning_rate': 0.05, 'num_leaves': 64, 'max_bin': 1024, 'seed': 3,
↪'num_threads': N_THREADS},
    freeze_defaults=True
)

pipeline_lvl2 = MLPipeline([model], pre_selection=None, features_pipeline=pipe1, post_
↪selection=None)
```

```
CPU times: user 41 µs, sys: 29 µs, total: 70 µs
Wall time: 81.5 µs
```

Step 4. Create AutoML pipeline

AutoML pipeline consist of: - Reader for data preparation - First level ML pipeline (as built in step 3.1) - Second level ML pipeline (as built in step 3.2) - `Skip_conn = False` equals here "not to use initial features on the second level pipeline"

[15]:
```
%%time

automl = AutoML(reader, [
    [pipeline_lvl1],
    [pipeline_lvl2],
], skip_conn=False)
```

```
CPU times: user 35 µs, sys: 24 µs, total: 59 µs
Wall time: 73.7 µs
```

Step 5. Train AutoML on loaded data

In cell below we train AutoML with target column `TARGET` to receive fitted model and OOF predictions:

[16]:

```
%%time

oof_pred = automl.fit_predict(train_data, roles={'target': TARGET_NAME})
print('oof_pred:\n{}\nShape = {}'.format(oof_pred, oof_pred.shape))
```

```
[LightGBM] [Warning] seed is set=42, random_state=42 will be ignored. Current value:␣
↪seed=42
[LightGBM] [Warning] seed is set=1, random_state=42 will be ignored. Current value:␣
↪seed=1
[LightGBM] [Warning] seed is set=1, random_state=42 will be ignored. Current value:␣
↪seed=1
[LightGBM] [Warning] seed is set=1, random_state=42 will be ignored. Current value:␣
↪seed=1
[LightGBM] [Warning] seed is set=1, random_state=42 will be ignored. Current value:␣
↪seed=1
[LightGBM] [Warning] seed is set=1, random_state=42 will be ignored. Current value:␣
↪seed=1
[LightGBM] [Warning] seed is set=1, random_state=42 will be ignored. Current value:␣
↪seed=1
[LightGBM] [Warning] seed is set=1, random_state=42 will be ignored. Current value:␣
↪seed=1
[LightGBM] [Warning] seed is set=1, random_state=42 will be ignored. Current value:␣
↪seed=1
[LightGBM] [Warning] seed is set=1, random_state=42 will be ignored. Current value:␣
↪seed=1
[LightGBM] [Warning] seed is set=1, random_state=42 will be ignored. Current value:␣
↪seed=1
[LightGBM] [Warning] seed is set=1, random_state=42 will be ignored. Current value:␣
↪seed=1
[LightGBM] [Warning] seed is set=1, random_state=42 will be ignored. Current value:␣
↪seed=1
[LightGBM] [Warning] seed is set=1, random_state=42 will be ignored. Current value:␣
↪seed=1
[LightGBM] [Warning] seed is set=1, random_state=42 will be ignored. Current value:␣
↪seed=1
[LightGBM] [Warning] seed is set=1, random_state=42 will be ignored. Current value:␣
↪seed=1
[LightGBM] [Warning] seed is set=1, random_state=42 will be ignored. Current value:␣
↪seed=1
[LightGBM] [Warning] seed is set=1, random_state=42 will be ignored. Current value:␣
↪seed=1
[LightGBM] [Warning] seed is set=1, random_state=42 will be ignored. Current value:␣
↪seed=1
[LightGBM] [Warning] seed is set=1, random_state=42 will be ignored. Current value:␣
↪seed=1
[LightGBM] [Warning] seed is set=1, random_state=42 will be ignored. Current value:␣
↪seed=1
[LightGBM] [Warning] seed is set=1, random_state=42 will be ignored. Current value:␣
↪seed=1
[LightGBM] [Warning] seed is set=1, random_state=42 will be ignored. Current value:␣
```

(continues on next page)

(continued from previous page)

```
↪seed=1
[LightGBM] [Warning] seed is set=1, random_state=42 will be ignored. Current value:␣
↪seed=1
[LightGBM] [Warning] seed is set=1, random_state=42 will be ignored. Current value:␣
↪seed=1
[LightGBM] [Warning] seed is set=1, random_state=42 will be ignored. Current value:␣
↪seed=1
[LightGBM] [Warning] seed is set=2, random_state=42 will be ignored. Current value:␣
↪seed=2
[LightGBM] [Warning] seed is set=2, random_state=42 will be ignored. Current value:␣
↪seed=2
[LightGBM] [Warning] seed is set=2, random_state=42 will be ignored. Current value:␣
↪seed=2
[LightGBM] [Warning] seed is set=2, random_state=42 will be ignored. Current value:␣
↪seed=2
[LightGBM] [Warning] seed is set=2, random_state=42 will be ignored. Current value:␣
↪seed=2
[LightGBM] [Warning] seed is set=3, random_state=42 will be ignored. Current value:␣
↪seed=3
[LightGBM] [Warning] seed is set=3, random_state=42 will be ignored. Current value:␣
↪seed=3
[LightGBM] [Warning] seed is set=3, random_state=42 will be ignored. Current value:␣
↪seed=3
[LightGBM] [Warning] seed is set=3, random_state=42 will be ignored. Current value:␣
↪seed=3
[LightGBM] [Warning] seed is set=3, random_state=42 will be ignored. Current value:␣
↪seed=3
oof_pred:
array([[0.07027727],
       [0.06983411],
       [0.06983411],
       ...,
       [0.04349083],
       [0.09716105],
       [0.12494681]], dtype=float32)
Shape = (8000, 1)
CPU times: user 4min 23s, sys: 2.63 s, total: 4min 26s
Wall time: 37.3 s
```

Step 6. Analyze fitted model

Below we analyze feature importances of different algos:

[17]:
```
print('Feature importances of selector:\n{}'
              .format(selector.get_features_score()))
print('=' * 70)

print('Feature importances of top level algorithm:\n{}'
              .format(automl.levels[-1][0].ml_algos[0].get_features_score()))
print('=' * 70)

print('Feature importances of lowest level algorithm - model 0:\n{}'
```

(continues on next page)

(continued from previous page)

```
              .format(automl.levels[0][0].ml_algos[0].get_features_score()))
print('=' * 70)

print('Feature importances of lowest level algorithm - model 1:\n{}'
              .format(automl.levels[0][0].ml_algos[1].get_features_score()))
print('=' * 70)
```

```
Feature importances of selector:
EXT_SOURCE_3              1029.681686
EXT_SOURCE_2               894.265428
BIRTH_DATE                 537.081401
EXT_SOURCE_1               424.764621
DAYS_LAST_PHONE_CHANGE     262.583100
                              ...
FLAG_DOCUMENT_16             0.000000
FLAG_DOCUMENT_14             0.000000
FLAG_DOCUMENT_13             0.000000
FLAG_DOCUMENT_11             0.000000
FLAG_PHONE                   0.000000
Length: 110, dtype: float64
======================================================================
Feature importances of top level algorithm:
Lvl_0_Pipe_0_Mod_0_LightGBM_prediction_0    2546.473691
Lvl_0_Pipe_0_Mod_1_LightGBM_prediction_0    1686.589227
dtype: float64
======================================================================
Feature importances of lowest level algorithm - model 0:
EXT_SOURCE_2                  1500.371550
EXT_SOURCE_3                  1382.049802
dtdiff__BIRTH_DATE             714.069627
EXT_SOURCE_1                   573.079861
DAYS_REGISTRATION              461.927863
                                  ...
ord__HOUSETYPE_MODE              1.985318
ELEVATORS_MEDI                   1.862320
FLAG_DOCUMENT_6                  0.000000
REG_REGION_NOT_WORK_REGION       0.000000
ord__FLAG_OWN_CAR                0.000000
Length: 85, dtype: float64
======================================================================
Feature importances of lowest level algorithm - model 1:
EXT_SOURCE_3                   2666.270588
EXT_SOURCE_2                   2425.430385
dtdiff__BIRTH_DATE             1607.440484
DAYS_REGISTRATION              1217.128893
SK_ID_CURR                     1136.992744
                                  ...
LIVE_REGION_NOT_WORK_REGION       9.561320
ord__EMERGENCYSTATE_MODE          7.256624
REG_REGION_NOT_WORK_REGION        5.843864
ord__NAME_CONTRACT_TYPE           3.890026
FLAG_DOCUMENT_6                   3.523548
```

(continues on next page)

(continued from previous page)

```
Length: 85, dtype: float64
======================================================================
```

Step 7. Predict to test data and check scores

[18]:
```
%%time

test_pred = automl.predict(test_data)
print('Prediction for test data:\n{}\nShape = {}'
              .format(test_pred, test_pred.shape))

print('Check scores...')
print('OOF score: {}'.format(roc_auc_score(train_data[TARGET_NAME].values, oof_pred.
↪data[:, 0])))
print('TEST score: {}'.format(roc_auc_score(test_data[TARGET_NAME].values, test_pred.
↪data[:, 0])))
```

```
Prediction for test data:
array([[0.060448  ],
       [0.07832611],
       [0.05339179],
       ...,
       [0.06192666],
       [0.07732402],
       [0.20730501]], dtype=float32)
Shape = (2000, 1)
Check scores...
OOF score: 0.6979918272484156
TEST score: 0.7158254076086956
CPU times: user 421 ms, sys: 11.6 ms, total: 433 ms
Wall time: 103 ms
```

2.1.7 Tutorial 7: ICE and PDP Interpretation Tutorial

Official LightAutoML github repository is here

Partial dependence plot (PDP) and Individual Conditional Expectation (ICE) are two model-agnostic interpretation methods (see details here).

Download library and make some imports

[1]:
```
# !pip install lightautoml
```

[2]:
```
# Standard python libraries
import os
```

(continues on next page)

(continued from previous page)

```
import requests

# Installed libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
from sklearn.model_selection import train_test_split


# Imports from our package
from lightautoml.automl.presets.tabular_presets import TabularAutoML
from lightautoml.tasks import Task
```

[3]:
```
plt.rcParams.update({'font.size': 20})
sns.set(rc={'figure.figsize':(15, 11)})
sns.set(style="whitegrid", font_scale=1.5)

N_THREADS = 8 # threads cnt for lgbm and linear models
N_FOLDS = 5 # folds cnt for AutoML
RANDOM_STATE = 42 # fixed random state for various reasons
TEST_SIZE = 0.2 # Test size for metric check
TIMEOUT = 120 # Time in seconds for automl run
TARGET_NAME = 'TARGET' # Target column name
```

Prepare data

Load a dataset from the repository if doesn't clone repository by git.

[4]:
```
DATASET_DIR = './data/'
DATASET_NAME = 'sampled_app_train.csv'
DATASET_FULLNAME = os.path.join(DATASET_DIR, DATASET_NAME)
DATASET_URL = 'https://raw.githubusercontent.com/AILab-MLTools/LightAutoML/master/
↪examples/data/sampled_app_train.csv'
```

[5]:
```
%%time

if not os.path.exists(DATASET_FULLNAME):
    os.makedirs(DATASET_DIR, exist_ok=True)

    dataset = requests.get(DATASET_URL).text
    with open(DATASET_FULLNAME, 'w') as output:
        output.write(dataset)

data = pd.read_csv(DATASET_FULLNAME)
data['EMP_DATE'] = (np.datetime64('2018-01-01') + np.clip(data['DAYS_EMPLOYED'], None,␣
↪0).astype(np.dtype('timedelta64[D]'))
                    ).astype(str)
```

```
CPU times: user 223 ms, sys: 52.9 ms, total: 276 ms
Wall time: 503 ms
```

[6]:
```
train_data, test_data = train_test_split(data,
                                         test_size=TEST_SIZE,
                                         stratify=data[TARGET_NAME],
                                         random_state=RANDOM_STATE)
```

Create AutoML from preset

Also works with `lightautoml.automl.presets.tabular_presets.TabularUtilizedAutoML`.

[7]:
```
%%time

task = Task('binary', )
roles = {'target': TARGET_NAME,}

automl = TabularAutoML(task = task,
                       timeout = TIMEOUT,
                       cpu_limit = N_THREADS,
                       reader_params = {'n_jobs': N_THREADS, 'cv': N_FOLDS, 'random_state
↪': RANDOM_STATE},
                      )
oof_pred = automl.fit_predict(train_data, roles = roles, verbose = 1, log_file = 'train.
↪log')
```

```
[16:58:33] Stdout logging level is INFO.
[16:58:33] Copying TaskTimer may affect the parent PipelineTimer, so copy will create␣
↪new unlimited TaskTimer
[16:58:33] Task: binary

[16:58:33] Start automl preset with listed constraints:
[16:58:33] - time: 120.00 seconds
[16:58:33] - CPU: 8 cores
[16:58:33] - memory: 16 GB

[16:58:33] Train data shape: (8000, 123)

[16:58:36] Layer 1 train process start. Time left 117.58 secs
[16:58:36] Start fitting Lvl_0_Pipe_0_Mod_0_LinearL2 ...
[16:58:40] Fitting Lvl_0_Pipe_0_Mod_0_LinearL2 finished. score = 0.7340989893230383
[16:58:40] Lvl_0_Pipe_0_Mod_0_LinearL2 fitting and predicting completed
[16:58:40] Time left 112.94 secs

[16:58:43] Selector_LightGBM fitting and predicting completed
[16:58:44] Start fitting Lvl_0_Pipe_1_Mod_0_LightGBM ...
[16:58:53] Time limit exceeded after calculating fold 3

[16:58:53] Fitting Lvl_0_Pipe_1_Mod_0_LightGBM finished. score = 0.7336652733096534
[16:58:53] Lvl_0_Pipe_1_Mod_0_LightGBM fitting and predicting completed
[16:58:53] Start hyperparameters optimization for Lvl_0_Pipe_1_Mod_1_Tuned_LightGBM ...␣
↪Time budget is 1.00 secs
[16:59:03] Hyperparameters optimization for Lvl_0_Pipe_1_Mod_1_Tuned_LightGBM completed
[16:59:03] Start fitting Lvl_0_Pipe_1_Mod_1_Tuned_LightGBM ...
[16:59:16] Fitting Lvl_0_Pipe_1_Mod_1_Tuned_LightGBM finished. score = 0.7146425170595188
[16:59:16] Lvl_0_Pipe_1_Mod_1_Tuned_LightGBM fitting and predicting completed
```

(continues on next page)

(continued from previous page)

```
[16:59:16] Start fitting Lvl_0_Pipe_1_Mod_2_CatBoost ...
[16:59:21] Fitting Lvl_0_Pipe_1_Mod_2_CatBoost finished. score = 0.7180592042951911
[16:59:21] Lvl_0_Pipe_1_Mod_2_CatBoost fitting and predicting completed
[16:59:21] Start hyperparameters optimization for Lvl_0_Pipe_1_Mod_3_Tuned_CatBoost ...␣
↪Time budget is 29.20 secs
[16:59:51] Hyperparameters optimization for Lvl_0_Pipe_1_Mod_3_Tuned_CatBoost completed
[16:59:51] Start fitting Lvl_0_Pipe_1_Mod_3_Tuned_CatBoost ...
[16:59:58] Fitting Lvl_0_Pipe_1_Mod_3_Tuned_CatBoost finished. score = 0.7424781750625415
[16:59:58] Lvl_0_Pipe_1_Mod_3_Tuned_CatBoost fitting and predicting completed
[16:59:58] Time left 35.17 secs

[16:59:58] Time limit exceeded in one of the tasks. AutoML will blend level 1 models.

[16:59:58] Layer 1 training completed.

[16:59:58] Blending: optimization starts with equal weights and score 0.7470969001073415
[16:59:58] Blending: iteration 0: score = 0.7483672886691461, weights = [0.18754406 0.
↪1279657  0.37286162 0.06386749 0.24776113]
[16:59:58] Blending: iteration 1: score = 0.7484541355819561, weights = [0.23439428 0.
↪12674679 0.31599942 0.06325912 0.25960034]
[16:59:59] Blending: iteration 2: score = 0.748450627689517, weights = [0.23445104 0.
↪1267374  0.315976   0.06325444 0.25958112]
[16:59:59] Blending: iteration 3: score = 0.748450627689517, weights = [0.23445104 0.
↪1267374  0.315976   0.06325444 0.25958112]
[16:59:59] Blending: no score update. Terminated

[16:59:59] Automl preset training completed in 85.25 seconds

[16:59:59] Model description:
Final prediction for new objects (level 0) =
         0.23445 * (5 averaged models Lvl_0_Pipe_0_Mod_0_LinearL2) +
         0.12674 * (4 averaged models Lvl_0_Pipe_1_Mod_0_LightGBM) +
         0.31598 * (5 averaged models Lvl_0_Pipe_1_Mod_1_Tuned_LightGBM) +
         0.06325 * (5 averaged models Lvl_0_Pipe_1_Mod_2_CatBoost) +
         0.25958 * (5 averaged models Lvl_0_Pipe_1_Mod_3_Tuned_CatBoost)

CPU times: user 10min 7s, sys: 49.1 s, total: 10min 56s
Wall time: 1min 25s
```

Calculate interpretation data

ICE shows the functional relationship between the predicted response and the feature separately for each instance. PDP averages the individual lines of an ICE plot.

Numeric features

For numeric features you can specify `n_bins` - number of bins into which the range of feature values is divided.

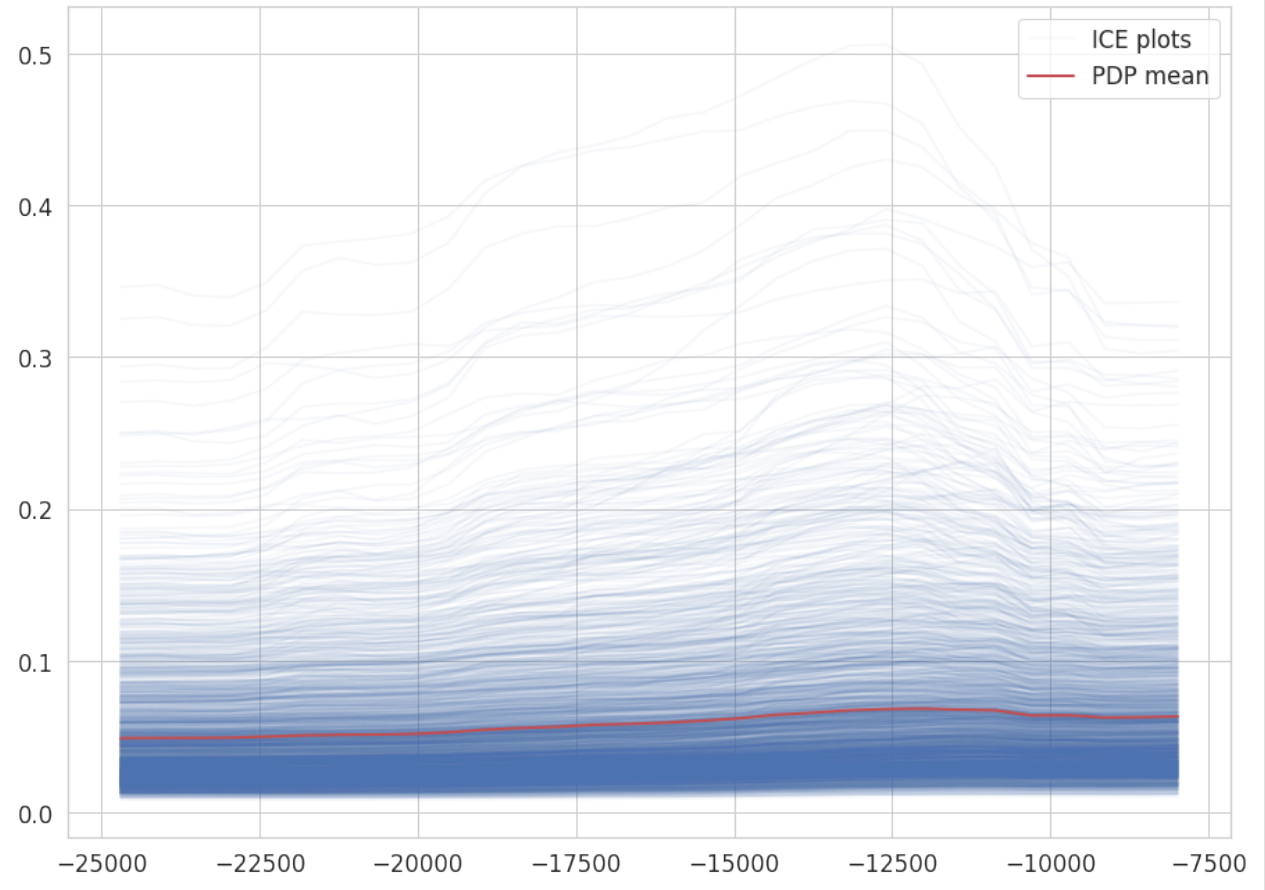
Calculate data for PDP plot manually:

[8]:
```
%%time

grid, ys, counts = automl.get_individual_pdp(test_data, feature_name='DAYS_BIRTH', n_
↪bins=30)
```

```
100%|| 30/30 [00:18<00:00,  1.63it/s]
```

```
CPU times: user 2min 2s, sys: 7.35 s, total: 2min 9s
Wall time: 18.4 s
```

[9]:
```
%%time

X = np.array([item.ravel() for item in ys]).T

plt.figure(figsize=(15, 11))
plt.plot(grid, X[0], alpha=0.05, color='m', label='ICE plots')
for i in range(1, X.shape[0]):
    plt.plot(grid, X[i], alpha=0.05, color='b')
plt.plot(grid, X.mean(axis=0), linewidth=2, color='r', label='PDP mean')
plt.legend()
plt.show()
```

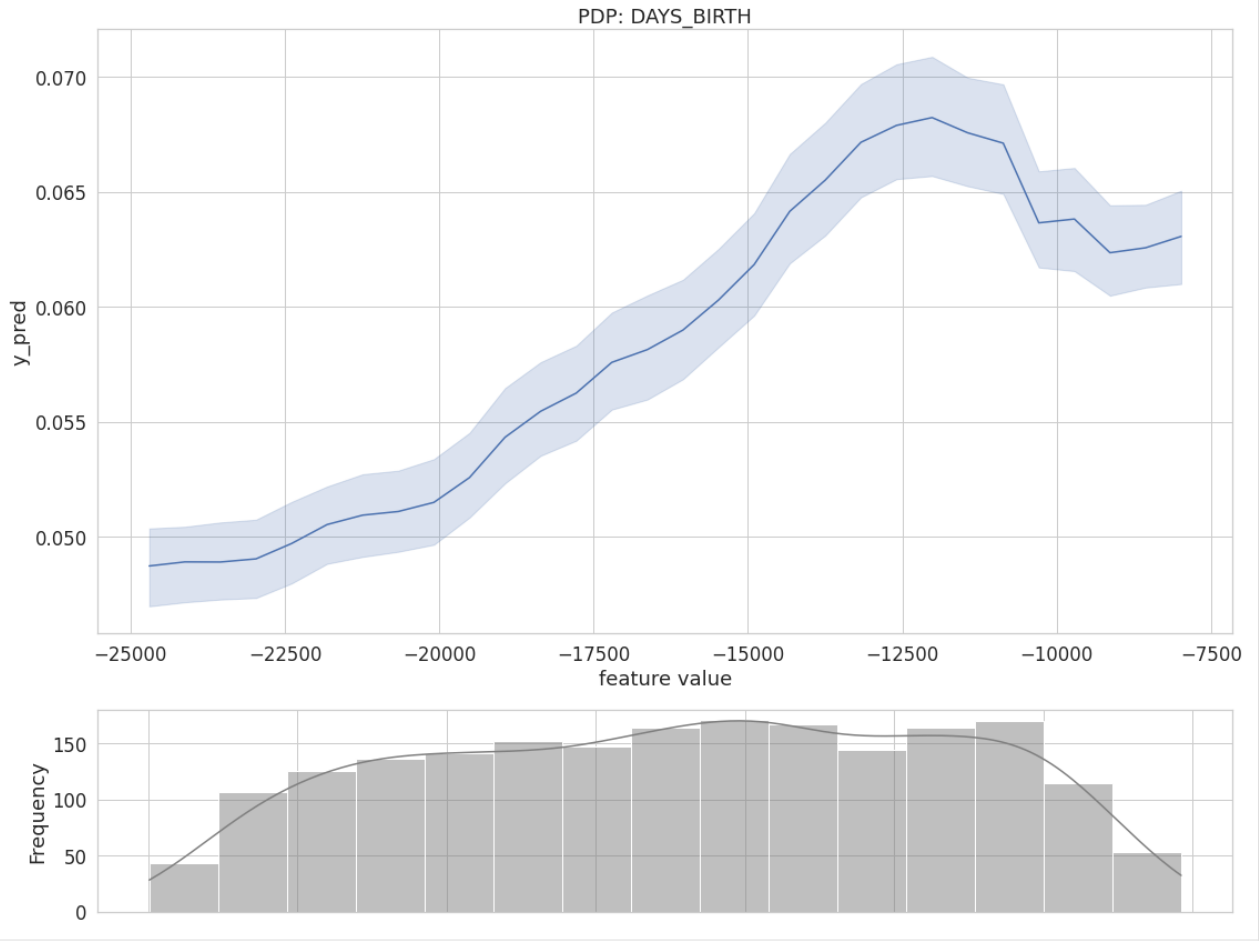


```
CPU times: user 5.9 s, sys: 3.63 s, total: 9.53 s
Wall time: 2.46 s
```

Built-in function:

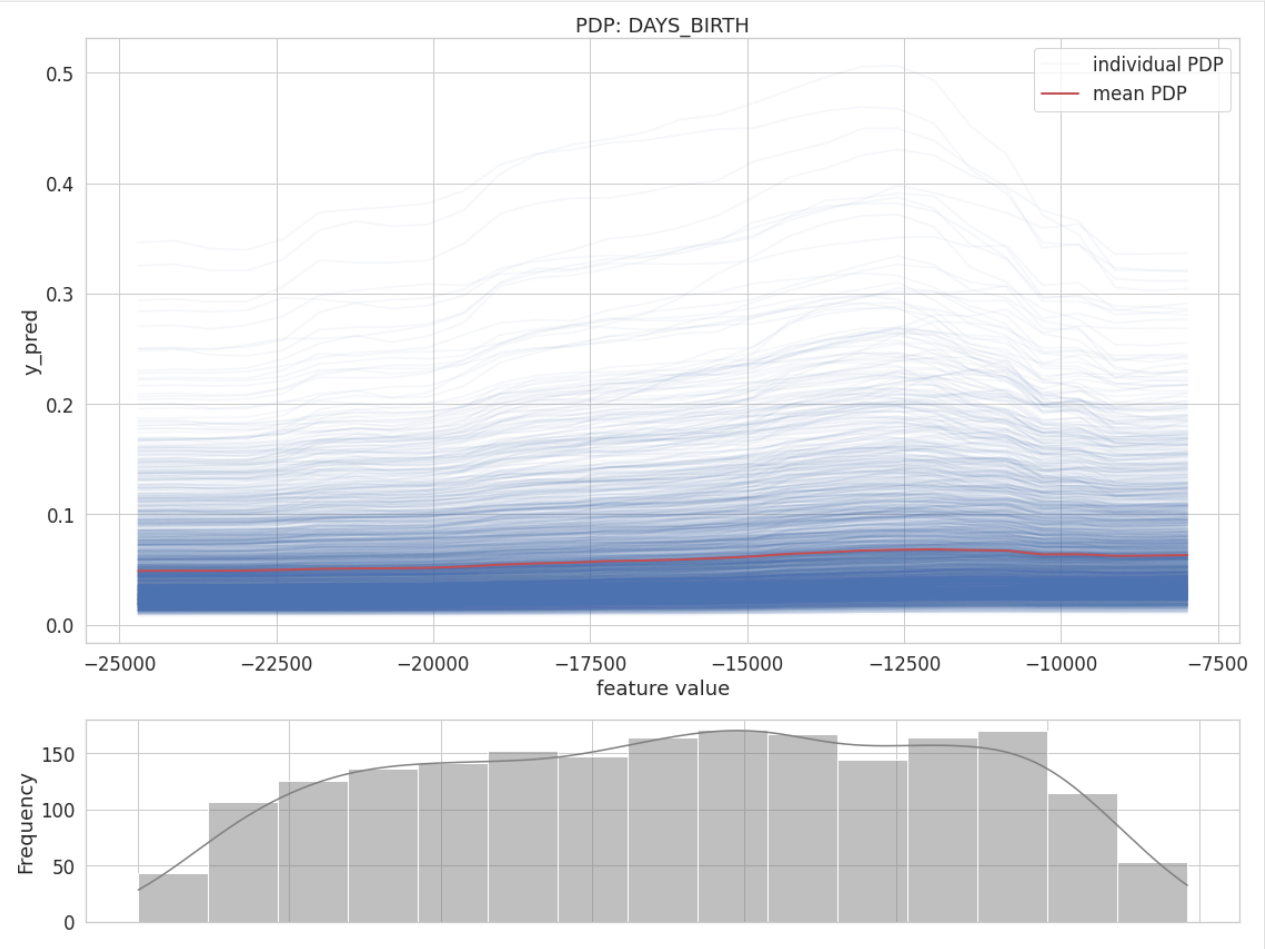
[10]:
```
automl.plot_pdp(test_data, feature_name='DAYS_BIRTH')
```

```
100%|| 30/30 [00:17<00:00,  1.67it/s]
```



[11]:
```
automl.plot_pdp(test_data, feature_name='DAYS_BIRTH', individual=True)
```

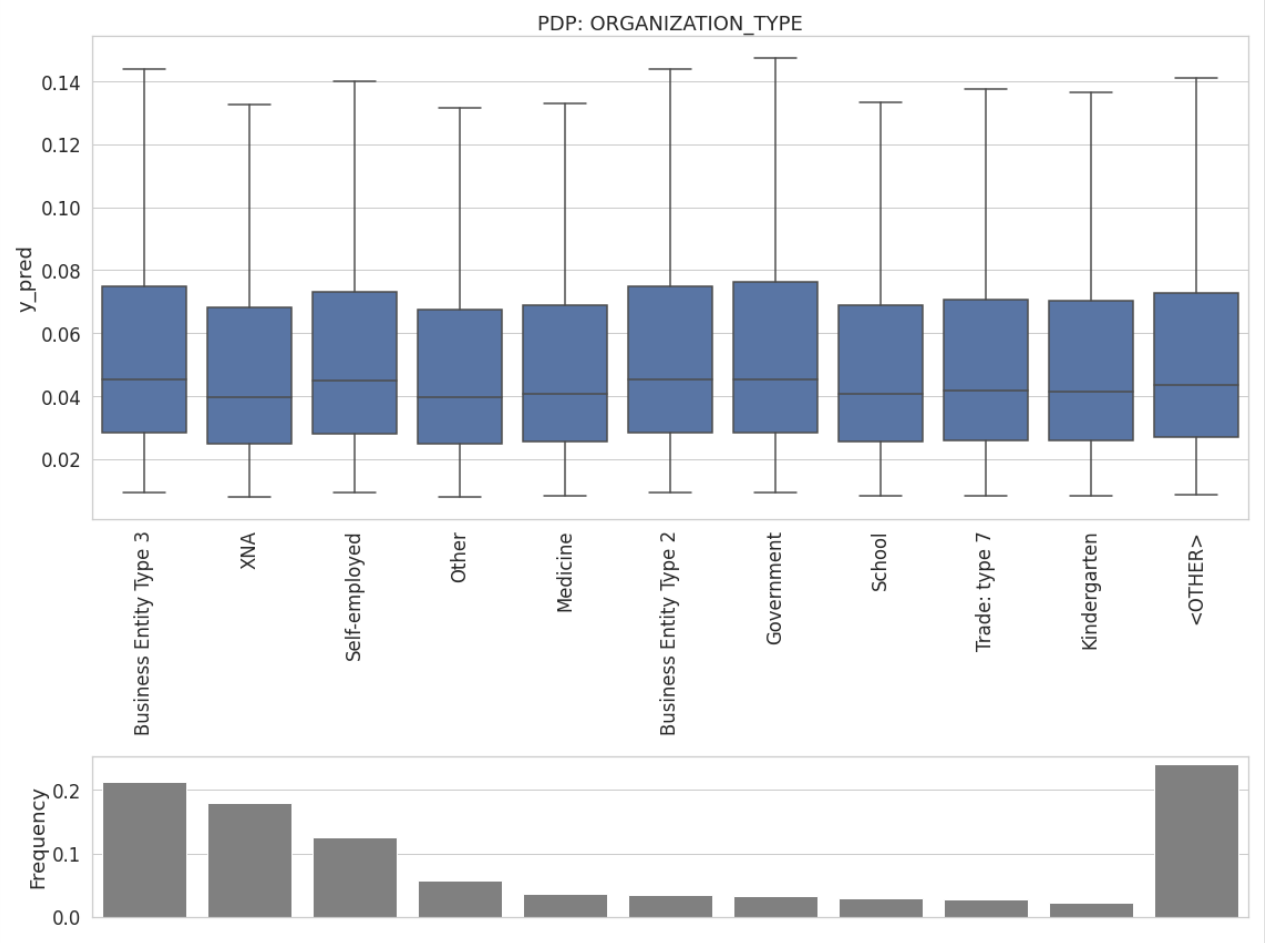
```
100%|| 30/30 [00:18<00:00,  1.63it/s]
```



Categorical features

[12]:
```
%%time

automl.plot_pdp(test_data, feature_name='ORGANIZATION_TYPE')
```

```
100%|| 10/10 [00:05<00:00,  1.69it/s]
```



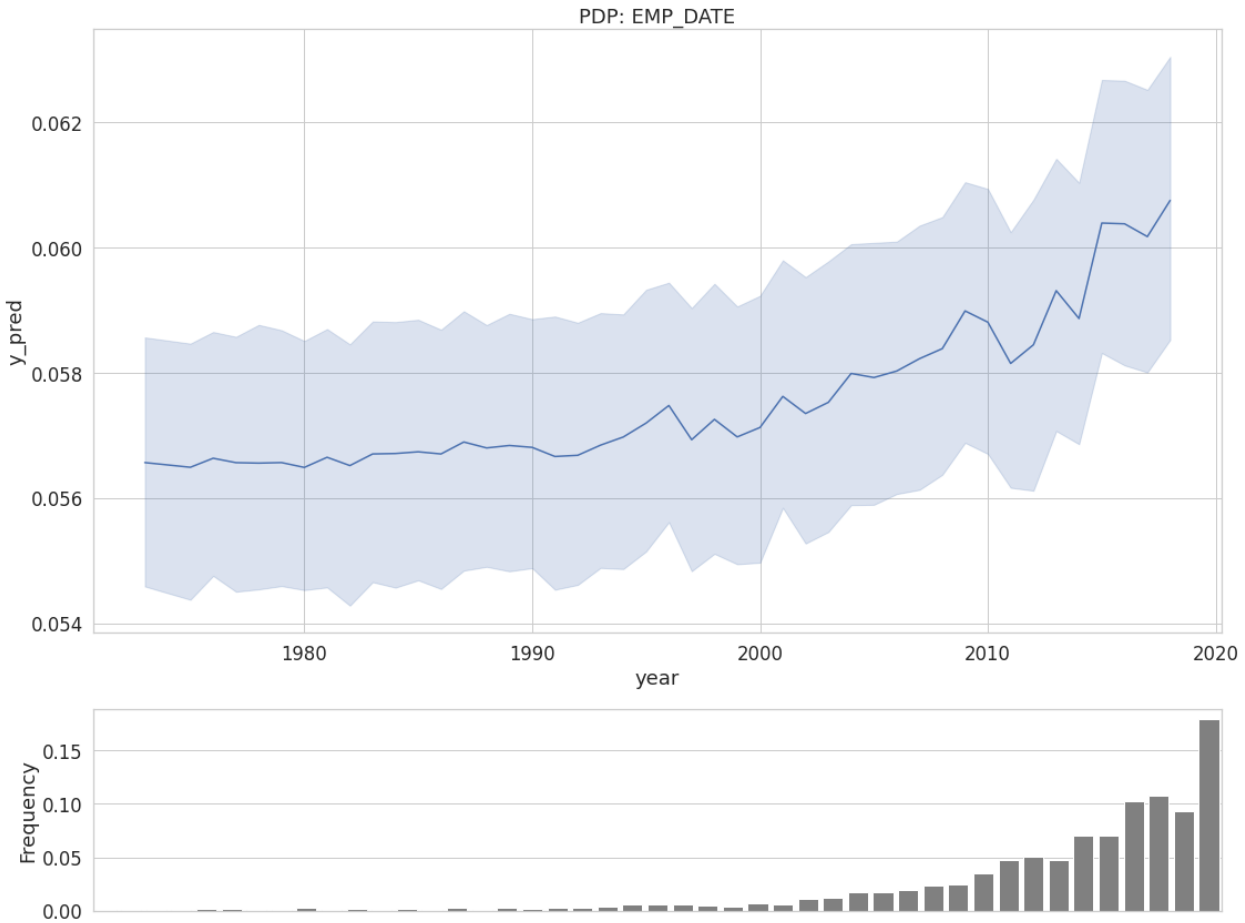

```
CPU times: user 43.8 s, sys: 2.54 s, total: 46.4 s
Wall time: 6.87 s
```

Datetime features

For datetime features you can specify groupby level, allowed values: `year`, `month`, `dayofweek`.

[13]:
```
%%time

automl.plot_pdp(test_data, feature_name='EMP_DATE', datetime_level='year')
```

```
100%|| 45/45 [00:27<00:00,  1.63it/s]
```




```
CPU times: user 3min 2s, sys: 10.2 s, total: 3min 12s
Wall time: 29.4 s
```

2.1.8 Tutorial 8: CV preset

Official LightAutoML github repository is here

In this tutorial we will look how to apply LightAutoML to computer vision tasks.

Basically, the corresponding modules are designed to solve problems where the image is more of an auxiliary value (complement the rest of the data from the table) than for solving full-fledged CV problems. In LightAutoML working with images goes essentially through tabular data, that is, not the images themselves are used, but the paths for them. They should be written in a separate column, which needs to specify the corresponding `'path'` role. The target variable and optionally other features are also specified in the table. To make predictions, numerical features are extracted from images, such as color histograms (RGB or HSV) and image embeddings based on EfficientNet (with the option to select a version and use AdvProp weights), and then standard machine learning models available in LightAutoML (as in conventional tabular presets) can be applied to them. By default, linear regression with L2 regularization and

CatBoost are used. Linear regression is trained on image embeddings, CatBoost is trained on histogram features, and weighted blending is finally applied to their predictions.

As an example, let's consider the Paddy Doctor competition - the task of multi-class classification, determining the type of paddy leaf disease based on photographs and other numerical features. Data is a set of images and a table, each row of which corresponds to a specific image with a specification of the path to it.

Importing libraries and preparing data

We will use the data from Kaggle. You can download the dataset from this link and import it in any convenient way. For example, we download the data using kaggle API and install some corresponding requirements. You can run next cell for loading data and installing packages in this way:

[ ]:
```
##Kaggle functionality for loading data; Note that you have to use your kaggle API token␣
↪(see the link above):
# !pip install opendatasets
# !pip install -q kaggle
# !pip install --upgrade --force-reinstall --no-deps kaggle
# !mkdir ~/.kaggle
# !ls ~/.kaggle
# import os.path
# assert os.path.isfile("kaggle.json")
# !cp kaggle.json ~/.kaggle/
# !chmod 600 ~/.kaggle/kaggle.json
# !kaggle competitions download -c paddy-disease-classification

# #Unpack data:
# !mkdir paddy-disease
# !unzip paddy-disease-classification.zip -d paddy-disease

# #Install LightAutoML, Pandas and torch EfficientNet:
# !pip install -U lightautoml[cv] #[cv] is for installing CV tasks functionality
```

Then we will import the libraries we use in this kernel: - Standard python libraries for timing, working with OS etc. - Essential python DS libraries like numpy, pandas, scikit-learn and torch (the last we will use in the next cell) - LightAutoML modules: `TabularCVAutoML` preset for AutoML model creation and Task class to setup what kind of ML problem we solve (binary/multiclass classification or regression)

[1]:
```
import os
os.environ["CUDA_DEVICE_ORDER"]="PCI_BUS_ID"   # see issue #152
os.environ["CUDA_VISIBLE_DEVICES"]="0"
```

[2]:
```
# Standard python libraries
import os
import time

# Essential DS libraries
import numpy as np
import pandas as pd
from sklearn.metrics import f1_score, accuracy_score, confusion_matrix
from sklearn.model_selection import train_test_split
import torch
import seaborn as sns
```

(continues on next page)

(continued from previous page)

```
import matplotlib.pyplot as plt

# LightAutoML presets, task and report generation
from lightautoml.automl.presets.image_presets import TabularCVAutoML
from lightautoml.tasks import Task
```

```
'nlp' extra dependecy package 'gensim' isn't installed. Look at README.md in repo
↪'LightAutoML' for installation instructions.
'nlp' extra dependecy package 'nltk' isn't installed. Look at README.md in repo
↪'LightAutoML' for installation instructions.
'nlp' extra dependecy package 'transformers' isn't installed. Look at README.md in repo
↪'LightAutoML' for installation instructions.
'nlp' extra dependecy package 'gensim' isn't installed. Look at README.md in repo
↪'LightAutoML' for installation instructions.
'nlp' extra dependecy package 'nltk' isn't installed. Look at README.md in repo
↪'LightAutoML' for installation instructions.
'nlp' extra dependecy package 'transformers' isn't installed. Look at README.md in repo
↪'LightAutoML' for installation instructions.
```

```
/home/dvladimirvasilyev/LightAutoML/lightautoml/ml_algo/dl_model.py:41: UserWarning:
↪'transformers' - package isn't installed
  warnings.warn("'transformers' - package isn't installed")
/home/dvladimirvasilyev/LightAutoML/lightautoml/text/nn_model.py:22: UserWarning:
↪'transformers' - package isn't installed
  warnings.warn("'transformers' - package isn't installed")
/home/dvladimirvasilyev/LightAutoML/lightautoml/text/dl_transformers.py:25: UserWarning:
↪'transformers' - package isn't installed
  warnings.warn("'transformers' - package isn't installed")
```

For better reproducibility fix numpy random seed with max number of threads for Torch (which usually try to use all the threads on server):

[3]:
```
np.random.seed(42)
torch.set_num_threads(2)
```

Let's check the data we have:

[4]:
```
INPUT_DIR = './paddy-disease/'
```

[5]:
```
train_data = pd.read_csv(INPUT_DIR + 'train.csv')
print(train_data.shape)
train_data.head()
```

```
(10407, 4)
```

[5]:

	image_id	label	variety	age
0	100330.jpg	bacterial_leaf_blight	ADT45	45
1	100365.jpg	bacterial_leaf_blight	ADT45	45
2	100382.jpg	bacterial_leaf_blight	ADT45	45
3	100632.jpg	bacterial_leaf_blight	ADT45	45
4	101918.jpg	bacterial_leaf_blight	ADT45	45

[6]:
```
train_data['label'].value_counts()
```

[6]:
```
normal                      1764
blast                       1738
hispa                       1594
dead_heart                  1442
tungro                      1088
brown_spot                   965
downy_mildew                 620
bacterial_leaf_blight        479
bacterial_leaf_streak        380
bacterial_panicle_blight     337
Name: label, dtype: int64
```

[7]:
```
train_data['variety'].value_counts()
```

[7]:
```
ADT45             6992
KarnatakaPonni     988
Ponni              657
AtchayaPonni       461
Zonal              399
AndraPonni         377
Onthanel           351
IR20               114
RR                  36
Surya               32
Name: variety, dtype: int64
```

[8]:
```
train_data['age'].value_counts()
```

[8]:
```
70    3077
60    1660
50    1066
75     866
65     774
55     563
72     552
45     505
67     415
68     253
80     225
57     213
47     112
77      42
73      38
66      36
62       5
82       5
Name: age, dtype: int64
```

[9]:
```
submission = pd.read_csv(INPUT_DIR + 'sample_submission.csv')
print(submission.shape)
submission.head()
```

```
(3469, 2)
```

[9]:
```
     image_id  label
0  200001.jpg    NaN
1  200002.jpg    NaN
2  200003.jpg    NaN
3  200004.jpg    NaN
4  200005.jpg    NaN
```

Add a column with the full path to the images:

[10]:
```
%%time

train_data['path'] = INPUT_DIR + 'train_images/' + train_data['label'] + '/' + train_
↪data['image_id']
train_data.head()
```

```
CPU times: user 4.89 ms, sys: 485 µs, total: 5.37 ms
Wall time: 5.14 ms
```

[10]:
```
     image_id                  label variety  age  \
0  100330.jpg  bacterial_leaf_blight   ADT45   45
1  100365.jpg  bacterial_leaf_blight   ADT45   45
2  100382.jpg  bacterial_leaf_blight   ADT45   45
3  100632.jpg  bacterial_leaf_blight   ADT45   45
4  101918.jpg  bacterial_leaf_blight   ADT45   45

                                            path
0  ./paddy-disease/train_images/bacterial_leaf_bl...
1  ./paddy-disease/train_images/bacterial_leaf_bl...
2  ./paddy-disease/train_images/bacterial_leaf_bl...
3  ./paddy-disease/train_images/bacterial_leaf_bl...
4  ./paddy-disease/train_images/bacterial_leaf_bl...
```

[11]:
```
submission['path'] = INPUT_DIR + 'test_images/' + submission['image_id']
submission.head()
```

[11]:
```
     image_id  label                                    path
0  200001.jpg    NaN  ./paddy-disease/test_images/200001.jpg
1  200002.jpg    NaN  ./paddy-disease/test_images/200002.jpg
2  200003.jpg    NaN  ./paddy-disease/test_images/200003.jpg
3  200004.jpg    NaN  ./paddy-disease/test_images/200004.jpg
4  200005.jpg    NaN  ./paddy-disease/test_images/200005.jpg
```

Let's expand the training data with augmentations: random rotations and flips:

[ ]:
```
os.mkdir('./paddy-disease/modified_train')
```

[12]:
```
from PIL import Image
from tqdm.notebook import tqdm
new_imgs = []

for i, p in tqdm(enumerate(train_data['path'].values)):
    if i % 1000 == 0:
        print(i)
```

(continues on next page)

(continued from previous page)

```
    img = Image.open(p)

    for it in range(10):
        new_img = img.rotate(np.random.rand() * 60 - 30, resample=3)

        if np.random.rand() > 0.5:
            new_img = new_img.transpose(Image.FLIP_LEFT_RIGHT)

        new_img_name = './paddy-disease/modified_train/' + p.split('/')[-1][:-4] + '_' +␣
↪str(it) + '.jpg'
        new_img.save(new_img_name)
        new_imgs.append([new_img_name, p.split('/')[-2], p.split('/')[-1]])
```

```
0it [00:00, ?it/s]
```

```
0
1000
2000
3000
4000
5000
6000
7000
8000
9000
10000
```

[13]:
```
train_data = pd.concat([train_data, pd.DataFrame(new_imgs, columns = ['path', 'label',
↪'image_id'])]).reset_index(drop = True)
train_data
```

[13]:
```
          image_id                  label variety   age  \
0       100330.jpg  bacterial_leaf_blight   ADT45  45.0
1       100365.jpg  bacterial_leaf_blight   ADT45  45.0
2       100382.jpg  bacterial_leaf_blight   ADT45  45.0
3       100632.jpg  bacterial_leaf_blight   ADT45  45.0
4       101918.jpg  bacterial_leaf_blight   ADT45  45.0
...            ...                    ...     ...   ...
114472  110381.jpg                tungro     NaN   NaN
114473  110381.jpg                tungro     NaN   NaN
114474  110381.jpg                tungro     NaN   NaN
114475  110381.jpg                tungro     NaN   NaN
114476  110381.jpg                tungro     NaN   NaN

                                                    path
0       ./paddy-disease/train_images/bacterial_leaf_bl...
1       ./paddy-disease/train_images/bacterial_leaf_bl...
2       ./paddy-disease/train_images/bacterial_leaf_bl...
3       ./paddy-disease/train_images/bacterial_leaf_bl...
4       ./paddy-disease/train_images/bacterial_leaf_bl...
...                                                   ...
114472        ./paddy-disease/modified_train/110381_5.jpg
114473        ./paddy-disease/modified_train/110381_6.jpg
```

(continues on next page)

(continued from previous page)

```
114474        ./paddy-disease/modified_train/110381_7.jpg
114475        ./paddy-disease/modified_train/110381_8.jpg
114476        ./paddy-disease/modified_train/110381_9.jpg

[114477 rows x 5 columns]
```

Let's do the same for the test dataset:

[22]:
```
os.mkdir('./paddy-disease/modified_test')
```

[14]:
```
new_imgs = []

for i, p in tqdm(enumerate(submission['path'].values)):
    if i % 1000 == 0:
        print(i)

    img = Image.open(p)

    for it in range(5):
        new_img = img.rotate(np.random.rand() * 60 - 30, resample=3)
        if np.random.rand() > 0.5:
            new_img = new_img.transpose(Image.FLIP_LEFT_RIGHT)

        new_img_name = './paddy-disease/modified_test/' + p.split('/')[-1][:-4] + '_' +␣
↪str(it) + '.jpg'
        new_img.save(new_img_name)
        new_imgs.append([new_img_name, p.split('/')[-1]])
```

```
0it [00:00, ?it/s]
```

```
0
1000
2000
3000
```

[15]:
```
submission = pd.concat([submission, pd.DataFrame(new_imgs, columns = ['path', 'image_id
↪'])]).reset_index(drop = True)
submission
```

[15]:

	image_id	label	path
0	200001.jpg	NaN	./paddy-disease/test_images/200001.jpg
1	200002.jpg	NaN	./paddy-disease/test_images/200002.jpg
2	200003.jpg	NaN	./paddy-disease/test_images/200003.jpg
3	200004.jpg	NaN	./paddy-disease/test_images/200004.jpg
4	200005.jpg	NaN	./paddy-disease/test_images/200005.jpg
...	...	...	...
20809	203469.jpg	NaN	./paddy-disease/modified_test/203469_0.jpg
20810	203469.jpg	NaN	./paddy-disease/modified_test/203469_1.jpg
20811	203469.jpg	NaN	./paddy-disease/modified_test/203469_2.jpg
20812	203469.jpg	NaN	./paddy-disease/modified_test/203469_3.jpg
20813	203469.jpg	NaN	./paddy-disease/modified_test/203469_4.jpg

[20814 rows x 3 columns]

Task definition

Task type

On the cell below we create `Task` object - the class to setup what task LightAutoML model should solve with specific loss and metric if necessary (more info can be found here in our documentation). In general, it can be any type of tasks available in LightAutoML (binary and multi-class classification, one-dimensional and multi-dimensional regression, multi-label classification), but in this case we have a multi-class classification task:

[16]:
```
task = Task('multiclass')
```

Default metric and loss in multi-class classification is cross-entropy.

Feature roles setup

Next we need to setup columns roles. It is necessary to specify the role of the target variable (`'target'`), as well as the role of the path to the images (`'path'`) in the case of using `TabularCVAutoML`. We will also group the images (the original ones and their augmentations) and apply group k-fold cross-validation, specifying the column with ids as the `'group'` role:

[17]:
```
roles = {
    'target': 'label',
    'path': ['path'],
    'drop': ['variety', 'age'],
    'group': 'image_id'
}
```

Then we initialize `TabularCVAutoML`. It is possible to specify many parameters (reader parameters, time and memory limits etc), including the EfficientNet parameters for getting embeddings: version (`B0` by default), device, batch size (128 by default), path for weights, AdvProp weights using (for better use of the shape in images, `True` by default) etc. Note that the `Utilized` version of `TabularCVAutoML` for more flexible use of time resources is not yet available.

[18]:
```
automl = TabularCVAutoML(task = task,
                         timeout=5 * 3600,
                        cpu_limit = 2,
                        reader_params = {'cv': 5, 'random_state': 42})
```

AutoML training

To run autoML training use fit_predict method: - `train_data` - Dataset to train. - `roles` - Roles dict. - `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;

Note: out-of-fold prediction is calculated during training and returned from the fit_predict method

[19]:
```
%%time

oof_pred = automl.fit_predict(train_data, roles = roles, verbose = 3)
```

```
[14:04:32] Stdout logging level is INFO3.
[14:04:32] Task: multiclass

[14:04:32] Start automl preset with listed constraints:
[14:04:32] - time: 18000.00 seconds
```

(continues on next page)

(continued from previous page)

```
[14:04:32] - CPU: 2 cores
[14:04:32] - memory: 16 GB

[14:04:32] Train data shape: (114477, 5)

[14:04:32] Layer 1 train process start. Time left 17999.83 secs
100%|| 895/895 [07:29<00:00,  1.99it/s]
[14:12:09] Feature path transformed
[14:12:16] Start fitting Lvl_0_Pipe_0_Mod_0_LinearL2 ...
[14:12:17] ===== Start working with fold 0 for Lvl_0_Pipe_0_Mod_0_LinearL2 =====
[14:12:26] Linear model: C = 1e-05 score = -0.9995305866945853
[14:12:32] Linear model: C = 5e-05 score = -0.6879959560713191
[14:12:38] Linear model: C = 0.0001 score = -0.5802952177399445
[14:12:45] Linear model: C = 0.0005 score = -0.3907926611544111
[14:12:51] Linear model: C = 0.001 score = -0.33425017155675657
[14:13:00] Linear model: C = 0.005 score = -0.2559518217619532
[14:13:07] Linear model: C = 0.01 score = -0.24141776919439237
[14:13:15] Linear model: C = 0.05 score = -0.2431661172897411
[14:13:23] Linear model: C = 0.1 score = -0.25925367786528475
[14:13:24] ===== Start working with fold 1 for Lvl_0_Pipe_0_Mod_0_LinearL2 =====
[14:13:32] Linear model: C = 1e-05 score = -0.9872444001968863
[14:13:39] Linear model: C = 5e-05 score = -0.6682540100549987
[14:13:45] Linear model: C = 0.0001 score = -0.5574685730009872
[14:13:51] Linear model: C = 0.0005 score = -0.3653461360638747
[14:13:58] Linear model: C = 0.001 score = -0.31059360297670363
[14:14:05] Linear model: C = 0.005 score = -0.2370436682635623
[14:14:14] Linear model: C = 0.01 score = -0.22495884629469698
[14:14:21] Linear model: C = 0.05 score = -0.23420873784566962
[14:14:29] Linear model: C = 0.1 score = -0.25263966927426823
[14:14:29] ===== Start working with fold 2 for Lvl_0_Pipe_0_Mod_0_LinearL2 =====
[14:14:37] Linear model: C = 1e-05 score = -0.9554531133528031
[14:14:43] Linear model: C = 5e-05 score = -0.640784196156178
[14:14:49] Linear model: C = 0.0001 score = -0.5345024606190905
[14:14:57] Linear model: C = 0.0005 score = -0.3546726337461952
[14:15:04] Linear model: C = 0.001 score = -0.30344210801693483
[14:15:12] Linear model: C = 0.005 score = -0.2331574262775805
[14:15:19] Linear model: C = 0.01 score = -0.22071779776854528
[14:15:28] Linear model: C = 0.05 score = -0.22603075278344578
[14:15:36] Linear model: C = 0.1 score = -0.24138537694410292
[14:15:36] ===== Start working with fold 3 for Lvl_0_Pipe_0_Mod_0_LinearL2 =====
[14:15:44] Linear model: C = 1e-05 score = -0.973115505822288
[14:15:51] Linear model: C = 5e-05 score = -0.6613476137718094
[14:15:56] Linear model: C = 0.0001 score = -0.5539538946164072
[14:16:04] Linear model: C = 0.0005 score = -0.3666276035478478
[14:16:10] Linear model: C = 0.001 score = -0.31130200709742806
[14:16:18] Linear model: C = 0.005 score = -0.2326339584928626
[14:16:25] Linear model: C = 0.01 score = -0.21658099282365262
[14:16:33] Linear model: C = 0.05 score = -0.21364841773406087
[14:16:42] Linear model: C = 0.1 score = -0.2256018292053085
[14:16:51] Linear model: C = 0.5 score = -0.2763179966937595
[14:16:51] ===== Start working with fold 4 for Lvl_0_Pipe_0_Mod_0_LinearL2 =====
```

(continues on next page)

(continued from previous page)

```
[14:16:58] Linear model: C = 1e-05 score = -0.9531496536787142
[14:17:05] Linear model: C = 5e-05 score = -0.6270339670737181
[14:17:10] Linear model: C = 0.0001 score = -0.517302736118502
[14:17:17] Linear model: C = 0.0005 score = -0.331531311465719
[14:17:23] Linear model: C = 0.001 score = -0.27798570249468424
[14:17:32] Linear model: C = 0.005 score = -0.20448637290477473
[14:17:39] Linear model: C = 0.01 score = -0.19081673660070902
[14:17:47] Linear model: C = 0.05 score = -0.1923892363102242
[14:17:56] Linear model: C = 0.1 score = -0.20661581389305533
[14:17:56] Fitting Lvl_0_Pipe_0_Mod_0_LinearL2 finished. score = -0.21831477243925082
[14:17:56] Lvl_0_Pipe_0_Mod_0_LinearL2 fitting and predicting completed
[14:17:56] Time left 17195.98 secs

[14:22:15] Start fitting Lvl_0_Pipe_1_Mod_0_CatBoost ...
[14:22:16] ===== Start working with fold 0 for Lvl_0_Pipe_1_Mod_0_CatBoost =====
[14:22:16] 0:   learn: 2.2636799        test: 2.2649649 best: 2.2649649 (0)     total: 6.
↪85ms   remaining: 27.4s
[14:22:35] bestTest = 0.2436411292
[14:22:35] bestIteration = 3999
[14:22:35] ===== Start working with fold 1 for Lvl_0_Pipe_1_Mod_0_CatBoost =====
[14:22:36] 0:   learn: 2.2634692        test: 2.2632526 best: 2.2632526 (0)     total: 6.
↪16ms   remaining: 24.6s
[14:22:55] bestTest = 0.2658199543
[14:22:55] bestIteration = 3999
[14:22:56] ===== Start working with fold 2 for Lvl_0_Pipe_1_Mod_0_CatBoost =====
[14:22:56] 0:   learn: 2.2631654        test: 2.2656298 best: 2.2656298 (0)     total: 6.
↪08ms   remaining: 24.3s
[14:23:16] bestTest = 0.2753673319
[14:23:16] bestIteration = 3999
[14:23:16] ===== Start working with fold 3 for Lvl_0_Pipe_1_Mod_0_CatBoost =====
[14:23:17] 0:   learn: 2.2645696        test: 2.2657045 best: 2.2657045 (0)     total: 6.
↪76ms   remaining: 27s
[14:23:37] bestTest = 0.2738943611
[14:23:37] bestIteration = 3996
[14:23:37] Shrink model to first 3997 iterations.
[14:23:37] ===== Start working with fold 4 for Lvl_0_Pipe_1_Mod_0_CatBoost =====
[14:23:38] 0:   learn: 2.2642805        test: 2.2644245 best: 2.2644245 (0)     total: 5.
↪84ms   remaining: 23.4s
[14:23:57] bestTest = 0.2538460334
[14:23:57] bestIteration = 3999
[14:23:58] Fitting Lvl_0_Pipe_1_Mod_0_CatBoost finished. score = -0.2625123265864018
[14:23:58] Lvl_0_Pipe_1_Mod_0_CatBoost fitting and predicting completed
[14:23:58] Time left 16834.07 secs

[14:23:58] Layer 1 training completed.

[14:23:58] Blending: optimization starts with equal weights and score -0.1879588701291192

/home/dvladimirvasilyev/anaconda3/envs/myenv/lib/python3.8/site-packages/sklearn/metrics/
↪_classification.py:2916: UserWarning: The y_pred values do not sum to one. Starting␣
↪from 1.5 thiswill result in an error.
  warnings.warn(
```

```
[14:23:59] Blending: iteration 0: score = -0.18573794844833624, weights = [0.63928086 0.
↪36071914]
[14:23:59] Blending: iteration 1: score = -0.18573794844833624, weights = [0.63928086 0.
↪36071914]
[14:23:59] Blending: no score update. Terminated

[14:23:59] Automl preset training completed in 1167.35 seconds

[14:23:59] Model description:
Final prediction for new objects (level 0) =
         0.63928 * (5 averaged models Lvl_0_Pipe_0_Mod_0_LinearL2) +
         0.36072 * (5 averaged models Lvl_0_Pipe_1_Mod_0_CatBoost)

CPU times: user 18min 40s, sys: 3min 1s, total: 21min 42s
Wall time: 19min 27s
```

onsider out-of-fold predictions on train data. In case of classification, LightAutoML returns class probabilities as an output.

[21]:
```
preds = train_data[['image_id', 'label']]
preds
```

[21]:
```
          image_id                  label
0       100330.jpg  bacterial_leaf_blight
1       100365.jpg  bacterial_leaf_blight
2       100382.jpg  bacterial_leaf_blight
3       100632.jpg  bacterial_leaf_blight
4       101918.jpg  bacterial_leaf_blight
...            ...                    ...
114472  110381.jpg                tungro
114473  110381.jpg                tungro
114474  110381.jpg                tungro
114475  110381.jpg                tungro
114476  110381.jpg                tungro

[114477 rows x 2 columns]
```

[22]:
```
for i in range(10):
    preds['pred_' + str(i)] = oof_pred.data[:,i]

preds
```

```
/tmp/ipykernel_12895/1432655611.py:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_
↪guide/indexing.html#returning-a-view-versus-a-copy
  preds['pred_' + str(i)] = oof_pred.data[:,i]
/tmp/ipykernel_12895/1432655611.py:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_
```

(continues on next page)

(continued from previous page)

```
↪guide/indexing.html#returning-a-view-versus-a-copy
  preds['pred_' + str(i)] = oof_pred.data[:,i]
```

[22]:

```
          image_id                  label    pred_0    pred_1    pred_2  \
0       100330.jpg  bacterial_leaf_blight  0.023245  0.315283  0.470886
1       100365.jpg  bacterial_leaf_blight  0.003717  0.011035  0.028317
2       100382.jpg  bacterial_leaf_blight  0.025734  0.095088  0.208473
3       100632.jpg  bacterial_leaf_blight  0.002876  0.542942  0.027466
4       101918.jpg  bacterial_leaf_blight  0.009988  0.033572  0.017635
...            ...                   ...       ...       ...       ...
114472  110381.jpg                tungro  0.001716  0.109143  0.020722
114473  110381.jpg                tungro  0.022644  0.137650  0.026389
114474  110381.jpg                tungro  0.016897  0.072329  0.010469
114475  110381.jpg                tungro  0.008637  0.114299  0.082281
114476  110381.jpg                tungro  0.004179  0.099988  0.008320

          pred_3    pred_4    pred_5    pred_6    pred_7        pred_8  \
0       0.002528  0.021895  0.007454  0.001554  0.157142  8.914904e-06
1       0.000110  0.003178  0.000015  0.000131  0.953496  1.555987e-07
2       0.000879  0.007030  0.003382  0.000142  0.659271  3.872871e-07
3       0.000317  0.036005  0.000398  0.000082  0.389901  3.837710e-06
4       0.000032  0.008310  0.000136  0.000041  0.930286  1.554736e-07
...          ...       ...       ...       ...       ...           ...
114472  0.001495  0.845324  0.000177  0.021384  0.000027  6.304998e-06
114473  0.004165  0.788036  0.001093  0.019688  0.000259  3.142513e-05
114474  0.005554  0.789777  0.001240  0.103631  0.000060  1.301366e-05
114475  0.003465  0.560001  0.000741  0.230260  0.000112  1.909918e-04
114476  0.004660  0.822037  0.000663  0.059627  0.000318  1.922170e-04

              pred_9
0       4.559626e-06
1       5.692390e-07
2       2.898941e-07
3       9.339438e-06
4       1.530466e-07
...              ...
114472  6.075803e-06
114473  4.477663e-05
114474  2.972130e-05
114475  1.351225e-05
114476  1.441010e-05

[114477 rows x 12 columns]
```

We will average forecasts for images by their augmentations:

[23]:

```
preds = preds.groupby(['image_id', 'label']).mean().reset_index()
preds
```

[23]:

```
         image_id       label    pred_0    pred_1    pred_2        pred_3  \
0      100001.jpg  brown_spot  0.001334  0.000791  0.002372  5.432664e-03
1      100002.jpg      normal  0.978428  0.011744  0.001621  3.187062e-03
2      100003.jpg       hispa  0.004639  0.002192  0.992883  1.573081e-07
```

(continues on next page)

(continued from previous page)

```
3      100004.jpg       blast  0.000259  0.982406  0.004401  7.787708e-03
4      100005.jpg       hispa  0.010951  0.047475  0.829855  1.200308e-05
...           ...         ...       ...       ...       ...           ...
10402  110403.jpg      tungro  0.001664  0.002167  0.007366  4.507852e-03
10403  110404.jpg      normal  0.932484  0.002359  0.049850  1.244102e-05
10404  110405.jpg  dead_heart  0.000192  0.000044  0.000152  9.994839e-01
10405  110406.jpg       blast  0.000226  0.977683  0.000268  9.254745e-03
10406  110407.jpg  brown_spot  0.000009  0.000188  0.000539  4.357956e-04

          pred_4    pred_5    pred_6    pred_7        pred_8        pred_9
0      0.005328  0.978495  0.002519  0.003511  7.897679e-05  1.378119e-04
1      0.002579  0.000282  0.000156  0.001969  3.391063e-05  1.971700e-07
2      0.000026  0.000037  0.000005  0.000218  1.920397e-07  1.528186e-07
3      0.002372  0.002163  0.000173  0.000115  3.223106e-04  4.848040e-07
4      0.091933  0.000418  0.018967  0.000370  1.118553e-05  8.759866e-06
...         ...       ...       ...       ...           ...           ...
10402  0.981122  0.000052  0.001666  0.001455  1.527430e-07  3.928369e-07
10403  0.011696  0.000593  0.002646  0.000304  4.828784e-05  7.773816e-06
10404  0.000001  0.000025  0.000058  0.000003  1.957294e-06  3.789358e-05
10405  0.004962  0.000595  0.004523  0.001717  5.624577e-04  2.080105e-04
10406  0.000232  0.997215  0.000039  0.000010  1.319862e-03  1.372061e-05

[10407 rows x 12 columns]
```

Assign classes by maximum class probability:

[24]:
```
OOFs = np.argmax(preds[['pred_' + str(i) for i in range(10)]].values, axis = 1)
OOFs
```

[24]:
```
array([5, 0, 2, ..., 3, 1, 5])
```

Let's see classification accuracy on train:

[25]:
```
accuracy = (OOFs == preds['label'].map(automl.reader.target_mapping)).mean()
print(f'Out-of-fold accuracy: {accuracy}')
```

```
Out-of-fold accuracy: 0.9686749303353512
```

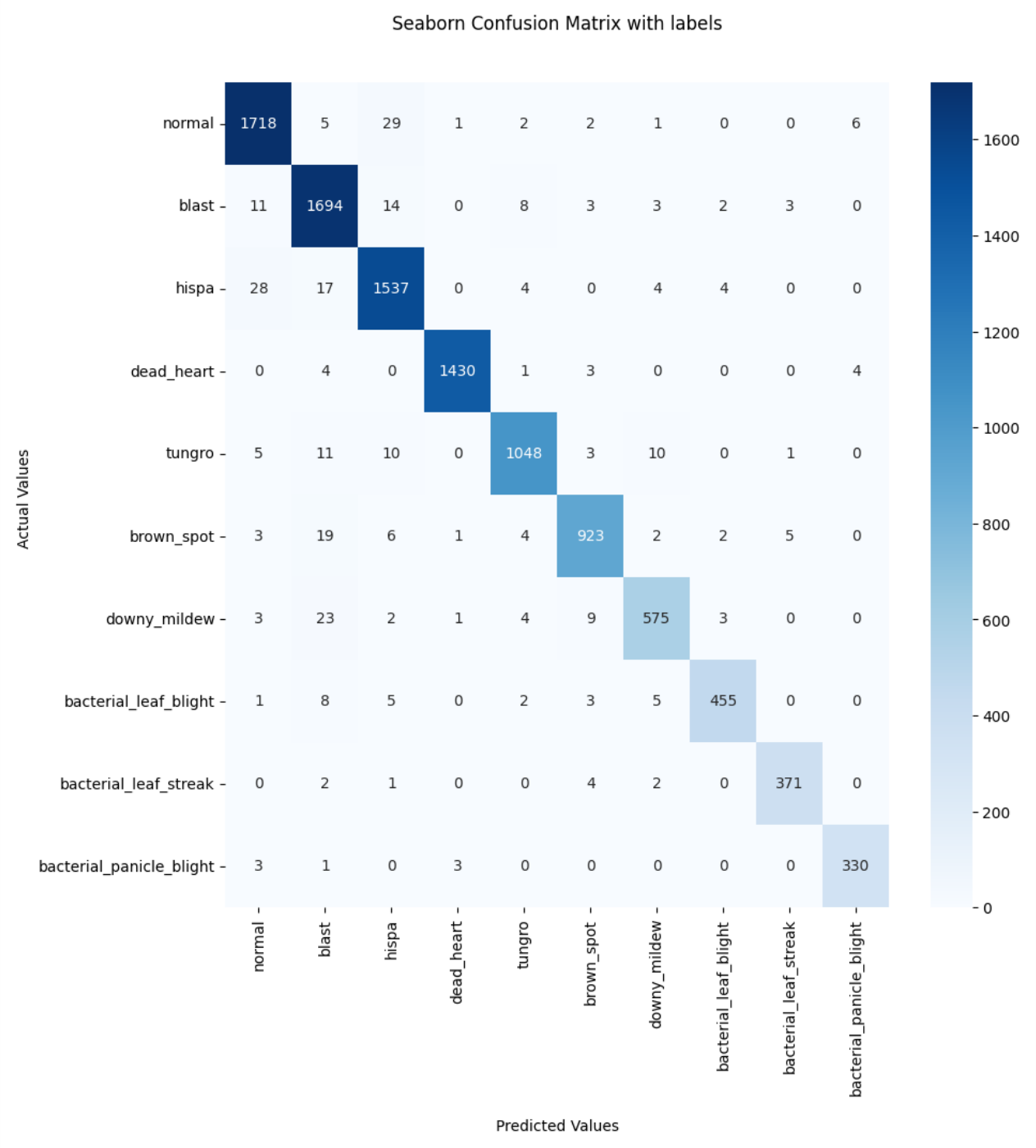
Also to estimate the quality of classification, we can use the confusion matrix:

[26]:
```
cf_matrix = confusion_matrix(preds['label'].map(automl.reader.target_mapping),
                             OOFs)

plt.figure(figsize = (10, 10))

ax = sns.heatmap(cf_matrix, annot=True, cmap='Blues', fmt = 'd')

ax.set_title('Seaborn Confusion Matrix with labels\n\n');
ax.set_xlabel('\nPredicted Values')
ax.set_ylabel('Actual Values ');

inverse_target_mapping = {y: x for x,y in automl.reader.target_mapping.items()}
labels = [inverse_target_mapping[i] for i in range(len(inverse_target_mapping))]
ax.xaxis.set_ticklabels(labels, rotation = 90)
```

(continues on next page)

(continued from previous page)

```
ax.yaxis.set_ticklabels(labels, rotation = 0)

plt.show()
```



Predict for test dataset

Now we are also ready to predict for our test competition dataset and submission file creation:

[27]:
```
%%time

te_pred = automl.predict(submission)
print(f'Prediction for te_data:\n{te_pred}\nShape = {te_pred.shape}')
```

```
100%|| 163/163 [01:28<00:00,  1.84it/s]
```

```
[14:28:22] Feature path transformed
Prediction for te_data:
array([[1.57098308e-01, 2.81519257e-03, 5.96348643e-01, ...,
        1.08084995e-02, 1.95845146e-07, 1.42198633e-05],
       [9.83384371e-01, 6.52049668e-04, 1.45791359e-02, ...,
        1.12365209e-03, 9.75986836e-07, 1.95965598e-07],
       [1.68020770e-01, 3.79674375e-01, 1.86414778e-01, ...,
        1.67078048e-03, 1.21877249e-03, 3.75247910e-03],
       ...,
       [1.05072348e-03, 1.24680300e-05, 5.70231769e-03, ...,
        4.37476301e-05, 1.52421890e-07, 1.81421214e-07],
       [6.52685121e-04, 4.47798493e-06, 5.04824053e-03, ...,
        2.13344283e-05, 1.52417726e-07, 1.62638599e-07],
       [1.57185504e-03, 1.01540554e-05, 2.53849756e-02, ...,
        1.17763964e-04, 1.52426963e-07, 1.77946404e-07]], dtype=float32)
Shape = (20814, 10)
CPU times: user 55.8 s, sys: 21.6 s, total: 1min 17s
Wall time: 2min 19s
```

[28]:
```
sub = submission[['image_id']]
for i in range(10):
    sub['pred_' + str(i)] = te_pred.data[:,i]

sub
```

```
/tmp/ipykernel_12895/1185757098.py:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_
↪guide/indexing.html#returning-a-view-versus-a-copy
  sub['pred_' + str(i)] = te_pred.data[:,i]
```

[28]:

	image_id	pred_0	pred_1	pred_2	pred_3	pred_4	\
0	200001.jpg	0.157098	0.002815	0.596349	0.020590	1.148577e-01	
1	200002.jpg	0.983384	0.000652	0.014579	0.000139	6.825896e-05	
2	200003.jpg	0.168021	0.379674	0.186415	0.000225	1.850213e-03	
3	200004.jpg	0.000013	0.990730	0.008530	0.000097	1.116415e-04	
4	200005.jpg	0.000340	0.999536	0.000031	0.000002	6.857538e-07	
...	...	...	...	...	...	...	
20809	203469.jpg	0.003061	0.000017	0.041731	0.943745	1.648944e-04	
20810	203469.jpg	0.000430	0.000003	0.002508	0.993409	2.613632e-05	
20811	203469.jpg	0.001051	0.000012	0.005702	0.989972	5.734707e-05	
20812	203469.jpg	0.000653	0.000004	0.005048	0.990724	3.223727e-05	
20813	203469.jpg	0.001572	0.000010	0.025385	0.965282	1.030424e-04	

(continues on next page)

(continued from previous page)

```
         pred_5    pred_6    pred_7        pred_8        pred_9
0      0.095614  0.001854  0.010808  1.958451e-07  1.421986e-05
1      0.000044  0.000008  0.001124  9.759868e-07  1.959656e-07
2      0.036919  0.220253  0.001671  1.218772e-03  3.752479e-03
3      0.000215  0.000111  0.000037  1.548404e-04  1.946677e-07
4      0.000003  0.000007  0.000029  5.404088e-07  4.985940e-05
...         ...       ...       ...           ...           ...
20809  0.010877  0.000146  0.000258  1.524480e-07  2.509265e-07
20810  0.003580  0.000007  0.000036  1.524176e-07  1.595918e-07
20811  0.003144  0.000018  0.000044  1.524219e-07  1.814212e-07
20812  0.003505  0.000012  0.000021  1.524177e-07  1.626386e-07
20813  0.007472  0.000058  0.000118  1.524270e-07  1.779464e-07

[20814 rows x 11 columns]
```

[29]:
```
sub = sub.groupby(['image_id']).mean().reset_index()
sub
```

[29]:
```
        image_id    pred_0    pred_1    pred_2    pred_3    pred_4    pred_5  \
0     200001.jpg  0.127650  0.001409  0.599914  0.017568  0.136898  0.106915
1     200002.jpg  0.937035  0.000638  0.060420  0.000098  0.000105  0.000096
2     200003.jpg  0.120163  0.523312  0.106169  0.000473  0.000748  0.042688
3     200004.jpg  0.000020  0.888623  0.006415  0.001150  0.000430  0.004390
4     200005.jpg  0.000680  0.998898  0.000085  0.000009  0.000001  0.000002
...          ...       ...       ...       ...       ...       ...       ...
3464  203465.jpg  0.000224  0.002143  0.001514  0.990281  0.002657  0.000401
3465  203466.jpg  0.250769  0.007148  0.741840  0.000002  0.000022  0.000013
3466  203467.jpg  0.960745  0.004105  0.001135  0.000646  0.016724  0.008584
3467  203468.jpg  0.003675  0.001097  0.038018  0.000038  0.000483  0.000310
3468  203469.jpg  0.001372  0.000012  0.015432  0.977533  0.000086  0.005415

        pred_6    pred_7        pred_8        pred_9
0     0.001796  0.007829  8.801418e-06  1.216593e-05
1     0.000016  0.001586  6.087082e-06  2.249314e-07
2     0.201373  0.002807  1.389023e-03  8.788786e-04
3     0.000616  0.001799  9.654120e-02  1.466518e-05
4     0.000021  0.000172  1.743805e-06  1.304403e-04
...        ...       ...           ...           ...
3464  0.001074  0.000134  1.530934e-03  4.091801e-05
3465  0.000076  0.000129  2.629060e-07  2.120279e-07
3466  0.000062  0.007749  2.438365e-04  6.326832e-06
3467  0.000223  0.000208  9.551883e-01  7.596347e-04
3468  0.000046  0.000104  1.524300e-07  1.962799e-07

[3469 rows x 11 columns]
```

[30]:
```
TEs = pd.Series(np.argmax(sub[['pred_' + str(i) for i in range(10)]].values, axis = 1)).
↪map(inverse_target_mapping)
TEs
```

[30]:
```
0                       hispa
1                      normal
2                       blast
3                       blast
4                       blast
                ...
3464               dead_heart
3465                    hispa
3466                   normal
3467    bacterial_leaf_streak
3468               dead_heart
Length: 3469, dtype: object
```

[31]:
```
sub['label'] = TEs
sub[['image_id', 'label']].to_csv('LightAutoML_TabularCVAutoML_with_aug.csv', index =␣
↪False)
sub[['image_id', 'label']]
```

[31]:
```
        image_id                  label
0     200001.jpg                  hispa
1     200002.jpg                 normal
2     200003.jpg                  blast
3     200004.jpg                  blast
4     200005.jpg                  blast
...          ...                    ...
3464  203465.jpg             dead_heart
3465  203466.jpg                  hispa
3466  203467.jpg                 normal
3467  203468.jpg  bacterial_leaf_streak
3468  203469.jpg             dead_heart

[3469 rows x 2 columns]
```

Now we can choose another model from `timm`. So we will use `tf_efficientnetv2_b0.in1k`, by default it uses `vit_base_patch16_224.augreg_in21k`

[35]:
```
automl = TabularCVAutoML(task = task,
                         timeout=5 * 3600,
                         autocv_features={"embed_model": 'timm/tf_efficientnetv2_b0.in1k
↪'},
                        cpu_limit = 2,
                        reader_params = {'cv': 5, 'random_state': 42})
```

[36]:
```
%%time

oof_pred = automl.fit_predict(train_data, roles = roles, verbose = 3)
```

```
[14:37:43] Stdout logging level is INFO3.
[14:37:43] Task: multiclass

[14:37:43] Start automl preset with listed constraints:
[14:37:43] - time: 18000.00 seconds
```

(continues on next page)

(continued from previous page)

```
[14:37:43] - CPU: 2 cores
[14:37:43] - memory: 16 GB

[14:37:43] Train data shape: (114477, 5)

[14:37:43] Layer 1 train process start. Time left 17999.80 secs

Downloading model.safetensors:   0%|          | 0.00/28.8M [00:00<?, ?B/s]

100%|| 895/895 [06:43<00:00,  2.22it/s]

[14:44:31] Feature path transformed
[14:44:41] Start fitting Lvl_0_Pipe_0_Mod_0_LinearL2 ...
[14:44:41] ===== Start working with fold 0 for Lvl_0_Pipe_0_Mod_0_LinearL2 =====
[14:44:53] Linear model: C = 1e-05 score = -1.2282992628176856
[14:45:04] Linear model: C = 5e-05 score = -0.9078946864858105
[14:45:14] Linear model: C = 0.0001 score = -0.7903223383077203
[14:45:25] Linear model: C = 0.0005 score = -0.5805263796419443
[14:45:37] Linear model: C = 0.001 score = -0.5191830537228186
[14:45:48] Linear model: C = 0.005 score = -0.442378006077887 24
[14:46:01] Linear model: C = 0.01 score = -0.4332587963951451
[14:46:16] Linear model: C = 0.05 score = -0.4659824021930572
[14:46:28] Linear model: C = 0.1 score = -0.49696980356910764
[14:46:29] ===== Start working with fold 1 for Lvl_0_Pipe_0_Mod_0_LinearL2 =====
[14:46:40] Linear model: C = 1e-05 score = -1.1941203869888553
[14:46:50] Linear model: C = 5e-05 score = -0.870315687726058
[14:47:00] Linear model: C = 0.0001 score = -0.7542737074009194
[14:47:11] Linear model: C = 0.0005 score = -0.5565397834768919
[14:47:23] Linear model: C = 0.001 score = -0.5021799803891854
[14:47:37] Linear model: C = 0.005 score = -0.4375446715586552
[14:47:49] Linear model: C = 0.01 score = -0.4337117229695793
[14:48:03] Linear model: C = 0.05 score = -0.47678539878379567
[14:48:16] Linear model: C = 0.1 score = -0.5100193461879381
[14:48:16] ===== Start working with fold 2 for Lvl_0_Pipe_0_Mod_0_LinearL2 =====
[14:48:27] Linear model: C = 1e-05 score = -1.1828501053814764
[14:48:39] Linear model: C = 5e-05 score = -0.8603329618510173
[14:48:48] Linear model: C = 0.0001 score = -0.7451147263666518
[14:48:59] Linear model: C = 0.0005 score = -0.5469582228988039
[14:49:12] Linear model: C = 0.001 score = -0.49160247842297417
[14:49:24] Linear model: C = 0.005 score = -0.4257572256164155
[14:49:37] Linear model: C = 0.01 score = -0.4188241529929714
[14:49:50] Linear model: C = 0.05 score = -0.4522382557188784
[14:50:03] Linear model: C = 0.1 score = -0.48277984079191094
[14:50:04] ===== Start working with fold 3 for Lvl_0_Pipe_0_Mod_0_LinearL2 =====
[14:50:15] Linear model: C = 1e-05 score = -1.1958343845422246
[14:50:26] Linear model: C = 5e-05 score = -0.878725101433787
[14:50:35] Linear model: C = 0.0001 score = -0.7660166437189271
[14:50:45] Linear model: C = 0.0005 score = -0.5679153687919936
[14:50:59] Linear model: C = 0.001 score = -0.5110457138416219
[14:51:10] Linear model: C = 0.005 score = -0.44229320617124224
[14:51:23] Linear model: C = 0.01 score = -0.43663952743918066
[14:51:37] Linear model: C = 0.05 score = -0.47363171137894655
[14:51:51] Linear model: C = 0.1 score = -0.5032655687259646
```

(continues on next page)

(continued from previous page)

```
[14:51:51] ===== Start working with fold 4 for Lvl_0_Pipe_0_Mod_0_LinearL2 =====
[14:52:02] Linear model: C = 1e-05 score = -1.1804715353776323
[14:52:13] Linear model: C = 5e-05 score = -0.8529105474280552
[14:52:21] Linear model: C = 0.0001 score = -0.7373622302487922
[14:52:32] Linear model: C = 0.0005 score = -0.537561225715503
[14:52:43] Linear model: C = 0.001 score = -0.48106564988541606
[14:52:57] Linear model: C = 0.005 score = -0.4138154861612588
[14:53:09] Linear model: C = 0.01 score = -0.40990101492044817
[14:53:23] Linear model: C = 0.05 score = -0.44904189928940963
[14:53:36] Linear model: C = 0.1 score = -0.4789966864522385
[14:53:36] Fitting Lvl_0_Pipe_0_Mod_0_LinearL2 finished. score = -0.4264683916927181
[14:53:36] Lvl_0_Pipe_0_Mod_0_LinearL2 fitting and predicting completed
[14:53:36] Time left 17046.53 secs

[14:58:02] Start fitting Lvl_0_Pipe_1_Mod_0_CatBoost ...
[14:58:02] ===== Start working with fold 0 for Lvl_0_Pipe_1_Mod_0_CatBoost =====
[14:58:02] 0:   learn: 2.2636799        test: 2.2649651 best: 2.2649651 (0)     total:␣
↪10.4ms   remaining: 41.6s
[14:58:22] bestTest = 0.2436411292
[14:58:22] bestIteration = 3999
[14:58:23] ===== Start working with fold 1 for Lvl_0_Pipe_1_Mod_0_CatBoost =====
[14:58:23] 0:   learn: 2.2634693        test: 2.2632523 best: 2.2632523 (0)     total: 6.
↪07ms   remaining: 24.3s
[14:58:43] bestTest = 0.2658199756
[14:58:43] bestIteration = 3999
[14:58:43] ===== Start working with fold 2 for Lvl_0_Pipe_1_Mod_0_CatBoost =====
[14:58:44] 0:   learn: 2.2631659        test: 2.2656305 best: 2.2656305 (0)     total: 6.
↪52ms   remaining: 26.1s
[14:59:03] bestTest = 0.2753673959
[14:59:03] bestIteration = 3999
[14:59:04] ===== Start working with fold 3 for Lvl_0_Pipe_1_Mod_0_CatBoost =====
[14:59:04] 0:   learn: 2.2645703        test: 2.2657044 best: 2.2657044 (0)     total: 6.
↪13ms   remaining: 24.5s
[14:59:24] bestTest = 0.2738942971
[14:59:24] bestIteration = 3996
[14:59:24] Shrink model to first 3997 iterations.
[14:59:24] ===== Start working with fold 4 for Lvl_0_Pipe_1_Mod_0_CatBoost =====
[14:59:25] 0:   learn: 2.2642798        test: 2.2644247 best: 2.2644247 (0)     total: 5.
↪95ms   remaining: 23.8s
[14:59:44] bestTest = 0.2538460547
[14:59:44] bestIteration = 3999
[14:59:45] Fitting Lvl_0_Pipe_1_Mod_0_CatBoost finished. score = -0.2625123265864018
[14:59:45] Lvl_0_Pipe_1_Mod_0_CatBoost fitting and predicting completed
[14:59:45] Time left 16678.32 secs

[14:59:45] Layer 1 training completed.

[14:59:45] Blending: optimization starts with equal weights and score -0.2561708318332855
/home/dvladimirvasilyev/anaconda3/envs/myenv/lib/python3.8/site-packages/sklearn/metrics/
↪_classification.py:2916: UserWarning: The y_pred values do not sum to one. Starting␣
↪from 1.5 thiswill result in an error.
  warnings.warn(
```

```
[14:59:45] Blending: iteration 0: score = -0.23692344794948073, weights = [0.19089036 0.
↪8091096 ]
[14:59:46] Blending: iteration 1: score = -0.23692344794948073, weights = [0.19089036 0.
↪8091096 ]
[14:59:46] Blending: no score update. Terminated

[14:59:46] Automl preset training completed in 1323.26 seconds

[14:59:46] Model description:
Final prediction for new objects (level 0) =
         0.19089 * (5 averaged models Lvl_0_Pipe_0_Mod_0_LinearL2) +
         0.80911 * (5 averaged models Lvl_0_Pipe_1_Mod_0_CatBoost)

CPU times: user 20min 56s, sys: 1min 25s, total: 22min 22s
Wall time: 22min 3s
```

[37]:
```
%%time

te_pred = automl.predict(submission)
print(f'Prediction for te_data:\n{te_pred}\nShape = {te_pred.shape}')
```

```
100%|| 163/163 [01:16<00:00,  2.13it/s]
[15:01:03] Feature path transformed
Prediction for te_data:
array([[5.8534566e-02, 6.8576052e-03, 4.5334366e-01, ..., 1.5735241e-02,
        4.2415738e-07, 2.8625556e-05],
       [9.6386713e-01, 1.4697504e-03, 3.2047924e-02, ..., 2.0407902e-03,
        6.7228694e-07, 1.4319470e-07],
       [3.5120246e-01, 2.9431397e-01, 1.9644174e-01, ..., 2.2667376e-04,
        6.4593733e-06, 5.9228983e-05],
       ...,
       [2.3565248e-03, 2.7670001e-05, 1.2790265e-02, ..., 9.7831573e-05,
        4.5524594e-08, 1.1057142e-07],
       [1.4637065e-03, 9.7479615e-06, 1.1323140e-02, ..., 4.7557736e-05,
        4.5515264e-08, 6.8441139e-08],
       [3.5254466e-03, 2.2519611e-05, 5.6939691e-02, ..., 2.6386546e-04,
        4.5536019e-08, 1.7701051e-07]], dtype=float32)
Shape = (20814, 10)
CPU times: user 13 s, sys: 3.3 s, total: 16.3 s
Wall time: 2min 7s
```

Our submission has 0.95770 accuracy on public and 0.95276 accuracy on private leaderboard (Alexander Ryzhkov account).

Additional materials

- Official LightAutoML github repo
- LightAutoML documentation
- LightAutoML tutorials
- LightAutoML course:
  - Part 1 - general overview

  - Part 2 - LightAutoML specific applications
  - Part 3 - LightAutoML customization
- OpenDataScience AutoML benchmark leaderboard

2.1.9 Tutorial 9: Neural Networks

Official LightAutoML github repository is here

In this tutorial you will learn how to: * train neural networks (nn) with LightAutoML on tabualr data * customize model architecture and pipelines

0. Prerequisites

0.0 install LightAutoML

[1]:
```
# !pip install -U lightautoml[all]
```

0.1 Import libraries

Here we will import the libraries we use in this kernel: - Standard python libraries for timing, working with OS etc. - Essential python DS libraries like numpy, pandas, scikit-learn and torch (the last we will use in the next cell) - LightAutoML modules: presets for AutoML, task and report generation module

[2]:
```
# Standard python libraries
import os

# Essential DS libraries
import optuna
import requests
import numpy as np
import pandas as pd
from sklearn.metrics import roc_auc_score
from sklearn.model_selection import train_test_split
import torch
from copy import deepcopy as copy
import torch.nn as nn
from collections import OrderedDict

# LightAutoML presets, task and report generation
from lightautoml.automl.presets.tabular_presets import TabularAutoML
from lightautoml.tasks import Task
```

0.2 Constants

Here we setup the constants to use in the kernel: - `N_THREADS` - number of vCPUs for LightAutoML model creation - `N_FOLDS` - number of folds in LightAutoML inner CV - `RANDOM_STATE` - random seed for better reproducibility - `TEST_SIZE` - houldout data part size - `TIMEOUT` - limit in seconds for model to train - `TARGET_NAME` - target column name in dataset

[3]:
```
N_THREADS = 4
N_FOLDS = 5
RANDOM_STATE = 42
TEST_SIZE = 0.2
TIMEOUT = 300
TARGET_NAME = 'TARGET'

np.random.seed(RANDOM_STATE)
torch.set_num_threads(N_THREADS)
```

0.3 Data loading

[4]:
```
DATASET_DIR = '../data/'
DATASET_NAME = 'sampled_app_train.csv'
DATASET_FULLNAME = os.path.join(DATASET_DIR, DATASET_NAME)
DATASET_URL = 'https://raw.githubusercontent.com/AILab-MLTools/LightAutoML/master/
↪examples/data/sampled_app_train.csv'

if not os.path.exists(DATASET_FULLNAME):
    os.makedirs(DATASET_DIR, exist_ok=True)

    dataset = requests.get(DATASET_URL).text
    with open(DATASET_FULLNAME, 'w') as output:
        output.write(dataset)

data = pd.read_csv(DATASET_FULLNAME)
data.head()

tr_data, te_data = train_test_split(
    data,
    test_size=TEST_SIZE,
    stratify=data[TARGET_NAME],
    random_state=RANDOM_STATE
)
```

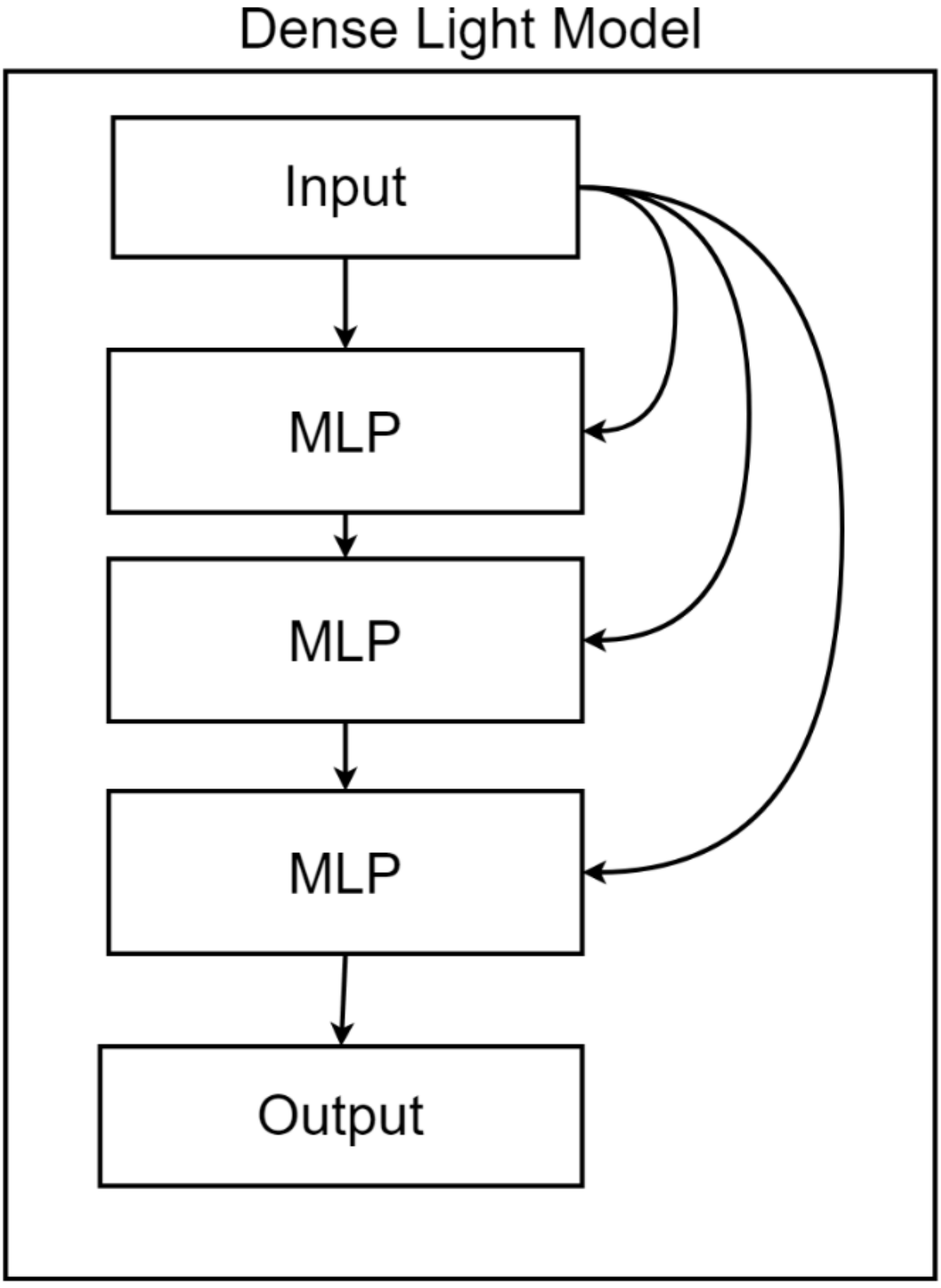
1. Available built-in models

To use different model pass it to the list in "`use_algo`". We support custom models inherited from `torch.nn.Module` class. For every model their parameters is listed below.

1.1 MLP ("mlp")

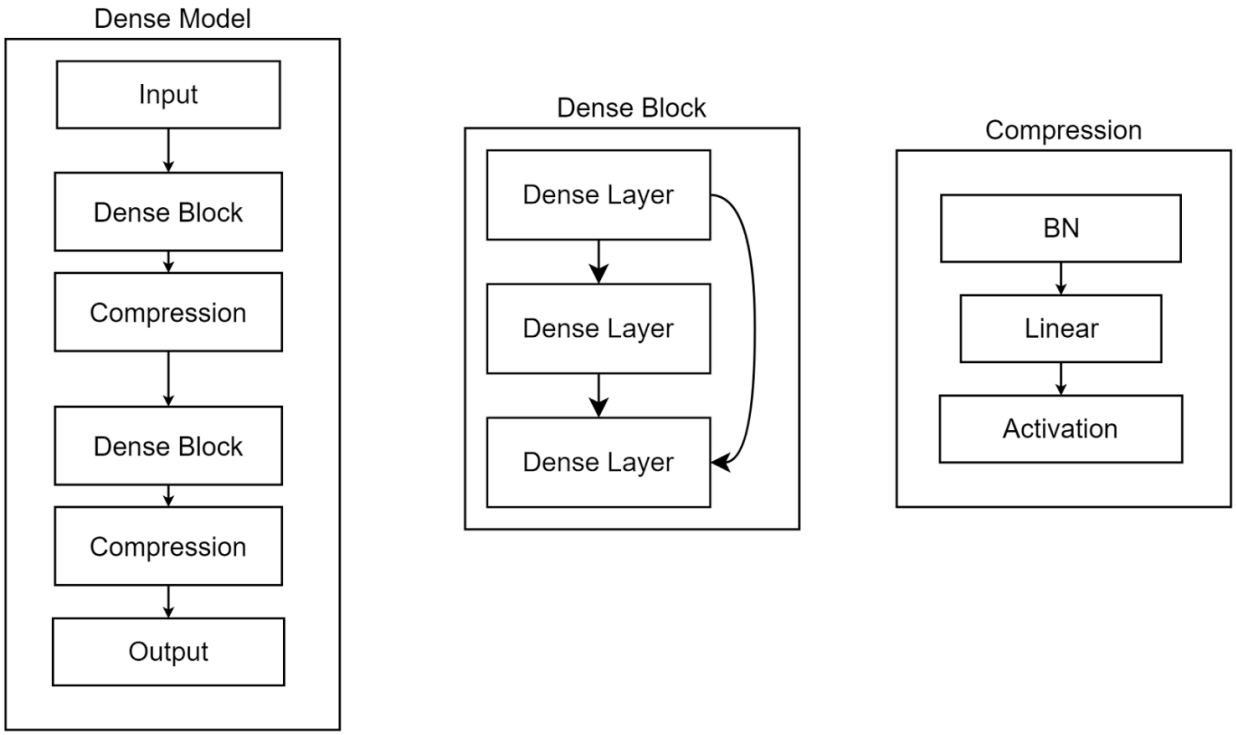
- `hidden_size` - define hidden layer dimensions

1.2 Dense Light ("denselight")



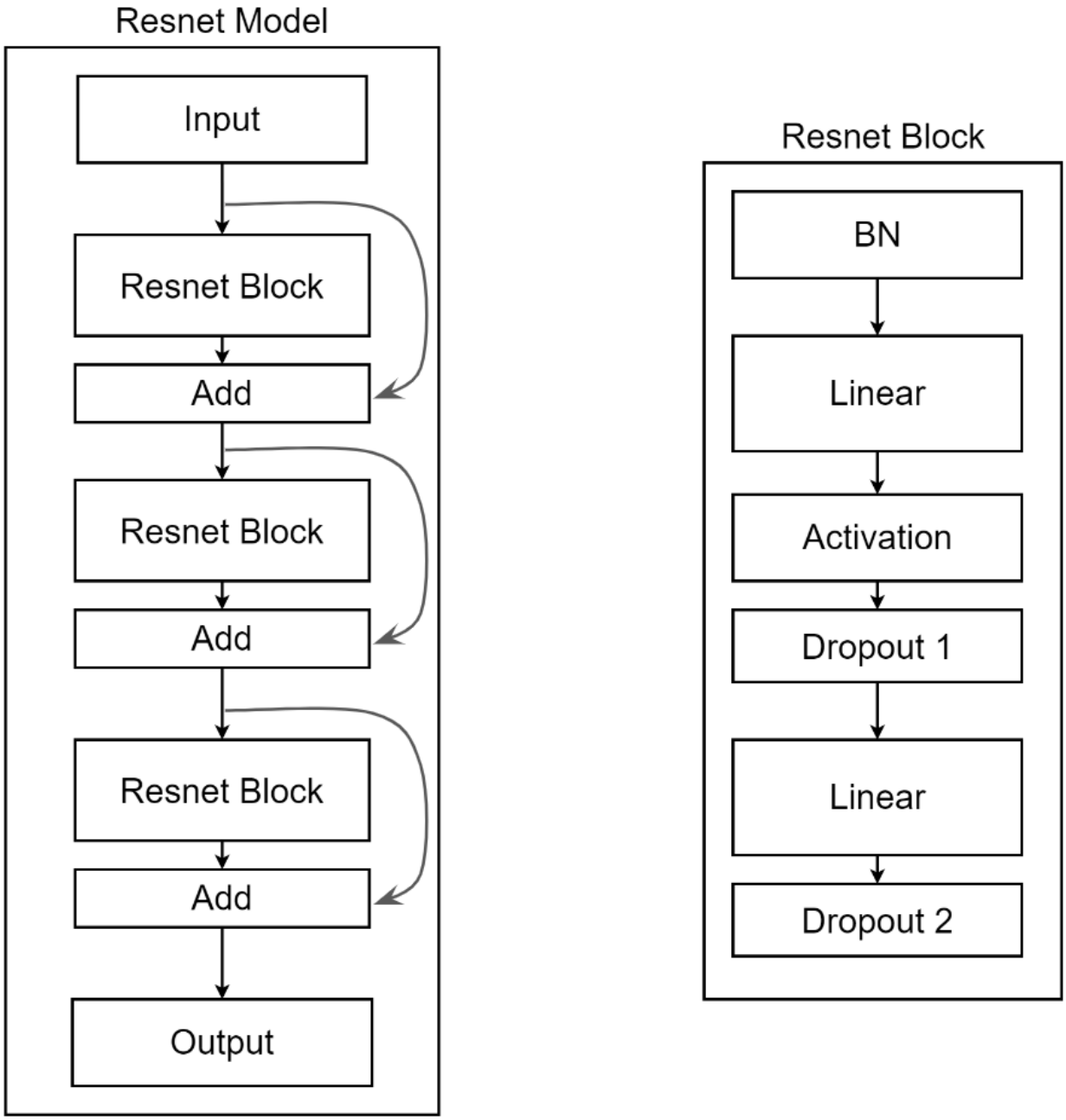
- `hidden_size` - define hidden layer dimensions

1.3 Dense (`"dense"`)



- `block_config` - set number of blocks and layers within each block
- `compression` - portion of neuron to drop after `DenseBlock`
- `growth_size` - output dim of every `DenseLayer`
- `bn_factor` - size of intermediate fc is increased times this factor in layer

1.4 Resnet (`"resnet"`)

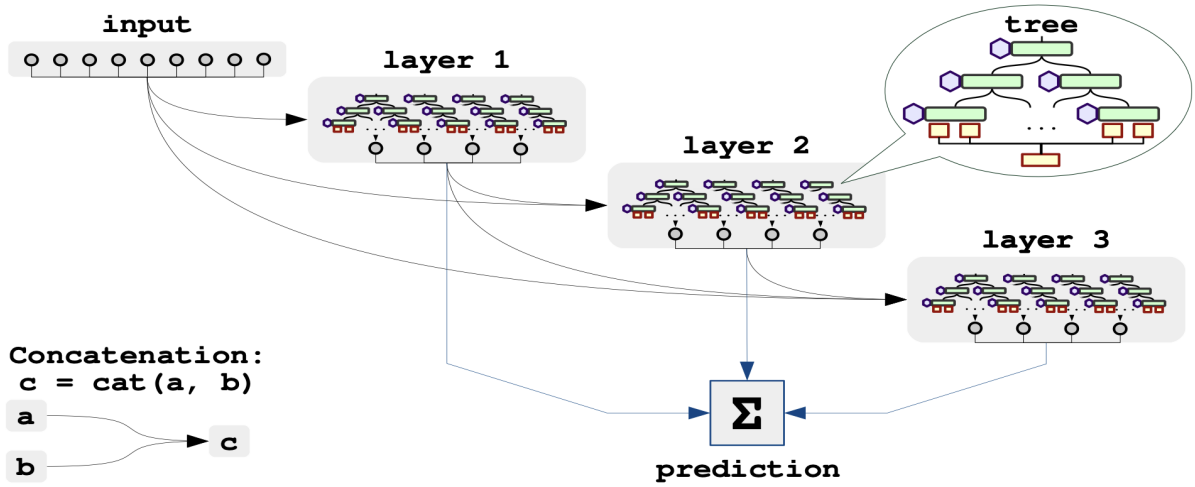


- `hid_factor` - size of intermediate fc is increased times this factor in layer

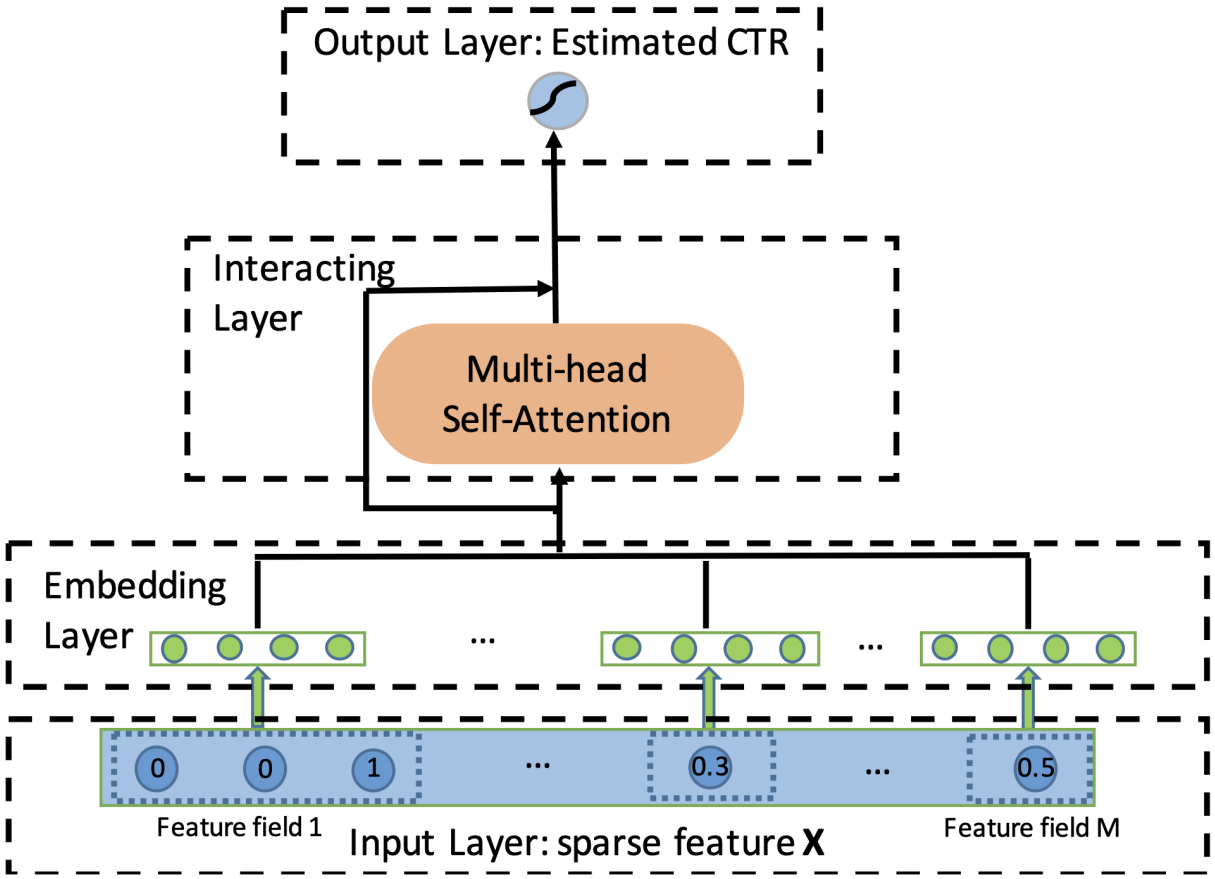
1.5 SNN (`"snn"`)

- `hidden_size` - define hidden layer dimensions

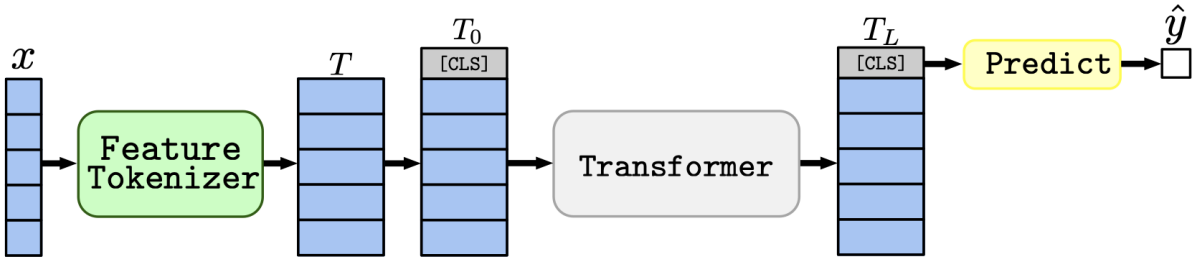
1.5 NODE ("node")



1.5 AutoInt ("autoint")



1.5 FTTransformer ("fttransformer")



- `pooling` - Pooling used for the last step.
- `n_out` - Output dimension, 1 for binary prediction.
- `embedding_size` - Embeddings size.
- `depth` - Number of Attention Blocks inside Transformer.
- `heads` - Number of heads in Attention.
- `attn_dropout` - Post-Attention dropout.
- `ff_dropout` - Feed-Forward Dropout.
- `dim_head` - Attention head dimension
- `return_attn` - Return attention scores or not.
- `num_enc_layers` - Number of Transformer layers.
- `device` - Device to compute on.

2. Example of usage

2.1 Task definition

[5]:
```
task = Task('binary')
roles = {
    'target': TARGET_NAME,
    'drop': ['SK_ID_CURR']
}
```

2.2 LightAutoML model creation - TabularAutoML preset with neural network

In next the cell we are going to create LightAutoML model with `TabularAutoML` class.

in just several lines. Let's discuss the params we can setup: - `task` - the type of the ML task (the only **must have** parameter) - `timeout` - time limit in seconds for model to train - `cpu_limit` - vCPU count for model to use - `nn_params` - network and training params, for example, `"hidden_size"`, `"batch_size"`, `"lr"`, etc. - `nn_pipeline_params` - data preprocessing params, which affect how data is fed to the model: use embeddings or target encoding for categorical columns, standard scalar or quantile transformer for numerical columns - `reader_params` - parameter change for Reader object inside preset, which works on the first step of data preparation: automatic feature typization, preliminary almost-constant features, correct CV setup etc.

[6]:
```
automl = TabularAutoML(
    task = task,
    timeout = TIMEOUT,
```

(continues on next page)

(continued from previous page)

```
    cpu_limit = N_THREADS,
    general_params = {"use_algos": [["mlp"]]}, # ['nn', 'mlp', 'dense', 'denselight', 'resnet',
↪'snn', 'node', 'autoint', 'fttransformer'] or custom torch model
    nn_params = {"n_epochs": 10, "bs": 512, "num_workers": 0, "path_to_save": None,
↪"freeze_defaults": True},
    nn_pipeline_params = {"use_qnt": True, "use_te": False},
    reader_params = {'n_jobs': N_THREADS, 'cv': N_FOLDS, 'random_state': RANDOM_STATE}
)
```

2.3 AutoML training

To run autoML training use fit_predict method:

- `train_data` - Dataset to train.
- `roles` - Roles dict.
- `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;

Note: out-of-fold prediction is calculated during training and returned from the fit_predict method

[7]:
```
%%time
oof_pred = automl.fit_predict(tr_data, roles = roles, verbose = 1)
```

```
[14:56:04] Stdout logging level is INFO.
[14:56:04] Copying TaskTimer may affect the parent PipelineTimer, so copy will create␣
↪new unlimited TaskTimer
[14:56:04] Task: binary

[14:56:04] Start automl preset with listed constraints:
[14:56:04] - time: 300.00 seconds
[14:56:04] - CPU: 4 cores
[14:56:04] - memory: 16 GB

[14:56:04] Train data shape: (8000, 122)

[14:56:08] Layer 1 train process start. Time left 296.45 secs
[14:56:08] Start fitting Lvl_0_Pipe_0_Mod_0_TorchNN_mlp_0 ...
[14:56:15] Fitting Lvl_0_Pipe_0_Mod_0_TorchNN_mlp_0 finished. score = 0.6035621265821923
[14:56:15] Lvl_0_Pipe_0_Mod_0_TorchNN_mlp_0 fitting and predicting completed
[14:56:15] Time left 289.10 secs

[14:56:15] Layer 1 training completed.

[14:56:15] Automl preset training completed in 10.90 seconds

[14:56:15] Model description:
Final prediction for new objects (level 0) =
         1.00000 * (5 averaged models Lvl_0_Pipe_0_Mod_0_TorchNN_mlp_0)

CPU times: user 10.9 s, sys: 822 ms, total: 11.8 s
```

(continues on next page)

(continued from previous page)

```
Wall time: 10.9 s
```

2.4 Prediction on holdout and model evaluation

[8]:
```
%%time

te_pred = automl.predict(te_data)
print(f'Prediction for te_data:\n{te_pred}\nShape = {te_pred.shape}')
```

```
Prediction for te_data:
array([[0.09815434],
       [0.08660936],
       [0.060364  ],
       ...,
       [0.09103375],
       [0.05593849],
       [0.09817966]], dtype=float32)
Shape = (2000, 1)
CPU times: user 1.39 s, sys: 59.4 ms, total: 1.45 s
Wall time: 1.35 s
```

[9]:
```
print(f'OOF score: {roc_auc_score(tr_data[TARGET_NAME].values, oof_pred.data[:, 0])}')
print(f'HOLDOUT score: {roc_auc_score(te_data[TARGET_NAME].values, te_pred.data[:, 0])}')
```

```
OOF score: 0.6035621265821923
HOLDOUT score: 0.5970482336956522
```

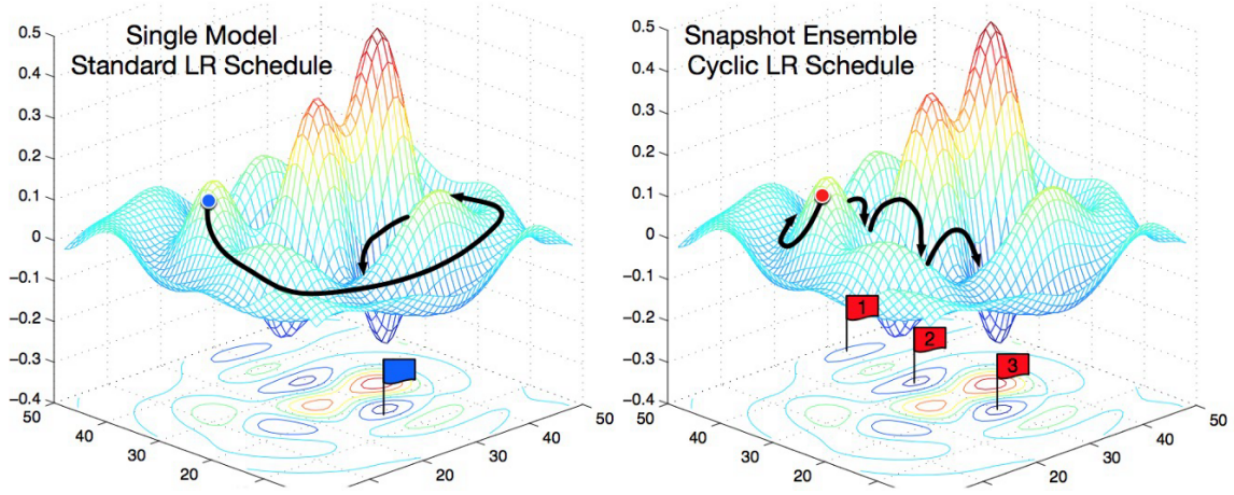
You can obtain the description of the resulting pipeline:

[10]:
```
print(automl.create_model_str_desc())
```

```
Final prediction for new objects (level 0) =
         1.00000 * (5 averaged models Lvl_0_Pipe_0_Mod_0_TorchNN_mlp_0)
```

3. Main training loop and pipeline params

3.1 Training loop params



- `bs` - batch_size
- `snap_params` - early stopping and checkpoint averaging params, stochastic weight averaging (swa)
- `opt` - lr optimizer
- `opt_params` - optimizer params
- `clip_grad` - use grad clipping for regularization
- `clip_grad_params`
- `emb_dropout` - embedding dropout for categorical columns

This set of params should be passed in `nn_params` as well.

3.2 Pipeline params

Transformation for numerical columns

- `use_qnt` - uses quantile transformation for numerical columns
- `output_distribution` - type of distribuiton of feature after qnt transformer
- `n_quantiles` - number of quantiles used to build feature distribution
- `qnt_factor` - decreses `n_quantiles` depending on train data shape

Transformation for categorical columns

- `use_te` - uses target encoding
- `top_intersections` - number of intersections of cat columns to use

Full list of default parametres you can find here: - nn_params - nn_pipeline_params

4. More use cases

Let's remember default Lama params to be more compact.

[11]:
```
default_lama_params = {
    "task": task,
    "timeout": TIMEOUT,
    "cpu_limit": N_THREADS,
    "reader_params": {'n_jobs': N_THREADS, 'cv': N_FOLDS, 'random_state': RANDOM_STATE}
}

default_nn_params = {
    "bs": 512, "num_workers": 0, "path_to_save": None, "n_epochs": 10, "freeze_defaults":
↪ True
}
```

4.1 Custom model

Consider simple neural network that we want to train.

[12]:
```
class SimpleNet(nn.Module):
    def __init__(
        self,
        n_in,
        n_out,
```

(continues on next page)

(continued from previous page)

```
        hidden_size,
        drop_rate,
        **kwargs, # kwargs is must-have to hold unnecessary parameters
    ):
        super(SimpleNet, self).__init__()
        self.features = nn.Sequential(OrderedDict([]))

        self.features.add_module("norm", nn.BatchNorm1d(n_in))
        self.features.add_module("dense1", nn.Linear(n_in, hidden_size))
        self.features.add_module("act", nn.SiLU())
        self.features.add_module("dropout", nn.Dropout(p=drop_rate))
        self.features.add_module("dense2", nn.Linear(hidden_size, n_out))

    def forward(self, x):
        """
        Args:
            x: data after feature pipeline transformation
            (by default concatenation of columns)
        """
        for layer in self.features:
            x = layer(x)
        return x
```

[13]:
```
automl = TabularAutoML(
    **default_lama_params,
    general_params={"use_algos": [[SimpleNet]]},
    nn_params={
        **default_nn_params,
        "hidden_size": 256,
        "drop_rate": 0.1
    },
)
automl.fit_predict(tr_data, roles=roles, verbose=1)
```

```
[14:56:17] Stdout logging level is INFO.
[14:56:17] Task: binary

[14:56:17] Start automl preset with listed constraints:
[14:56:17] - time: 300.00 seconds
[14:56:17] - CPU: 4 cores
[14:56:17] - memory: 16 GB

[14:56:17] Train data shape: (8000, 122)

[14:56:17] Layer 1 train process start. Time left 299.22 secs
[14:56:18] Start fitting Lvl_0_Pipe_0_Mod_0_TorchNN_0 ...
[14:56:23] Fitting Lvl_0_Pipe_0_Mod_0_TorchNN_0 finished. score = 0.70579837612218
[14:56:23] Lvl_0_Pipe_0_Mod_0_TorchNN_0 fitting and predicting completed
[14:56:23] Time left 293.15 secs

[14:56:23] Layer 1 training completed.
```

(continues on next page)

(continued from previous page)

```
[14:56:23] Automl preset training completed in 6.86 seconds

[14:56:23] Model description:
Final prediction for new objects (level 0) =
         1.00000 * (5 averaged models Lvl_0_Pipe_0_Mod_0_TorchNN_0)
```

[13]:
```
array([[0.04888836],
       [0.02840128],
       [0.04246276],
       ...,
       [0.05778075],
       [0.17132443],
       [0.20606528]], dtype=float32)
```

4.1.1 Define the pipeline by yourself

[14]:
```
from typing import Sequence
from typing import Dict
from typing import Optional
from typing import Any
from typing import Callable
from typing import Union


class CatEmbedder(nn.Module):
    """Category data model.

    Args:
        cat_dims: Sequence with number of unique categories
            for category features
    """

    def __init__(
        self,
        cat_dims: Sequence[int],
        **kwargs
    ):
        super(CatEmbedder, self).__init__()
        emb_dims = [
            (int(x), 5)
            for x in cat_dims
        ]
        self.no_of_embs = sum([y for x, y in emb_dims])
        self.emb_layers = nn.ModuleList([nn.Embedding(x, y) for x, y in emb_dims])

    def get_out_shape(self) -> int:
        """Output shape.

        Returns:
            Int with module output shape.
```

(continues on next page)

(continued from previous page)

```
        """
        return self.no_of_embs

    def forward(self, inp: Dict[str, torch.Tensor]) -> torch.Tensor:
        """Concat all categorical embeddings
        """
        output = torch.cat(
            [
                emb_layer(inp["cat"][:, i])
                for i, emb_layer in enumerate(self.emb_layers)
            ],
            dim=1,
        )
        return output


class ContEmbedder(nn.Module):
    """Numeric data model.

    Class for working with numeric data.

    Args:
        num_dims: Sequence with number of numeric features.
        input_bn: Use 1d batch norm for input data.

    """

    def __init__(self, num_dims: int,  **kwargs):
        super(ContEmbedder, self).__init__()
        self.n_out = num_dims

    def get_out_shape(self) -> int:
        """Output shape.

        Returns:
            int with module output shape.

        """
        return self.n_out

    def forward(self, inp: Dict[str, torch.Tensor]) -> torch.Tensor:
        """Forward-pass."""
        return (inp["cont"] - inp["cont"].mean(axis=0)) / (inp["cont"].std(axis=0) + 1e-
↪6)
```

[15]:
```
from lightautoml.text.nn_model import TorchUniversalModel

class SimpleNet_plus(TorchUniversalModel):
    """Mixed data model.

    Class for preparing input for DL model with mixed data.
```

(continues on next page)

(continued from previous page)

```
    Args:
            n_out: Number of output dimensions.
            cont_params: Dict with numeric model params.
            cat_params: Dict with category model para
            **kwargs: Loss, task and other parameters.

        """

    def __init__(
            self,
            n_out: int = 1,
            cont_params: Optional[Dict] = None,
            cat_params: Optional[Dict] = None,
            **kwargs,
    ):
        # init parent class (need some helper functions to be used)
        super(SimpleNet_plus, self).__init__(**{
                **kwargs,
                "cont_params": cont_params,
                "cat_params": cat_params,
                "torch_model": None, # dont need any model inside parent class
        })

        n_in = 0

        # add cont columns processing
        self.cont_embedder = ContEmbedder(**cont_params)
        n_in += self.cont_embedder.get_out_shape()

        # add cat columns processing
        self.cat_embedder = CatEmbedder(**cat_params)
        n_in += self.cat_embedder.get_out_shape()

        self.torch_model = SimpleNet(
                **{
                    **kwargs,
                    **{"n_in": n_in, "n_out": n_out},
                }
        )

    def get_logits(self, inp: Dict[str, torch.Tensor]) -> torch.Tensor:
        outputs = []
        outputs.append(self.cont_embedder(inp))
        outputs.append(self.cat_embedder(inp))

        if len(outputs) > 1:
            output = torch.cat(outputs, dim=1)
        else:
            output = outputs[0]

        logits = self.torch_model(output)
```

(continues on next page)

(continued from previous page)

```
        return logits
```

[16]:
```
automl = TabularAutoML(
    **default_lama_params,
    general_params={"use_algos": [[SimpleNet_plus]]},
    nn_params={
        **default_nn_params,
        "hidden_size": 256,
        "drop_rate": 0.1,
        "model_with_emb": True,
    },
    debug=True
)
automl.fit_predict(tr_data, roles = roles, verbose = 1)
```

```
[14:56:24] Stdout logging level is INFO.
[14:56:24] Task: binary

[14:56:24] Start automl preset with listed constraints:
[14:56:24] - time: 300.00 seconds
[14:56:24] - CPU: 4 cores
[14:56:24] - memory: 16 GB

[14:56:24] Train data shape: (8000, 122)

[14:56:24] Layer 1 train process start. Time left 299.21 secs
[14:56:25] Start fitting Lvl_0_Pipe_0_Mod_0_TorchNN_0 ...
[14:56:30] Fitting Lvl_0_Pipe_0_Mod_0_TorchNN_0 finished. score = 0.6600159152016962
[14:56:30] Lvl_0_Pipe_0_Mod_0_TorchNN_0 fitting and predicting completed
[14:56:30] Time left 293.19 secs

[14:56:30] Layer 1 training completed.

[14:56:30] Automl preset training completed in 6.82 seconds

[14:56:30] Model description:
Final prediction for new objects (level 0) =
         1.00000 * (5 averaged models Lvl_0_Pipe_0_Mod_0_TorchNN_0)
```

[16]:
```
array([[0.07509199],
       [0.06439159],
       [0.04291169],
       ...,
       [0.11671165],
       [0.2381251 ],
       [0.04382631]], dtype=float32)
```

4.2 Tuning network

One can try optimize metric with the help of Optuna. Among validation stratagies there are: - `fit_on_holdout = True` - holdout - `fit_on_holdout = False` - cross-validation.

4.2.1 Built-in models

Use "`_tuned`" in model name to tune it.

[17]:
```
automl = TabularAutoML(
    **default_lama_params,
    general_params={"use_algos": [["denselight_tuned"]]},
    nn_params={
        **default_nn_params,
        "n_epochs": 3,
        "tuning_params": {
            "max_tuning_iter": 5,
            "max_tuning_time": 100,
            "fit_on_holdout": True
        }
    },
)
automl.fit_predict(tr_data, roles = roles, verbose = 3)
```

```
[14:56:31] Stdout logging level is INFO3.
[14:56:31] Task: binary

[14:56:31] Start automl preset with listed constraints:
[14:56:31] - time: 300.00 seconds
[14:56:31] - CPU: 4 cores
[14:56:31] - memory: 16 GB

[14:56:31] Train data shape: (8000, 122)

[14:56:31] Feats was rejected during automatic roles guess: []
[14:56:31] Layer 1 train process start. Time left 299.23 secs
[14:56:31] Start hyperparameters optimization for Lvl_0_Pipe_0_Mod_0_Tuned_TorchNN_
↪denselight_tuned_0 ... Time budget is 100.00 secs
[14:56:32] Epoch: 0, train loss: 0.307483434677124, val loss: 0.2785775661468506, val␣
↪metric: 0.6090575236140289
[14:56:32] Epoch: 1, train loss: 0.27614495158195496, val loss: 0.2799951434135437, val␣
↪metric: 0.585088014710992
[14:56:32] Epoch: 2, train loss: 0.27499067783355713, val loss: 0.28620871901512146, val␣
↪metric: 0.626435952125129
[14:56:32] Early stopping: val loss: 0.27636581659317017, val metric: 0.6215073421321315
[14:56:33] Trial 1 with hyperparameters {'bs': 128, 'weight_decay_bin': 0, 'lr': 0.
↪029154431891537533} scored 0.6215073421321315 in 0:00:01.209556
[14:56:33] Epoch: 0, train loss: 0.275651752948761, val loss: 0.30943918228149414, val␣
↪metric: 0.5956214485409284
[14:56:33] Epoch: 1, train loss: 0.28080031275749207, val loss: 0.3092973828315735, val␣
↪metric: 0.590257175083257
[14:56:33] Epoch: 2, train loss: 0.27882617712020874, val loss: 0.3090418577194214, val␣
↪metric: 0.5908826060693533
[14:56:33] Early stopping: val loss: 0.30920112133026123, val metric: 0.5908826060693533
```

(continues on next page)

(continued from previous page)

```
[14:56:33] Trial 2 with hyperparameters {'bs': 512, 'weight_decay_bin': 0, 'lr': 5.
↪415244119402538e-05} scored 0.5908826060693533 in 0:00:00.625572
[14:56:33] Epoch: 0, train loss: 0.2746148705482483, val loss: 0.2774512767791748, val␣
↪metric: 0.5961372954653581
[14:56:33] Epoch: 1, train loss: 0.2780302464962006, val loss: 0.276823490858078, val␣
↪metric: 0.5987940407652709
[14:56:34] Epoch: 2, train loss: 0.2752102017402649, val loss: 0.2738468050956726, val␣
↪metric: 0.6048131458109488
[14:56:34] Early stopping: val loss: 0.2762503921985626, val metric: 0.6007344804913642
[14:56:34] Trial 3 with hyperparameters {'bs': 1024, 'weight_decay_bin': 1, 'weight_decay
↪': 2.9204338471814107e-05, 'lr': 0.0006672367170464204} scored 0.6007344804913642 in 0:
↪00:00.504866
[14:56:34] Epoch: 0, train loss: 0.2786032557487488, val loss: 0.2767927944660187, val␣
↪metric: 0.5910777191547594
[14:56:35] Epoch: 1, train loss: 0.27806466817855835, val loss: 0.27634257078170776, val␣
↪metric: 0.592954012113048
[14:56:35] Epoch: 2, train loss: 0.2776066064834595, val loss: 0.2759130001068115, val␣
↪metric: 0.5941113267155251
[14:56:35] Early stopping: val loss: 0.27634990215301514, val metric: 0.593207926402275
[14:56:36] Trial 4 with hyperparameters {'bs': 64, 'weight_decay_bin': 0, 'lr': 1.
↪8205657658407255e-05} scored 0.593207926402275 in 0:00:01.881862
[14:56:36] Epoch: 0, train loss: 0.27860182523727417, val loss: 0.28250277042388916, val␣
↪metric: 0.5923793639847972
[14:56:36] Epoch: 1, train loss: 0.2779836356639862, val loss: 0.2820056080818176, val␣
↪metric: 0.5916817678849207
[14:56:37] Epoch: 2, train loss: 0.2774103283882141, val loss: 0.2815183401107788, val␣
↪metric: 0.5948837607111739
[14:56:37] Early stopping: val loss: 0.2820083796977997, val metric: 0.5934698590374777
[14:56:37] Trial 5 with hyperparameters {'bs': 128, 'weight_decay_bin': 0, 'lr': 3.
↪077180271250682e-05} scored 0.5934698590374777 in 0:00:01.121957
[14:56:37] Hyperparameters optimization for Lvl_0_Pipe_0_Mod_0_Tuned_TorchNN_denselight_
↪tuned_0 completed
[14:56:37] The set of hyperparameters {'bs': 128, 'weight_decay_bin': 0, 'lr': 0.
↪029154431891537533}
 achieve 0.6215 auc
[14:56:37] Start fitting Lvl_0_Pipe_0_Mod_0_Tuned_TorchNN_denselight_tuned_0 ...
[14:56:37] ===== Start working with fold 0 for Lvl_0_Pipe_0_Mod_0_Tuned_TorchNN_
↪denselight_tuned_0 =====
[14:56:37] Epoch: 0, train loss: 0.2774243652820587, val loss: 0.2794122099876404, val␣
↪metric: 0.5975324876651111
[14:56:37] Epoch: 1, train loss: 0.2743901014328003, val loss: 0.2776646912097931, val␣
↪metric: 0.608838355490696
[14:56:38] Epoch: 2, train loss: 0.27337446808815, val loss: 0.2764543294906616, val␣
↪metric: 0.6192034040551448
[14:56:38] Early stopping: val loss: 0.2777901291847229, val metric: 0.6107948319087405
[14:56:38] ===== Start working with fold 1 for Lvl_0_Pipe_0_Mod_0_Tuned_TorchNN_
↪denselight_tuned_0 =====
[14:56:38] Epoch: 0, train loss: 0.27729275822639465, val loss: 0.27042776346206665, val␣
↪metric: 0.6243578040081522
[14:56:39] Epoch: 1, train loss: 0.2746446132659912, val loss: 0.268259733915329, val␣
↪metric: 0.6331256368885869
[14:56:39] Epoch: 2, train loss: 0.2735038101673126, val loss: 0.26757094264030457, val␣
```

(continues on next page)

(continued from previous page)

```
↪metric: 0.6373237941576086
[14:56:39] Early stopping: val loss: 0.2686738669872284, val metric: 0.6330725628396741
[14:56:39] ===== Start working with fold 2 for Lvl_0_Pipe_0_Mod_0_Tuned_TorchNN_
↪denselight_tuned_0 =====
[14:56:39] Epoch: 0, train loss: 0.2761422395706177, val loss: 0.27418816089630127, val␣
↪metric: 0.5687123174252717
[14:56:40] Epoch: 1, train loss: 0.2733106017112732, val loss: 0.2741503119468689, val␣
↪metric: 0.5761931046195653
[14:56:40] Epoch: 2, train loss: 0.27197468280792236, val loss: 0.2742484509944916, val␣
↪metric: 0.5801471212635869
[14:56:40] Early stopping: val loss: 0.2739076316356659, val metric: 0.5770820949388586
[14:56:40] ===== Start working with fold 3 for Lvl_0_Pipe_0_Mod_0_Tuned_TorchNN_
↪denselight_tuned_0 =====
[14:56:41] Epoch: 0, train loss: 0.2768873870372772, val loss: 0.27838170528411865, val␣
↪metric: 0.6020985478940217
[14:56:41] Epoch: 1, train loss: 0.2738708257675171, val loss: 0.2780988812446594, val␣
↪metric: 0.5976827870244564
[14:56:41] Epoch: 2, train loss: 0.2721073031425476, val loss: 0.2787047028541565, val␣
↪metric: 0.5934050186820653
[14:56:41] Early stopping: val loss: 0.2780289649963379, val metric: 0.5992405103600543
[14:56:41] ===== Start working with fold 4 for Lvl_0_Pipe_0_Mod_0_Tuned_TorchNN_
↪denselight_tuned_0 =====
[14:56:42] Epoch: 0, train loss: 0.27685776352882385, val loss: 0.27540671825408936, val␣
↪metric: 0.5904859459918478
[14:56:42] Epoch: 1, train loss: 0.27350443601608276, val loss: 0.2741386592388153, val␣
↪metric: 0.5980489979619567
[14:56:42] Epoch: 2, train loss: 0.27244144678115845, val loss: 0.27352553606033325, val␣
↪metric: 0.6023214588994564
[14:56:42] Early stopping: val loss: 0.2742116153240204, val metric: 0.5989326808763588
[14:56:43] Fitting Lvl_0_Pipe_0_Mod_0_Tuned_TorchNN_denselight_tuned_0 finished. score =␣
↪0.6015718759719786
[14:56:43] Lvl_0_Pipe_0_Mod_0_Tuned_TorchNN_denselight_tuned_0 fitting and predicting␣
↪completed
[14:56:43] Time left 288.04 secs

[14:56:43] Layer 1 training completed.

[14:56:43] Automl preset training completed in 11.97 seconds

[14:56:43] Model description:
Final prediction for new objects (level 0) =
         1.00000 * (5 averaged models Lvl_0_Pipe_0_Mod_0_Tuned_TorchNN_denselight_tuned_
↪0)
```

[17]:
```
array([[0.07784943],
       [0.04554275],
       [0.05328501],
       ...,
       [0.07100379],
       [0.09577154],
       [0.07620702]], dtype=float32)
```

4.2.2 Custom model

There is a spesial flag `tuned` to mark that you need optimize parameters for the model.

[18]:
```
automl = TabularAutoML(
    **default_lama_params,
    general_params={"use_algos": [[SimpleNet]]},
    nn_params={
        **default_nn_params,
        "hidden_size": 256,
        "drop_rate": 0.1,

        "tuned": True,
        "tuning_params": {
            "max_tuning_iter": 5,
            "max_tuning_time": 100,
            "fit_on_holdout": True
        }
    },
)
automl.fit_predict(tr_data, roles = roles, verbose = 2)
```

```
[14:56:43] Stdout logging level is INFO2.
[14:56:43] Task: binary

[14:56:43] Start automl preset with listed constraints:
[14:56:43] - time: 300.00 seconds
[14:56:43] - CPU: 4 cores
[14:56:43] - memory: 16 GB

[14:56:43] Train data shape: (8000, 122)

[14:56:43] Layer 1 train process start. Time left 299.22 secs
[14:56:43] Start hyperparameters optimization for Lvl_0_Pipe_0_Mod_0_Tuned_TorchNN_0 ...␣
↪Time budget is 100.00 secs
```

```
Optimization Progress: 100%|| 5/5 [00:12<00:00,  2.42s/it, best_trial=0, best_value=0.
↪767]
```

```
[14:56:56] Hyperparameters optimization for Lvl_0_Pipe_0_Mod_0_Tuned_TorchNN_0 completed
[14:56:56] The set of hyperparameters {'bs': 128, 'weight_decay_bin': 0, 'lr': 0.
↪029154431891537533}
 achieve 0.7667 auc
[14:56:56] Start fitting Lvl_0_Pipe_0_Mod_0_Tuned_TorchNN_0 ...
[14:56:56] ===== Start working with fold 0 for Lvl_0_Pipe_0_Mod_0_Tuned_TorchNN_0 =====
```

```
[14:56:58] ===== Start working with fold 1 for Lvl_0_Pipe_0_Mod_0_Tuned_TorchNN_0 =====
[14:57:01] ===== Start working with fold 2 for Lvl_0_Pipe_0_Mod_0_Tuned_TorchNN_0 =====
[14:57:04] ===== Start working with fold 3 for Lvl_0_Pipe_0_Mod_0_Tuned_TorchNN_0 =====
[14:57:06] ===== Start working with fold 4 for Lvl_0_Pipe_0_Mod_0_Tuned_TorchNN_0 =====
[14:57:09] Fitting Lvl_0_Pipe_0_Mod_0_Tuned_TorchNN_0 finished. score = 0.
↪7271980081974132
[14:57:09] Lvl_0_Pipe_0_Mod_0_Tuned_TorchNN_0 fitting and predicting completed
[14:57:09] Time left 273.65 secs
```

(continues on next page)

(continued from previous page)

```
[14:57:09] Layer 1 training completed.

[14:57:09] Automl preset training completed in 26.35 seconds

[14:57:09] Model description:
Final prediction for new objects (level 0) =
         1.00000 * (5 averaged models Lvl_0_Pipe_0_Mod_0_Tuned_TorchNN_0)
```

[18]:
```
array([[0.03879727],
       [0.02351108],
       [0.0386253 ],
       ...,
       [0.04145308],
       [0.182652  ],
       [0.28383675]], dtype=float32)
```

Sometimes we need to tune parameters that we define by ourself. To this purpose we have `optimization_search_space` which describes necessary parameter grid. See example below.

Here is the grid:

- `bs` in `[64, 128, 256, 512, 1024]`
- `hidden_size` in `[64, 128, 256, 512, 1024]`
- `drop_rate` in `[0.0, 0.3]`

[19]:
```
def my_opt_space(trial: optuna.trial.Trial, estimated_n_trials, suggested_params):
    '''
        This function needs for parameter tuning
    '''
    # optionally
    trial_values = copy(suggested_params)

    trial_values["bs"] = trial.suggest_categorical(
        "bs", [2 ** i for i in range(6, 11)]
    )
    trial_values["hidden_size"] = trial.suggest_categorical(
        "hidden_size", [2 ** i for i in range(6, 11)]
    )
    trial_values["drop_rate"] = trial.suggest_float(
        "drop_rate", 0.0, 0.3
    )
    return trial_values
```

[20]:
```
automl = TabularAutoML(
    **default_lama_params,
    general_params={"use_algos": [[SimpleNet]]},
    nn_params={
        **default_nn_params,
        "n_epochs": 3,
        "tuned": True,
```

(continues on next page)

(continued from previous page)

```
        "tuning_params": {
            "max_tuning_iter": 5,
            "max_tuning_time": 3600,
            "fit_on_holdout": True
        },
        "optimization_search_space": my_opt_space,
    },
)
automl.fit_predict(tr_data, roles = roles, verbose = 3)
```

```
[14:57:09] Stdout logging level is INFO3.
[14:57:09] Task: binary

[14:57:09] Start automl preset with listed constraints:
[14:57:09] - time: 300.00 seconds
[14:57:09] - CPU: 4 cores
[14:57:09] - memory: 16 GB

[14:57:09] Train data shape: (8000, 122)

[14:57:10] Feats was rejected during automatic roles guess: []
[14:57:10] Layer 1 train process start. Time left 299.19 secs
[14:57:10] Start hyperparameters optimization for Lvl_0_Pipe_0_Mod_0_Tuned_TorchNN_0 ...␣
↪Time budget is 156.97 secs
[14:57:10] Epoch: 0, train loss: 0.27667880058288574, val loss: 0.2776942551136017, val␣
↪metric: 0.6380251348418515
[14:57:10] Epoch: 1, train loss: 0.2685483694076538, val loss: 0.2682102620601654, val␣
↪metric: 0.6962383266246506
[14:57:11] Epoch: 2, train loss: 0.2586376965045929, val loss: 0.259479820728302, val␣
↪metric: 0.741352213865324
[14:57:11] Early stopping: val loss: 0.2688170075416565, val metric: 0.7005254689396005
[14:57:11] Trial 1 with hyperparameters {'bs': 128, 'hidden_size': 256, 'drop_rate': 0.
↪006175348288740734} scored 0.7005254689396005 in 0:00:01.061573
[14:57:11] Epoch: 0, train loss: 0.2708968222141266, val loss: 0.2574772536754608, val␣
↪metric: 0.73635678432253
[14:57:12] Epoch: 1, train loss: 0.2547965347766876, val loss: 0.2453528791666031, val␣
↪metric: 0.7694297886898558
[14:57:12] Epoch: 2, train loss: 0.244949072599411, val loss: 0.24259500205516815, val␣
↪metric: 0.7720972251177362
[14:57:12] Early stopping: val loss: 0.2467520385980606, val metric: 0.7681415077697773
[14:57:13] Trial 2 with hyperparameters {'bs': 64, 'hidden_size': 1024, 'drop_rate': 0.
↪04184815819561255} scored 0.7681415077697773 in 0:00:01.623261
[14:57:13] Epoch: 0, train loss: 0.2758210599422455, val loss: 0.30738407373428345, val␣
↪metric: 0.6267513404001689
[14:57:13] Epoch: 1, train loss: 0.27457138895988464, val loss: 0.30406495928764343, val␣
↪metric: 0.6331232526687729
[14:57:13] Epoch: 2, train loss: 0.27020180225372314, val loss: 0.30057811737060547, val␣
↪metric: 0.6501381828289794
[14:57:13] Early stopping: val loss: 0.30382469296455383, val metric: 0.6392439234301416
[14:57:13] Trial 3 with hyperparameters {'bs': 512, 'hidden_size': 512, 'drop_rate': 0.
↪019515477895583853} scored 0.6392439234301416 in 0:00:00.557979
[14:57:13] Epoch: 0, train loss: 0.27815869450569153, val loss: 0.28139299154281616, val␣
```

(continues on next page)

(continued from previous page)

```
↪metric: 0.6253187292525297
[14:57:14] Epoch: 1, train loss: 0.2752159833908081, val loss: 0.2780429720878601, val␣
↪metric: 0.6364214656467331
[14:57:14] Epoch: 2, train loss: 0.2704654037952423, val loss: 0.27340632677078247, val␣
↪metric: 0.6653997680025231
[14:57:14] Early stopping: val loss: 0.2779545187950134, val metric: 0.6449262579448445
[14:57:14] Trial 4 with hyperparameters {'bs': 128, 'hidden_size': 64, 'drop_rate': 0.
↪2727961206236346} scored 0.6449262579448445 in 0:00:01.034966
[14:57:15] Epoch: 0, train loss: 0.2777031064033508, val loss: 0.28032609820365906, val␣
↪metric: 0.627515756049842
[14:57:15] Epoch: 1, train loss: 0.27274399995803833, val loss: 0.27464091777801514, val␣
↪metric: 0.6614868151664342
[14:57:15] Epoch: 2, train loss: 0.2659642994403839, val loss: 0.26697367429733276, val␣
↪metric: 0.7082605000240552
[14:57:15] Early stopping: val loss: 0.27455854415893555, val metric: 0.6712371238727542
[14:57:15] Trial 5 with hyperparameters {'bs': 128, 'hidden_size': 128, 'drop_rate': 0.
↪17936999364332554} scored 0.6712371238727542 in 0:00:01.005326
[14:57:15] Hyperparameters optimization for Lvl_0_Pipe_0_Mod_0_Tuned_TorchNN_0 completed
[14:57:15] The set of hyperparameters {'bs': 64, 'hidden_size': 1024, 'drop_rate': 0.
↪04184815819561255}
 achieve 0.7681 auc
[14:57:15] Start fitting Lvl_0_Pipe_0_Mod_0_Tuned_TorchNN_0 ...
[14:57:15] ===== Start working with fold 0 for Lvl_0_Pipe_0_Mod_0_Tuned_TorchNN_0 =====
[14:57:16] Epoch: 0, train loss: 0.2708968222141266, val loss: 0.2574772536754608, val␣
↪metric: 0.73635678432253
[14:57:16] Epoch: 1, train loss: 0.2547965347766876, val loss: 0.2453528791666031, val␣
↪metric: 0.7694297886898558
[14:57:17] Epoch: 2, train loss: 0.244949072599411, val loss: 0.242595002055168 15, val␣
↪metric: 0.7720972251177362
[14:57:17] Early stopping: val loss: 0.2467520385980606, val metric: 0.7681415077697773
[14:57:17] ===== Start working with fold 1 for Lvl_0_Pipe_0_Mod_0_Tuned_TorchNN_0 =====
[14:57:17] Epoch: 0, train loss: 0.27157771587371826, val loss: 0.2592219412326813, val␣
↪metric: 0.7320822010869567
[14:57:18] Epoch: 1, train loss: 0.2547703981399536, val loss: 0.2508712708950043, val␣
↪metric: 0.742293648097826
[14:57:18] Epoch: 2, train loss: 0.24370187520980835, val loss: 0.2526074945926666, val␣
↪metric: 0.7376868206521738
[14:57:18] Early stopping: val loss: 0.25174227356910706, val metric: 0.7418584408967391
[14:57:18] ===== Start working with fold 2 for Lvl_0_Pipe_0_Mod_0_Tuned_TorchNN_0 =====
[14:57:19] Epoch: 0, train loss: 0.2685704529285431, val loss: 0.27050256729125977, val␣
↪metric: 0.6421482252038044
[14:57:19] Epoch: 1, train loss: 0.2485620081424713, val loss: 0.2682766616344452, val␣
↪metric: 0.673721976902174
[14:57:20] Epoch: 2, train loss: 0.2378905713558197, val loss: 0.2720091640949249, val␣
↪metric: 0.6822032099184783
[14:57:20] Early stopping: val loss: 0.26766759157180786, val metric: 0.6720713739809782
[14:57:20] ===== Start working with fold 3 for Lvl_0_Pipe_0_Mod_0_Tuned_TorchNN_0 =====
[14:57:21] Epoch: 0, train loss: 0.2702709436416626, val loss: 0.2615496516227722, val␣
↪metric: 0.7016389266304347
[14:57:21] Epoch: 1, train loss: 0.2521918714046478, val loss: 0.2561715841293335, val␣
↪metric: 0.7150401239809783
[14:57:22] Epoch: 2, train loss: 0.24233928322792053, val loss: 0.2551616132259369, val␣
```

(continues on next page)

(continued from previous page)

```
↪metric: 0.7239884086277175
[14:57:22] Early stopping: val loss: 0.2555495500564575, val metric: 0.7176938264266304
[14:57:22] ===== Start working with fold 4 for Lvl_0_Pipe_0_Mod_0_Tuned_TorchNN_0 =====
[14:57:22] Epoch: 0, train loss: 0.2711535096168518, val loss: 0.2612263262271881, val␣
↪metric: 0.701416015625
[14:57:23] Epoch: 1, train loss: 0.25223615765571594, val loss: 0.254193514585495, val␣
↪metric: 0.7256443189538043
[14:57:23] Epoch: 2, train loss: 0.24515873193740845, val loss: 0.25219428539276123, val␣
↪metric: 0.7378141983695652
[14:57:23] Early stopping: val loss: 0.25415900349617004, val metric: 0.7281971807065218
[14:57:23] Fitting Lvl_0_Pipe_0_Mod_0_Tuned_TorchNN_0 finished. score = 0.
↪7241582599492865
[14:57:23] Lvl_0_Pipe_0_Mod_0_Tuned_TorchNN_0 fitting and predicting completed
[14:57:23] Time left 285.60 secs

[14:57:23] Layer 1 training completed.

[14:57:23] Automl preset training completed in 14.40 seconds

[14:57:23] Model description:
Final prediction for new objects (level 0) =
         1.00000 * (5 averaged models Lvl_0_Pipe_0_Mod_0_Tuned_TorchNN_0)
```

[20]:
```
array([[0.04496425],
       [0.03032025],
       [0.03665409],
       ...,
       [0.05365612],
       [0.16432838],
       [0.1691863 ]], dtype=float32)
```

4.2.3 One more example

Tuning NODE params

[21]:
```
TIMEOUT = 3000
```

[22]:
```
default_lama_params = {
    "task": task,
    "timeout": TIMEOUT,
    "cpu_limit": N_THREADS,
    "reader_params": {'n_jobs': N_THREADS, 'cv': N_FOLDS, 'random_state': RANDOM_STATE}
}

default_nn_params = {
    "bs": 512, "num_workers": 0, "path_to_save": None, "n_epochs": 10, "freeze_defaults":
↪ True
}
```

[23]:
```
def my_opt_space_NODE(trial: optuna.trial.Trial, estimated_n_trials, suggested_params):
```

(continues on next page)

(continued from previous page)

```
    '''
        This function needs for parameter tuning
    '''
    # optionally
    trial_values = copy(suggested_params)

    trial_values["layer_dim"] = trial.suggest_categorical(
        "layer_dim", [2 ** i for i in range(8, 10)]
    )
    trial_values["use_original_head"] = trial.suggest_categorical(
        "use_original_head", [True, False]
    )
    trial_values["num_layers"] = trial.suggest_int(
        "num_layers", 1, 3
    )
    trial_values["drop_rate"] = trial.suggest_float(
        "drop_rate", 0.0, 0.3
    )
    trial_values["tree_dim"] = trial.suggest_int(
        "tree_dim", 1, 3
    )
    return trial_values
```

[24]:
```
automl = TabularAutoML(
    task = task,
    timeout = TIMEOUT,
    cpu_limit = N_THREADS,
    general_params = {"use_algos": [["node_tuned"]]}, # ['nn', 'mlp', 'dense', 'denselight',
↪'resnet', 'snn'] or custom torch model
    nn_params = {"n_epochs": 10, "bs": 512, "num_workers": 0, "path_to_save": None,
↪"freeze_defaults": True, "optimization_search_space": my_opt_space_NODE,},
    nn_pipeline_params = {"use_qnt": True, "use_te": False},
    reader_params = {'n_jobs': N_THREADS, 'cv': N_FOLDS, 'random_state': RANDOM_STATE}
)
```

[25]:
```
oof_pred = automl.fit_predict(tr_data, roles = roles, verbose = 2)
```

```
[14:57:24] Stdout logging level is INFO2.
[14:57:24] Task: binary

[14:57:24] Start automl preset with listed constraints:
[14:57:24] - time: 3000.00 seconds
[14:57:24] - CPU: 4 cores
[14:57:24] - memory: 16 GB

[14:57:24] Train data shape: (8000, 122)

[14:57:25] Layer 1 train process start. Time left 2999.22 secs
[14:57:25] Start hyperparameters optimization for Lvl_0_Pipe_0_Mod_0_Tuned_TorchNN_node_
↪tuned_0 ... Time budget is 1574.34 secs
```

```
Optimization Progress: 100%|| 25/25 [03:48<00:00,  9.14s/it, best_trial=13, best_value=0.
↪732]
```

```
[15:01:14] Hyperparameters optimization for Lvl_0_Pipe_0_Mod_0_Tuned_TorchNN_node_tuned_
↪0 completed
[15:01:14] The set of hyperparameters {'layer_dim': 512, 'use_original_head': False,
↪'num_layers': 3, 'drop_rate': 0.1310638585198816, 'tree_dim': 3}
 achieve 0.7315 auc
[15:01:14] Start fitting Lvl_0_Pipe_0_Mod_0_Tuned_TorchNN_node_tuned_0 ...
[15:01:14] ===== Start working with fold 0 for Lvl_0_Pipe_0_Mod_0_Tuned_TorchNN_node_
↪tuned_0 =====
```

```
[15:01:27] ===== Start working with fold 1 for Lvl_0_Pipe_0_Mod_0_Tuned_TorchNN_node_
↪tuned_0 =====
[15:01:41] ===== Start working with fold 2 for Lvl_0_Pipe_0_Mod_0_Tuned_TorchNN_node_
↪tuned_0 =====
[15:01:54] ===== Start working with fold 3 for Lvl_0_Pipe_0_Mod_0_Tuned_TorchNN_node_
↪tuned_0 =====
[15:02:07] ===== Start working with fold 4 for Lvl_0_Pipe_0_Mod_0_Tuned_TorchNN_node_
↪tuned_0 =====
[15:02:20] Fitting Lvl_0_Pipe_0_Mod_0_Tuned_TorchNN_node_tuned_0 finished. score = 0.
↪6942477367184283
[15:02:20] Lvl_0_Pipe_0_Mod_0_Tuned_TorchNN_node_tuned_0 fitting and predicting completed
[15:02:20] Time left 2703.48 secs

[15:02:20] Layer 1 training completed.

[15:02:20] Automl preset training completed in 296.53 seconds

[15:02:20] Model description:
Final prediction for new objects (level 0) =
         1.00000 * (5 averaged models Lvl_0_Pipe_0_Mod_0_Tuned_TorchNN_node_tuned_0)
```

4.3 Several models

If you have several neural networks you can either define one set parameters for all or use unique for each one of them as below.

Note: numeration starts with 0. Each id (string of number) corresponds to the serial number in *the list of used neural networks*.

[26]:
```
automl = TabularAutoML(
    **default_lama_params,
    general_params = {"use_algos": [["lgb", "mlp", "dense"]]},
    nn_params = {"0": {**default_nn_params, "n_epochs": 2},
                 "1": {**default_nn_params, "n_epochs": 5}},
)
automl.fit_predict(tr_data, roles = roles, verbose = 3)
```

```
[15:02:20] Stdout logging level is INFO3.
[15:02:20] Task: binary

[15:02:20] Start automl preset with listed constraints:
```

(continues on next page)

(continued from previous page)

```
[15:02:20] - time: 3000.00 seconds
[15:02:20] - CPU: 4 cores
[15:02:20] - memory: 16 GB

[15:02:20] Train data shape: (8000, 122)

[15:02:21] Feats was rejected during automatic roles guess: []
[15:02:21] Layer 1 train process start. Time left 2999.21 secs
[15:02:21] Training until validation scores don't improve for 200 rounds
[15:02:24] Selector_LightGBM fitting and predicting completed
[15:02:25] Start fitting Lvl_0_Pipe_0_Mod_0_LightGBM ...
[15:02:25] ===== Start working with fold 0 for Lvl_0_Pipe_0_Mod_0_LightGBM =====
[15:02:25] Training until validation scores don't improve for 200 rounds
[15:02:27] ===== Start working with fold 1 for Lvl_0_Pipe_0_Mod_0_LightGBM =====
[15:02:27] Training until validation scores don't improve for 200 rounds
[15:02:31] ===== Start working with fold 2 for Lvl_0_Pipe_0_Mod_0_LightGBM =====
[15:02:31] Training until validation scores don't improve for 200 rounds
[15:02:33] ===== Start working with fold 3 for Lvl_0_Pipe_0_Mod_0_LightGBM =====
[15:02:33] Training until validation scores don't improve for 200 rounds
[15:02:37] ===== Start working with fold 4 for Lvl_0_Pipe_0_Mod_0_LightGBM =====
[15:02:37] Training until validation scores don't improve for 200 rounds
[15:02:39] Fitting Lvl_0_Pipe_0_Mod_0_LightGBM finished. score = 0.7324164765495265
[15:02:39] Lvl_0_Pipe_0_Mod_0_LightGBM fitting and predicting completed
[15:02:39] Time left 2981.34 secs

[15:02:39] Start fitting Lvl_0_Pipe_1_Mod_0_TorchNN_mlp_0 ...
[15:02:39] ===== Start working with fold 0 for Lvl_0_Pipe_1_Mod_0_TorchNN_mlp_0 =====
[15:02:39] Epoch: 0, train loss: 0.27922049164772034, val loss: 0.30915024876594543, val␣
↪metric: 0.5770001764036115
[15:02:39] Epoch: 1, train loss: 0.2804635167121887, val loss: 0.30738765001296997, val␣
↪metric: 0.5919196454821967
[15:02:40] Early stopping: val loss: 0.3083692789077759, val metric: 0.5864190601429403
[15:02:40] ===== Start working with fold 1 for Lvl_0_Pipe_1_Mod_0_TorchNN_mlp_0 =====
[15:02:40] Epoch: 0, train loss: 0.2781796455383301, val loss: 0.2599707245826721, val␣
↪metric: 0.6234980044157609
[15:02:40] Epoch: 1, train loss: 0.27900466322898865, val loss: 0.25812670588493347, val␣
↪metric: 0.6304188603940217
[15:02:40] Early stopping: val loss: 0.259158730506897, val metric: 0.6298456606657609
[15:02:40] ===== Start working with fold 2 for Lvl_0_Pipe_1_Mod_0_TorchNN_mlp_0 =====
[15:02:40] Epoch: 0, train loss: 0.27802005410194397, val loss: 0.26105043292045593, val␣
↪metric: 0.54180908203125
[15:02:41] Epoch: 1, train loss: 0.2754856050014496, val loss: 0.26127585768699646, val␣
↪metric: 0.5531536599864131
[15:02:41] Early stopping: val loss: 0.2610853612422943, val metric: 0.5479577105978262
[15:02:41] ===== Start working with fold 3 for Lvl_0_Pipe_1_Mod_0_TorchNN_mlp_0 =====
[15:02:41] Epoch: 0, train loss: 0.2771044075489044, val loss: 0.2935408353805542, val␣
↪metric: 0.5986620032269022
[15:02:41] Epoch: 1, train loss: 0.2794908881187439, val loss: 0.292603075504303, val␣
↪metric: 0.5987601902173911
[15:02:41] Early stopping: val loss: 0.2930262088775635, val metric: 0.6013183593749999
[15:02:41] ===== Start working with fold 4 for Lvl_0_Pipe_1_Mod_0_TorchNN_mlp_0 =====
[15:02:41] Epoch: 0, train loss: 0.27787843346595764, val loss: 0.2770363688468933, val␣
```

(continues on next page)

(continued from previous page)

```
↪metric: 0.5949680494225544
[15:02:41] Epoch: 1, train loss: 0.27761200070381165, val loss: 0.2755982279777527, val␣
↪metric: 0.5874899159307065
[15:02:41] Early stopping: val loss: 0.27642762660980225, val metric: 0.5912050993546196
[15:02:42] Fitting Lvl_0_Pipe_1_Mod_0_TorchNN_mlp_0 finished. score = 0.5890259518134635
[15:02:42] Lvl_0_Pipe_1_Mod_0_TorchNN_mlp_0 fitting and predicting completed
[15:02:42] Start fitting Lvl_0_Pipe_1_Mod_1_TorchNN_dense_1 ...
[15:02:42] ===== Start working with fold 0 for Lvl_0_Pipe_1_Mod_1_TorchNN_dense_1 =====
[15:02:42] Epoch: 0, train loss: 0.27331802248954773, val loss: 0.30547928811011963, val␣
↪metric: 0.6767831465058721
[15:02:42] Epoch: 1, train loss: 0.24652224779129028, val loss: 0.28541794419288635, val␣
↪metric: 0.7536603749378579
[15:02:42] Epoch: 2, train loss: 0.22167454659938812, val loss: 0.29559990763664246, val␣
↪metric: 0.7280871968397026
[15:02:42] Epoch: 3, train loss: 0.18925495445728302, val loss: 0.32088974118232727, val␣
↪metric: 0.7067904699285298
[15:02:43] Epoch: 4, train loss: 0.15982066094875336, val loss: 0.3512553572654724, val␣
↪metric: 0.7054808067525163
[15:02:43] Early stopping: val loss: 0.2940850555896759, val metric: 0.733935243837901
[15:02:43] ===== Start working with fold 1 for Lvl_0_Pipe_1_Mod_1_TorchNN_dense_1 =====
[15:02:43] Epoch: 0, train loss: 0.2741259038448334, val loss: 0.2579110264778137, val␣
↪metric: 0.714859672214674
[15:02:43] Epoch: 1, train loss: 0.24377931654453278, val loss: 0.2441163808107376, val␣
↪metric: 0.7128348972486412
[15:02:43] Epoch: 2, train loss: 0.21424879133701324, val loss: 0.2476673424243927, val␣
↪metric: 0.6905942170516304
[15:02:44] Epoch: 3, train loss: 0.18909378349781036, val loss: 0.2624853551387787, val␣
↪metric: 0.6963155995244565
[15:02:44] Epoch: 4, train loss: 0.15743489563465118, val loss: 0.2790760099887848, val␣
↪metric: 0.6850798233695653
[15:02:44] Early stopping: val loss: 0.24882426857948303, val metric: 0.7349694293478262
[15:02:44] ===== Start working with fold 2 for Lvl_0_Pipe_1_Mod_1_TorchNN_dense_1 =====
[15:02:44] Epoch: 0, train loss: 0.2707020044326782, val loss: 0.2598980665206909, val␣
↪metric: 0.6082498301630435
[15:02:44] Epoch: 1, train loss: 0.2435956597328186, val loss: 0.2571799159049988, val␣
↪metric: 0.6500668733016304
[15:02:45] Epoch: 2, train loss: 0.21632780134677887, val loss: 0.2744308412075043, val␣
↪metric: 0.630631156589674
[15:02:45] Epoch: 3, train loss: 0.18905353546142578, val loss: 0.2583690583705902, val␣
↪metric: 0.6252043350883152
[15:02:45] Epoch: 4, train loss: 0.1545938104391098, val loss: 0.31239771842956543, val␣
↪metric: 0.6080746858016305
[15:02:45] Early stopping: val loss: 0.25516635179519653, val metric: 0.6447541610054348
[15:02:45] ===== Start working with fold 3 for Lvl_0_Pipe_1_Mod_1_TorchNN_dense_1 =====
[15:02:45] Epoch: 0, train loss: 0.27545467019081116, val loss: 0.28978264331817627, val␣
↪metric: 0.6915867017663043
[15:02:46] Epoch: 1, train loss: 0.24367769062519073, val loss: 0.2738344967365265, val␣
↪metric: 0.7211224099864131
[15:02:46] Epoch: 2, train loss: 0.21612469851970673, val loss: 0.2693524658679962, val␣
↪metric: 0.7317690641983695
[15:02:46] Epoch: 3, train loss: 0.18013358116149902, val loss: 0.31211358308792114, val␣
↪metric: 0.7004659901494565
```

(continues on next page)

(continued from previous page)

```
[15:02:46] Epoch: 4, train loss: 0.16005361080169678, val loss: 0.31500178575515747, val␣
↪metric: 0.7096371858016304
[15:02:46] Early stopping: val loss: 0.27935606241226196, val metric: 0.7314346976902173
[15:02:46] ===== Start working with fold 4 for Lvl_0_Pipe_1_Mod_1_TorchNN_dense_1 =====
[15:02:47] Epoch: 0, train loss: 0.2715047299861908, val loss: 0.27490749955177307, val␣
↪metric: 0.672867484714674
[15:02:47] Epoch: 1, train loss: 0.24714724719524384, val loss: 0.2625226378440857, val␣
↪metric: 0.7219928243885869
[15:02:47] Epoch: 2, train loss: 0.21466588973999023, val loss: 0.26622310280799866, val␣
↪metric: 0.7148384425951086
[15:02:47] Epoch: 3, train loss: 0.18601463735103607, val loss: 0.28521138429641724, val␣
↪metric: 0.6925048828124999
[15:02:47] Epoch: 4, train loss: 0.15070100128650665, val loss: 0.2958201467990875, val␣
↪metric: 0.6602305536684783
[15:02:47] Early stopping: val loss: 0.2677777111530304, val metric: 0.7166058084239131
[15:02:47] Fitting Lvl_0_Pipe_1_Mod_1_TorchNN_dense_1 finished. score = 0.
↪7086451902861568
[15:02:47] Lvl_0_Pipe_1_Mod_1_TorchNN_dense_1 fitting and predicting completed
[15:02:47] Time left 2973.02 secs

[15:02:47] Layer 1 training completed.

[15:02:47] Blending: optimization starts with equal weights. Score = 0.7380476
[15:02:47] Blending: iteration 0: score = 0.7394669, weights = [0.45736045 0.0553336  0.
↪487306  ]
[15:02:48] Blending: iteration 1: score = 0.7395753, weights = [0.4892029  0.         0.
↪51079714]
[15:02:48] Blending: no improvements for score. Terminated.

[15:02:48] Blending: best score = 0.7395753, best weights = [0.4892029  0.         0.
↪51079714]
[15:02:48] Automl preset training completed in 27.27 seconds

[15:02:48] Model description:
Final prediction for new objects (level 0) =
         0.48920 * (5 averaged models Lvl_0_Pipe_0_Mod_0_LightGBM) +
         0.51080 * (5 averaged models Lvl_0_Pipe_1_Mod_1_TorchNN_dense_1)
```

[26]:
```
array([[0.06395157],
       [0.04285344],
       [0.04808115],
       ...,
       [0.04276791],
       [0.19339147],
       [0.10395089]], dtype=float32)
```

2.1.10 Tutorial 10: Relational datasets (with star scheme)

Official LightAutoML github repository is here

In this tutorial, we will look at how to use LightAutoML with relational datasets.

Install LightAutoML

[1]:
```
#! pip install -U lightautoml
```

Import necessary libraries

[2]:
```
# Standard python libraries
from os.path import join as pjoin

# ML and DS libraries
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_absolute_error

# Imports from lightautoml package
from lightautoml.automl.base import AutoML
from lightautoml.ml_algo.boost_lgbm import BoostLGBM

from lightautoml.pipelines.features.lgb_pipeline import LGBSimpleFeatures
from lightautoml.pipelines.ml.base import MLPipeline
from lightautoml.reader.base import DictToPandasSeqReader
from lightautoml.tasks import Task

# Import Feature Generator Transformer
from lightautoml.pipelines.features.generator_pipeline import FeatureGeneratorPipeline
```

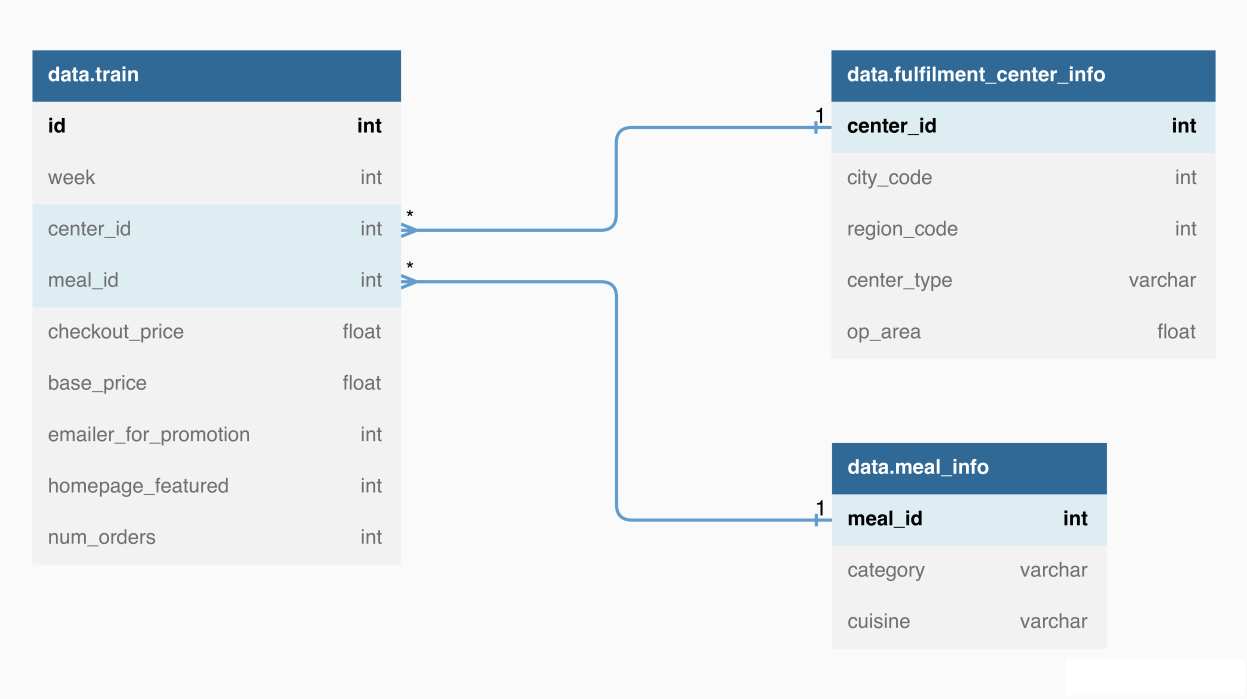
Relational data

Consider data that is a set of linked tables. Usually in this case there is a separate main table containing the objects identifiers and the corresponding values of the target variable, as well as possibly the values of other features (so called fact table). Other tables contain additional or auxiliary information, for example, records about all customer transactions (there can be an arbitrary number for a user with a specific identifier etc), the correspondence between the values of one feature and the values of another (the correspondence between an employee's department and his salary, for example), etc (so called dimension tables). However the organization of the data may differ from this scheme. To apply machine learning algorithms and LightAutoML, it is necessary to create a single dataset with all the features for each of the objects. For this we need to set the correspondence between the columns of the main and auxiliary tables for the correct aggregation of features. Such tables can form different schemas.

In this example, we use Meal delivery company dataset and will consider one of the simplest and most common schemes for organizing tables - the so-called star scheme, in which there is one main table, and there are connections only between the main and auxiliary tables by specified columns, but not between separate auxiliary tables, not sequentially, etc. At the present moment, this is the only scheme supported in LightAutoML, support for more complex schemes is

in development. Note that the connection between the main and each auxiliary table is carried out by a single key, but they may differ for different tables. Also, the columns for binding must be the primary key.

Consider an example of data with a star scheme organization. The dataset contains data on the sale of meals in the restaurant chain, consists of three tables: the main one containing information about completed orders (`train` and `test` parts), and two auxiliary tables containing information about restaurants (`fulfilment_center_info`) and available dishes (`meal_info`). The tables and the scheme of their organization are shown in the image below.



For the convenience of further use, we will save datasets and paths to them in dictionaries.

[3]:
```
data_dir = '../data/meal_delivery_company'

fulfilment_center_info = pd.read_csv(pjoin(data_dir, 'fulfilment_center_info.csv'))
meal_info = pd.read_csv(pjoin(data_dir, 'meal_info.csv'))
df_main = pd.read_csv(pjoin(data_dir, 'relational_main.csv.zip'))
```

[4]:
```
fulfilment_center_info.head()
```

[4]:

	center_id	city_code	region_code	center_type	op_area
0	11	679	56	TYPE_A	3.7
1	13	590	56	TYPE_B	6.7
2	124	590	56	TYPE_C	4.0
3	66	648	34	TYPE_A	4.1
4	94	632	34	TYPE_C	3.6

[5]:
```
meal_info.head()
```

[5]:

	meal_id	category	cuisine
0	1885	Beverages	Thai
1	1993	Beverages	Thai
2	2539	Beverages	Thai

(continues on next page)

(continued from previous page)

```
3     1248  Beverages  Indian
4     2631  Beverages  Indian
```

[6]:
```
df_main.head()
```

[6]:
```
        id  week  center_id  meal_id  checkout_price  base_price  \
0  1476796   135         43     1770          486.03      486.03
1  1168999    65         23     2760          241.53      241.53
2  1190875   105         75     2444          709.13      708.13
3  1375454    68         10     2760          222.13      224.13
4  1397113    33         36     1438          256.08      243.50

   emailer_for_promotion  homepage_featured  num_orders
0                      0                  0          40
1                      0                  0          68
2                      0                  0          80
3                      0                  1         634
4                      0                  1         122
```

[7]:
```
df_main.shape
```

[7]:
```
(45655, 9)
```

Create sequential star scheme dictionary

For further use of LightAutoML, you need to specify the data schema. It is necessary to specify secondary tables in the dictionary as the key to which the dictionary of the remaining parameters corresponds. The following parameters are specified in this dictionary:

- `'case'` - the type of column that plays the role of a key for binding. If `'ids'`, then the column is treated as a set of unique identifiers (ids), and if `'next_values'`, then it is treated as a set of timestamps.
- `'params'` - dictionary of timestamp processing and interpretation parameters in case of linking by `'next_values'` type column. In case of `'ids'` it might be set empty.
- `'scheme'` - dictionary describing the scheme of relationship between the main and secondary table. Consists of the next keys:
  - `'to'` - the name of the table, the relationship with which is being considered (in case of star scheme, the name `'plain'` should be specified here)
  - `'from_id'` - the name of column for link in secondary table (from which the link exists);
  - `'to_id'` - the name of column for link in main table (to which the link exists).

In our example, columns for linkage are IDs. Now we set a dictionary of parameters for communication taking into account the table schema:

[8]:
```
seq_params = {
   'fulfilment_center_info': {
      'case': 'ids',
      'params': {},
      'scheme': {'to': 'plain', 'from_id': 'center_id', 'to_id': 'center_id'},
   },
   'meal_info':{
      'case': 'ids',
```

(continues on next page)

(continued from previous page)

```
        'params': {},
        'scheme': {'to': 'plain', 'from_id': 'meal_id', 'to_id': 'meal_id'},
    },
}
```

Create a dict with second-level tables.

[9]:
```
seq_data = {
        'fulfilment_center_info': fulfilment_center_info,
        'meal_info': meal_info
}
```

Define train and test data samples. They must be specified in the form of a dictionary, where the main dataset is specified by the `'plain'` key, and the dictionary with secondary tables is specified by the `'seq'` key (like the `seq_data` dictionary). Note that train and test data differ only in plain data, and train plain data must contain a column with the target variable.

[10]:
```
train, test = train_test_split(df_main.sort_values(by='week', ascending=True),␣
↪shuffle=False, test_size=0.2)

train = {
    'plain': train,
    'seq': seq_data
}

test = {
    'plain': test,
    'seq': seq_data
}
```

Create Task snd Sequential Reader for the star scheme data

To work with linked tables in LightAutoML, it is not possible to use tabular presets like `TabularAutoML`, so we have to set all the pipeline manually. You can see more details about creating custom pipelines in this tutorial.

First we will set task and roles for our objective. Than it is necessary to create `DictToPandasSeqReader` to process data in form of relational tables. It requires setting the task and sequential data parameters dict as arguments (more details about this reader you can see here):

[11]:
```
task = Task('reg', metric='mae')
roles={'target': 'num_orders'}
reader = DictToPandasSeqReader(task=task, seq_params=seq_params)
```

Create Feature Generator Pipeline

In addition to aggregating data from all related tables into one, LightAutoML has the ability to perform additional feature generation by using `FeatureGeneratorPipeline`. Features can be generated using various aggregations (taking the average, median, counting unique values, etc.), extracting date features (year, day, difference between dates, weekend or weekday, etc.), different transformations, as well as using so-called interesting values, that is, constructing features by objects with a certain value of a set of categorical features (conditional feature generation, like "where" clause). For aggregation and transformation LightAutoML uses according primitives from FeatureTools, detailed info is available here.

Define interesting values parameters for feature generation in corresponding tables.

[12]:
```
interesting_values = {
    'fulfilment_center_info': {'center_type': ['TYPE_A', 'TYPE_C'], 'city_code': [647,␣
↪456, 703]},
    'meal_info': {'category': ['Extras', 'Seafood'], 'cuisine': ['Continental', 'Thai']}
}
```

So, in our example we want to generate features by orders where `'center_type'` feature was equal to `'TYPE_A'` or `'TYPE_C'`, and `'city_code'` feature was equal to `647`, `456` or `703`, and similarly for features of ordered meal from `meal_info` table.

Params of feature generator: - seq_params: secondary tables or sequence related parameters. - max_gener_features: maximum number of generated features. - max_depth: maximum allowed depth of features (that is, the number of consecutively applied aggregation and transformation primitives in a superposition to obtain features). - agg_primitives: list of aggregation primitives. By default it is ["entropy", "count", "mean", "std", "median", "max", "sum", "num_unique", "min", "percent_true"]. - trans_primitives: list of transform primitives. By default it is ["hour", "month", "weekday", "is_weekend", "day", "time_since_previous", "week", "age", "time_since"]. - interesting_values: categorical values if the form of {'table_name': {'column': [values]}} for feature generation in corresponding slices (like the `interesting_values` dictionary above). - generate_interesting_values: whether generate feature in slices of unique categories or not. - per_top_categories: percent of most frequent categories for feature generation in corresponding slices. If number of unique values is less than 10, then the all values are be used. - sample_size: size of data to make generated feature selection on it. - n_jobs: number of processes to run in parallel

More details about `FeatureGeneratorPipeline` are available FeatureGeneratorPipeline class in lightautoml/pipelines/features/generator_pipeline.py

[13]:
```
generator = FeatureGeneratorPipeline(
    seq_params,
    max_gener_features=500,
    interesting_values = interesting_values,
    generate_interesting_values = True,
    per_top_categories = 25,
    sample_size = None,
    n_jobs = 16
)
```

Create one-level ML pipeline for AutoML

Next we will compose the entire pipeline. We will add the basic simplest transformations to the pipeline of feature generation (encoding categorical features, converting date features to appropriate format, defining numeric types, defining roles). The set of algorithms will consist only of LightGBM gradient boosting, and no pre-selection or post-selection of features will be used.

[14]:
```
simpletransf = LGBSimpleFeatures()
feats = generator.append(simpletransf)

model = BoostLGBM()

pipeline_lvl1 = MLPipeline([model], pre_selection=None, features_pipeline=feats,post_
↪selection=None)
```

Initialize `AutoML` instance:

[15]:
```
automl = AutoML(reader, [[pipeline_lvl1],], skip_conn=False)
```

Train AutoML on loaded data

Let's train our model on train data and look at the logs of training. For more detailed info we will set verbosity level to 3:

[16]:
```
%%time

train_pred = automl.fit_predict(train, roles=roles, verbose=3)
```

```
[17:03:13] Feats was rejected during automatic roles guess: []
[17:03:13] Layer 1 train process start. Time left 9999999997.76 secs
[17:03:13] This selector only for holdout training. fit_on_holout argument added just to␣
↪be compatible
[17:03:13] Copying TaskTimer may affect the parent PipelineTimer, so copy will create␣
↪new unlimited TaskTimer
```

```
/home/rinchin/lama_gitlab/LightAutoML/.venv/lib/python3.8/site-packages/featuretools/
↪synthesis/dfs.py:321: UnusedPrimitiveWarning: Some specified primitives were not used␣
↪during DFS:
  trans_primitives: ['age', 'day', 'hour', 'is_weekend', 'month', 'time_since', 'time_
↪since_previous', 'week', 'weekday']
  agg_primitives: ['percent_true']
  where_primitives: ['entropy', 'num_unique', 'percent_true']
This may be caused by a using a value of max_depth that is too small, not setting␣
↪interesting values, or it may indicate no compatible columns for the primitive were␣
↪found in the data. If the DFS call contained multiple instances of a primitive in the␣
↪list above, none of them were used.
  warnings.warn(warning_msg, UnusedPrimitiveWarning)
```

```
EntitySet scattered to 16 workers in 4 seconds
[17:03:22] Training until validation scores don't improve for 100 rounds
[17:03:23] [100]        valid's l1: 208.397
[17:03:23] [200]        valid's l1: 208.432
[17:03:24] Early stopping, best iteration is:
[125]   valid's l1: 208.028
[17:03:24] LightGBM fitting and predicting completed
[17:03:24] Started iteration 0, chunk = ['ft__plain_center_id.MAX(fulfilment_center_info.
↪op_area)', 'ft__plain_center_id.MAX(fulfilment_center_info.city_code WHERE center_type␣
↪= TYPE_A)', 'ft__plain_meal_id.COUNT(meal_info WHERE cuisine = Italian)', 'ft__plain_
↪meal_id.COUNT(meal_info WHERE cuisine = Continental)', 'ft__plain_center_id.
↪MAX(fulfilment_center_info.op_area WHERE center_type = TYPE_B)', 'ft__plain_meal_id.
↪COUNT(meal_info WHERE cuisine = Indian)', 'ft__plain_center_id.MAX(fulfilment_center_
↪info.city_code)', 'ft__plain_center_id.MAX(fulfilment_center_info.op_area WHERE center_
↪type = TYPE_A)', 'ft__plain_center_id.MAX(fulfilment_center_info.region_code WHERE␣
↪center_type = TYPE_A)', 'ft__plain_meal_id.COUNT(meal_info WHERE cuisine = Thai)', 'ft_
↪_plain_center_id.MEDIAN(fulfilment_center_info.op_area)', 'ft__plain_center_id.
↪MEAN(fulfilment_center_info.op_area)'], feats to check = ['ft__plain_center_id.
↪MAX(fulfilment_center_info.op_area)', 'ft__plain_center_id.MAX(fulfilment_center_info.
↪city_code WHERE center_type = TYPE_A)', 'ft__plain_meal_id.COUNT(meal_info WHERE␣
↪cuisine = Italian)', 'ft__plain_meal_id.COUNT(meal_info WHERE cuisine = Continental)',
↪'ft__plain_center_id.MAX(fulfilment_center_info.op_area WHERE center_type = TYPE_B)',
↪'ft__plain_meal_id.COUNT(meal_info WHERE cuisine = Indian)', 'ft__plain_center_id.
↪MAX(fulfilment_center_info.city_code)', 'ft__plain_center_id.MAX(fulfilment_center_
↪info.op_area WHERE center_type = TYPE_A)', 'ft__plain_center_id.MAX(fulfilment_center_
↪info.region_code WHERE center_type = TYPE_A)', 'ft__plain_meal_id.COUNT(meal_info␣
```

(continues on next page)

(continued from previous page)

```
↪WHERE cuisine = Thai)', 'ft__plain_center_id.MEDIAN(fulfilment_center_info.op_area)',
↪'ft__plain_center_id.MEAN(fulfilment_center_info.op_area)']
[17:03:24] Features in SCI = ['ft__plain_center_id.MAX(fulfilment_center_info.city_code␣
↪WHERE center_type = TYPE_A)', 'ft__plain_center_id.MAX(fulfilment_center_info.city_
↪code)', 'ft__plain_center_id.MAX(fulfilment_center_info.op_area WHERE center_type =␣
↪TYPE_A)', 'ft__plain_center_id.MAX(fulfilment_center_info.op_area WHERE center_type =␣
↪TYPE_B)', 'ft__plain_center_id.MAX(fulfilment_center_info.op_area)', 'ft__plain_center_
↪id.MAX(fulfilment_center_info.region_code WHERE center_type = TYPE_A)', 'ft__plain_
↪center_id.MEAN(fulfilment_center_info.op_area)', 'ft__plain_center_id.
↪MEDIAN(fulfilment_center_info.op_area)', 'ft__plain_meal_id.COUNT(meal_info WHERE␣
↪cuisine = Continental)', 'ft__plain_meal_id.COUNT(meal_info WHERE cuisine = Indian)',
↪'ft__plain_meal_id.COUNT(meal_info WHERE cuisine = Italian)', 'ft__plain_meal_id.
↪COUNT(meal_info WHERE cuisine = Thai)']
[17:03:24] Training until validation scores don't improve for 100 rounds
[17:03:24] [100]        valid's l1: 208.366
[17:03:25] [200]        valid's l1: 208.36
[17:03:25] Early stopping, best iteration is:
[125]   valid's l1: 208.044
[17:03:25] LightGBM fitting and predicting completed
[17:03:25] Update best score from None to -208.04352023579347
[17:03:25] Started iteration 1, chunk = ['ft__plain_center_id.MAX(fulfilment_center_info.
↪city_code WHERE center_type = TYPE_B)', 'ft__plain_center_id.MAX(fulfilment_center_
↪info.op_area WHERE center_type = TYPE_C)', 'ft__plain_center_id.MAX(fulfilment_center_
↪info.region_code)', 'ft__plain_center_id.MAX(fulfilment_center_info.city_code WHERE␣
↪center_type = TYPE_C)', 'ft__plain_center_id.MEAN(fulfilment_center_info.op_area WHERE␣
↪center_type = TYPE_B)', 'ft__plain_center_id.MEAN(fulfilment_center_info.op_area WHERE␣
↪center_type = TYPE_A)', 'ft__plain_center_id.COUNT(fulfilment_center_info WHERE center_
↪type = TYPE_C)', 'ft__plain_center_id.MEAN(fulfilment_center_info.region_code WHERE␣
↪center_type = TYPE_A)', 'ft__plain_center_id.MEAN(fulfilment_center_info.city_code␣
↪WHERE center_type = TYPE_A)', 'ft__plain_center_id.MEAN(fulfilment_center_info.city_
↪code)', 'ft__plain_center_id.MAX(fulfilment_center_info.region_code WHERE center_type␣
↪= TYPE_C)', 'ft__plain_center_id.MEDIAN(fulfilment_center_info.city_code WHERE center_
↪type = TYPE_A)'], feats to check = ['ft__plain_center_id.MAX(fulfilment_center_info.op_
↪area)', 'ft__plain_center_id.MAX(fulfilment_center_info.city_code WHERE center_type =␣
↪TYPE_A)', 'ft__plain_meal_id.COUNT(meal_info WHERE cuisine = Italian)', 'ft__plain_
↪meal_id.COUNT(meal_info WHERE cuisine = Continental)', 'ft__plain_center_id.
↪MAX(fulfilment_center_info.op_area WHERE center_type = TYPE_B)', 'ft__plain_meal_id.
↪COUNT(meal_info WHERE cuisine = Indian)', 'ft__plain_center_id.MAX(fulfilment_center_
↪info.city_code)', 'ft__plain_center_id.MAX(fulfilment_center_info.op_area WHERE center_
↪type = TYPE_A)', 'ft__plain_center_id.MAX(fulfilment_center_info.region_code WHERE␣
↪center_type = TYPE_A)', 'ft__plain_meal_id.COUNT(meal_info WHERE cuisine = Thai)', 'ft_
↪_plain_center_id.MEDIAN(fulfilment_center_info.op_area)', 'ft__plain_center_id.
↪MEAN(fulfilment_center_info.op_area)', 'ft__plain_center_id.MAX(fulfilment_center_info.
↪city_code WHERE center_type = TYPE_B)', 'ft__plain_center_id.MAX(fulfilment_center_
↪info.op_area WHERE center_type = TYPE_C)', 'ft__plain_center_id.MAX(fulfilment_center_
↪info.region_code)', 'ft__plain_center_id.MAX(fulfilment_center_info.city_code WHERE␣
↪center_type = TYPE_C)', 'ft__plain_center_id.MEAN(fulfilment_center_info.op_area WHERE␣
↪center_type = TYPE_B)', 'ft__plain_center_id.MEAN(fulfilment_center_info.op_area WHERE␣
↪center_type = TYPE_A)', 'ft__plain_center_id.COUNT(fulfilment_center_info WHERE center_
↪type = TYPE_C)', 'ft__plain_center_id.MEAN(fulfilment_center_info.region_code WHERE␣
↪center_type = TYPE_A)', 'ft__plain_center_id.MEAN(fulfilment_center_info.city_code␣
↪WHERE center_type = TYPE_A)', 'ft__plain_center_id.MEAN(fulfilment_center_info.city_
```

(continues on next page)

(continued from previous page)

```
↪code)', 'ft__plain_center_id.MAX(fulfilment_center_info.region_code WHERE center_type␣
↪= TYPE_C)', 'ft__plain_center_id.MEDIAN(fulfilment_center_info.city_code WHERE center_
↪type = TYPE_A)']
[17:03:25] Features in SCI = ['ft__plain_center_id.COUNT(fulfilment_center_info WHERE␣
↪center_type = TYPE_C)', 'ft__plain_center_id.MAX(fulfilment_center_info.city_code␣
↪WHERE center_type = TYPE_A)', 'ft__plain_center_id.MAX(fulfilment_center_info.city_
↪code WHERE center_type = TYPE_B)', 'ft__plain_center_id.MAX(fulfilment_center_info.
↪city_code WHERE center_type = TYPE_C)', 'ft__plain_center_id.MAX(fulfilment_center_
↪info.city_code)', 'ft__plain_center_id.MAX(fulfilment_center_info.op_area WHERE center_
↪type = TYPE_A)', 'ft__plain_center_id.MAX(fulfilment_center_info.op_area WHERE center_
↪type = TYPE_B)', 'ft__plain_center_id.MAX(fulfilment_center_info.op_area WHERE center_
↪type = TYPE_C)', 'ft__plain_center_id.MAX(fulfilment_center_info.op_area)', 'ft__plain_
↪center_id.MAX(fulfilment_center_info.region_code WHERE center_type = TYPE_A)', 'ft__
↪plain_center_id.MAX(fulfilment_center_info.region_code WHERE center_type = TYPE_C)',
↪'ft__plain_center_id.MAX(fulfilment_center_info.region_code)', 'ft__plain_center_id.
↪MEAN(fulfilment_center_info.city_code WHERE center_type = TYPE_A)', 'ft__plain_center_
↪id.MEAN(fulfilment_center_info.city_code)', 'ft__plain_center_id.MEAN(fulfilment_
↪center_info.op_area WHERE center_type = TYPE_A)', 'ft__plain_center_id.MEAN(fulfilment_
↪center_info.op_area WHERE center_type = TYPE_B)', 'ft__plain_center_id.MEAN(fulfilment_
↪center_info.op_area)', 'ft__plain_center_id.MEAN(fulfilment_center_info.region_code␣
↪WHERE center_type = TYPE_A)', 'ft__plain_center_id.MEDIAN(fulfilment_center_info.city_
↪code WHERE center_type = TYPE_A)', 'ft__plain_center_id.MEDIAN(fulfilment_center_info.
↪op_area)', 'ft__plain_meal_id.COUNT(meal_info WHERE cuisine = Continental)', 'ft__
↪plain_meal_id.COUNT(meal_info WHERE cuisine = Indian)', 'ft__plain_meal_id.COUNT(meal_
↪info WHERE cuisine = Italian)', 'ft__plain_meal_id.COUNT(meal_info WHERE cuisine =␣
↪Thai)']
[17:03:25] Training until validation scores don't improve for 100 rounds
[17:03:26] [100]        valid's l1: 208.434
[17:03:26] [200]        valid's l1: 208.345
[17:03:27] Early stopping, best iteration is:
[125]   valid's l1: 208.067
[17:03:27] LightGBM fitting and predicting completed
[17:03:27] Started iteration 2, chunk = ['ft__plain_center_id.MAX(fulfilment_center_info.
↪region_code WHERE center_type = TYPE_B)', 'ft__plain_center_id.MEAN(fulfilment_center_
↪info.region_code)', 'ft__plain_center_id.MEDIAN(fulfilment_center_info.city_code)',
↪'ft__plain_center_id.MEAN(fulfilment_center_info.op_area WHERE center_type = TYPE_C)',
↪'ft__plain_center_id.COUNT(fulfilment_center_info WHERE center_type = TYPE_A)', 'ft__
↪plain_center_id.MEAN(fulfilment_center_info.city_code WHERE center_type = TYPE_B)',
↪'ft__plain_center_id.MEDIAN(fulfilment_center_info.region_code WHERE center_type =␣
↪TYPE_A)', 'ft__plain_center_id.MEDIAN(fulfilment_center_info.op_area WHERE center_type␣
↪= TYPE_A)', 'ft__plain_center_id.MEAN(fulfilment_center_info.city_code WHERE center_
↪type = TYPE_C)', 'ft__plain_center_id.MIN(fulfilment_center_info.op_area WHERE center_
↪type = TYPE_B)', 'ft__plain_center_id.MIN(fulfilment_center_info.city_code)', 'ft__
↪plain_center_id.MEDIAN(fulfilment_center_info.op_area WHERE center_type = TYPE_B)'],␣
↪feats to check = ['ft__plain_center_id.MAX(fulfilment_center_info.op_area)', 'ft__
↪plain_center_id.MAX(fulfilment_center_info.city_code WHERE center_type = TYPE_A)', 'ft_
↪_plain_meal_id.COUNT(meal_info WHERE cuisine = Italian)', 'ft__plain_meal_id.
↪COUNT(meal_info WHERE cuisine = Continental)', 'ft__plain_center_id.MAX(fulfilment_
↪center_info.op_area WHERE center_type = TYPE_B)', 'ft__plain_meal_id.COUNT(meal_info␣
↪WHERE cuisine = Indian)', 'ft__plain_center_id.MAX(fulfilment_center_info.city_code)',
↪'ft__plain_center_id.MAX(fulfilment_center_info.op_area WHERE center_type = TYPE_A)',
↪'ft__plain_center_id.MAX(fulfilment_center_info.region_code WHERE center_type = TYPE_A)
```

(continues on next page)

(continued from previous page)

```
↪', 'ft__plain_meal_id.COUNT(meal_info WHERE cuisine = Thai)', 'ft__plain_center_id.
↪MEDIAN(fulfilment_center_info.op_area)', 'ft__plain_center_id.MEAN(fulfilment_center_
↪info.op_area)', 'ft__plain_center_id.MAX(fulfilment_center_info.region_code WHERE␣
↪center_type = TYPE_B)', 'ft__plain_center_id.MEAN(fulfilment_center_info.region_code)',
↪ 'ft__plain_center_id.MEDIAN(fulfilment_center_info.city_code)', 'ft__plain_center_id.
↪MEAN(fulfilment_center_info.op_area WHERE center_type = TYPE_C)', 'ft__plain_center_id.
↪COUNT(fulfilment_center_info WHERE center_type = TYPE_A)', 'ft__plain_center_id.
↪MEAN(fulfilment_center_info.city_code WHERE center_type = TYPE_B)', 'ft__plain_center_
↪id.MEDIAN(fulfilment_center_info.region_code WHERE center_type = TYPE_A)', 'ft__plain_
↪center_id.MEDIAN(fulfilment_center_info.op_area WHERE center_type = TYPE_A)', 'ft__
↪plain_center_id.MEAN(fulfilment_center_info.city_code WHERE center_type = TYPE_C)',
↪'ft__plain_center_id.MIN(fulfilment_center_info.op_area WHERE center_type = TYPE_B)',
↪'ft__plain_center_id.MIN(fulfilment_center_info.city_code)', 'ft__plain_center_id.
↪MEDIAN(fulfilment_center_info.op_area WHERE center_type = TYPE_B)']
[17:03:27] Features in SCI = ['ft__plain_center_id.COUNT(fulfilment_center_info WHERE␣
↪center_type = TYPE_A)', 'ft__plain_center_id.MAX(fulfilment_center_info.city_code␣
↪WHERE center_type = TYPE_A)', 'ft__plain_center_id.MAX(fulfilment_center_info.city_
↪code)', 'ft__plain_center_id.MAX(fulfilment_center_info.op_area WHERE center_type =␣
↪TYPE_A)', 'ft__plain_center_id.MAX(fulfilment_center_info.op_area WHERE center_type =␣
↪TYPE_B)', 'ft__plain_center_id.MAX(fulfilment_center_info.op_area)', 'ft__plain_center_
↪id.MAX(fulfilment_center_info.region_code WHERE center_type = TYPE_A)', 'ft__plain_
↪center_id.MAX(fulfilment_center_info.region_code WHERE center_type = TYPE_B)', 'ft__
↪plain_center_id.MEAN(fulfilment_center_info.city_code WHERE center_type = TYPE_B)',
↪'ft__plain_center_id.MEAN(fulfilment_center_info.city_code WHERE center_type = TYPE_C)
↪', 'ft__plain_center_id.MEAN(fulfilment_center_info.op_area WHERE center_type = TYPE_C)
↪', 'ft__plain_center_id.MEAN(fulfilment_center_info.op_area)', 'ft__plain_center_id.
↪MEAN(fulfilment_center_info.region_code)', 'ft__plain_center_id.MEDIAN(fulfilment_
↪center_info.city_code)', 'ft__plain_center_id.MEDIAN(fulfilment_center_info.op_area␣
↪WHERE center_type = TYPE_A)', 'ft__plain_center_id.MEDIAN(fulfilment_center_info.op_
↪area WHERE center_type = TYPE_B)', 'ft__plain_center_id.MEDIAN(fulfilment_center_info.
↪op_area)', 'ft__plain_center_id.MEDIAN(fulfilment_center_info.region_code WHERE center_
↪type = TYPE_A)', 'ft__plain_center_id.MIN(fulfilment_center_info.city_code)', 'ft__
↪plain_center_id.MIN(fulfilment_center_info.op_area WHERE center_type = TYPE_B)', 'ft__
↪plain_meal_id.COUNT(meal_info WHERE cuisine = Continental)', 'ft__plain_meal_id.
↪COUNT(meal_info WHERE cuisine = Indian)', 'ft__plain_meal_id.COUNT(meal_info WHERE␣
↪cuisine = Italian)', 'ft__plain_meal_id.COUNT(meal_info WHERE cuisine = Thai)']
[17:03:27] Training until validation scores don't improve for 100 rounds
[17:03:27] [100]        valid's l1: 208.433
[17:03:28] [200]        valid's l1: 208.34
[17:03:28] Early stopping, best iteration is:
[125]   valid's l1: 208.057
[17:03:28] LightGBM fitting and predicting completed
[17:03:28] Started iteration 3, chunk = ['ft__plain_center_id.COUNT(fulfilment_center_
↪info WHERE center_type = TYPE_B)', 'ft__plain_center_id.MIN(fulfilment_center_info.op_
↪area WHERE center_type = TYPE_A)', 'ft__plain_center_id.MEDIAN(fulfilment_center_info.
↪city_code WHERE center_type = TYPE_B)', 'ft__plain_center_id.MEDIAN(fulfilment_center_
↪info.op_area WHERE center_type = TYPE_C)', 'ft__plain_center_id.MEDIAN(fulfilment_
↪center_info.region_code)', 'ft__plain_center_id.MIN(fulfilment_center_info.op_area)',
↪'ft__plain_center_id.SUM(fulfilment_center_info.city_code WHERE center_type = TYPE_A)',
↪ 'ft__plain_center_id.MIN(fulfilment_center_info.city_code WHERE center_type = TYPE_B)
↪', 'ft__plain_center_id.MEAN(fulfilment_center_info.region_code WHERE center_type =␣
↪TYPE_C)', 'ft__plain_center_id.MEDIAN(fulfilment_center_info.city_code WHERE center_
```

(continues on next page)

(continued from previous page)

```
↪type = TYPE_C)', 'ft__plain_center_id.SUM(fulfilment_center_info.city_code)', 'ft__
↪plain_center_id.MIN(fulfilment_center_info.city_code WHERE center_type = TYPE_C)'],␣
↪feats to check = ['ft__plain_center_id.MAX(fulfilment_center_info.op_area)', 'ft__
↪plain_center_id.MAX(fulfilment_center_info.city_code WHERE center_type = TYPE_A)', 'ft_
↪_plain_meal_id.COUNT(meal_info WHERE cuisine = Italian)', 'ft__plain_meal_id.
↪COUNT(meal_info WHERE cuisine = Continental)', 'ft__plain_center_id.MAX(fulfilment_
↪center_info.op_area WHERE center_type = TYPE_B)', 'ft__plain_meal_id.COUNT(meal_info␣
↪WHERE cuisine = Indian)', 'ft__plain_center_id.MAX(fulfilment_center_info.city_code)',
↪'ft__plain_center_id.MAX(fulfilment_center_info.op_area WHERE center_type = TYPE_A)',
↪'ft__plain_center_id.MAX(fulfilment_center_info.region_code WHERE center_type = TYPE_A)
↪', 'ft__plain_meal_id.COUNT(meal_info WHERE cuisine = Thai)', 'ft__plain_center_id.
↪MEDIAN(fulfilment_center_info.op_area)', 'ft__plain_center_id.MEAN(fulfilment_center_
↪info.op_area)', 'ft__plain_center_id.COUNT(fulfilment_center_info WHERE center_type =␣
↪TYPE_B)', 'ft__plain_center_id.MIN(fulfilment_center_info.op_area WHERE center_type =␣
↪TYPE_A)', 'ft__plain_center_id.MEDIAN(fulfilment_center_info.city_code WHERE center_
↪type = TYPE_B)', 'ft__plain_center_id.MEDIAN(fulfilment_center_info.op_area WHERE␣
↪center_type = TYPE_C)', 'ft__plain_center_id.MEDIAN(fulfilment_center_info.region_code)
↪', 'ft__plain_center_id.MIN(fulfilment_center_info.op_area)', 'ft__plain_center_id.
↪SUM(fulfilment_center_info.city_code WHERE center_type = TYPE_A)', 'ft__plain_center_
↪id.MIN(fulfilment_center_info.city_code WHERE center_type = TYPE_B)', 'ft__plain_
↪center_id.MEAN(fulfilment_center_info.region_code WHERE center_type = TYPE_C)', 'ft__
↪plain_center_id.MEDIAN(fulfilment_center_info.city_code WHERE center_type = TYPE_C)',
↪'ft__plain_center_id.SUM(fulfilment_center_info.city_code)', 'ft__plain_center_id.
↪MIN(fulfilment_center_info.city_code WHERE center_type = TYPE_C)']
[17:03:28] Features in SCI = ['ft__plain_center_id.COUNT(fulfilment_center_info WHERE␣
↪center_type = TYPE_B)', 'ft__plain_center_id.MAX(fulfilment_center_info.city_code␣
↪WHERE center_type = TYPE_A)', 'ft__plain_center_id.MAX(fulfilment_center_info.city_
↪code)', 'ft__plain_center_id.MAX(fulfilment_center_info.op_area WHERE center_type =␣
↪TYPE_A)', 'ft__plain_center_id.MAX(fulfilment_center_info.op_area WHERE center_type =␣
↪TYPE_B)', 'ft__plain_center_id.MAX(fulfilment_center_info.op_area)', 'ft__plain_center_
↪id.MAX(fulfilment_center_info.region_code WHERE center_type = TYPE_A)', 'ft__plain_
↪center_id.MEAN(fulfilment_center_info.op_area)', 'ft__plain_center_id.MEAN(fulfilment_
↪center_info.region_code WHERE center_type = TYPE_C)', 'ft__plain_center_id.
↪MEDIAN(fulfilment_center_info.city_code WHERE center_type = TYPE_B)', 'ft__plain_
↪center_id.MEDIAN(fulfilment_center_info.city_code WHERE center_type = TYPE_C)', 'ft__
↪plain_center_id.MEDIAN(fulfilment_center_info.op_area WHERE center_type = TYPE_C)',
↪'ft__plain_center_id.MEDIAN(fulfilment_center_info.op_area)', 'ft__plain_center_id.
↪MEDIAN(fulfilment_center_info.region_code)', 'ft__plain_center_id.MIN(fulfilment_
↪center_info.city_code WHERE center_type = TYPE_B)', 'ft__plain_center_id.
↪MIN(fulfilment_center_info.city_code WHERE center_type = TYPE_C)', 'ft__plain_center_
↪id.MIN(fulfilment_center_info.op_area WHERE center_type = TYPE_A)', 'ft__plain_center_
↪id.MIN(fulfilment_center_info.op_area)', 'ft__plain_center_id.SUM(fulfilment_center_
↪info.city_code WHERE center_type = TYPE_A)', 'ft__plain_center_id.SUM(fulfilment_
↪center_info.city_code)', 'ft__plain_meal_id.COUNT(meal_info WHERE cuisine =␣
↪Continental)', 'ft__plain_meal_id.COUNT(meal_info WHERE cuisine = Indian)', 'ft__plain_
↪meal_id.COUNT(meal_info WHERE cuisine = Italian)', 'ft__plain_meal_id.COUNT(meal_info␣
↪WHERE cuisine = Thai)']
[17:03:28] Training until validation scores don't improve for 100 rounds
[17:03:29] [100]        valid's l1: 208.432
[17:03:29] [200]        valid's l1: 208.338
[17:03:30] Early stopping, best iteration is:
[125]   valid's l1: 208.095
```

(continues on next page)

(continued from previous page)

```
[17:03:30] LightGBM fitting and predicting completed
[17:03:30] Started iteration 4, chunk = ['ft__plain_center_id.MEAN(fulfilment_center_
↪info.region_code WHERE center_type = TYPE_B)', 'ft__plain_center_id.MEDIAN(fulfilment_
↪center_info.region_code WHERE center_type = TYPE_C)', 'ft__plain_center_id.
↪MEDIAN(fulfilment_center_info.region_code WHERE center_type = TYPE_B)', 'ft__plain_
↪center_id.MIN(fulfilment_center_info.op_area WHERE center_type = TYPE_C)', 'ft__plain_
↪center_id.SUM(fulfilment_center_info.op_area WHERE center_type = TYPE_A)', 'ft__plain_
↪center_id.MIN(fulfilment_center_info.region_code)', 'ft__plain_meal_id.COUNT(meal_info)
↪', 'ft__plain_center_id.SUM(fulfilment_center_info.op_area)', 'ft__plain_center_id.
↪SUM(fulfilment_center_info.city_code WHERE center_type = TYPE_C)', 'ft__plain_center_
↪id.SUM(fulfilment_center_info.op_area WHERE center_type = TYPE_B)', 'ft__plain_center_
↪id.SUM(fulfilment_center_info.op_area WHERE center_type = TYPE_C)', 'ft__plain_center_
↪id.SUM(fulfilment_center_info.region_code WHERE center_type = TYPE_B)'], feats to␣
↪check = ['ft__plain_center_id.MAX(fulfilment_center_info.op_area)', 'ft__plain_center_
↪id.MAX(fulfilment_center_info.city_code WHERE center_type = TYPE_A)', 'ft__plain_meal_
↪id.COUNT(meal_info WHERE cuisine = Italian)', 'ft__plain_meal_id.COUNT(meal_info WHERE␣
↪cuisine = Continental)', 'ft__plain_center_id.MAX(fulfilment_center_info.op_area WHERE␣
↪center_type = TYPE_B)', 'ft__plain_meal_id.COUNT(meal_info WHERE cuisine = Indian)',
↪'ft__plain_center_id.MAX(fulfilment_center_info.city_code)', 'ft__plain_center_id.
↪MAX(fulfilment_center_info.op_area WHERE center_type = TYPE_A)', 'ft__plain_center_id.
↪MAX(fulfilment_center_info.region_code WHERE center_type = TYPE_A)', 'ft__plain_meal_
↪id.COUNT(meal_info WHERE cuisine = Thai)', 'ft__plain_center_id.MEDIAN(fulfilment_
↪center_info.op_area)', 'ft__plain_center_id.MEAN(fulfilment_center_info.op_area)', 'ft_
↪_plain_center_id.MEAN(fulfilment_center_info.region_code WHERE center_type = TYPE_B)',
↪'ft__plain_center_id.MEDIAN(fulfilment_center_info.region_code WHERE center_type =␣
↪TYPE_C)', 'ft__plain_center_id.MEDIAN(fulfilment_center_info.region_code WHERE center_
↪type = TYPE_B)', 'ft__plain_center_id.MIN(fulfilment_center_info.op_area WHERE center_
↪type = TYPE_C)', 'ft__plain_center_id.SUM(fulfilment_center_info.op_area WHERE center_
↪type = TYPE_A)', 'ft__plain_center_id.MIN(fulfilment_center_info.region_code)', 'ft__
↪plain_meal_id.COUNT(meal_info)', 'ft__plain_center_id.SUM(fulfilment_center_info.op_
↪area)', 'ft__plain_center_id.SUM(fulfilment_center_info.city_code WHERE center_type =␣
↪TYPE_C)', 'ft__plain_center_id.SUM(fulfilment_center_info.op_area WHERE center_type =␣
↪TYPE_B)', 'ft__plain_center_id.SUM(fulfilment_center_info.op_area WHERE center_type =␣
↪TYPE_C)', 'ft__plain_center_id.SUM(fulfilment_center_info.region_code WHERE center_
↪type = TYPE_B)']
[17:03:30] Features in SCI = ['ft__plain_center_id.MAX(fulfilment_center_info.city_code␣
↪WHERE center_type = TYPE_A)', 'ft__plain_center_id.MAX(fulfilment_center_info.city_
↪code)', 'ft__plain_center_id.MAX(fulfilment_center_info.op_area WHERE center_type =␣
↪TYPE_A)', 'ft__plain_center_id.MAX(fulfilment_center_info.op_area WHERE center_type =␣
↪TYPE_B)', 'ft__plain_center_id.MAX(fulfilment_center_info.op_area)', 'ft__plain_center_
↪id.MAX(fulfilment_center_info.region_code WHERE center_type = TYPE_A)', 'ft__plain_
↪center_id.MEAN(fulfilment_center_info.op_area)', 'ft__plain_center_id.MEAN(fulfilment_
↪center_info.region_code WHERE center_type = TYPE_B)', 'ft__plain_center_id.
↪MEDIAN(fulfilment_center_info.op_area)', 'ft__plain_center_id.MEDIAN(fulfilment_center_
↪info.region_code WHERE center_type = TYPE_B)', 'ft__plain_center_id.MEDIAN(fulfilment_
↪center_info.region_code WHERE center_type = TYPE_C)', 'ft__plain_center_id.
↪MIN(fulfilment_center_info.op_area WHERE center_type = TYPE_C)', 'ft__plain_center_id.
↪MIN(fulfilment_center_info.region_code)', 'ft__plain_center_id.SUM(fulfilment_center_
↪info.city_code WHERE center_type = TYPE_C)', 'ft__plain_center_id.SUM(fulfilment_
↪center_info.op_area WHERE center_type = TYPE_A)', 'ft__plain_center_id.SUM(fulfilment_
↪center_info.op_area WHERE center_type = TYPE_B)', 'ft__plain_center_id.SUM(fulfilment_
↪center_info.op_area WHERE center_type = TYPE_C)', 'ft__plain_center_id.SUM(fulfilment_
```

(continues on next page)

(continued from previous page)

```
↪center_info.op_area)', 'ft__plain_center_id.SUM(fulfilment_center_info.region_code␣
↪WHERE center_type = TYPE_B)', 'ft__plain_meal_id.COUNT(meal_info WHERE cuisine =␣
↪Continental)', 'ft__plain_meal_id.COUNT(meal_info WHERE cuisine = Indian)', 'ft__plain_
↪meal_id.COUNT(meal_info WHERE cuisine = Italian)', 'ft__plain_meal_id.COUNT(meal_info␣
↪WHERE cuisine = Thai)', 'ft__plain_meal_id.COUNT(meal_info)']
[17:03:30] Training until validation scores don't improve for 100 rounds
[17:03:30] [100]        valid's l1: 208.386
[17:03:31] [200]        valid's l1: 208.392
[17:03:31] Early stopping, best iteration is:
[125]   valid's l1: 208.077
[17:03:31] LightGBM fitting and predicting completed
[17:03:31] Started iteration 5, chunk = ['ft__plain_center_id.SUM(fulfilment_center_info.
↪region_code WHERE center_type = TYPE_A)', 'ft__plain_center_id.COUNT(fulfilment_center_
↪info)', 'ft__plain_center_id.SUM(fulfilment_center_info.region_code WHERE center_type␣
↪= TYPE_C)', 'ft__plain_center_id.SUM(fulfilment_center_info.city_code WHERE center_
↪type = TYPE_B)', 'ft__plain_meal_id.ENTROPY(meal_info.cuisine)', 'ft__plain_center_id.
↪ENTROPY(fulfilment_center_info.center_type)', 'ft__plain_center_id.SUM(fulfilment_
↪center_info.region_code)', 'ft__plain_center_id.STD(fulfilment_center_info.region_code␣
↪WHERE center_type = TYPE_C)', 'ft__plain_center_id.STD(fulfilment_center_info.region_
↪code)', 'ft__plain_center_id.STD(fulfilment_center_info.city_code WHERE center_type =␣
↪TYPE_C)', 'ft__plain_center_id.MIN(fulfilment_center_info.region_code WHERE center_
↪type = TYPE_A)', 'ft__plain_center_id.MIN(fulfilment_center_info.region_code WHERE␣
↪center_type = TYPE_B)'], feats to check = ['ft__plain_center_id.MAX(fulfilment_center_
↪info.op_area)', 'ft__plain_center_id.MAX(fulfilment_center_info.city_code WHERE center_
↪type = TYPE_A)', 'ft__plain_meal_id.COUNT(meal_info WHERE cuisine = Italian)', 'ft__
↪plain_meal_id.COUNT(meal_info WHERE cuisine = Continental)', 'ft__plain_center_id.
↪MAX(fulfilment_center_info.op_area WHERE center_type = TYPE_B)', 'ft__plain_meal_id.
↪COUNT(meal_info WHERE cuisine = Indian)', 'ft__plain_center_id.MAX(fulfilment_center_
↪info.city_code)', 'ft__plain_center_id.MAX(fulfilment_center_info.op_area WHERE center_
↪type = TYPE_A)', 'ft__plain_center_id.MAX(fulfilment_center_info.region_code WHERE␣
↪center_type = TYPE_A)', 'ft__plain_meal_id.COUNT(meal_info WHERE cuisine = Thai)', 'ft_
↪_plain_center_id.MEDIAN(fulfilment_center_info.op_area)', 'ft__plain_center_id.
↪MEAN(fulfilment_center_info.op_area)', 'ft__plain_center_id.SUM(fulfilment_center_info.
↪region_code WHERE center_type = TYPE_A)', 'ft__plain_center_id.COUNT(fulfilment_center_
↪info)', 'ft__plain_center_id.SUM(fulfilment_center_info.region_code WHERE center_type␣
↪= TYPE_C)', 'ft__plain_center_id.SUM(fulfilment_center_info.city_code WHERE center_
↪type = TYPE_B)', 'ft__plain_meal_id.ENTROPY(meal_info.cuisine)', 'ft__plain_center_id.
↪ENTROPY(fulfilment_center_info.center_type)', 'ft__plain_center_id.SUM(fulfilment_
↪center_info.region_code)', 'ft__plain_center_id.STD(fulfilment_center_info.region_code␣
↪WHERE center_type = TYPE_C)', 'ft__plain_center_id.STD(fulfilment_center_info.region_
↪code)', 'ft__plain_center_id.STD(fulfilment_center_info.city_code WHERE center_type =␣
↪TYPE_C)', 'ft__plain_center_id.MIN(fulfilment_center_info.region_code WHERE center_
↪type = TYPE_A)', 'ft__plain_center_id.MIN(fulfilment_center_info.region_code WHERE␣
↪center_type = TYPE_B)']
[17:03:31] Features in SCI = ['ft__plain_center_id.COUNT(fulfilment_center_info)', 'ft__
↪plain_center_id.ENTROPY(fulfilment_center_info.center_type)', 'ft__plain_center_id.
↪MAX(fulfilment_center_info.city_code WHERE center_type = TYPE_A)', 'ft__plain_center_
↪id.MAX(fulfilment_center_info.city_code)', 'ft__plain_center_id.MAX(fulfilment_center_
↪info.op_area WHERE center_type = TYPE_A)', 'ft__plain_center_id.MAX(fulfilment_center_
↪info.op_area WHERE center_type = TYPE_B)', 'ft__plain_center_id.MAX(fulfilment_center_
↪info.op_area)', 'ft__plain_center_id.MAX(fulfilment_center_info.region_code WHERE␣
↪center_type = TYPE_A)', 'ft__plain_center_id.MEAN(fulfilment_center_info.op_area)',
```

(continues on next page)

(continued from previous page)

```
↪'ft__plain_center_id.MEDIAN(fulfilment_center_info.op_area)', 'ft__plain_center_id.
↪MIN(fulfilment_center_info.region_code WHERE center_type = TYPE_A)', 'ft__plain_center_
↪id.MIN(fulfilment_center_info.region_code WHERE center_type = TYPE_B)', 'ft__plain_
↪center_id.STD(fulfilment_center_info.city_code WHERE center_type = TYPE_C)', 'ft__
↪plain_center_id.STD(fulfilment_center_info.region_code WHERE center_type = TYPE_C)',
↪'ft__plain_center_id.STD(fulfilment_center_info.region_code)', 'ft__plain_center_id.
↪SUM(fulfilment_center_info.city_code WHERE center_type = TYPE_B)', 'ft__plain_center_
↪id.SUM(fulfilment_center_info.region_code WHERE center_type = TYPE_A)', 'ft__plain_
↪center_id.SUM(fulfilment_center_info.region_code WHERE center_type = TYPE_C)', 'ft__
↪plain_center_id.SUM(fulfilment_center_info.region_code)', 'ft__plain_meal_id.
↪COUNT(meal_info WHERE cuisine = Continental)', 'ft__plain_meal_id.COUNT(meal_info␣
↪WHERE cuisine = Indian)', 'ft__plain_meal_id.COUNT(meal_info WHERE cuisine = Italian)',
↪ 'ft__plain_meal_id.COUNT(meal_info WHERE cuisine = Thai)', 'ft__plain_meal_id.
↪ENTROPY(meal_info.cuisine)']
[17:03:31] Training until validation scores don't improve for 100 rounds
[17:03:32] [100]        valid's l1: 208.447
[17:03:32] [200]        valid's l1: 208.39
[17:03:33] Early stopping, best iteration is:
[125]   valid's l1: 208.101
[17:03:33] LightGBM fitting and predicting completed
[17:03:33] Started iteration 6, chunk = ['ft__plain_center_id.MIN(fulfilment_center_info.
↪region_code WHERE center_type = TYPE_C)', 'ft__plain_center_id.NUM_UNIQUE(fulfilment_
↪center_info.center_type)', 'ft__plain_center_id.STD(fulfilment_center_info.city_code␣
↪WHERE center_type = TYPE_A)', 'ft__plain_center_id.STD(fulfilment_center_info.city_
↪code WHERE center_type = TYPE_B)', 'ft__plain_center_id.STD(fulfilment_center_info.
↪city_code)', 'ft__plain_center_id.MIN(fulfilment_center_info.city_code WHERE center_
↪type = TYPE_A)', 'ft__plain_center_id.STD(fulfilment_center_info.op_area WHERE center_
↪type = TYPE_A)', 'ft__plain_center_id.STD(fulfilment_center_info.op_area WHERE center_
↪type = TYPE_B)', 'ft__plain_center_id.STD(fulfilment_center_info.op_area WHERE center_
↪type = TYPE_C)', 'ft__plain_center_id.STD(fulfilment_center_info.op_area)', 'ft__plain_
↪center_id.STD(fulfilment_center_info.region_code WHERE center_type = TYPE_A)', 'ft__
↪plain_center_id.STD(fulfilment_center_info.region_code WHERE center_type = TYPE_B)'],␣
↪feats to check = ['ft__plain_center_id.MAX(fulfilment_center_info.op_area)', 'ft__
↪plain_center_id.MAX(fulfilment_center_info.city_code WHERE center_type = TYPE_A)', 'ft_
↪_plain_meal_id.COUNT(meal_info WHERE cuisine = Italian)', 'ft__plain_meal_id.
↪COUNT(meal_info WHERE cuisine = Continental)', 'ft__plain_center_id.MAX(fulfilment_
↪center_info.op_area WHERE center_type = TYPE_B)', 'ft__plain_meal_id.COUNT(meal_info␣
↪WHERE cuisine = Indian)', 'ft__plain_center_id.MAX(fulfilment_center_info.city_code)',
↪'ft__plain_center_id.MAX(fulfilment_center_info.op_area WHERE center_type = TYPE_A)',
↪'ft__plain_center_id.MAX(fulfilment_center_info.region_code WHERE center_type = TYPE_A)
↪', 'ft__plain_meal_id.COUNT(meal_info WHERE cuisine = Thai)', 'ft__plain_center_id.
↪MEDIAN(fulfilment_center_info.op_area)', 'ft__plain_center_id.MEAN(fulfilment_center_
↪info.op_area)', 'ft__plain_center_id.MIN(fulfilment_center_info.region_code WHERE␣
↪center_type = TYPE_C)', 'ft__plain_center_id.NUM_UNIQUE(fulfilment_center_info.center_
↪type)', 'ft__plain_center_id.STD(fulfilment_center_info.city_code WHERE center_type =␣
↪TYPE_A)', 'ft__plain_center_id.STD(fulfilment_center_info.city_code WHERE center_type␣
↪= TYPE_B)', 'ft__plain_center_id.STD(fulfilment_center_info.city_code)', 'ft__plain_
↪center_id.MIN(fulfilment_center_info.city_code WHERE center_type = TYPE_A)', 'ft__
↪plain_center_id.STD(fulfilment_center_info.op_area WHERE center_type = TYPE_A)', 'ft__
↪plain_center_id.STD(fulfilment_center_info.op_area WHERE center_type = TYPE_B)', 'ft__
↪plain_center_id.STD(fulfilment_center_info.op_area WHERE center_type = TYPE_C)', 'ft__
↪plain_center_id.STD(fulfilment_center_info.op_area)', 'ft__plain_center_id.
```

(continues on next page)

(continued from previous page)

```
↪STD(fulfilment_center_info.region_code WHERE center_type = TYPE_A)', 'ft__plain_center_
↪id.STD(fulfilment_center_info.region_code WHERE center_type = TYPE_B)']
[17:03:33] Features in SCI = ['ft__plain_center_id.MAX(fulfilment_center_info.city_code␣
↪WHERE center_type = TYPE_A)', 'ft__plain_center_id.MAX(fulfilment_center_info.city_
↪code)', 'ft__plain_center_id.MAX(fulfilment_center_info.op_area WHERE center_type =␣
↪TYPE_A)', 'ft__plain_center_id.MAX(fulfilment_center_info.op_area WHERE center_type =␣
↪TYPE_B)', 'ft__plain_center_id.MAX(fulfilment_center_info.op_area)', 'ft__plain_center_
↪id.MAX(fulfilment_center_info.region_code WHERE center_type = TYPE_A)', 'ft__plain_
↪center_id.MEAN(fulfilment_center_info.op_area)', 'ft__plain_center_id.
↪MEDIAN(fulfilment_center_info.op_area)', 'ft__plain_center_id.MIN(fulfilment_center_
↪info.city_code WHERE center_type = TYPE_A)', 'ft__plain_center_id.MIN(fulfilment_
↪center_info.region_code WHERE center_type = TYPE_C)', 'ft__plain_center_id.NUM_
↪UNIQUE(fulfilment_center_info.center_type)', 'ft__plain_center_id.STD(fulfilment_
↪center_info.city_code WHERE center_type = TYPE_A)', 'ft__plain_center_id.
↪STD(fulfilment_center_info.city_code WHERE center_type = TYPE_B)', 'ft__plain_center_
↪id.STD(fulfilment_center_info.city_code)', 'ft__plain_center_id.STD(fulfilment_center_
↪info.op_area WHERE center_type = TYPE_A)', 'ft__plain_center_id.STD(fulfilment_center_
↪info.op_area WHERE center_type = TYPE_B)', 'ft__plain_center_id.STD(fulfilment_center_
↪info.op_area WHERE center_type = TYPE_C)', 'ft__plain_center_id.STD(fulfilment_center_
↪info.op_area)', 'ft__plain_center_id.STD(fulfilment_center_info.region_code WHERE␣
↪center_type = TYPE_A)', 'ft__plain_center_id.STD(fulfilment_center_info.region_code␣
↪WHERE center_type = TYPE_B)', 'ft__plain_meal_id.COUNT(meal_info WHERE cuisine =␣
↪Continental)', 'ft__plain_meal_id.COUNT(meal_info WHERE cuisine = Indian)', 'ft__plain_
↪meal_id.COUNT(meal_info WHERE cuisine = Italian)', 'ft__plain_meal_id.COUNT(meal_info␣
↪WHERE cuisine = Thai)']
[17:03:33] Training until validation scores don't improve for 100 rounds
[17:03:33] [100]        valid's l1: 208.41
[17:03:34] [200]        valid's l1: 208.413
[17:03:34] Early stopping, best iteration is:
[125]   valid's l1: 207.977
[17:03:34] LightGBM fitting and predicting completed
[17:03:34] Update best score from -208.04352023579347 to -207.97743742493378
[17:03:34] Started iteration 7, chunk = ['ft__plain_meal_id.NUM_UNIQUE(meal_info.cuisine)
↪'], feats to check = ['ft__plain_center_id.MAX(fulfilment_center_info.op_area)', 'ft__
↪plain_center_id.MAX(fulfilment_center_info.city_code WHERE center_type = TYPE_A)', 'ft_
↪_plain_meal_id.COUNT(meal_info WHERE cuisine = Italian)', 'ft__plain_meal_id.
↪COUNT(meal_info WHERE cuisine = Continental)', 'ft__plain_center_id.MAX(fulfilment_
↪center_info.op_area WHERE center_type = TYPE_B)', 'ft__plain_meal_id.COUNT(meal_info␣
↪WHERE cuisine = Indian)', 'ft__plain_center_id.MAX(fulfilment_center_info.city_code)',
↪'ft__plain_center_id.MAX(fulfilment_center_info.op_area WHERE center_type = TYPE_A)',
↪'ft__plain_center_id.MAX(fulfilment_center_info.region_code WHERE center_type = TYPE_A)
↪', 'ft__plain_meal_id.COUNT(meal_info WHERE cuisine = Thai)', 'ft__plain_center_id.
↪MEDIAN(fulfilment_center_info.op_area)', 'ft__plain_center_id.MEAN(fulfilment_center_
↪info.op_area)', 'ft__plain_center_id.MIN(fulfilment_center_info.region_code WHERE␣
↪center_type = TYPE_C)', 'ft__plain_center_id.NUM_UNIQUE(fulfilment_center_info.center_
↪type)', 'ft__plain_center_id.STD(fulfilment_center_info.city_code WHERE center_type =␣
↪TYPE_A)', 'ft__plain_center_id.STD(fulfilment_center_info.city_code WHERE center_type␣
↪= TYPE_B)', 'ft__plain_center_id.STD(fulfilment_center_info.city_code)', 'ft__plain_
↪center_id.MIN(fulfilment_center_info.city_code WHERE center_type = TYPE_A)', 'ft__
↪plain_center_id.STD(fulfilment_center_info.op_area WHERE center_type = TYPE_A)', 'ft__
↪plain_center_id.STD(fulfilment_center_info.op_area WHERE center_type = TYPE_B)', 'ft__
↪plain_center_id.STD(fulfilment_center_info.op_area WHERE center_type = TYPE_C)', 'ft__
```

(continues on next page)

(continued from previous page)

```
↪plain_center_id.STD(fulfilment_center_info.op_area)', 'ft__plain_center_id.
↪STD(fulfilment_center_info.region_code WHERE center_type = TYPE_A)', 'ft__plain_center_
↪id.STD(fulfilment_center_info.region_code WHERE center_type = TYPE_B)', 'ft__plain_
↪meal_id.NUM_UNIQUE(meal_info.cuisine)']
[17:03:34] Features in SCI = ['ft__plain_center_id.MAX(fulfilment_center_info.city_code␣
↪WHERE center_type = TYPE_A)', 'ft__plain_center_id.MAX(fulfilment_center_info.city_
↪code)', 'ft__plain_center_id.MAX(fulfilment_center_info.op_area WHERE center_type =␣
↪TYPE_A)', 'ft__plain_center_id.MAX(fulfilment_center_info.op_area WHERE center_type =␣
↪TYPE_B)', 'ft__plain_center_id.MAX(fulfilment_center_info.op_area)', 'ft__plain_center_
↪id.MAX(fulfilment_center_info.region_code WHERE center_type = TYPE_A)', 'ft__plain_
↪center_id.MEAN(fulfilment_center_info.op_area)', 'ft__plain_center_id.
↪MEDIAN(fulfilment_center_info.op_area)', 'ft__plain_center_id.MIN(fulfilment_center_
↪info.city_code WHERE center_type = TYPE_A)', 'ft__plain_center_id.MIN(fulfilment_
↪center_info.region_code WHERE center_type = TYPE_C)', 'ft__plain_center_id.NUM_
↪UNIQUE(fulfilment_center_info.center_type)', 'ft__plain_center_id.STD(fulfilment_
↪center_info.city_code WHERE center_type = TYPE_A)', 'ft__plain_center_id.
↪STD(fulfilment_center_info.city_code WHERE center_type = TYPE_B)', 'ft__plain_center_
↪id.STD(fulfilment_center_info.city_code)', 'ft__plain_center_id.STD(fulfilment_center_
↪info.op_area WHERE center_type = TYPE_A)', 'ft__plain_center_id.STD(fulfilment_center_
↪info.op_area WHERE center_type = TYPE_B)', 'ft__plain_center_id.STD(fulfilment_center_
↪info.op_area WHERE center_type = TYPE_C)', 'ft__plain_center_id.STD(fulfilment_center_
↪info.op_area)', 'ft__plain_center_id.STD(fulfilment_center_info.region_code WHERE␣
↪center_type = TYPE_A)', 'ft__plain_center_id.STD(fulfilment_center_info.region_code␣
↪WHERE center_type = TYPE_B)', 'ft__plain_meal_id.COUNT(meal_info WHERE cuisine =␣
↪Continental)', 'ft__plain_meal_id.COUNT(meal_info WHERE cuisine = Indian)', 'ft__plain_
↪meal_id.COUNT(meal_info WHERE cuisine = Italian)', 'ft__plain_meal_id.COUNT(meal_info␣
↪WHERE cuisine = Thai)', 'ft__plain_meal_id.NUM_UNIQUE(meal_info.cuisine)']
[17:03:34] Training until validation scores don't improve for 100 rounds
[17:03:35] [100]        valid's l1: 208.41
[17:03:35] [200]        valid's l1: 208.413
[17:03:35] Early stopping, best iteration is:
[125]   valid's l1: 207.977
[17:03:35] LightGBM fitting and predicting completed
[17:03:35] Finally selected feats = ['ft__plain_center_id.MAX(fulfilment_center_info.op_
↪area)', 'ft__plain_center_id.MAX(fulfilment_center_info.city_code WHERE center_type =␣
↪TYPE_A)', 'ft__plain_meal_id.COUNT(meal_info WHERE cuisine = Italian)', 'ft__plain_
↪meal_id.COUNT(meal_info WHERE cuisine = Continental)', 'ft__plain_center_id.
↪MAX(fulfilment_center_info.op_area WHERE center_type = TYPE_B)', 'ft__plain_meal_id.
↪COUNT(meal_info WHERE cuisine = Indian)', 'ft__plain_center_id.MAX(fulfilment_center_
↪info.city_code)', 'ft__plain_center_id.MAX(fulfilment_center_info.op_area WHERE center_
↪type = TYPE_A)', 'ft__plain_center_id.MAX(fulfilment_center_info.region_code WHERE␣
↪center_type = TYPE_A)', 'ft__plain_meal_id.COUNT(meal_info WHERE cuisine = Thai)', 'ft_
↪_plain_center_id.MEDIAN(fulfilment_center_info.op_area)', 'ft__plain_center_id.
↪MEAN(fulfilment_center_info.op_area)', 'ft__plain_center_id.NUM_UNIQUE(fulfilment_
↪center_info.center_type)', 'ft__plain_center_id.STD(fulfilment_center_info.city_code␣
↪WHERE center_type = TYPE_A)', 'ft__plain_center_id.MIN(fulfilment_center_info.region_
↪code WHERE center_type = TYPE_C)', 'ft__plain_center_id.STD(fulfilment_center_info.
↪city_code)', 'ft__plain_center_id.STD(fulfilment_center_info.op_area WHERE center_type␣
↪= TYPE_A)', 'ft__plain_center_id.STD(fulfilment_center_info.op_area WHERE center_type␣
↪= TYPE_B)', 'ft__plain_center_id.STD(fulfilment_center_info.op_area WHERE center_type␣
↪= TYPE_C)', 'ft__plain_center_id.STD(fulfilment_center_info.op_area)', 'ft__plain_
↪center_id.STD(fulfilment_center_info.region_code WHERE center_type = TYPE_A)', 'ft__
```

(continues on next page)

(continued from previous page)

```
↪plain_center_id.STD(fulfilment_center_info.region_code WHERE center_type = TYPE_B)',
↪'ft__plain_center_id.MIN(fulfilment_center_info.city_code WHERE center_type = TYPE_A)',
↪  'ft__plain_center_id.STD(fulfilment_center_info.city_code WHERE center_type = TYPE_B)
↪']
```

```
/home/rinchin/lama_gitlab/LightAutoML/.venv/lib/python3.8/site-packages/featuretools/
↪entityset/entityset.py:1738: UserWarning: Woodwork typing information on new dataframe␣
↪will be replaced with existing typing information from plain
  warnings.warn(
2023-03-30 17:03:40,001 - distributed.worker - WARNING - Could not find data: {'bytes-
↪7832c43e218d1860da556fd287d63146': ['tcp://127.0.0.1:33721', 'tcp://127.0.0.1:43615',
↪'tcp://127.0.0.1:45185']} on workers: [] (who_has: {'bytes-
↪7832c43e218d1860da556fd287d63146': ['tcp://127.0.0.1:33721', 'tcp://127.0.0.1:43615',
↪'tcp://127.0.0.1:45185']})
2023-03-30 17:03:40,004 - distributed.scheduler - WARNING - Worker tcp://127.0.0.1:45367␣
↪failed to acquire keys: {'bytes-7832c43e218d1860da556fd287d63146': ('tcp://127.0.0.1:
↪33721', 'tcp://127.0.0.1:43615', 'tcp://127.0.0.1:45185')}
2023-03-30 17:03:40,143 - distributed.worker - WARNING - Could not find data: {'bytes-
↪7832c43e218d1860da556fd287d63146': ['tcp://127.0.0.1:33721', 'tcp://127.0.0.1:43615',
↪'tcp://127.0.0.1:45185']} on workers: [] (who_has: {'EntitySet-
↪12f0afe6b03337e360b57b2fea5ba056': ['tcp://127.0.0.1:42689', 'tcp://127.0.0.1:44001',
↪'tcp://127.0.0.1:43877'], 'bytes-7832c43e218d1860da556fd287d63146': ['tcp://127.0.0.1:
↪33721', 'tcp://127.0.0.1:43615', 'tcp://127.0.0.1:45185']})
2023-03-30 17:03:40,144 - distributed.scheduler - WARNING - Worker tcp://127.0.0.1:45425␣
↪failed to acquire keys: {'bytes-7832c43e218d1860da556fd287d63146': ('tcp://127.0.0.1:
↪33721', 'tcp://127.0.0.1:43615', 'tcp://127.0.0.1:45185')}
```

```
EntitySet scattered to 16 workers in 4 seconds
[17:03:42] Start fitting Lvl_0_Pipe_0_Mod_0_LightGBM ...
[17:03:42] ===== Start working with fold 0 for Lvl_0_Pipe_0_Mod_0_LightGBM =====
[17:03:42] Training until validation scores don't improve for 100 rounds
[17:03:42] [100]        valid's l1: 95.8142
[17:03:43] [200]        valid's l1: 91.9066
[17:03:43] [300]        valid's l1: 90.8876
[17:03:44] [400]        valid's l1: 90.3315
[17:03:45] [500]        valid's l1: 90.0936
[17:03:45] [600]        valid's l1: 90.0952
[17:03:46] [700]        valid's l1: 89.995
[17:03:46] Early stopping, best iteration is:
[679]   valid's l1: 89.9358
[17:03:47] ===== Start working with fold 1 for Lvl_0_Pipe_0_Mod_0_LightGBM =====
[17:03:47] Training until validation scores don't improve for 100 rounds
[17:03:47] [100]        valid's l1: 93.3286
[17:03:48] [200]        valid's l1: 89.2285
[17:03:48] [300]        valid's l1: 88.3445
[17:03:49] [400]        valid's l1: 88.2788
[17:03:50] [500]        valid's l1: 88.2329
[17:03:50] Early stopping, best iteration is:
[442]   valid's l1: 88.1347
[17:03:50] ===== Start working with fold 2 for Lvl_0_Pipe_0_Mod_0_LightGBM =====
[17:03:50] Training until validation scores don't improve for 100 rounds
[17:03:51] [100]        valid's l1: 95.2671
[17:03:51] [200]        valid's l1: 91.0065
```

(continues on next page)

(continued from previous page)

```
[17:03:52] [300]        valid's l1: 90.3078
[17:03:52] [400]        valid's l1: 89.9448
[17:03:53] [500]        valid's l1: 89.8401
[17:03:53] Early stopping, best iteration is:
[484]   valid's l1: 89.7472
[17:03:54] ===== Start working with fold 3 for Lvl_0_Pipe_0_Mod_0_LightGBM =====
[17:03:54] Training until validation scores don't improve for 100 rounds
[17:03:54] [100]        valid's l1: 92.7871
[17:03:55] [200]        valid's l1: 88.5443
[17:03:55] [300]        valid's l1: 87.5072
[17:03:56] [400]        valid's l1: 87.3209
[17:03:57] [500]        valid's l1: 87.0954
[17:03:57] [600]        valid's l1: 87.1619
[17:03:57] Early stopping, best iteration is:
[509]   valid's l1: 87.0716
[17:03:58] ===== Start working with fold 4 for Lvl_0_Pipe_0_Mod_0_LightGBM =====
[17:03:58] Training until validation scores don't improve for 100 rounds
[17:03:58] [100]        valid's l1: 92.7026
[17:03:59] [200]        valid's l1: 88.2765
[17:03:59] [300]        valid's l1: 87.0135
[17:04:00] [400]        valid's l1: 86.527
[17:04:00] Early stopping, best iteration is:
[396]   valid's l1: 86.4735
[17:04:01] Fitting Lvl_0_Pipe_0_Mod_0_LightGBM finished. score = -88.27262874082102
[17:04:01] Lvl_0_Pipe_0_Mod_0_LightGBM fitting and predicting completed
[17:04:01] Time left 9999999950.01 secs

[17:04:01] Layer 1 training completed.

CPU times: user 2min 13s, sys: 5.83 s, total: 2min 19s
Wall time: 50 s
```

In the **"Finally selected feats"** line, we can see the features generated by `FeatureGenerationPipeline` and selected using LightGBM, obtained using aggregations, tarnsformations and interesting values. For example, `'ft__plain_center_id.MEDIAN(fulfilment_center_info.region_code WHERE center_type = TYPE_A)'` feature is median over `'region_code'` column in `fulfilment_center_info` table (which linked with `'plain'` dataset by `'center_id'` key) where `'center_type'` value equals `'TYPE_A'`.

Analyze fitted model

Let's see the generated features and their importances (received from LightGBM) which we get as a result of training the model:

[17]:
```
feature_imps = model.get_features_score()
feature_imps
```

[17]:
```
ord__checkout_price                                                    ␣
↪ 7.618603e+09
meal_id                                                                ␣
↪ 5.803957e+09
ord__base_price                                                        ␣
↪ 4.911443e+09
homepage_featured                                                      ␣
```

(continues on next page)

(continued from previous page)

```
↪ 2.367179e+09
week                                                                                ␣
↪ 2.110229e+09
ft__plain_center_id.MAX(fulfilment_center_info.op_area)                             ␣
↪ 1.824585e+09
emailer_for_promotion                                                               ␣
↪ 1.425533e+09
center_id                                                                           ␣
↪ 1.280621e+09
id                                                                                  ␣
↪ 1.044121e+09
ft__plain_center_id.MEDIAN(fulfilment_center_info.op_area)                          ␣
↪ 9.298832e+08
ft__plain_center_id.MAX(fulfilment_center_info.city_code WHERE center_type = TYPE_A)    ␣
↪ 8.705599e+08
ft__plain_meal_id.COUNT(meal_info WHERE cuisine = Italian)                          ␣
↪ 8.173745e+08
ft__plain_meal_id.COUNT(meal_info WHERE cuisine = Thai)                             ␣
↪ 8.160740e+08
ft__plain_meal_id.COUNT(meal_info WHERE cuisine = Indian)                           ␣
↪ 8.016899e+08
ft__plain_center_id.MAX(fulfilment_center_info.city_code)                           ␣
↪ 7.446240e+08
ft__plain_center_id.MAX(fulfilment_center_info.op_area WHERE center_type = TYPE_B)      ␣
↪ 5.920852e+08
ft__plain_center_id.MAX(fulfilment_center_info.region_code WHERE center_type = TYPE_A)  ␣
↪ 5.403963e+08
ft__plain_center_id.MAX(fulfilment_center_info.op_area WHERE center_type = TYPE_A)      ␣
↪ 5.356365e+08
ft__plain_center_id.MEAN(fulfilment_center_info.op_area)                            ␣
↪ 3.982109e+08
ft__plain_meal_id.COUNT(meal_info WHERE cuisine = Continental)                      ␣
↪ 3.171587e+08
ft__plain_center_id.MIN(fulfilment_center_info.region_code WHERE center_type = TYPE_C)  ␣
↪ 2.466869e+08
ft__plain_center_id.MIN(fulfilment_center_info.city_code WHERE center_type = TYPE_A)    ␣
↪ 1.435612e+08
ft__plain_center_id.STD(fulfilment_center_info.city_code WHERE center_type = TYPE_B)    ␣
↪ 0.000000e+00
ft__plain_center_id.STD(fulfilment_center_info.op_area)                             ␣
↪ 0.000000e+00
ft__plain_center_id.STD(fulfilment_center_info.region_code WHERE center_type = TYPE_B)  ␣
↪ 0.000000e+00
ft__plain_center_id.STD(fulfilment_center_info.region_code WHERE center_type = TYPE_A)  ␣
↪ 0.000000e+00
ft__plain_center_id.STD(fulfilment_center_info.op_area WHERE center_type = TYPE_C)      ␣
↪ 0.000000e+00
ft__plain_center_id.STD(fulfilment_center_info.op_area WHERE center_type = TYPE_B)      ␣
↪ 0.000000e+00
ft__plain_center_id.STD(fulfilment_center_info.city_code)                           ␣
↪ 0.000000e+00
ft__plain_center_id.STD(fulfilment_center_info.city_code WHERE center_type = TYPE_A)    ␣
```

(continues on next page)

(continued from previous page)

```
↪ 0.000000e+00
ft__plain_center_id.NUM_UNIQUE(fulfilment_center_info.center_type)                ␣
↪ 0.000000e+00
ft__plain_center_id.STD(fulfilment_center_info.op_area WHERE center_type = TYPE_A)      ␣
↪ 0.000000e+00
dtype: float64
```

Quite a large number of features heve non-zero importances:

[18]:
```
feature_imps.index[feature_imps > 0]
```

[18]:
```
Index(['ord__checkout_price', 'meal_id', 'ord__base_price',
       'homepage_featured', 'week',
       'ft__plain_center_id.MAX(fulfilment_center_info.op_area)',
       'emailer_for_promotion', 'center_id', 'id',
       'ft__plain_center_id.MEDIAN(fulfilment_center_info.op_area)',
       'ft__plain_center_id.MAX(fulfilment_center_info.city_code WHERE center_type =␣
↪TYPE_A)',
       'ft__plain_meal_id.COUNT(meal_info WHERE cuisine = Italian)',
       'ft__plain_meal_id.COUNT(meal_info WHERE cuisine = Thai)',
       'ft__plain_meal_id.COUNT(meal_info WHERE cuisine = Indian)',
       'ft__plain_center_id.MAX(fulfilment_center_info.city_code)',
       'ft__plain_center_id.MAX(fulfilment_center_info.op_area WHERE center_type = TYPE_
↪B)',
       'ft__plain_center_id.MAX(fulfilment_center_info.region_code WHERE center_type =␣
↪TYPE_A)',
       'ft__plain_center_id.MAX(fulfilment_center_info.op_area WHERE center_type = TYPE_
↪A)',
       'ft__plain_center_id.MEAN(fulfilment_center_info.op_area)',
       'ft__plain_meal_id.COUNT(meal_info WHERE cuisine = Continental)',
       'ft__plain_center_id.MIN(fulfilment_center_info.region_code WHERE center_type =␣
↪TYPE_C)',
       'ft__plain_center_id.MIN(fulfilment_center_info.city_code WHERE center_type =␣
↪TYPE_A)'],
      dtype='object')
```

Evaluation

[21]:
```
test_pred = automl.predict(test)
```

```
EntitySet scattered to 16 workers in 4 seconds
```

[22]:
```
print(f"OOF MAE on train: {mean_absolute_error(train['plain'][roles['target']], train_
↪pred.data[:, 0])}")
print(f"MAE on test: {mean_absolute_error(test['plain'][roles['target']], test_pred.
↪data[:, 0])}")
```

```
OOF MAE on train: 88.27262874082102
MAE on test: 95.97085837200986
```

Additional materials

- Official LightAutoML github repo
- LightAutoML documentation
- LightAutoML tutorials
- LightAutoML course:
  - Part 1 - general overview
  - Part 2 - LightAutoML specific applications
  - Part 3 - LightAutoML customization
- OpenDataScience AutoML benchmark leaderboard

2.1.11 Tutorial 11: time series

There are some features how LightAutoML makes forecast for time-series tasks:

- multi-output strategy for forecasting time series if horizon is more than 1 point (forecast simultaneously over the entire horizon)
- global-modelling approach for handling multiple time series (one model for all time series in the dataset)

In this tutorial you will learn how to:

- run LightAutoML training on data with one or many time series
- configure time series features transformers' parameters

Official LightAutoML github repository is here.

0. Prerequisites

0.0. install LightAutoML

[ ]:
```
# !pip install -U lightautoml
```

0.1. Import libraries

[2]:
```
# Standard python libraries
import os

# Installed libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import mean_absolute_error

# Imports from our package
```

(continues on next page)

(continued from previous page)

```
from lightautoml.tasks import Task
from lightautoml.addons.autots.base import AutoTS
from lightautoml.dataset.roles import DatetimeRole
from lightautoml.automl.base import AutoML
from lightautoml.ml_algo.boost_cb import BoostCB
from lightautoml.ml_algo.linear_sklearn import LinearLBFGS
from lightautoml.pipelines.features.lgb_pipeline import LGBSeqSimpleFeatures
from lightautoml.pipelines.features.linear_pipeline import LinearTrendFeatures
from lightautoml.pipelines.ml.base import MLPipeline
from lightautoml.reader.base import DictToPandasSeqReader
from lightautoml.automl.blend import WeightedBlender
from lightautoml.ml_algo.random_forest import RandomForestSklearn

# Disable warnings
import warnings
warnings.filterwarnings("ignore")
```

0.2. Constants and functions

Here we setup the constants to use in the kernel: - `HORIZON` - number of points to forecast - `TARGET_COLUMN` - target column name in dataset - `DATE_COLUMN` - date column name in dataset - `ID_COLUMN` - id column name in dataset

[3]:
```
HORIZON = 30
TARGET_COLUMN = "value"
DATE_COLUMN = "date"
ID_COLUMN = "ID"
```

[4]:
```
def draw_raw_ts(
    data: pd.DataFrame,
    id_column: str,
    target_column: str,
    date_column: str,
    ncols: int = 2,
):
    """Draw graphs of time series with specified parameters.
    Args:
        - data: pd.DataFrame with time series data
        - id_column: id column name in dataset
        - target_column: target column name in dataset
        - date_column: date column name in dataset
        - ncols: number of columns for subplot's grid
    """
    # Initialize grid's shape
    num_ts = data[id_column].nunique()
    nrows = num_ts // ncols + num_ts % ncols
    fig, ax = plt.subplots(
        nrows,
        ncols,
        figsize=(24, 5 * nrows)
    )
    axes_to_del = nrows * ncols - num_ts
```

(continues on next page)

(continued from previous page)

```
    for i in range(axes_to_del):
        i_row = (nrows - 1) - i // ncols
        i_col = (ncols - 1) - i % ncols
        fig.delaxes(ax[i_row][i_col])

    # Draw graphs
    for i, ts_id in enumerate(data[id_column].unique()):
        i_row = i // ncols
        i_col = i % ncols

        ts_df = data[data[id_column] == ts_id]
        ax = ax.reshape(nrows, ncols)
        ax[i_row, i_col].plot(ts_df[date_column], ts_df[target_column])
        ax[i_row, i_col].title.set_text(f"TS with ID {ts_id}")
```

0.3. Data loading

[5]:
```
df = pd.read_csv('../data/ts_data.csv')
df[DATE_COLUMN] = pd.to_datetime(df[DATE_COLUMN])
display(df.head())
print(f"data shape: {df.shape}")
print(f"number of time series in data: {df[ID_COLUMN].nunique()}")
```

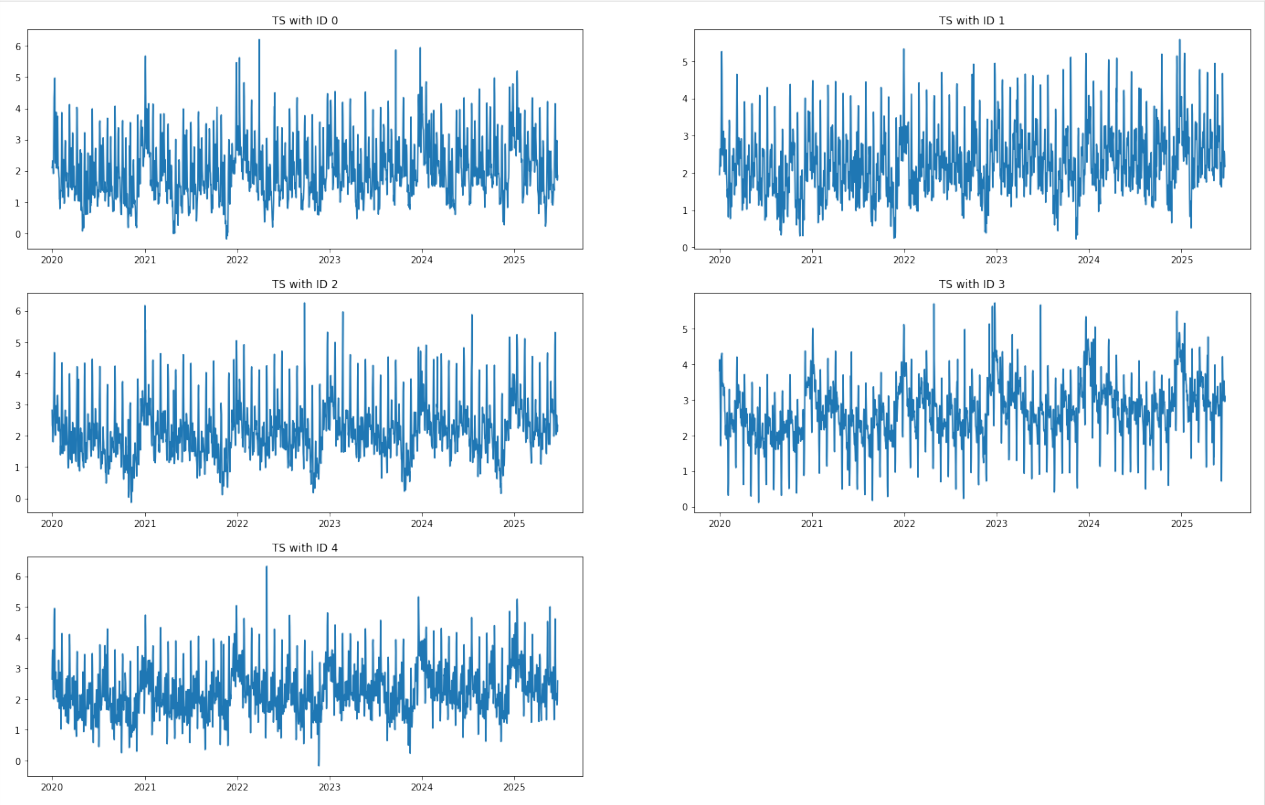
```
        date     value  ID
0 2020-01-01  2.100760   0
1 2020-01-02  2.106072   0
2 2020-01-03  2.291461   0
3 2020-01-04  2.322224   0
4 2020-01-05  2.140932   0
```

```
data shape: (10000, 3)
number of time series in data: 5
```

This data is simulated and presented time series, contained such components as trend, seasonality, events and future reg.

On the cell below we draw our time series:

[6]:
```
draw_raw_ts(df, ID_COLUMN, TARGET_COLUMN, DATE_COLUMN)
```



0.4. Data splitting for train-holdout

Let's make forecast of 30 next points. So, we should make train-test-split for each times series in our data.

[7]:
```
# Assume, that our time series are aligned and have identical timestamps.
test_start = df[df[ID_COLUMN] == 0][DATE_COLUMN].values[-HORIZON]

train = df[df[DATE_COLUMN] < test_start].copy()
test = df[df[DATE_COLUMN] >= test_start].copy()
```

1. Task definition

1.1. Task type

On the cell below we create `Task` object - the class to setup what task LightAutoML model should solve with specific loss and metric if necessary (more info can be found here in our documentation). For time series forecasting it should set as "multi:reg":

[8]:
```
task = Task("multi:reg", greater_is_better=False, metric="mae", loss="mae")
```

```
multi:reg isn`t supported in lgb
```

1.2. Feature roles setup

The only role you must setup is `"target"` role, everything else (date, categorical, etc.) is up to user.

But if we have several time series in data, it is recommended to setup `"id"` role to group our observations. Also we can define seasonal features through `DatetimeRole`. Valid are: `y`, `m`, `d`, `wd`, `hour`, `min`, `sec`, `ms`, `ns`.

[9]:
```
univariate_roles = {
    "target": TARGET_COLUMN,
    DatetimeRole(seasonality=('d', 'm', 'wd'), base_date=True): DATE_COLUMN,
}

global_modelling_roles = {
    "target": TARGET_COLUMN,
    DatetimeRole(seasonality=('d', 'm', 'wd'), base_date=True): DATE_COLUMN,
    "id": ID_COLUMN
}
```

1.3. LightAutoML (AutoTS) model creation

In this part we are going to create LightAutoML model with `AutoTS` class.

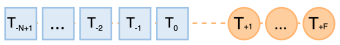
The params we can setup: - `task` - the type of the ML task (we defined it earlier) - `seq_params` - parameter for Reader object, which works on the first step of data preparation: - `case` - the type of problem we are solving (`next_values`: prediction of next values) - `n_target` - forecasting horizon - `history` - history size for feature generating (i.e., features for observation y_t are counted from observations $(y_{t-history}, \ldots, y_{t-1})$) - `step` - in how many points to take the next observation in the training sample (the higher the step value –> the fewer observations fall into the training sample) - `test_last` - technical parameter: test data are built by the last observation from the training sample - `from_last` - technical parameter: build train features from last possible observation. - `transformers_params` - parameter for Transformer objects, which are needed to make features from raw time series - `lag_features` - *bool/int/list/array*: lags to make lag features for features other than date - `lag_time_features` - *bool/int/list/array*: lags to make lag features for date features - `diff_features` - *bool/int/list/array*: lags to make difference features for features other than date - `trend_params` - parameter for TrendModel object, which is needed to detrend the time series before using the main AutoML class

Note: Parameters within the transformers_params may be configured as True/False, or integer, list, numpy-array: - **True**: use default values (lag_other_features=30, lag_time_features=30, diff_features=7) - **int**: use all lags and diffs up to the input number (range) - **list**: use certain lags and diffs specified in the list

---

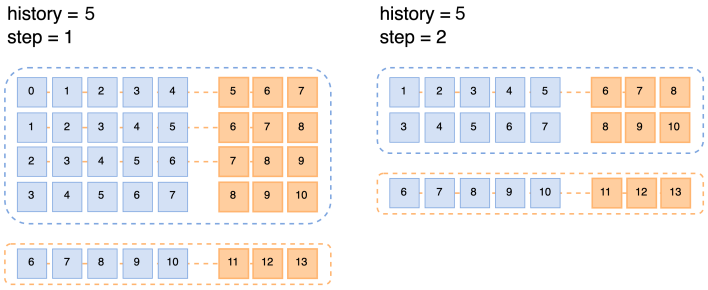
There are some figures which can be helpful for better understanding these params:

Firstly, let's generalise regression task of time series forecasting: - with known values for timestamps from T_{N+1} to T_0 we should predict ones for timestamps $T_1, ..., T_F$

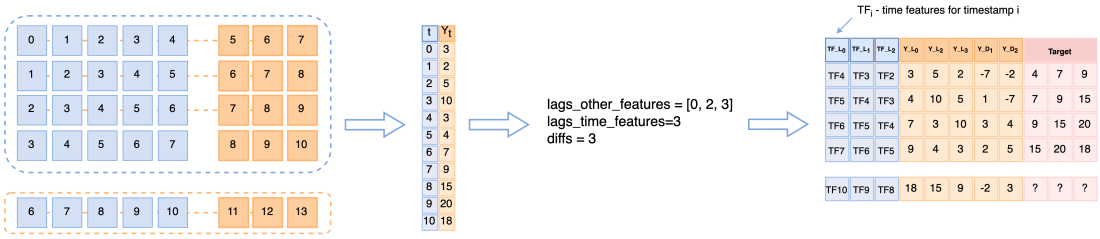


- assume that $N = 10$ and $F = 3$ and get train and test index arrays for values in time series:

Firstly, the graph below is about `history` and `step` parameters:



Now let's take the first case and understand how `transformers_params` work and how train and test samples look like after transformations:



[10]:
```
seq_params = {
    "seq0": {
        "case": "next_values",
        "params": {
            "n_target": HORIZON,
            "history": HORIZON,
            "step": 1,
            "from_last": True,
            "test_last": True
        }
    }
}

transformers_params = {
    "lag_features": 30,
    "lag_time_features": 30,
    "diff_features": [1, 2, 3, 4, 5, 6, 7, 14],
}

# You can try specify parameters for trend model too
# trend_params = {
#     'trend': False,
#     'train_on_trend': False,
#     'trend_type': 'decompose',  # one of 'decompose', 'decompose_STL', 'linear' or 'rolling'
#     'trend_size': 1,
#     'decompose_period': 30,
#     'detect_step_quantile': 0.01,
#     'detect_step_window': 1,
#     'detect_step_threshold': 0.7,
#     'rolling_size': 1,
#     'verbose': 0
```

(continues on next page)

(continued from previous page)

```
# }

automl = AutoTS(
    task,
    reader_params = {
        "seq_params": seq_params
    },
    time_series_trend_params={
        "trend": False,
    },
    time_series_pipeline_params=transformers_params
)
```

Important note: reader_params, time_series_trend_params, time_series_pipeline_params, general_params keys are the YAML config keys, which is used inside TabularAutoML preset. More details on its structure with explanation comments can be found inside the lightautoml/automl/presets/time_series_config.yaml file. Each key from this config can be modified with user settings during both to providing .yml file and modifying while AutoTS class initialization like in the cell above.

2. AutoML training

2.1. Univariate LightAutoML

[11]:
```
# leave only ts with ID = 4
ID = 4

univariate_train = train[train[ID_COLUMN] == ID].drop("ID", axis=1)
univariate_test = test[test[ID_COLUMN] == ID].drop("ID", axis=1)
```

[12]:
```
os.environ["CUDA_DEVICE_ORDER"] = "PCI_BUS_ID"
os.environ["CUDA_VISIBLE_DEVICES"] = "1"
```

[13]:
```
univariate_train_pred, _ = automl.fit_predict(univariate_train, univariate_roles,␣
↪verbose=4)
univariate_forecast, _ = automl.predict(univariate_train)
```

```
[10:24:17] Stdout logging level is DEBUG.
[10:24:17] Task: multi:reg

[10:24:17] Start automl preset with listed constraints:
[10:24:17] - time: 3600.00 seconds
[10:24:17] - CPU: 4 cores
[10:24:17] - memory: 16 GB

[17:58:26] Layer 1 train process start. Time left 3599.96 secs
[17:58:26] Start fitting Lvl_0_Pipe_0_Mod_0_RFSklearn ...
[17:58:26] Training params: {'bootstrap': True, 'ccp_alpha': 0.0, 'max_depth': None,
↪'max_features': 'sqrt', 'max_leaf_nodes': None, 'max_samples': None, 'min_samples_leaf
↪': 32, 'min_samples_split': 2, 'min_weight_fraction_leaf': 0.0, 'n_estimators': 500,
↪'n_jobs': 4, 'oob_score': False, 'random_state': 42, 'warm_start': False, 'verbose': 0,
↪ 'criterion': 'mse'}
[17:58:26] ===== Start working with fold 0 for Lvl_0_Pipe_0_Mod_0_RFSklearn =====
```

(continues on next page)

(continued from previous page)

```
[17:58:27] Score for RF model: -0.474507
[17:58:27] ===== Start working with fold 1 for Lvl_0_Pipe_0_Mod_0_RFSklearn =====
[17:58:28] Score for RF model: -0.468268
[17:58:28] Fitting Lvl_0_Pipe_0_Mod_0_RFSklearn finished. score = -0.4713888963184421
[17:58:28] Lvl_0_Pipe_0_Mod_0_RFSklearn fitting and predicting completed
[17:58:28] Time left 3598.34 secs

[17:58:28] Start fitting Lvl_0_Pipe_1_Mod_0_LinearL2 ...
[17:58:28] Training params: {'tol': 1e-06, 'max_iter': 100, 'cs': [1e-05, 5e-05, 0.0001,␣
↪0.0005, 0.001, 0.005, 0.01, 0.05, 0.1, 0.5, 1, 5, 10, 50, 100, 500, 1000, 5000, 10000,␣
↪50000, 100000], 'early_stopping': 2, 'categorical_idx': [], 'embed_sizes': (), 'data_
↪size': 188}
[17:58:28] ===== Start working with fold 0 for Lvl_0_Pipe_1_Mod_0_LinearL2 =====
[17:58:28] Linear model: C = 1e-05 score = -0.6003076791232048
[17:58:28] Linear model: C = 5e-05 score = -0.5183533156110102
[17:58:29] Linear model: C = 0.0001 score = -0.46330821971023245
[17:58:29] Linear model: C = 0.0005 score = -0.36810350419844357
[17:58:29] Linear model: C = 0.001 score = -0.35480725252195533
[17:58:29] Linear model: C = 0.005 score = -0.3475977814992737
[17:58:29] Linear model: C = 0.01 score = -0.3480328256824163
[17:58:30] Linear model: C = 0.05 score = -0.3484652342803444
[17:58:30] ===== Start working with fold 1 for Lvl_0_Pipe_1_Mod_0_LinearL2 =====
[17:58:30] Linear model: C = 1e-05 score = -0.6028475283394467
[17:58:30] Linear model: C = 5e-05 score = -0.5207041477349572
[17:58:30] Linear model: C = 0.0001 score = -0.4659881593860412
[17:58:30] Linear model: C = 0.0005 score = -0.3684143016750015
[17:58:31] Linear model: C = 0.001 score = -0.3556188196228592
[17:58:31] Linear model: C = 0.005 score = -0.34978738837360196
[17:58:31] Linear model: C = 0.01 score = -0.3501108550228322
[17:58:31] Linear model: C = 0.05 score = -0.35125563103009994
[17:58:31] Fitting Lvl_0_Pipe_1_Mod_0_LinearL2 finished. score = -0.34869201204086625
[17:58:31] Lvl_0_Pipe_1_Mod_0_LinearL2 fitting and predicting completed
[17:58:31] Time left 3594.93 secs

[17:58:31] Start fitting Lvl_0_Pipe_2_Mod_0_CatBoost ...
[17:58:31] Training params: {'task_type': 'GPU', 'thread_count': 4, 'random_seed': 42,
↪'num_trees': 3000, 'learning_rate': 0.03, 'l2_leaf_reg': 0.01, 'bootstrap_type':
↪'Bernoulli', 'grow_policy': 'SymmetricTree', 'max_depth': 5, 'min_data_in_leaf': 1,
↪'one_hot_max_size': 10, 'fold_permutation_block': 1, 'boosting_type': 'Plain', 'boost_
↪from_average': True, 'od_type': 'Iter', 'od_wait': 100, 'max_bin': 32, 'feature_border_
↪type': 'GreedyLogSum', 'nan_mode': 'Min', 'verbose': 100, 'allow_writing_files': False,
↪ 'devices': '0'}
[17:58:31] ===== Start working with fold 0 for Lvl_0_Pipe_2_Mod_0_CatBoost =====
[17:58:32] 0:   learn: 4.4726230        test: 4.4551552 best: 4.4551552 (0)     total:␣
↪59.3ms   remaining: 2m 57s
[17:58:33] 100: learn: 2.5849946        test: 2.8315490 best: 2.8315490 (100)   total: 1.
↪29s    remaining: 36.9s
[17:58:34] 200: learn: 1.8855638        test: 2.2618428 best: 2.2618428 (200)   total: 2.
↪5s     remaining: 34.7s
[17:58:35] 300: learn: 1.5189472        test: 1.9874703 best: 1.9874703 (300)   total: 3.
↪72s    remaining: 33.3s
[17:58:36] 400: learn: 1.2936364        test: 1.8444599 best: 1.8444599 (400)   total: 4.
```

(continues on next page)

(continued from previous page)

```
↪9s     remaining: 31.8s
[17:58:38] 500: learn: 1.1450053        test: 1.7661319 best: 1.7661319 (500)   total: 6.
↪12s    remaining: 30.5s
[17:58:39] 600: learn: 1.0355444        test: 1.7198144 best: 1.7198144 (600)   total: 7.
↪27s    remaining: 29s
[17:58:40] 700: learn: 0.9461529        test: 1.6905038 best: 1.6905038 (700)   total: 8.
↪49s    remaining: 27.8s
[17:58:41] 800: learn: 0.8698428        test: 1.6713547 best: 1.6713547 (800)   total: 9.
↪64s    remaining: 26.5s
[17:58:42] 900: learn: 0.8034182        test: 1.6592994 best: 1.6592994 (900)   total:␣
↪10.8s    remaining: 25.2s
[17:58:44] 1000:        learn: 0.7435150        test: 1.6494466 best: 1.6493530 (997)  ␣
↪total: 12s      remaining: 23.9s
[17:58:45] 1100:        learn: 0.6882164        test: 1.6402117 best: 1.6402017 (1099) ␣
↪total: 13.2s    remaining: 22.7s
[17:58:46] 1200:        learn: 0.6392184        test: 1.6336748 best: 1.6336748 (1200) ␣
↪total: 14.4s    remaining: 21.6s
[17:58:47] 1300:        learn: 0.5943148        test: 1.6284005 best: 1.6283444 (1299) ␣
↪total: 15.7s    remaining: 20.5s
[17:58:48] 1400:        learn: 0.5537223        test: 1.6237273 best: 1.6237273 (1400) ␣
↪total: 16.9s    remaining: 19.3s
[17:58:50] 1500:        learn: 0.5174100        test: 1.6217630 best: 1.6217572 (1499) ␣
↪total: 18.1s    remaining: 18s
[17:58:51] 1600:        learn: 0.4829392        test: 1.6189425 best: 1.6189425 (1600) ␣
↪total: 19.3s    remaining: 16.8s
[17:58:52] 1700:        learn: 0.4513613        test: 1.6168994 best: 1.6168994 (1700) ␣
↪total: 20.4s    remaining: 15.6s
[17:58:53] 1800:        learn: 0.4221131        test: 1.6147198 best: 1.6147198 (1800) ␣
↪total: 21.6s    remaining: 14.4s
[17:58:55] 1900:        learn: 0.3947037        test: 1.6129983 best: 1.6129983 (1900) ␣
↪total: 22.9s    remaining: 13.2s
[17:58:56] 2000:        learn: 0.3704698        test: 1.6118408 best: 1.6118361 (1999) ␣
↪total: 24.1s    remaining: 12s
[17:58:57] 2100:        learn: 0.3473040        test: 1.6108620 best: 1.6108613 (2099) ␣
↪total: 25.2s    remaining: 10.8s
[17:58:58] 2200:        learn: 0.3258960        test: 1.6102695 best: 1.6102651 (2188) ␣
↪total: 26.4s    remaining: 9.59s
[17:58:59] 2300:        learn: 0.3054025        test: 1.6097875 best: 1.6097492 (2296) ␣
↪total: 27.6s    remaining: 8.38s
[17:59:00] 2400:        learn: 0.2867054        test: 1.6090802 best: 1.6090795 (2398) ␣
↪total: 28.8s    remaining: 7.17s
[17:59:02] 2500:        learn: 0.2689507        test: 1.6081654 best: 1.6081654 (2500) ␣
↪total: 29.9s    remaining: 5.97s
[17:59:03] 2600:        learn: 0.2525349        test: 1.6076868 best: 1.6076019 (2590) ␣
↪total: 31.1s    remaining: 4.77s
[17:59:04] 2700:        learn: 0.2377121        test: 1.6073861 best: 1.6073861 (2700) ␣
↪total: 32.3s    remaining: 3.57s
[17:59:05] 2800:        learn: 0.2229549        test: 1.6071654 best: 1.6071654 (2800) ␣
↪total: 33.4s    remaining: 2.38s
[17:59:06] 2900:        learn: 0.2094542        test: 1.6066872 best: 1.6066869 (2899) ␣
↪total: 34.6s    remaining: 1.18s
[17:59:07] 2999:        learn: 0.1970519        test: 1.6066699 best: 1.6066511 (2993) ␣
```

(continues on next page)

(continued from previous page)

```
↪total: 35.8s    remaining: 0us
[17:59:07] bestTest = 1.606651141
[17:59:07] bestIteration = 2993
[17:59:07] Shrink model to first 2994 iterations.
[17:59:08] ===== Start working with fold 1 for Lvl_0_Pipe_2_Mod_0_CatBoost =====
[17:59:08] 0:   learn: 4.4577437        test: 4.4867925 best: 4.4867925 (0)     total:␣
↪15.6ms   remaining: 46.8s
[17:59:09] 100: learn: 2.6137326        test: 2.8394343 best: 2.8394343 (100)   total: 1.
↪19s    remaining: 34.2s
[17:59:10] 200: learn: 1.8909167        test: 2.2558778 best: 2.2558778 (200)   total: 2.
↪38s    remaining: 33.2s
[17:59:11] 300: learn: 1.4986037        test: 1.9673679 best: 1.9673679 (300)   total: 3.
↪58s    remaining: 32.1s
[17:59:12] 400: learn: 1.2687985        test: 1.8194061 best: 1.8194061 (400)   total: 4.
↪75s    remaining: 30.8s
[17:59:14] 500: learn: 1.1222801        test: 1.7438983 best: 1.7438983 (500)   total: 6.
↪06s    remaining: 30.2s
[17:59:15] 600: learn: 1.0116269        test: 1.6987285 best: 1.6987285 (600)   total: 7.
↪25s    remaining: 29s
[17:59:16] 700: learn: 0.9254741        test: 1.6695112 best: 1.6695112 (700)   total: 8.
↪44s    remaining: 27.7s
[17:59:17] 800: learn: 0.8495263        test: 1.6502003 best: 1.6501991 (799)   total: 9.
↪64s    remaining: 26.5s
[17:59:18] 900: learn: 0.7812408        test: 1.6338258 best: 1.6337881 (899)   total:␣
↪10.8s    remaining: 25.2s
[17:59:20] 1000:        learn: 0.7249770        test: 1.6227465 best: 1.6227465 (1000) ␣
↪total: 12s      remaining: 24s
[17:59:21] 1100:        learn: 0.6742388        test: 1.6164241 best: 1.6164241 (1100) ␣
↪total: 13.2s    remaining: 22.8s
[17:59:22] 1200:        learn: 0.6261038        test: 1.6104160 best: 1.6103175 (1199) ␣
↪total: 14.4s    remaining: 21.6s
[17:59:23] 1300:        learn: 0.5812009        test: 1.6058196 best: 1.6058196 (1300) ␣
↪total: 15.6s    remaining: 20.3s
[17:59:24] 1400:        learn: 0.5414582        test: 1.6023364 best: 1.6023274 (1396) ␣
↪total: 16.8s    remaining: 19.1s
[17:59:26] 1500:        learn: 0.5047374        test: 1.5997954 best: 1.5997954 (1500) ␣
↪total: 17.9s    remaining: 17.9s
[17:59:27] 1600:        learn: 0.4710768        test: 1.5971879 best: 1.5971516 (1599) ␣
↪total: 19.1s    remaining: 16.7s
[17:59:28] 1700:        learn: 0.4406853        test: 1.5948181 best: 1.5948181 (1700) ␣
↪total: 20.3s    remaining: 15.5s
[17:59:29] 1800:        learn: 0.4122589        test: 1.5932432 best: 1.5932432 (1800) ␣
↪total: 21.5s    remaining: 14.3s
[17:59:30] 1900:        learn: 0.3856189        test: 1.5923377 best: 1.5922775 (1884) ␣
↪total: 22.7s    remaining: 13.1s
[17:59:31] 2000:        learn: 0.3613043        test: 1.5910058 best: 1.5910058 (2000) ␣
↪total: 23.8s    remaining: 11.9s
[17:59:33] 2100:        learn: 0.3383675        test: 1.5898274 best: 1.5898274 (2100) ␣
↪total: 25s      remaining: 10.7s
[17:59:34] 2200:        learn: 0.3172203        test: 1.5889919 best: 1.5889524 (2197) ␣
↪total: 26.2s    remaining: 9.51s
[17:59:35] 2300:        learn: 0.2974622        test: 1.5885660 best: 1.5885542 (2299) ␣
```

(continues on next page)

(continued from previous page)

```
↪total: 27.4s    remaining: 8.32s
[17:59:36] 2400:        learn: 0.2794474        test: 1.5876316 best: 1.5876312 (2399) ␣
↪total: 28.6s    remaining: 7.13s
[17:59:37] 2500:        learn: 0.2620195        test: 1.5870893 best: 1.5870893 (2500) ␣
↪total: 29.8s    remaining: 5.94s
[17:59:39] 2600:        learn: 0.2461638        test: 1.5866054 best: 1.5865939 (2598) ␣
↪total: 30.9s    remaining: 4.75s
[17:59:40] 2700:        learn: 0.2310694        test: 1.5863323 best: 1.5863064 (2696) ␣
↪total: 32.1s    remaining: 3.55s
[17:59:41] 2800:        learn: 0.2173924        test: 1.5859899 best: 1.5859823 (2799) ␣
↪total: 33.3s    remaining: 2.37s
[17:59:42] 2900:        learn: 0.2046222        test: 1.5855330 best: 1.5855330 (2900) ␣
↪total: 34.5s    remaining: 1.18s
[17:59:43] 2999:        learn: 0.1924596        test: 1.5853173 best: 1.5852992 (2953) ␣
↪total: 35.6s    remaining: 0us
[17:59:43] bestTest = 1.585299221
[17:59:43] bestIteration = 2953
[17:59:43] Shrink model to first 2954 iterations.
[17:59:43] Fitting Lvl_0_Pipe_2_Mod_0_CatBoost finished. score = -0.22094300934924996
[17:59:43] Lvl_0_Pipe_2_Mod_0_CatBoost fitting and predicting completed
[17:59:43] Time left 3522.77 secs

[17:59:43] Layer 1 training completed.

[17:59:43] Blending: optimization starts with equal weights and score -0.3009012970426841
[17:59:43] Blending: iteration 0: score = -0.22094300934924996, weights = [0. 0. 1.]
[17:59:43] Blending: iteration 1: score = -0.22094300934924996, weights = [0. 0. 1.]
[17:59:43] Blending: no score update. Terminated

[17:59:43] Automl preset training completed in 77.27 seconds

[10:25:54] Model description:
Final prediction for new objects (level 0) =
         1.00000 * (2 averaged models Lvl_0_Pipe_2_Mod_0_CatBoost)
```

Let's take a look on forecasts of univariate time series and check MAE

[14]:
```
print(univariate_forecast, "\n")
print(f"MAE: {mean_absolute_error(univariate_test.value, univariate_forecast)}")
```

```
[3.44697142 3.073035   2.49203897 2.1270709  2.29297566 2.70116901
 2.83488226 2.49551201 2.16700649 1.99528813 2.31517577 2.87421489
 3.00472379 2.59042096 1.8664093  1.47073889 1.81992817 2.58693051
 3.36398268 3.59384632 3.24508047 2.63568759 2.20735097 1.9374007
 1.74858916 1.72028816 1.79335809 2.08676887 2.57858872 3.18161464]

MAE: 0.22544950642226508
```

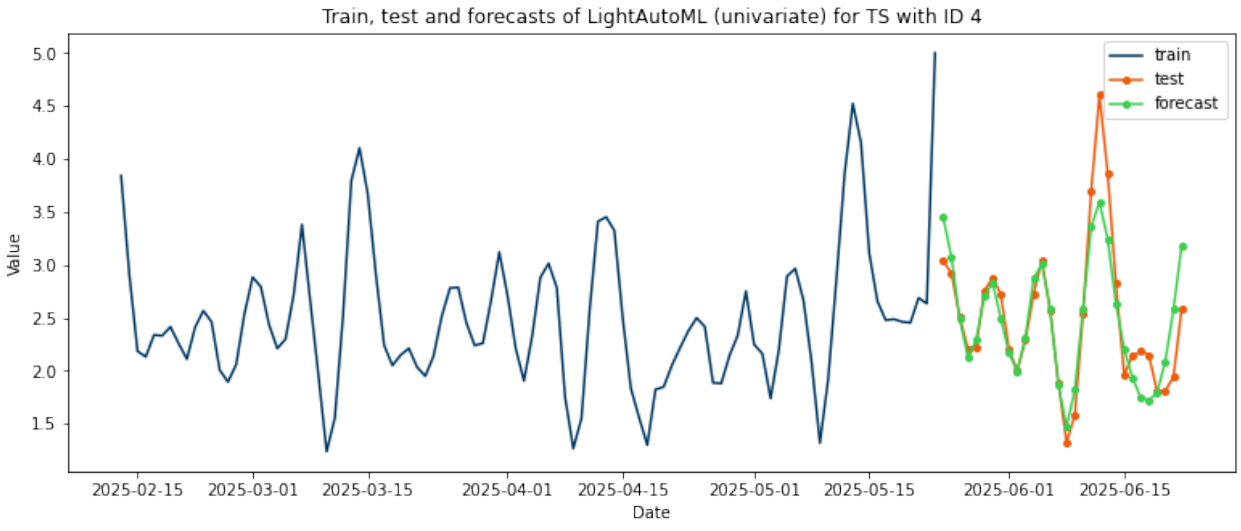
Get a graph of model's predictions in comparison with true values

[15]:
```
last_N = min(len(train), 100)
```

(continues on next page)

(continued from previous page)

```
fig = plt.figure(figsize=(13, 5))
plt.plot(
    univariate_train[DATE_COLUMN][-last_N:],
    univariate_train[TARGET_COLUMN][-last_N:],
    c="#003865",
    label="train"
)
plt.plot(
    univariate_test[DATE_COLUMN],
    univariate_test[TARGET_COLUMN],
    c="#EF5B0C",
    label="test",
    marker="o",
    markersize=4
)
plt.plot(
    univariate_test[DATE_COLUMN],
    univariate_forecast,
    c="#3CCF4E",
    label="forecast",
    marker="o",
    markersize=4
)

plt.xlabel("Date")
plt.ylabel("Value")
plt.title(f"Train, test and forecasts of LightAutoML (univariate) for TS with ID {ID}")
plt.legend()
plt.show()
```



One by one make forecasts for all time series in dataset and collect them to a new pd.DataFrame:

[16]:
```
ID_array = np.repeat(df[ID_COLUMN].unique(), HORIZON)
test_array = np.array([])
pred_array = np.array([])
```

(continues on next page)

(continued from previous page)

```
date_array = np.array([]).astype("datetime64")

for ts_id in df[ID_COLUMN].unique():
    univariate_train = train[train[ID_COLUMN] == ts_id]
    univariate_test = test[test[ID_COLUMN] == ts_id]

    # Model fit_predict
    _, _ = automl.fit_predict(univariate_train, univariate_roles)
    univariate_forecast, _ = automl.predict(univariate_train)
    pred_array = np.append(pred_array, univariate_forecast)
    test_array = np.append(test_array, univariate_test[TARGET_COLUMN].values)
    date_array = np.append(date_array, univariate_test[DATE_COLUMN].values)

# Collect results
res_df_dict = {
    "ID": ID_array,
    "pred": pred_array,
    "date": date_array,
    "test": test_array
}

res_df_univariate = pd.DataFrame(res_df_dict)
```

[17]:
```
res_df_univariate.head()
```

[17]:

	ID	pred	date	test
0	0	2.427282	2025-05-24	1.581092
1	0	3.297728	2025-05-25	2.376163
2	0	3.508930	2025-05-26	2.627754
3	0	3.071776	2025-05-27	2.105468
4	0	2.333910	2025-05-28	1.969833

Check MAE for all time series:

[18]:
```
mae_df_univariate = res_df_univariate.groupby("ID").apply(
    lambda x: mean_absolute_error(x["test"], x["pred"])
).reset_index()

mae_df_univariate.columns = ["ID", "MAE_univariate"]
mae_df_univariate
```

[18]:

	ID	MAE_univariate
0	0	0.371187
1	1	0.292606
2	2	0.435712
3	3	0.376864
4	4	0.225450

2.2. Global-modelling LightAutoML

[19]:
```
# Duplicate ID column for better feature generating
train["id_2"] = train[ID_COLUMN]
```

[20]:
```
oof_pred_seq = automl.fit_predict(train, roles=global_modelling_roles, verbose=4)
seq_test = automl.predict(train, return_raw=True)
```

```
[10:34:19] Stdout logging level is DEBUG.
[10:34:19] Task: multi:reg

[10:34:19] Start automl preset with listed constraints:
[10:34:19] - time: 3600.00 seconds
[10:34:19] - CPU: 4 cores
[10:34:19] - memory: 16 GB

[18:05:37] Layer 1 train process start. Time left 3599.91 secs
[18:05:38] Start fitting Lvl_0_Pipe_0_Mod_0_RFSklearn ...
[18:05:38] Training params: {'bootstrap': True, 'ccp_alpha': 0.0, 'max_depth': None,
↪'max_features': 'sqrt', 'max_leaf_nodes': None, 'max_samples': None, 'min_samples_leaf
↪': 32, 'min_samples_split': 2, 'min_weight_fraction_leaf': 0.0, 'n_estimators': 500,
↪'n_jobs': 4, 'oob_score': False, 'random_state': 42, 'warm_start': False, 'verbose': 0,
↪ 'criterion': 'mse'}
[18:05:38] ===== Start working with fold 0 for Lvl_0_Pipe_0_Mod_0_RFSklearn =====
[18:05:40] Score for RF model: -0.478922
[18:05:40] ===== Start working with fold 1 for Lvl_0_Pipe_0_Mod_0_RFSklearn =====
[18:05:43] Score for RF model: -0.477559
[18:05:43] Fitting Lvl_0_Pipe_0_Mod_0_RFSklearn finished. score = -0.47824063661560884
[18:05:43] Lvl_0_Pipe_0_Mod_0_RFSklearn fitting and predicting completed
[18:05:43] Time left 3593.91 secs

[18:05:44] Start fitting Lvl_0_Pipe_1_Mod_0_LinearL2 ...
[18:05:44] Training params: {'tol': 1e-06, 'max_iter': 100, 'cs': [1e-05, 5e-05, 0.0001,␣
↪0.0005, 0.001, 0.005, 0.01, 0.05, 0.1, 0.5, 1, 5, 10, 50, 100, 500, 1000, 5000, 10000,␣
↪50000, 100000], 'early_stopping': 2, 'categorical_idx': [], 'embed_sizes': (), 'data_
↪size': 226}
[18:05:44] ===== Start working with fold 0 for Lvl_0_Pipe_1_Mod_0_LinearL2 =====
[18:05:44] Linear model: C = 1e-05 score = -0.5964374876210202
[18:05:44] Linear model: C = 5e-05 score = -0.4531700575784085
[18:05:44] Linear model: C = 0.0001 score = -0.4146961763297737
[18:05:45] Linear model: C = 0.0005 score = -0.38449422448982795
[18:05:45] Linear model: C = 0.001 score = -0.38138838560544014
[18:05:45] Linear model: C = 0.005 score = -0.38019812717729135
[18:05:45] Linear model: C = 0.01 score = -0.38034723995869585
[18:05:45] Linear model: C = 0.05 score = -0.38041431355765454
[18:05:45] ===== Start working with fold 1 for Lvl_0_Pipe_1_Mod_0_LinearL2 =====
[18:05:46] Linear model: C = 1e-05 score = -0.5906956397709
[18:05:46] Linear model: C = 5e-05 score = -0.45173117819934433
[18:05:46] Linear model: C = 0.0001 score = -0.4145036642322005
[18:05:46] Linear model: C = 0.0005 score = -0.3849509035123634
[18:05:47] Linear model: C = 0.001 score = -0.3813145757461596
[18:05:47] Linear model: C = 0.005 score = -0.37985413506750626
[18:05:47] Linear model: C = 0.01 score = -0.37999213463071496
[18:05:47] Linear model: C = 0.05 score = -0.38003681603029926
[18:05:47] Fitting Lvl_0_Pipe_1_Mod_0_LinearL2 finished. score = -0.3800261491230324
[18:05:47] Lvl_0_Pipe_1_Mod_0_LinearL2 fitting and predicting completed
[18:05:47] Time left 3589.22 secs
```

(continues on next page)

(continued from previous page)

```
[18:05:48] Start fitting Lvl_0_Pipe_2_Mod_0_CatBoost ...
[18:05:48] Training params: {'task_type': 'GPU', 'thread_count': 4, 'random_seed': 42,
↪'num_trees': 3000, 'learning_rate': 0.03, 'l2_leaf_reg': 0.01, 'bootstrap_type':
↪'Bernoulli', 'grow_policy': 'SymmetricTree', 'max_depth': 5, 'min_data_in_leaf': 1,
↪'one_hot_max_size': 10, 'fold_permutation_block': 1, 'boosting_type': 'Plain', 'boost_
↪from_average': True, 'od_type': 'Iter', 'od_wait': 100, 'max_bin': 32, 'feature_border_
↪type': 'GreedyLogSum', 'nan_mode': 'Min', 'verbose': 100, 'allow_writing_files': False,
↪ 'devices': '0'}
[18:05:49] ===== Start working with fold 0 for Lvl_0_Pipe_2_Mod_0_CatBoost =====
[18:05:49] 0:   learn: 4.9703982        test: 4.9996176 best: 4.9996176 (0)     total:␣
↪43.1ms   remaining: 2m 9s
[18:05:53] 100: learn: 3.1817923        test: 3.2535349 best: 3.2535349 (100)   total: 4.
↪01s    remaining: 1m 55s
[18:05:57] 200: learn: 2.5726766        test: 2.6805614 best: 2.6805614 (200)   total: 8.
↪03s    remaining: 1m 51s
[18:06:01] 300: learn: 2.2560121        test: 2.3947249 best: 2.3947249 (300)   total:␣
↪12.1s    remaining: 1m 48s
[18:06:05] 400: learn: 2.0597074        test: 2.2313428 best: 2.2313428 (400)   total:␣
↪16.2s    remaining: 1m 45s
[18:06:09] 500: learn: 1.9274641        test: 2.1288114 best: 2.1288114 (500)   total:␣
↪20.4s    remaining: 1m 41s
[18:06:13] 600: learn: 1.8216448        test: 2.0494447 best: 2.0494447 (600)   total:␣
↪24.6s    remaining: 1m 38s
[18:06:17] 700: learn: 1.7391222        test: 1.9927232 best: 1.9927232 (700)   total:␣
↪28.7s    remaining: 1m 34s
[18:06:22] 800: learn: 1.6764093        test: 1.9538165 best: 1.9538165 (800)   total:␣
↪32.9s    remaining: 1m 30s
[18:06:26] 900: learn: 1.6219885        test: 1.9225213 best: 1.9225213 (900)   total:␣
↪37.1s    remaining: 1m 26s
[18:06:30] 1000:        learn: 1.5738942        test: 1.8971965 best: 1.8971965 (1000) ␣
↪total: 41.2s    remaining: 1m 22s
[18:06:34] 1100:        learn: 1.5308305        test: 1.8747704 best: 1.8747704 (1100) ␣
↪total: 45.4s    remaining: 1m 18s
[18:06:38] 1200:        learn: 1.4944545        test: 1.8602100 best: 1.8602100 (1200) ␣
↪total: 49.5s    remaining: 1m 14s
[18:06:42] 1300:        learn: 1.4584951        test: 1.8450711 best: 1.8450711 (1300) ␣
↪total: 53.6s    remaining: 1m 10s
[18:06:46] 1400:        learn: 1.4277984        test: 1.8356888 best: 1.8356888 (1400) ␣
↪total: 57.7s    remaining: 1m 5s
[18:06:50] 1500:        learn: 1.3976121        test: 1.8263948 best: 1.8263948 (1500) ␣
↪total: 1m 1s    remaining: 1m 1s
[18:06:55] 1600:        learn: 1.3703607        test: 1.8202220 best: 1.8202220 (1600) ␣
↪total: 1m 5s    remaining: 57.6s
[18:06:59] 1700:        learn: 1.3433452        test: 1.8133380 best: 1.8133380 (1700) ␣
↪total: 1m 10s   remaining: 53.5s
[18:07:03] 1800:        learn: 1.3172573        test: 1.8065216 best: 1.8065216 (1800) ␣
↪total: 1m 14s   remaining: 49.3s
[18:07:07] 1900:        learn: 1.2922313        test: 1.8003041 best: 1.8003041 (1900) ␣
↪total: 1m 18s   remaining: 45.2s
[18:07:11] 2000:        learn: 1.2687614        test: 1.7968015 best: 1.7968015 (2000) ␣
↪total: 1m 22s   remaining: 41.1s
[18:07:15] 2100:        learn: 1.2466742        test: 1.7934962 best: 1.7934962 (2100) ␣
```

(continues on next page)

(continued from previous page)

```
↪total: 1m 26s   remaining: 36.9s
[18:07:19] 2200:        learn: 1.2247532        test: 1.7906494 best: 1.7906494 (2200) ␣
↪total: 1m 30s   remaining: 32.8s
[18:07:23] 2300:        learn: 1.2037321        test: 1.7880290 best: 1.7880053 (2299) ␣
↪total: 1m 34s   remaining: 28.7s
[18:07:27] 2400:        learn: 1.1833635        test: 1.7857023 best: 1.7857023 (2400) ␣
↪total: 1m 38s   remaining: 24.6s
[18:07:31] 2500:        learn: 1.1636590        test: 1.7833589 best: 1.7833589 (2500) ␣
↪total: 1m 42s   remaining: 20.5s
[18:07:35] 2600:        learn: 1.1448807        test: 1.7812717 best: 1.7812717 (2600) ␣
↪total: 1m 46s   remaining: 16.3s
[18:07:39] 2700:        learn: 1.1258719        test: 1.7781959 best: 1.7781959 (2700) ␣
↪total: 1m 50s   remaining: 12.2s
[18:07:43] 2800:        learn: 1.1079550        test: 1.7766084 best: 1.7766084 (2800) ␣
↪total: 1m 54s   remaining: 8.14s
[18:07:47] 2900:        learn: 1.0903868        test: 1.7753964 best: 1.7753941 (2899) ␣
↪total: 1m 58s   remaining: 4.05s
[18:07:51] 2999:        learn: 1.0737443        test: 1.7740101 best: 1.7740101 (2999) ␣
↪total: 2m 2s    remaining: 0us
[18:07:51] bestTest = 1.774010135
[18:07:51] bestIteration = 2999
[18:07:51] ===== Start working with fold 1 for Lvl_0_Pipe_2_Mod_0_CatBoost =====
[18:07:52] 0:   learn: 4.9945826        test: 4.9698499 best: 4.9698499 (0)     total:␣
↪41.8ms   remaining: 2m 5s
[18:07:56] 100: learn: 3.1884268        test: 3.2469980 best: 3.2469980 (100)   total: 3.
↪98s    remaining: 1m 54s
[18:08:00] 200: learn: 2.5842575        test: 2.6880010 best: 2.6880010 (200)   total: 7.
↪96s    remaining: 1m 50s
[18:08:04] 300: learn: 2.2647479        test: 2.4010427 best: 2.4010427 (300)   total:␣
↪12s      remaining: 1m 47s
[18:08:08] 400: learn: 2.0722203        test: 2.2444863 best: 2.2444863 (400)   total:␣
↪16.1s    remaining: 1m 44s
[18:08:12] 500: learn: 1.9358628        test: 2.1373584 best: 2.1373584 (500)   total:␣
↪20.3s    remaining: 1m 41s
[18:08:16] 600: learn: 1.8365382        test: 2.0652895 best: 2.0652895 (600)   total:␣
↪24.4s    remaining: 1m 37s
[18:08:20] 700: learn: 1.7542585        test: 2.0068865 best: 2.0068865 (700)   total:␣
↪28.6s    remaining: 1m 33s
[18:08:24] 800: learn: 1.6883776        test: 1.9628955 best: 1.9628955 (800)   total:␣
↪32.7s    remaining: 1m 29s
[18:08:29] 900: learn: 1.6305606        test: 1.9266863 best: 1.9266863 (900)   total:␣
↪36.9s    remaining: 1m 25s
[18:08:33] 1000:        learn: 1.5848000        test: 1.9039980 best: 1.9039980 (1000) ␣
↪total: 41s      remaining: 1m 21s
[18:08:37] 1100:        learn: 1.5422530        test: 1.8820810 best: 1.8820810 (1100) ␣
↪total: 45.2s    remaining: 1m 17s
[18:08:41] 1200:        learn: 1.5048244        test: 1.8662495 best: 1.8662495 (1200) ␣
↪total: 49.3s    remaining: 1m 13s
[18:08:45] 1300:        learn: 1.4705212        test: 1.8536784 best: 1.8536784 (1300) ␣
↪total: 53.4s    remaining: 1m 9s
[18:08:49] 1400:        learn: 1.4377168        test: 1.8420271 best: 1.8420271 (1400) ␣
↪total: 57.5s    remaining: 1m 5s
```

(continues on next page)

(continued from previous page)

```
[18:08:53] 1500:        learn: 1.4068988        test: 1.8320774 best: 1.8320661 (1499) ␣
↪total: 1m 1s    remaining: 1m 1s
[18:08:57] 1600:        learn: 1.3788303        test: 1.8235099 best: 1.8235099 (1600) ␣
↪total: 1m 5s    remaining: 57.4s
[18:09:01] 1700:        learn: 1.3520362        test: 1.8165897 best: 1.8165897 (1700) ␣
↪total: 1m 9s    remaining: 53.3s
[18:09:06] 1800:        learn: 1.3263396        test: 1.8109005 best: 1.8109005 (1800) ␣
↪total: 1m 13s   remaining: 49.2s
[18:09:10] 1900:        learn: 1.3017944        test: 1.8052676 best: 1.8052676 (1900) ␣
↪total: 1m 18s   remaining: 45.1s
[18:09:14] 2000:        learn: 1.2771414        test: 1.7995645 best: 1.7995645 (2000) ␣
↪total: 1m 22s   remaining: 41s
[18:09:18] 2100:        learn: 1.2540940        test: 1.7948637 best: 1.7948637 (2100) ␣
↪total: 1m 26s   remaining: 36.9s
[18:09:22] 2200:        learn: 1.2319167        test: 1.7913644 best: 1.7913580 (2194) ␣
↪total: 1m 30s   remaining: 32.7s
[18:09:26] 2300:        learn: 1.2107906        test: 1.7883785 best: 1.7883785 (2300) ␣
↪total: 1m 34s   remaining: 28.6s
[18:09:30] 2400:        learn: 1.1906539        test: 1.7857203 best: 1.7857203 (2400) ␣
↪total: 1m 38s   remaining: 24.5s
[18:09:34] 2500:        learn: 1.1705045        test: 1.7827000 best: 1.7827000 (2500) ␣
↪total: 1m 42s   remaining: 20.4s
[18:09:38] 2600:        learn: 1.1510307        test: 1.7808157 best: 1.7808028 (2598) ␣
↪total: 1m 46s   remaining: 16.3s
[18:09:42] 2700:        learn: 1.1323293        test: 1.7789892 best: 1.7789676 (2697) ␣
↪total: 1m 50s   remaining: 12.2s
[18:09:46] 2800:        learn: 1.1142749        test: 1.7767626 best: 1.7767626 (2800) ␣
↪total: 1m 54s   remaining: 8.14s
[18:09:50] 2900:        learn: 1.0964777        test: 1.7755168 best: 1.7755025 (2899) ␣
↪total: 1m 58s   remaining: 4.05s
[18:09:54] 2999:        learn: 1.0793822        test: 1.7741527 best: 1.7741169 (2997) ␣
↪total: 2m 2s    remaining: 0us
[18:09:54] bestTest = 1.774116879
[18:09:54] bestIteration = 2997
[18:09:54] Shrink model to first 2998 iterations.
[18:09:54] Fitting Lvl_0_Pipe_2_Mod_0_CatBoost finished. score = -0.24088612521804945
[18:09:54] Lvl_0_Pipe_2_Mod_0_CatBoost fitting and predicting completed
[18:09:54] Time left 3342.18 secs

[18:09:54] Layer 1 training completed.

[18:09:54] Blending: optimization starts with equal weights and score -0.
↪31863480146881884
[18:09:54] Blending: iteration 0: score = -0.24088612521804945, weights = [0. 0. 1.]
[18:09:55] Blending: iteration 1: score = -0.24088612521804945, weights = [0. 0. 1.]
[18:09:55] Blending: no score update. Terminated

[18:09:55] Automl preset training completed in 257.95 seconds

[10:40:24] Model description:
Final prediction for new objects (level 0) =
         1.00000 * (2 averaged models Lvl_0_Pipe_2_Mod_0_CatBoost)
```

(continues on next page)

(continued from previous page)

`seq_test` is consisted of: - seq_test.date — base date, from which predictions are made - seq_test.id — id of time series - seq_test.data — predictions themselves

We can collect these values to the df.DataFrame:

[21]:
```
freq = pd.infer_freq(train[DATE_COLUMN].iloc[:10])
date_list = [seq_test.date[0] + pd.Timedelta(1+i, unit=freq) for i in range(HORIZON)]
date_list = date_list * len(seq_test.id)
pred_list = list(seq_test.data.reshape(-1))
id_list = np.repeat(seq_test.id, HORIZON)

df_dict = {
    "ID": id_list,
    "date": date_list,
    "pred": pred_list
}

res_df_global = pd.DataFrame(df_dict)
res_df_global = res_df_global.merge(test, on=["ID", "date"])

res_df_global
```

[21]:

	ID	date	pred	value
0	0	2025-05-24	2.384080	1.581092
1	0	2025-05-25	3.279628	2.376163
2	0	2025-05-26	3.530711	2.627754
3	0	2025-05-27	3.079974	2.105468
4	0	2025-05-28	2.362342	1.969833
..	..	...	...	...
145	4	2025-06-18	1.874377	2.149213
146	4	2025-06-19	1.947542	1.807998
147	4	2025-06-20	2.211135	1.810103
148	4	2025-06-21	2.608910	1.941590
149	4	2025-06-22	3.128166	2.590021

[150 rows x 4 columns]

Check MAE for all time series:

[22]:
```
mae_df_global = res_df_global.groupby("ID").apply(
    lambda x: mean_absolute_error(x["value"], x["pred"])
).reset_index()

mae_df_global.columns = ["ID", "MAE_global"]
mae_df_global
```

[22]:

	ID	MAE_global
0	0	0.344390
1	1	0.227540
2	2	0.477782
3	3	0.342722
4	4	0.193969

Compare MAE for two strategies (univariate/local- and global-modelling):

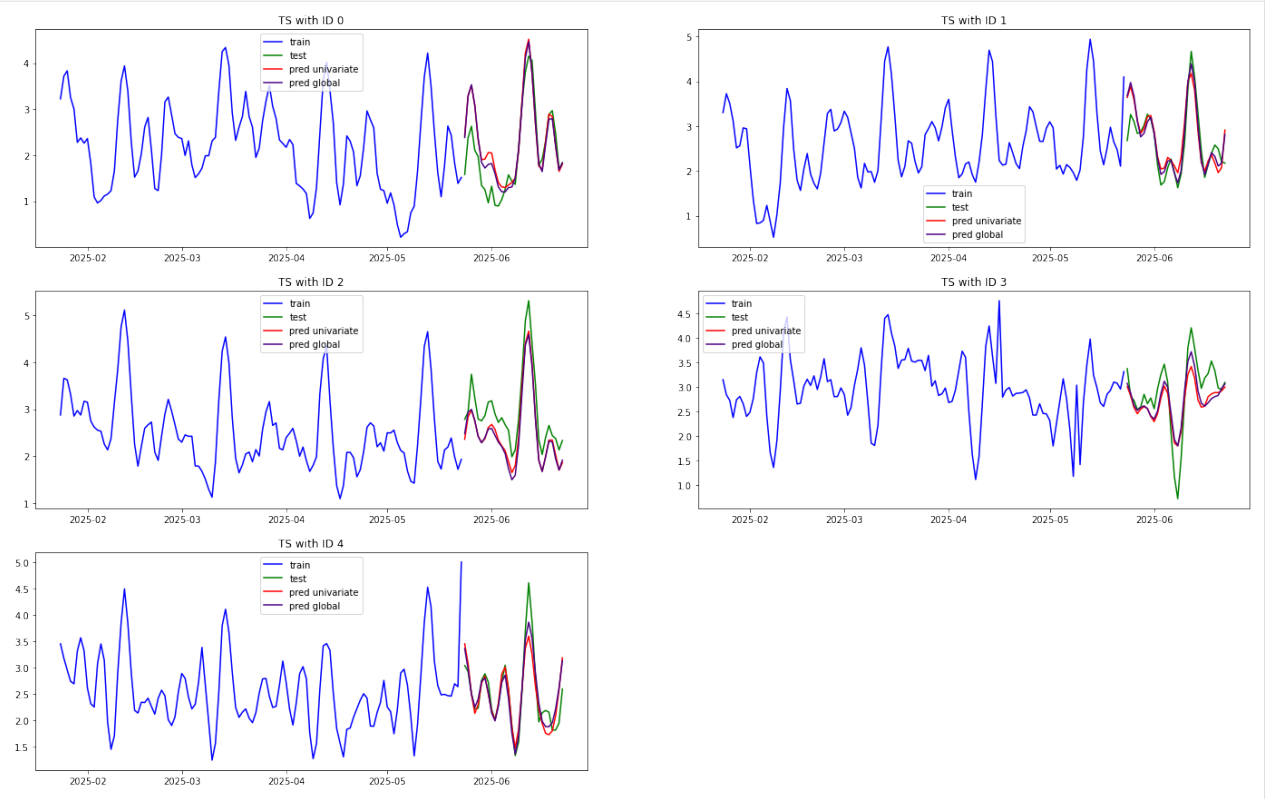
[23]:
```
mae_df_global.merge(mae_df_univariate, on=["ID"])
```

[23]:

	ID	MAE_global	MAE_univariate
0	0	0.344390	0.371187
1	1	0.227540	0.292606
2	2	0.477782	0.435712
3	3	0.342722	0.376864
4	4	0.193969	0.225450

Lastly, get a graph of models' predictions in comparison with each other and the true values

[24]:
```
LAST_TRAIN_N = 120

num_ids = len(df[ID_COLUMN].value_counts())
subplots_num_columns = 2
subplots_num_rows = num_ids // 2 + num_ids % 2
fig, ax = plt.subplots(
    subplots_num_rows,
    subplots_num_columns,
    figsize=(24, 5 * subplots_num_rows)
)

for i, ts_id in enumerate(df[ID_COLUMN].unique()):
    i_row = i // 2
    i_col = i % 2

    current_train = train[train[ID_COLUMN] == ts_id]
    current_res_df_univariate = res_df_univariate[res_df_univariate[ID_COLUMN] == ts_id]
    current_res_df_global = res_df_global[res_df_global[ID_COLUMN] == ts_id]

    ax[i_row, i_col].plot(current_train[DATE_COLUMN][-LAST_TRAIN_N:], current_
↪train[TARGET_COLUMN][-LAST_TRAIN_N:], c='b', label='train')
    ax[i_row, i_col].plot(current_res_df_univariate[DATE_COLUMN], current_res_df_
↪univariate["test"], c='g', label='test')
    ax[i_row, i_col].plot(current_res_df_univariate[DATE_COLUMN], current_res_df_
↪univariate["pred"], c='r', label='pred univariate')
    ax[i_row, i_col].plot(current_res_df_global[DATE_COLUMN], current_res_df_global["pred
↪"], c='indigo', label='pred global')

    ax[i_row, i_col].title.set_text(f"TS with ID {ts_id}")
    ax[i_row, i_col].legend()

axes_to_del = i_row * subplots_num_rows + i_col * subplots_num_columns - num_ids
for i in range(axes_to_del):
    i_row = (subplots_num_rows - 1) - i // 2
    i_col = (subplots_num_columns - 1) - i % 2
    fig.delaxes(ax[i_row][i_col])

plt.show();
```



Additional materials

- Official LightAutoML github repo
- LightAutoML documentation

2.2 HypEx Tutorials

2.2.1 AA test

An A/A test is a variation of an A/B test, the peculiarity of which is that the original is compared with itself, as opposed to an A/B test, which compares samples before and after exposure.

0. Import Libraries

[1]:
```
import numpy as np
import pandas as pd

from lightautoml.addons.hypex import AATest
from lightautoml.addons.hypex.utils.tutorial_data_creation import create_test_data

pd.options.display.float_format = '{:,.2f}'.format

np.random.seed(42)  # needed to create example data
```

[2]:
```
def show_result(result):
    for k, v in result.items():
        print(k)
        display(v)
        print()
```

1. Create or upload your dataset

In this case we will create random dataset with known effect size
If you have your own dataset, go to the part 2

[3]:
```
data = create_test_data(rs=52, na_step=10, nan_cols=['age', 'gender'])
data
```

[3]:
```
      user_id  signup_month  treat  pre_spends  post_spends   age gender  \
0           0             0      0      488.00       414.44   NaN      M
1           1             8      1      512.50       462.22 26.00    NaN
2           2             7      1      483.00       479.44 25.00      M
3           3             0      0      501.50       424.33 39.00      M
4           4             1      1      543.00       514.56 18.00      F
...       ...           ...    ...         ...          ...   ...    ...
9995     9995            10      1      538.50       450.44 42.00      M
9996     9996             0      0      500.50       430.89 26.00      F
9997     9997             3      1      473.00       534.11 22.00      F
9998     9998             2      1      495.00       523.22 67.00      F
9999     9999             7      1      508.00       475.89 38.00      F

        industry
0     E-commerce
1     E-commerce
2      Logistics
3     E-commerce
4     E-commerce
...          ...
9995   Logistics
9996   Logistics
9997  E-commerce
9998  E-commerce
9999  E-commerce

[10000 rows x 8 columns]
```

2. AATest

2.0 Initialize parameters

`info_col` used to define informative attributes that should NOT be part of testing, such as user_id and signup_month

[4]:
```
info_cols = ['user_id', 'signup_month']
target = ['post_spends', 'pre_spends']
```

2.1 Simple AA-test

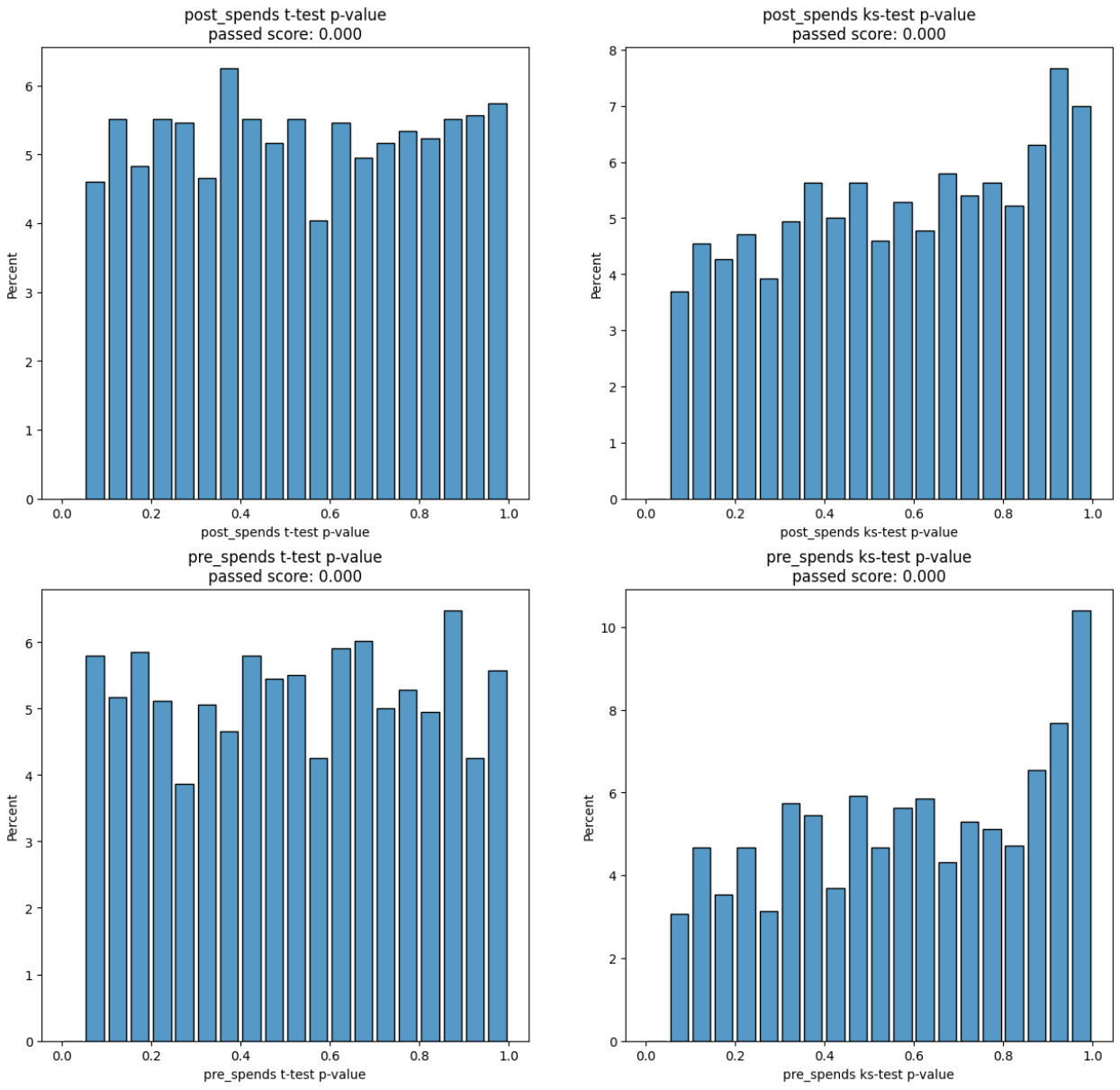
This is the easiest way to initialize and calculate metrics on a AA-test (default - on 2000 iterations)Use it when you are clear about each attribute or if you don't have any additional task conditions (like grouping)

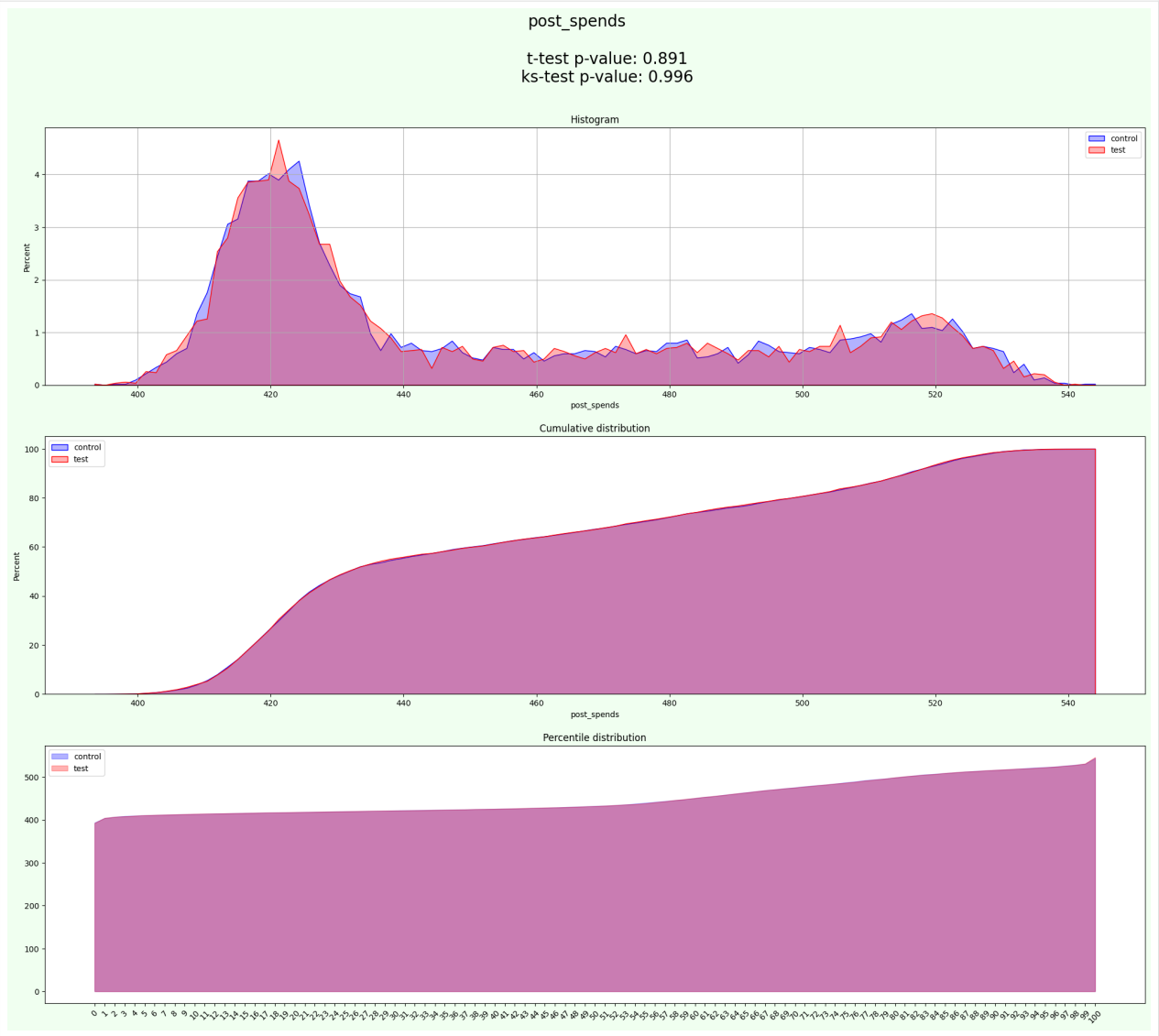
You can also add some extra arguments to the process(): * plot_set - types of plot, that you want to show ("hist", "cumulative", "percentile") * figsize - size of figure for plots * alpha - value to change the transparency of the histogram plot * bins - generic bin parameter that can be the name of a reference rule, the number of bins, or the breaks of the bins * title_size - size of title for plots

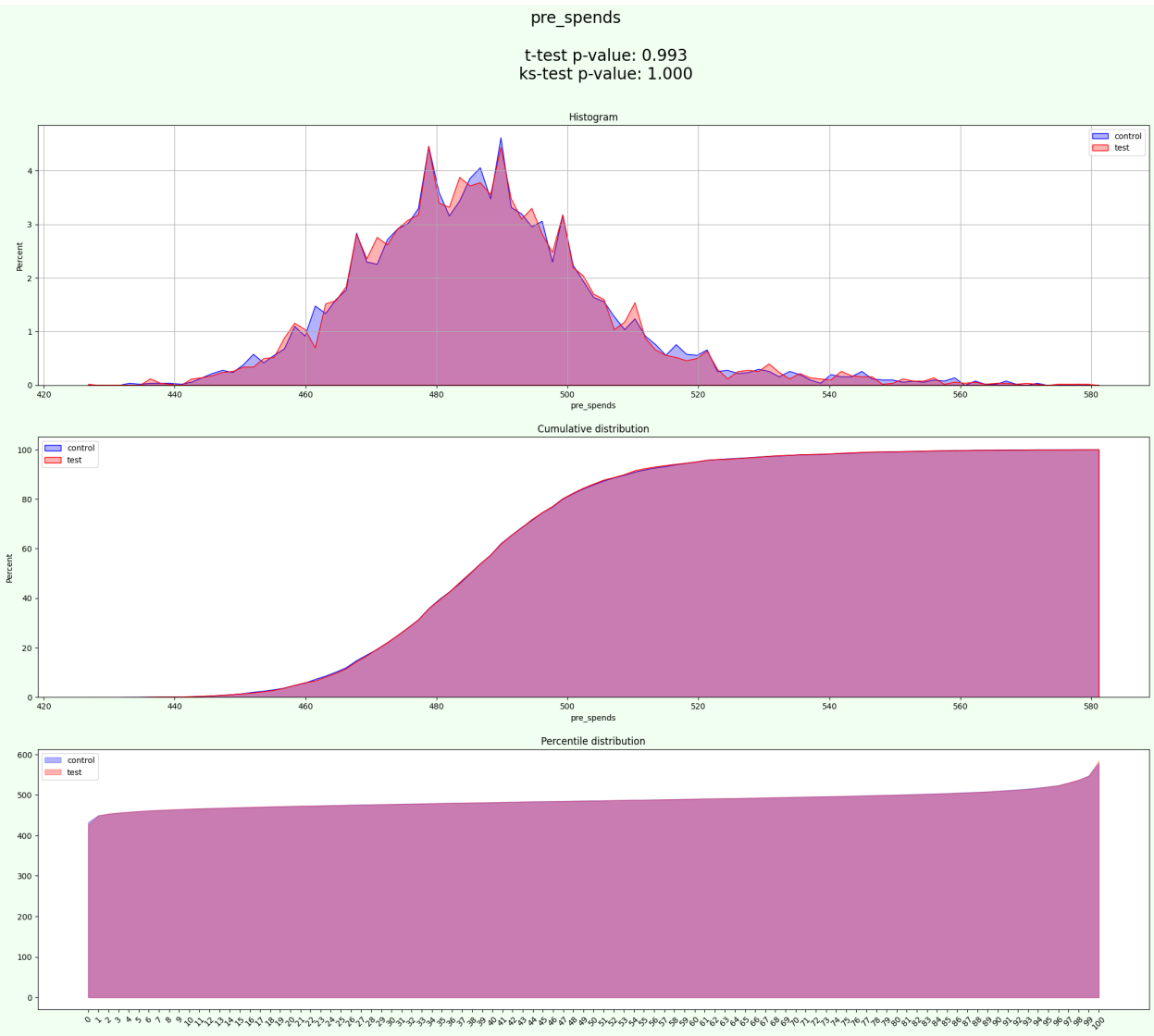
[5]:
```
experiment = AATest(info_cols=info_cols, target_fields=target)
```

[6]:
```
results = experiment.process(data, iterations=2000)
```

```
  0%|          | 0/2000 [00:00<?, ?it/s]
```




post_spends
t-test p-value: 0.891
ks-test p-value: 0.996
Histogram
control
test
Percent
post_spends
Cumulative distribution
Percentile distribution



[7]:
```
show_result(results)
```

experiments

	random_state	post_spends a mean	post_spends b mean	\
0	1	452.18	452.15	
1	2	452.82	451.50	
2	4	452.41	451.92	
3	5	452.64	451.69	
4	6	452.70	451.63	
...	...	...	...	
1755	1993	452.29	452.04	
1756	1994	452.56	451.77	
1757	1995	452.30	452.03	
1758	1996	451.89	452.44	
1759	1997	452.52	451.81	

(continues on next page)

(continued from previous page)

```
      post_spends ab delta  post_spends ab delta %  \
0                    -0.03                    -0.01
1                    -1.32                    -0.29
2                    -0.50                    -0.11
3                    -0.96                    -0.21
4                    -1.07                    -0.24
...                    ...                      ...
1755                 -0.24                    -0.05
1756                 -0.78                    -0.17
1757                 -0.26                    -0.06
1758                  0.55                     0.12
1759                 -0.70                    -0.16

      post_spends t-test p-value  post_spends ks-test p-value  \
0                          0.97                         0.56
1                          0.09                         0.18
2                          0.53                         0.06
3                          0.23                         0.41
4                          0.17                         0.53
...                         ...                          ...
1755                       0.76                         0.95
1756                       0.32                         0.11
1757                       0.74                         0.91
1758                       0.48                         0.78
1759                       0.37                         0.10

      post_spends t-test passed  post_spends ks-test passed  \
0                         False                        False
1                         False                        False
2                         False                        False
3                         False                        False
4                         False                        False
...                         ...                          ...
1755                      False                        False
1756                      False                        False
1757                      False                        False
1758                      False                        False
1759                      False                        False

      pre_spends a mean  ...  pre_spends ks-test passed  control %  test %  \
0                487.33  ...                      False      50.00   50.00
1                487.04  ...                      False      50.00   50.00
2                487.20  ...                      False      50.00   50.00
3                486.90  ...                      False      50.00   50.00
4                487.31  ...                      False      50.00   50.00
...                 ...  ...                        ...        ...     ...
1755             486.96  ...                      False      50.00   50.00
1756             487.12  ...                      False      50.00   50.00
1757             486.94  ...                      False      50.00   50.00
1758             487.30  ...                      False      50.00   50.00
1759             487.10  ...                      False      50.00   50.00
```

(continues on next page)

(continued from previous page)

```
      control size  test size  t-test mean p-value  ks-test mean p-value  \
0             5000       5000                 0.59                  0.49
1             5000       5000                 0.44                  0.45
2             5000       5000                 0.56                  0.36
3             5000       5000                 0.26                  0.40
4             5000       5000                 0.21                  0.35
...            ...        ...                  ...                   ...
1755          5000       5000                 0.61                  0.71
1756          5000       5000                 0.61                  0.21
1757          5000       5000                 0.57                  0.89
1758          5000       5000                 0.38                  0.86
1759          5000       5000                 0.66                  0.40

      t-test passed %  ks-test passed %  mean_tests_score
0                0.00              0.00              0.52
1                0.00              0.00              0.45
2                0.00              0.00              0.43
3                0.00              0.00              0.35
4                0.00              0.00              0.30
...               ...               ...               ...
1755             0.00              0.00              0.68
1756             0.00              0.00              0.34
1757             0.00              0.00              0.79
1758             0.00              0.00              0.70
1759             0.00              0.00              0.49

[1760 rows x 26 columns]
```

```
aa_score
```

```
             t-test passed score  ks-test passed score t-test aa passed  \
post_spends                 0.00                  0.00             0.00
pre_spends                  0.00                  0.00             0.00
mean                        0.00                  0.00             0.00

            ks-test aa passed
post_spends              0.00
pre_spends               0.00
mean                     0.00
```

```
split
```

```
      user_id  signup_month  treat  pre_spends  post_spends   age gender  \
0           0             0      0      488.00       414.44   NaN      M
1           1             8      1      512.50       462.22 26.00    NaN
2           4             1      1      543.00       514.56 18.00      F
3           5             6      1      486.50       486.56 44.00      M
4           8             4      1      465.50       506.00 66.00      M
...       ...           ...    ...         ...          ...   ...    ...
9995     9990             0      0      490.00       426.00   NaN      M
9996     9992             0      0      491.50       424.00 29.00      M
9997     9996             0      0      500.50       430.89 26.00      F
```

(continues on next page)

(continued from previous page)

```
9998     9997             3      1      473.00       534.11 22.00      F
9999     9998             2      1      495.00       523.22 67.00      F

        industry    group
0     E-commerce     test
1     E-commerce     test
2     E-commerce     test
3     E-commerce     test
4      Logistics     test
...          ...      ...
9995   Logistics  control
9996  E-commerce  control
9997   Logistics  control
9998  E-commerce  control
9999  E-commerce  control

[10000 rows x 9 columns]
```

```
best_experiment_stat
```

```
            a mean b mean ab delta ab delta % t-test p-value ks-test p-value  \
post_spends 452.22 452.11    -0.11      -0.02           0.89            1.00
pre_spends  487.09 487.10     0.00       0.00           0.99            1.00

            t-test passed ks-test passed
post_spends         False          False
pre_spends          False          False
```

```
split_stat
```

```
control %              50.00
test %                 50.00
control size            5000
test size               5000
t-test mean p-value     0.94
ks-test mean p-value    1.00
t-test passed %         0.00
ks-test passed %        0.00
mean_tests_score        0.98
Name: 60, dtype: object
```

```
resume
```

```
            aa test passed split is uniform
post_spends         not OK               OK
pre_spends          not OK               OK
```

[8]:
```
results.keys()
```

[8]:
```
dict_keys(['experiments', 'aa_score', 'split', 'best_experiment_stat', 'split_stat',
↪'resume'])
```

`results` is a dictionary with dataframes as values. * 'split' - result of separation, column 'group' contains values 'test' and 'control'

* 'resume' - summary of all results

* 'aa_score' - score of T-test and Kolmogorov-Smirnov test * 'experiments' - is a table of results of experiments, which includes - means of all targets in a and b samples, - p_values of Student t-test and test Kolmogorova-Smirnova, - and results of tests (did data on the random_state passes the uniform test) * 'best_experiment_stat' - like previous point but only for the best experiment * 'split_stat' - metrics and statistics tests for result of split

[9]:
```
results['aa_score']
```

[9]:
```
             t-test passed score  ks-test passed score t-test aa passed  \
post_spends                 0.00                  0.00             0.00
pre_spends                  0.00                  0.00             0.00
mean                        0.00                  0.00             0.00

            ks-test aa passed
post_spends              0.00
pre_spends               0.00
mean                     0.00
```

[10]:
```
results['resume']
```

[10]:
```
            aa test passed split is uniform
post_spends         not OK               OK
pre_spends          not OK               OK
```

2.2 Single experiment

To get stable results lets fix `random_state`

[11]:
```
random_state = 11
```

To perform single experiment you can use `sampling_metrics()`

[12]:
```
experiment = AATest(info_cols=info_cols, target_fields=target)
metrics, dict_of_datas = experiment.sampling_metrics(data=data, random_
↪state).values()
```

The results contains the same info as in multisampling, but on one experiment

[13]:
```
metrics
```

[13]:
```
{'random_state': 11,
 'post_spends a mean': 451.8546,
 'post_spends b mean': 452.4745111111112,
 'post_spends ab delta': 0.6199111111112074,
 'post_spends ab delta %': 0.13700464797208323,
 'post_spends t-test p-value': 0.43154056610193947,
 'post_spends ks-test p-value': 0.95721723072851,
 'post_spends t-test passed': False,
 'post_spends ks-test passed': False,
 'pre_spends a mean': 487.2131,
 'pre_spends b mean': 486.9744,
```

(continues on next page)

(continued from previous page)

```
 'pre_spends ab delta': -0.23869999999999436,
 'pre_spends ab delta %': -0.04901695037766718,
 'pre_spends t-test p-value': 0.5271083329122467,
 'pre_spends ks-test p-value': 0.14861030130677552,
 'pre_spends t-test passed': False,
 'pre_spends ks-test passed': False,
 'control %': 50.0,
 'test %': 50.0,
 'control size': 5000,
 'test size': 5000,
 't-test mean p-value': 0.4793244495070931,
 'ks-test mean p-value': 0.5529137660176427,
 't-test passed %': 0.0,
 'ks-test passed %': 0.0,
 'mean_tests_score': 0.5283839938474595}
```

[14]:
```
dict_of_datas[random_state]
```

[14]:
```
      user_id  signup_month  treat  pre_spends  post_spends   age gender  \
0           1             8      1      512.50       462.22 26.00    NaN
1           2             7      1      483.00       479.44 25.00      M
2           5             6      1      486.50       486.56 44.00      M
3           6            11      1      483.50       433.89 28.00      F
4          11             4      1      498.50       516.89 58.00    NaN
...       ...           ...    ...         ...          ...   ...    ...
9995     9986             0      0      494.00       432.11 39.00      M
9996     9989             6      1      466.50       487.44 19.00      F
9997     9991             0      0      482.50       421.89 43.00    NaN
9998     9995            10      1      538.50       450.44 42.00      M
9999     9998             2      1      495.00       523.22 67.00      F

        industry    group
0     E-commerce     test
1      Logistics     test
2     E-commerce     test
3      Logistics     test
4     E-commerce     test
...          ...      ...
9995   Logistics  control
9996  E-commerce  control
9997   Logistics  control
9998   Logistics  control
9999  E-commerce  control

[10000 rows x 9 columns]
```

[15]:
```
results = experiment.experiment_result_transform(pd.Series(metrics))
```

[16]:
```
results.keys()
```

[16]:
```
dict_keys(['best_experiment_stat', 'best_split_stat'])
```

[17]:
```
results['best_experiment_stat']
```

[17]:
```
            a mean b mean ab delta ab delta % t-test p-value ks-test p-value  \
post_spends 451.85 452.47     0.62       0.14           0.43            0.96
pre_spends  487.21 486.97    -0.24      -0.05           0.53            0.15

            t-test passed ks-test passed
post_spends         False          False
pre_spends          False          False
```

[18]:
```
results['best_split_stat']
```

[18]:
```
control %              50.00
test %                 50.00
control size            5000
test size               5000
t-test mean p-value     0.48
ks-test mean p-value    0.55
t-test passed %         0.00
ks-test passed %        0.00
mean_tests_score        0.53
dtype: object
```

2.3 AA-test with grouping

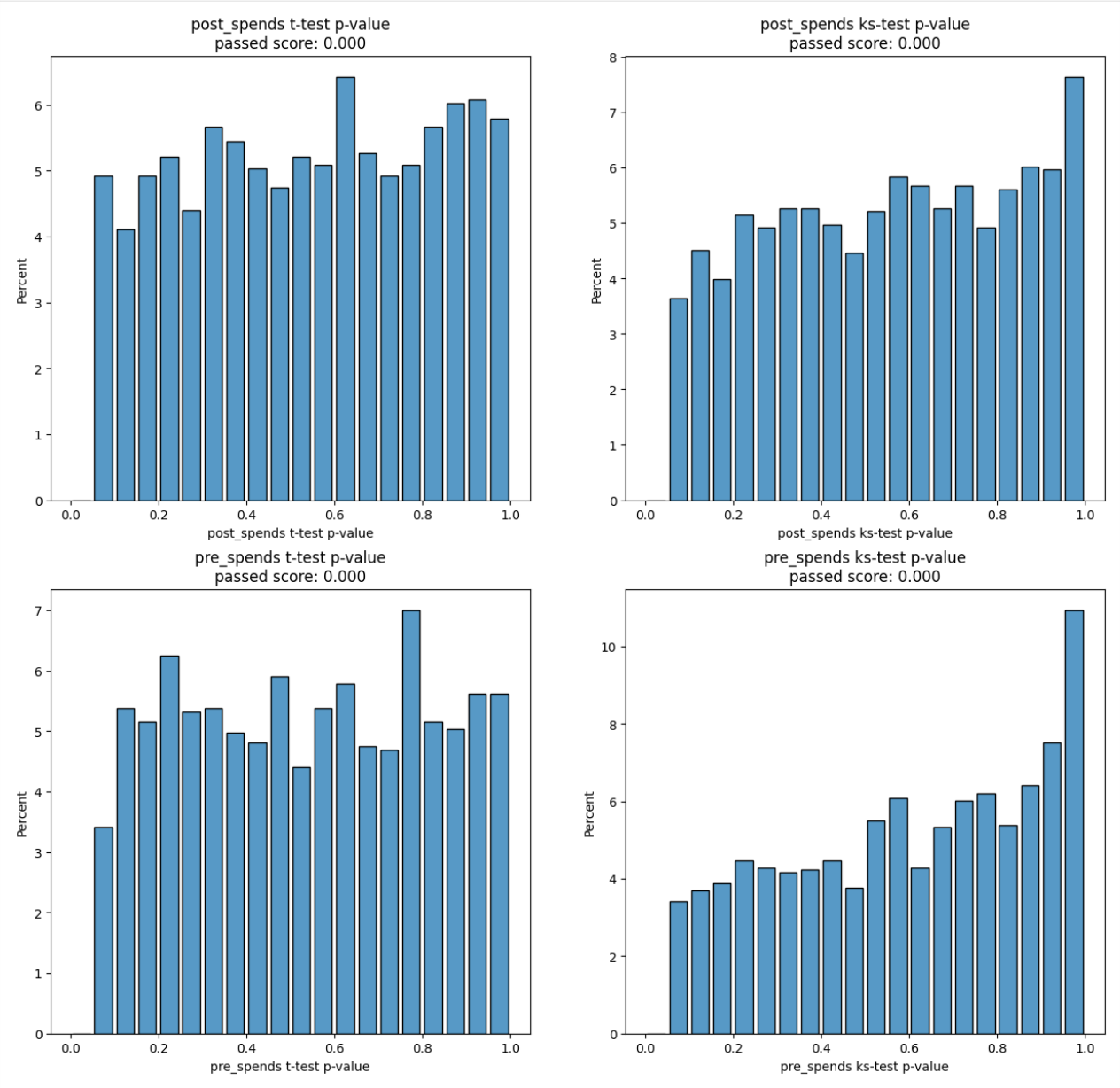
To perform experiment that separates samples by groups `group_col` can be used

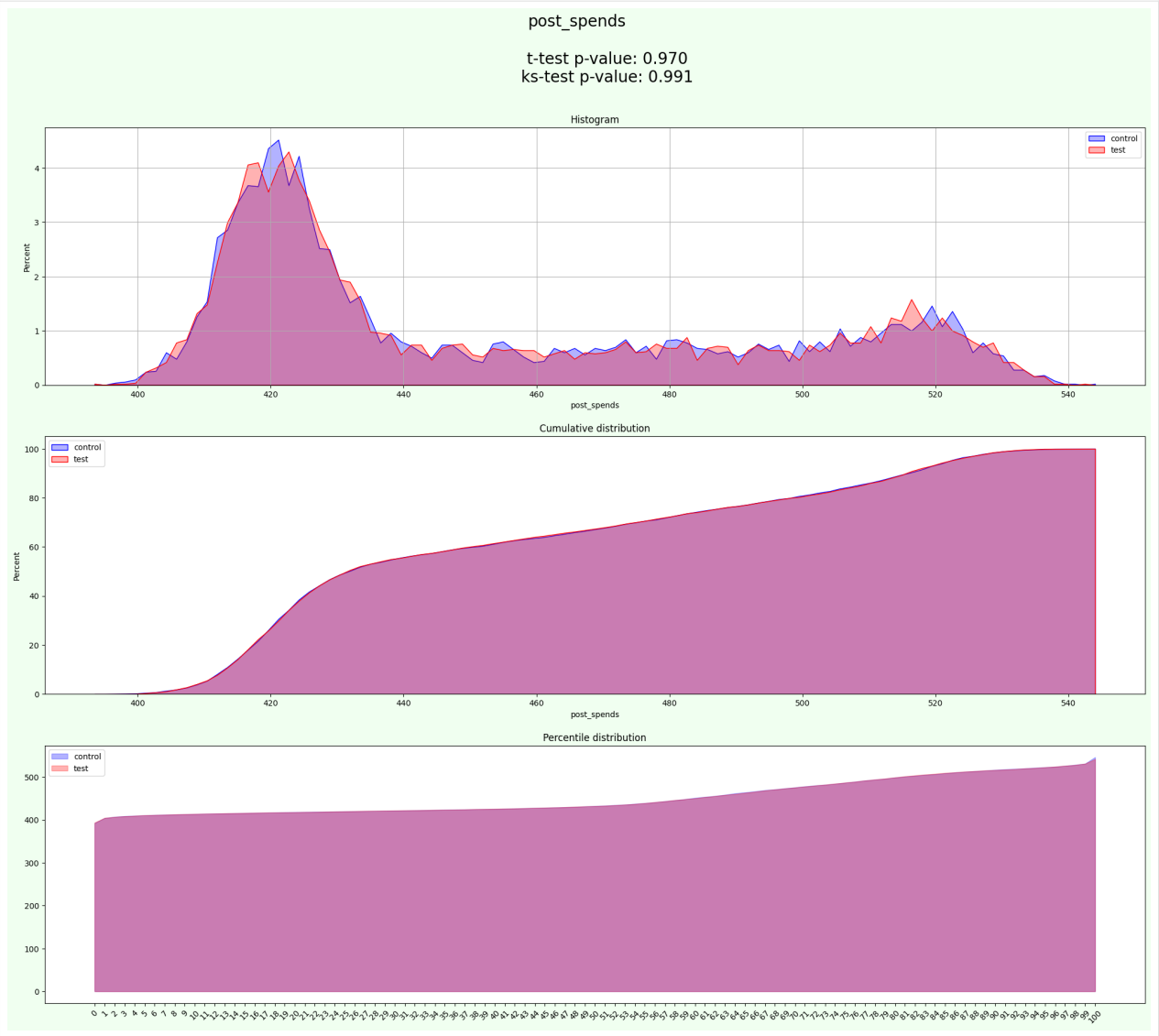
[19]:
```
info_cols = ['user_id', 'signup_month']
target = ['post_spends', 'pre_spends']

group_cols = 'industry'
```

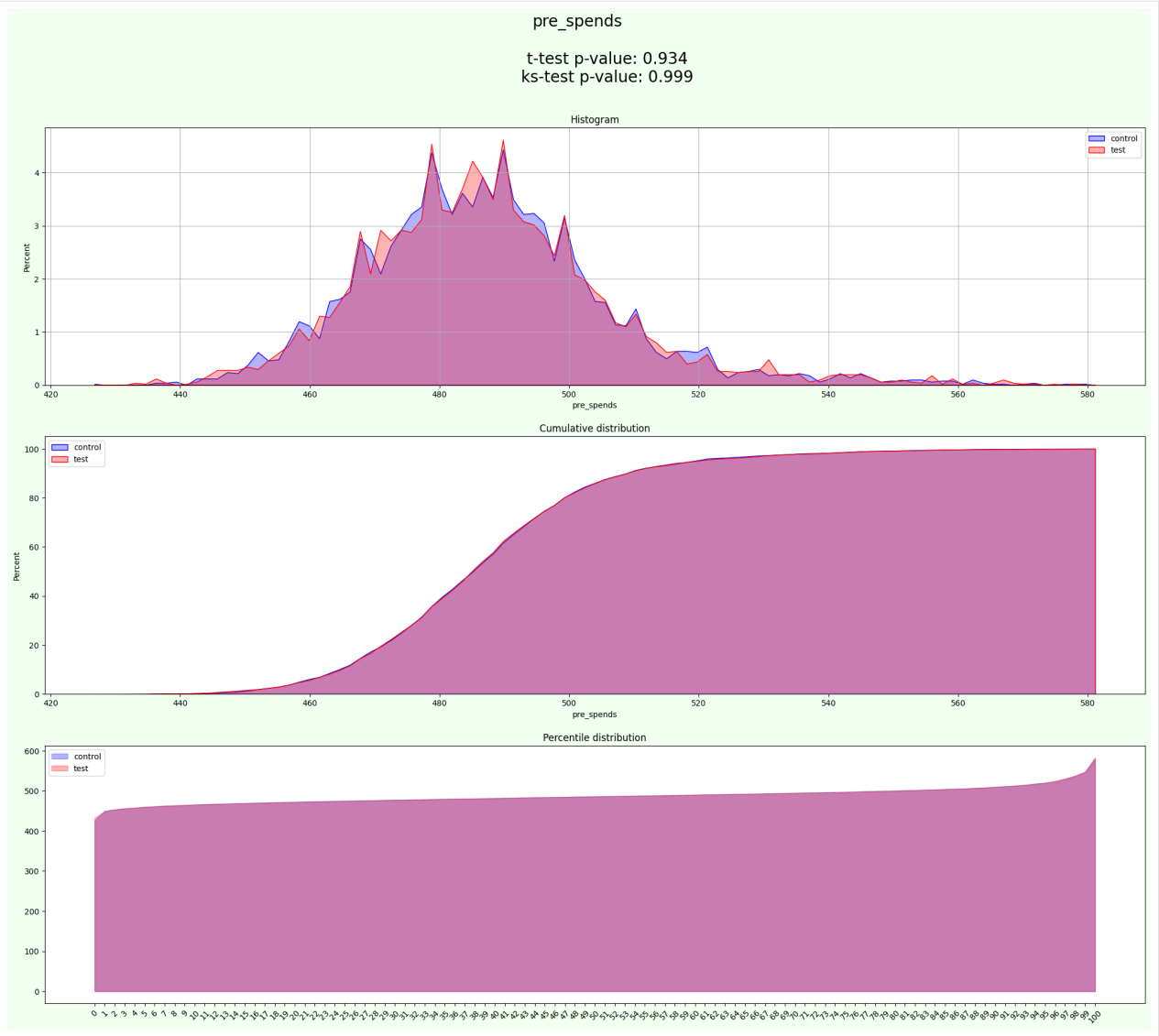
[20]:
```
experiment = AATest(info_cols=info_cols, target_fields=target, group_cols=group_cols)
```

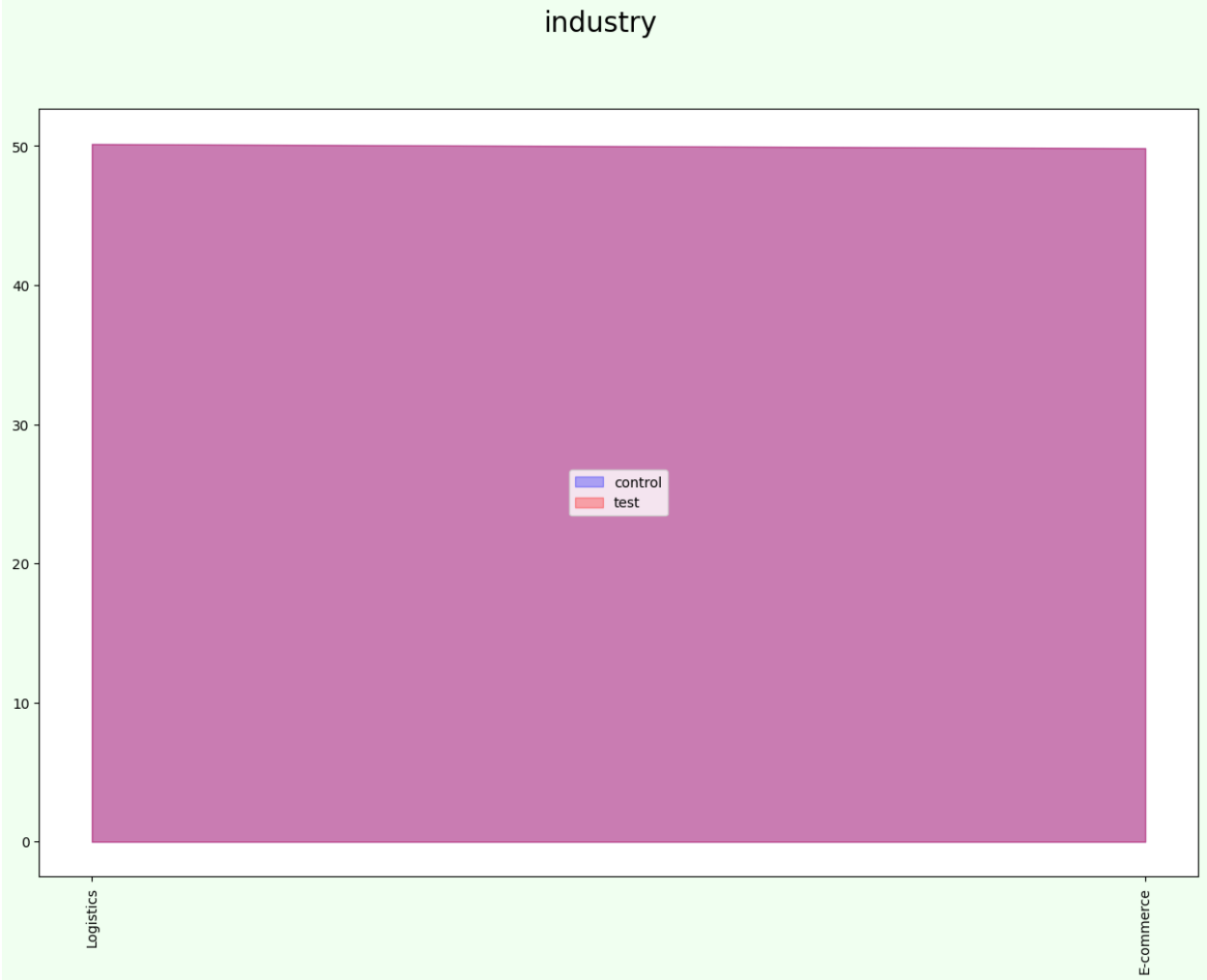
[21]:
```
results = experiment.process(data=data, iterations=2000)
```

```
  0%|          | 0/2000 [00:00<?, ?it/s]
```


post_spends t-test p-value
passed score: 0.000
Percent
post_spends t-test p-value
post_spends ks-test p-value
passed score: 0.000
Percent
post_spends ks-test p-value
pre_spends t-test p-value
passed score: 0.000
Percent
pre_spends t-test p-value
pre_spends ks-test p-value
passed score: 0.000
Percent
pre_spends ks-test p-value


post_spends
t-test p-value: 0.970
ks-test p-value: 0.991
Histogram
control
test
Percent
post_spends
Cumulative distribution
control
test
Percent
post_spends
Percentile distribution
control
test


pre_spends
t-test p-value: 0.934
ks-test p-value: 0.999
Histogram
control
test
Percent
pre_spends
Cumulative distribution
Percentile distribution



The result is in the same format as without groups

In this regime groups equally divided on each sample (test and control):

[22]:
```
results['split']['industry'].value_counts(normalize=True) * 100
```

[22]:
```
industry
Logistics    50.15
E-commerce   49.85
Name: proportion, dtype: float64
```

[23]:
```
results['split'].groupby(['industry', 'group'])[['user_id']].count()
```

[23]:
```
                    user_id
industry   group
E-commerce control     2493
           test        2492
Logistics  control     2508
           test        2507
```

[24]:
```
show_result(results)
```

```
experiments
```

```
      random_state  post_spends a mean  post_spends b mean  \
0                0              451.53              452.80
1                2              452.53              451.80
2                3              452.10              452.23
3                4              452.18              452.15
4                7              452.38              451.95
...            ...                 ...                 ...
1723          1995              452.70              451.63
1724          1996              452.36              451.96
1725          1997              452.08              452.25
1726          1998              451.96              452.36
1727          1999              452.38              451.95

      post_spends ab delta  post_spends ab delta %  \
0                     1.27                    0.28
1                    -0.73                   -0.16
2                     0.13                    0.03
3                    -0.03                   -0.01
4                    -0.42                   -0.09
...                    ...                     ...
1723                 -1.07                   -0.24
1724                 -0.40                   -0.09
1725                  0.18                    0.04
1726                  0.40                    0.09
1727                 -0.42                   -0.09

      post_spends t-test p-value  post_spends ks-test p-value  \
0                            0.11                         0.19
1                            0.35                         0.83
2                            0.87                         0.85
3                            0.97                         0.30
4                            0.59                         0.40
...                           ...                          ...
1723                         0.18                         0.14
1724                         0.61                         0.81
1725                         0.82                         0.59
1726                         0.61                         0.44
1727                         0.59                         0.54

      post_spends t-test passed  post_spends ks-test passed  \
0                         False                       False
1                         False                       False
2                         False                       False
3                         False                       False
4                         False                       False
...                         ...                         ...
1723                      False                       False
1724                      False                       False
1725                      False                       False
1726                      False                       False
```

(continues on next page)

(continued from previous page)

```
1727                     False                         False

      pre_spends a mean  ...  pre_spends ks-test passed  control %  test %  \
0                487.00  ...                      False      50.01   49.99
1                487.19  ...                      False      50.01   49.99
2                487.11  ...                      False      50.01   49.99
3                487.20  ...                      False      50.01   49.99
4                487.20  ...                      False      50.01   49.99
...                 ...  ...                        ...        ...     ...
1723             487.40  ...                      False      50.01   49.99
1724             487.08  ...                      False      50.01   49.99
1725             487.04  ...                      False      50.01   49.99
1726             486.85  ...                      False      50.01   49.99
1727             487.21  ...                      False      50.01   49.99

      control size  test size  t-test mean p-value  ks-test mean p-value  \
0             5001       4999                 0.36                  0.52
1             5001       4999                 0.47                  0.92
2             5001       4999                 0.90                  0.93
3             5001       4999                 0.77                  0.47
4             5001       4999                 0.58                  0.50
...            ...        ...                  ...                   ...
1723          5001       4999                 0.14                  0.16
1724          5001       4999                 0.78                  0.69
1725          5001       4999                 0.79                  0.66
1726          5001       4999                 0.41                  0.54
1727          5001       4999                 0.57                  0.62

      t-test passed %  ks-test passed %  mean_tests_score
0                0.00              0.00              0.47
1                0.00              0.00              0.77
2                0.00              0.00              0.92
3                0.00              0.00              0.57
4                0.00              0.00              0.53
...               ...               ...               ...
1723             0.00              0.00              0.16
1724             0.00              0.00              0.72
1725             0.00              0.00              0.70
1726             0.00              0.00              0.50
1727             0.00              0.00              0.60

[1728 rows x 26 columns]
```

```
aa_score
```

```
             t-test passed score  ks-test passed score t-test aa passed  \
post_spends                 0.00                  0.00             0.00
pre_spends                  0.00                  0.00             0.00
mean                        0.00                  0.00             0.00

            ks-test aa passed
post_spends              0.00
```

(continues on next page)

(continued from previous page)

```
pre_spends               0.00
mean                     0.00
```

```
split
```

```
      user_id  signup_month  treat  pre_spends  post_spends   age gender  \
0           0             0      0      488.00       414.44   NaN      M
1           2             7      1      483.00       479.44 25.00      M
2           4             1      1      543.00       514.56 18.00      F
3           5             6      1      486.50       486.56 44.00      M
4           7            11      1      496.00       432.89 57.00      M
...       ...           ...    ...         ...          ...   ...    ...
9995     9983             0      0      494.50       428.33 31.00      F
9996     9984             0      0      460.00       417.11 56.00      M
9997     9985             0      0      484.00       411.33 52.00      M
9998     9991             0      0      482.50       421.89 43.00    NaN
9999     9994             0      0      486.00       423.78 69.00      F

        industry    group
0     E-commerce     test
1      Logistics     test
2     E-commerce     test
3     E-commerce     test
4     E-commerce     test
...          ...      ...
9995   Logistics  control
9996   Logistics  control
9997  E-commerce  control
9998   Logistics  control
9999   Logistics  control

[10000 rows x 9 columns]
```

```
best_experiment_stat
```

```
            a mean b mean ab delta ab delta % t-test p-value ks-test p-value  \
post_spends 452.18 452.15    -0.03      -0.01           0.97            0.99
pre_spends  487.08 487.11     0.03       0.01           0.93            1.00

            t-test passed ks-test passed
post_spends         False          False
pre_spends          False          False
```

```
split_stat
```

```
control %              50.01
test %                 49.99
control size            5001
test size               4999
t-test mean p-value     0.95
ks-test mean p-value    0.99
t-test passed %         0.00
```

(continues on next page)

(continued from previous page)

```
ks-test passed %        0.00
mean_tests_score        0.98
Name: 1395, dtype: object
```

```
resume
```

```
            aa test passed split is uniform
post_spends         not OK               OK
pre_spends          not OK               OK
```

2.4 AA with optimize group

If you have many columns for grouping and don't know which colun or columns will make best result, you can use parametr ``optimize_group=True``. AA-Test will choose optimal number and names of group columns.

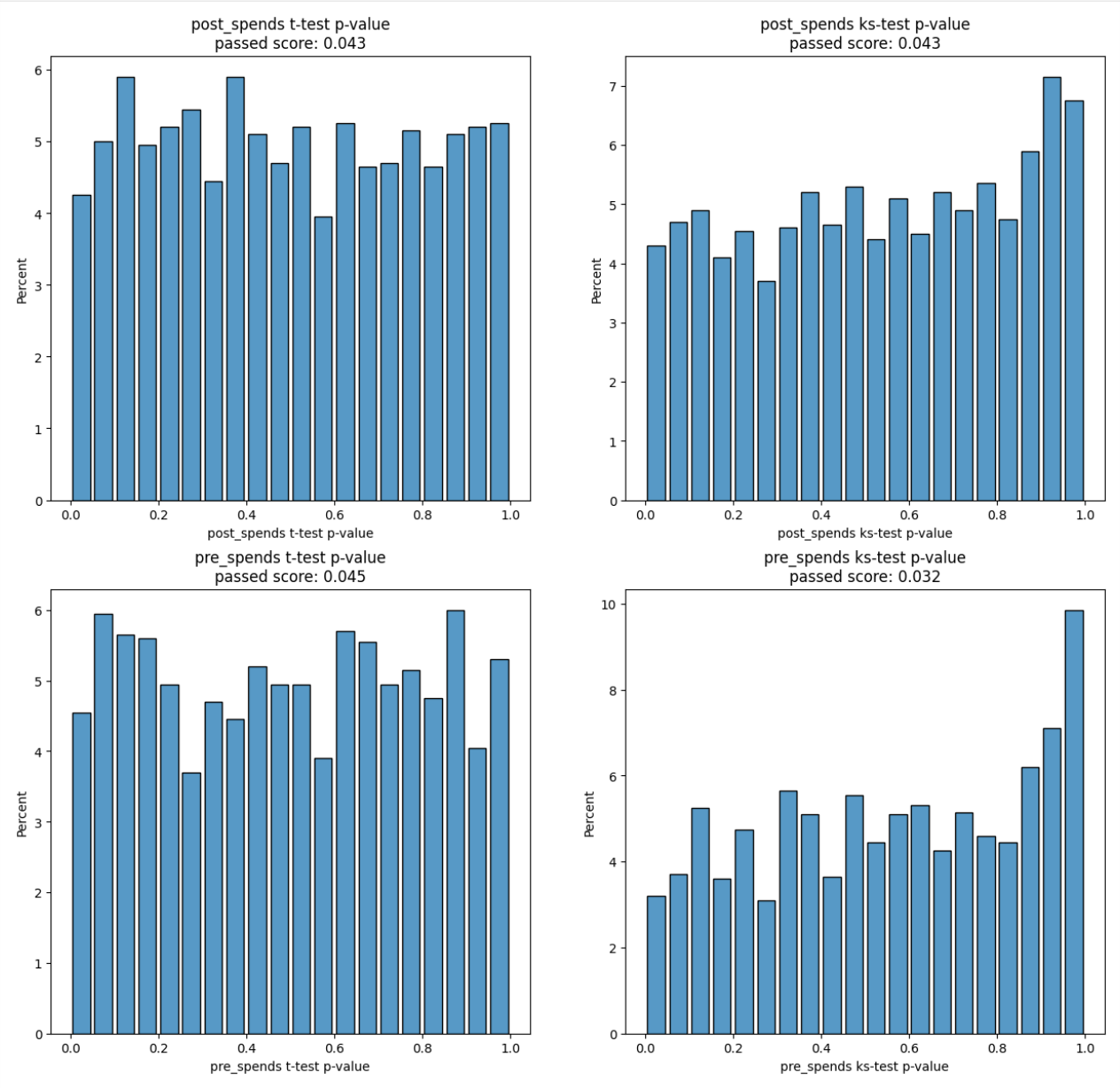
You can use `columns_labeling` to automatically name columns as target and group.

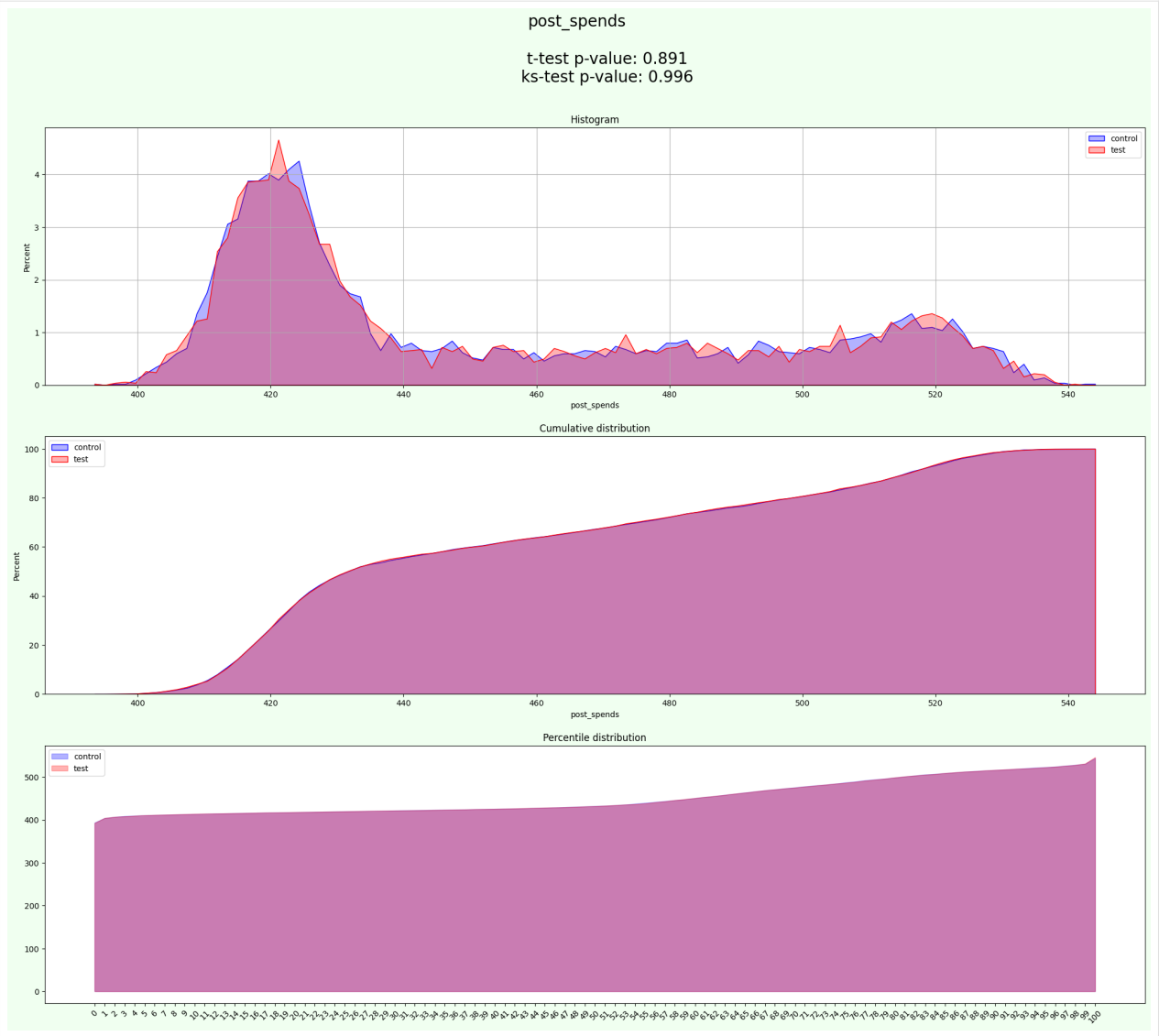
[25]:
```
experiment.columns_labeling(data)
```

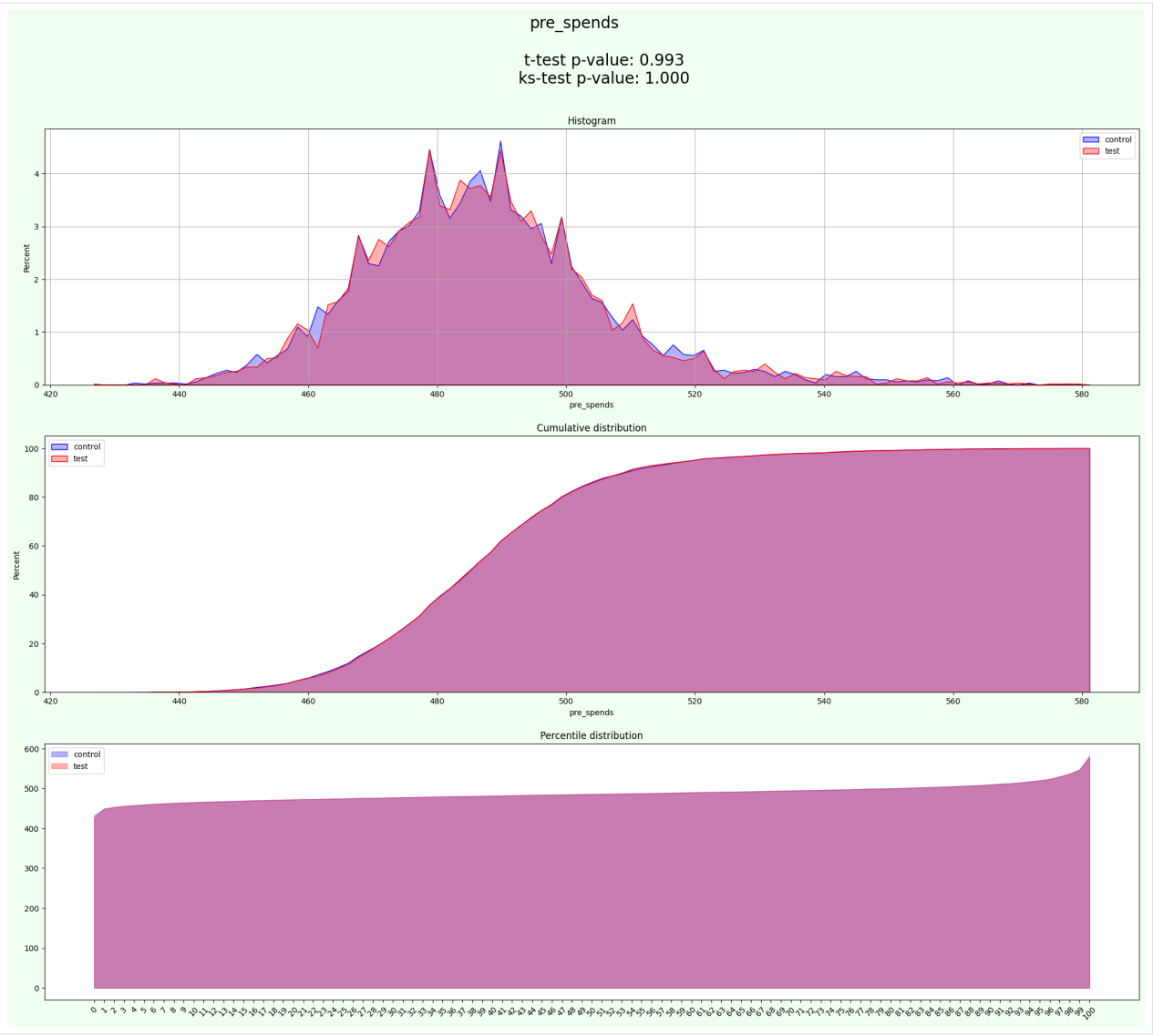
[25]:
```
{'target_field': ['treat', 'pre_spends', 'post_spends', 'age'],
 'group_col': ['gender', 'industry']}
```

[26]:
```
results = experiment.process(data=data, optimize_groups=True, iterations=2000)
```

```
Group optimization:   0%|          | 0/3 [00:00<?, ?it/s]
```


post_spends t-test p-value
passed score: 0.043
Percent
post_spends t-test p-value
post_spends ks-test p-value
passed score: 0.043
Percent
post_spends ks-test p-value
pre_spends t-test p-value
passed score: 0.045
Percent
pre_spends t-test p-value
pre_spends ks-test p-value
passed score: 0.032
Percent
pre_spends ks-test p-value


post_spends
t-test p-value: 0.891
ks-test p-value: 0.996
Histogram
control
test
Percent
post_spends
Cumulative distribution
control
test
Percent
post_spends
Percentile distribution
control
test


pre_spends
t-test p-value: 0.993
ks-test p-value: 1.000
Histogram
control
test
Percent
pre_spends
Cumulative distribution
Percentile distribution

[27]:
```
experiment.group_cols
```

[27]:
```
['industry']
```

[28]:
```
show_result(results)
```

```
experiments
```

```
      random_state  post_spends a mean  post_spends b mean  \
0                0              452.60              451.72
1                1              452.18              452.15
2                2              452.82              451.50
3                3              451.25              453.08
4                4              452.41              451.92
...            ...                 ...                 ...
1995          1995              452.30              452.03
1996          1996              451.89              452.44
1997          1997              452.52              451.81
1998          1998              452.27              452.06
1999          1999              451.47              452.86
```

(continues on next page)

(continued from previous page)

```
      post_spends ab delta  post_spends ab delta %  \
0                    -0.88                    -0.19
1                    -0.03                    -0.01
2                    -1.32                    -0.29
3                     1.83                     0.40
4                    -0.50                    -0.11
...                    ...                      ...
1995                 -0.26                    -0.06
1996                  0.55                     0.12
1997                 -0.70                    -0.16
1998                 -0.21                    -0.05
1999                  1.38                     0.31

      post_spends t-test p-value  post_spends ks-test p-value  \
0                           0.26                         0.48
1                           0.97                         0.56
2                           0.09                         0.18
3                           0.02                         0.08
4                           0.53                         0.06
...                          ...                          ...
1995                        0.74                         0.91
1996                        0.48                         0.78
1997                        0.37                         0.10
1998                        0.79                         0.86
1999                        0.08                         0.02

      post_spends t-test passed  post_spends ks-test passed  \
0                         False                        False
1                         False                        False
2                         False                        False
3                          True                        False
4                         False                        False
...                         ...                          ...
1995                      False                        False
1996                      False                        False
1997                      False                        False
1998                      False                        False
1999                      False                         True

      pre_spends a mean  ...  pre_spends ks-test passed  control %  test %  \
0                487.38  ...                       True      50.00   50.00
1                487.33  ...                      False      50.00   50.00
2                487.04  ...                      False      50.00   50.00
3                486.67  ...                      False      50.00   50.00
4                487.20  ...                      False      50.00   50.00
...                 ...  ...                        ...        ...     ...
1995             486.94  ...                      False      50.00   50.00
1996             487.30  ...                      False      50.00   50.00
1997             487.10  ...                      False      50.00   50.00
1998             486.73  ...                       True      50.00   50.00
1999             486.75  ...                      False      50.00   50.00
```

(continues on next page)

(continued from previous page)

```
      control size  test size  t-test mean p-value  ks-test mean p-value  \
0             5000       5000                 0.20                  0.25
1             5000       5000                 0.59                  0.49
2             5000       5000                 0.44                  0.45
3             5000       5000                 0.02                  0.13
4             5000       5000                 0.56                  0.36
...            ...        ...                  ...                   ...
1995          5000       5000                 0.57                  0.89
1996          5000       5000                 0.38                  0.86
1997          5000       5000                 0.66                  0.40
1998          5000       5000                 0.42                  0.45
1999          5000       5000                 0.07                  0.36

      t-test passed %  ks-test passed %  mean_tests_score
0                0.00             50.00              0.23
1                0.00              0.00              0.52
2                0.00              0.00              0.45
3              100.00              0.00              0.09
4                0.00              0.00              0.43
...               ...               ...               ...
1995             0.00              0.00              0.79
1996             0.00              0.00              0.70
1997             0.00              0.00              0.49
1998             0.00             50.00              0.44
1999             0.00             50.00              0.26

[2000 rows x 26 columns]
```

```
aa_score
```

```
             t-test passed score  ks-test passed score t-test aa passed  \
post_spends                 0.04                  0.04             1.00
pre_spends                  0.05                  0.03             1.00
mean                        0.04                  0.04             1.00

            ks-test aa passed
post_spends              1.00
pre_spends               0.00
mean                     0.50
```

```
split
```

```
      user_id  signup_month  treat  pre_spends  post_spends   age gender  \
0           0             0      0      488.00       414.44   NaN      M
1           1             8      1      512.50       462.22 26.00    NaN
2           4             1      1      543.00       514.56 18.00      F
3           5             6      1      486.50       486.56 44.00      M
4           8             4      1      465.50       506.00 66.00      M
...       ...           ...    ...         ...          ...   ...    ...
9995     9990             0      0      490.00       426.00   NaN      M
9996     9992             0      0      491.50       424.00 29.00      M
9997     9996             0      0      500.50       430.89 26.00      F
```

(continues on next page)

(continued from previous page)

```
9998     9997             3      1      473.00       534.11 22.00      F
9999     9998             2      1      495.00       523.22 67.00      F

        industry    group
0     E-commerce     test
1     E-commerce     test
2     E-commerce     test
3     E-commerce     test
4      Logistics     test
...          ...      ...
9995   Logistics  control
9996  E-commerce  control
9997   Logistics  control
9998  E-commerce  control
9999  E-commerce  control

[10000 rows x 9 columns]
```

```
best_experiment_stat
```

```
            a mean b mean ab delta ab delta % t-test p-value ks-test p-value  \
post_spends 452.22 452.11    -0.11      -0.02           0.89            1.00
pre_spends  487.09 487.10     0.00       0.00           0.99            1.00

            t-test passed ks-test passed
post_spends         False          False
pre_spends          False          False
```

```
split_stat
```

```
control %              50.00
test %                 50.00
control size            5000
test size               5000
t-test mean p-value     0.94
ks-test mean p-value    1.00
t-test passed %         0.00
ks-test passed %        0.00
mean_tests_score        0.98
Name: 69, dtype: object
```

```
resume
```

```
            aa test passed split is uniform
post_spends             OK               OK
pre_spends              OK               OK
```

2.5 AA test with quantization

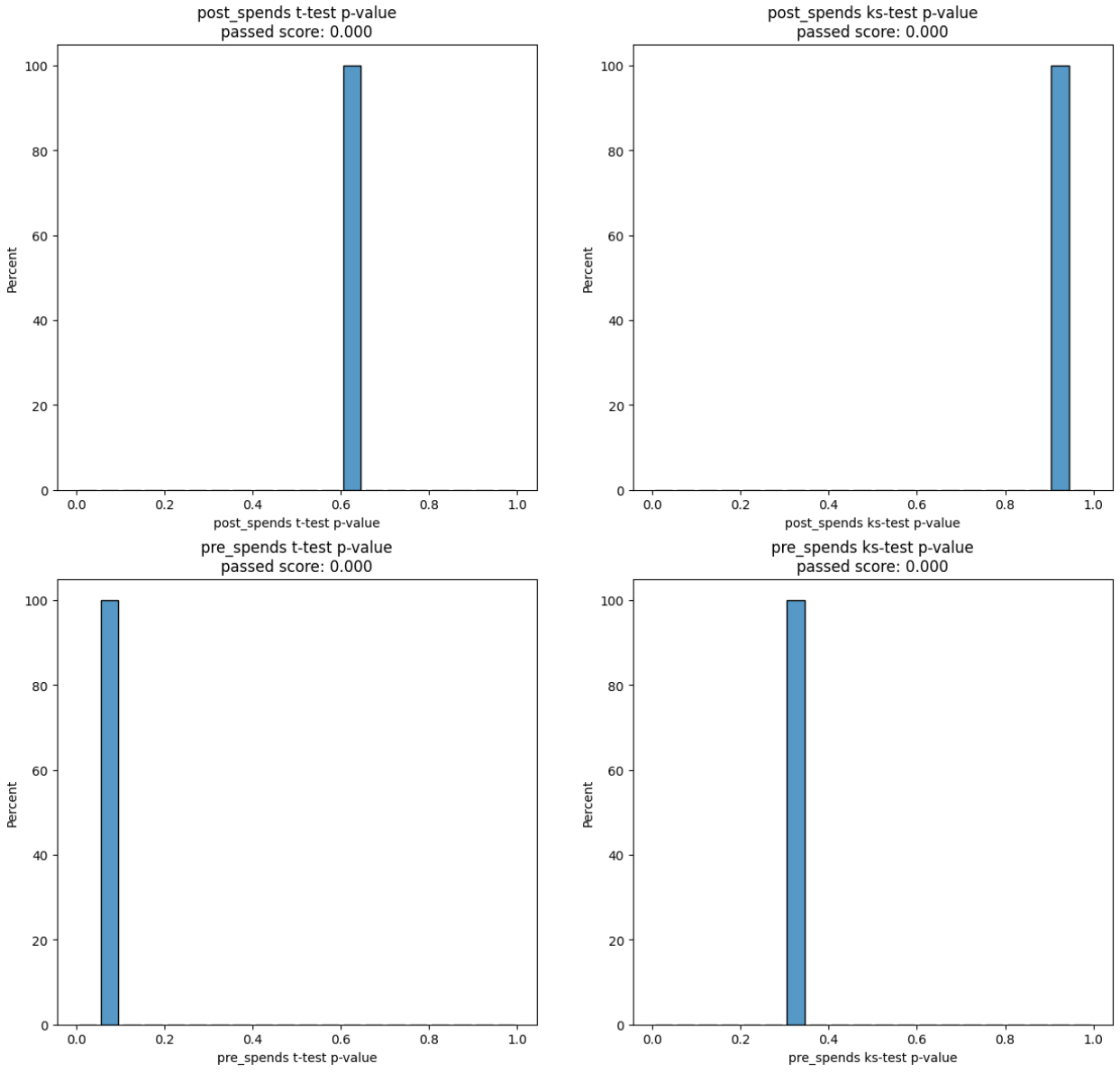
If you want make one column as parameter for quantization, you may use ``quant_field``.

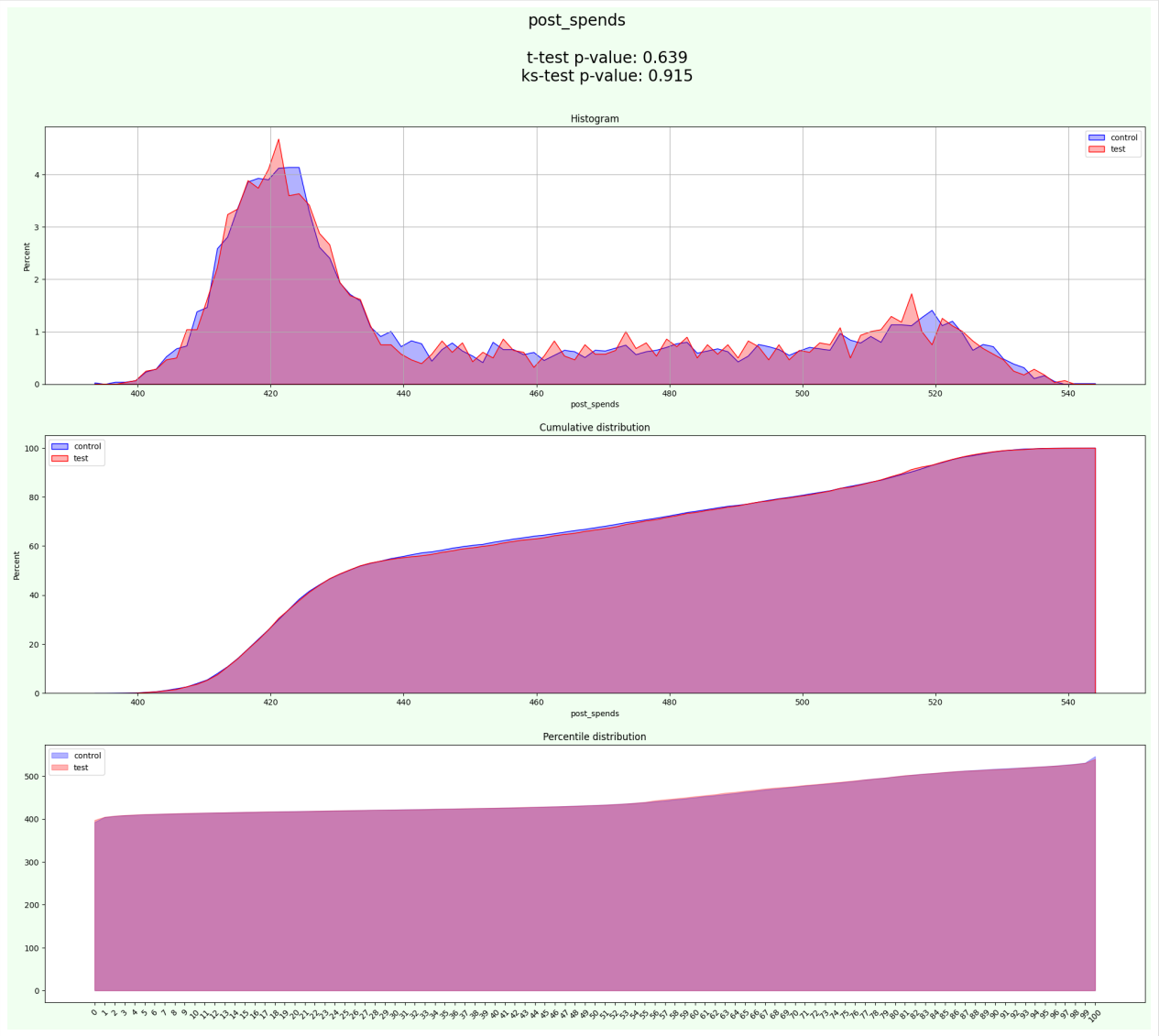
[29]:
```
info_cols = ['user_id', 'signup_month']
target = ['post_spends', 'pre_spends']

group_cols = 'industry'
quant_field = 'gender'
```

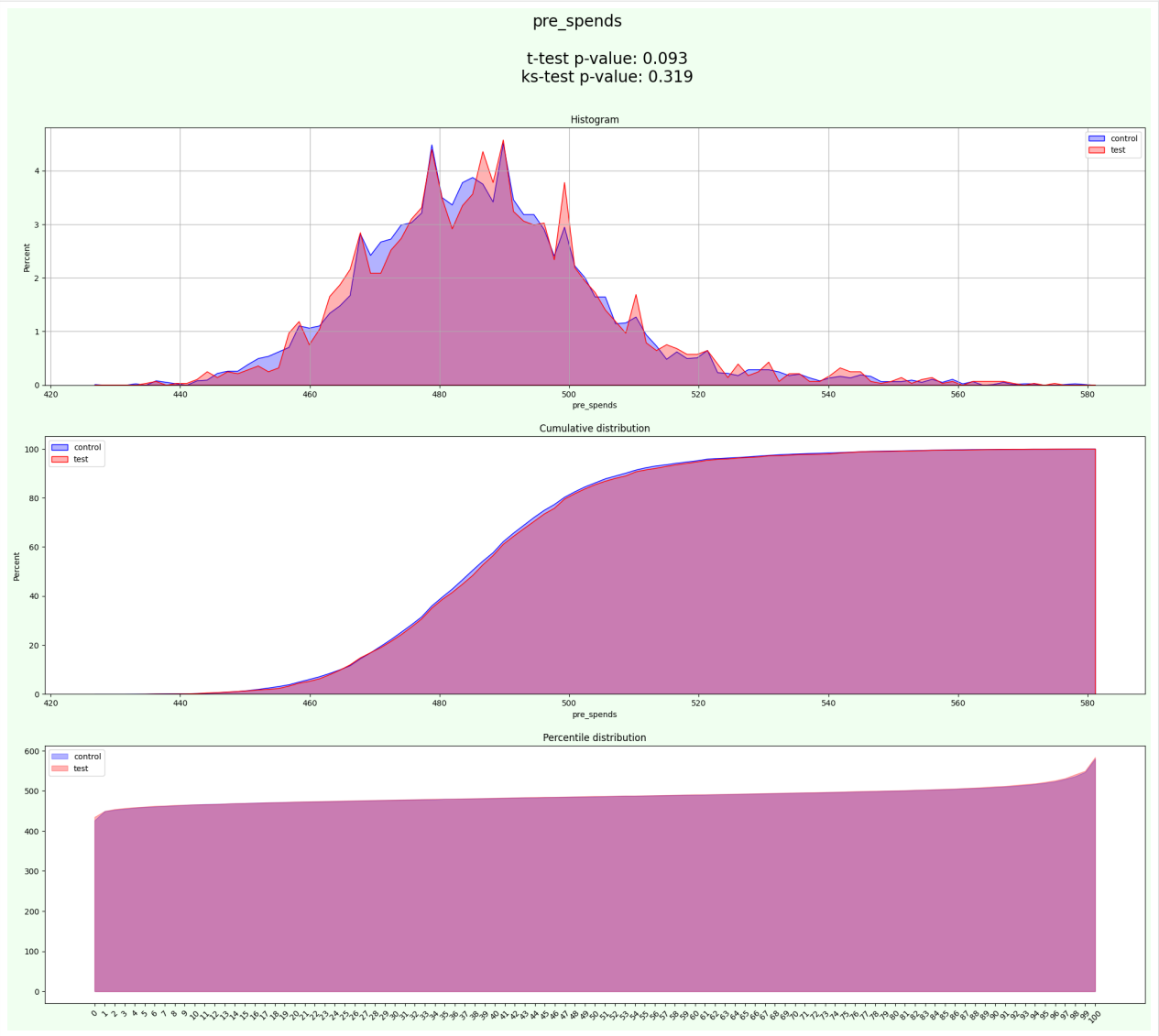
[30]:
```
experiment = AATest(info_cols=info_cols, target_fields=target, group_cols=group_cols,␣
↪quant_field=quant_field)
```

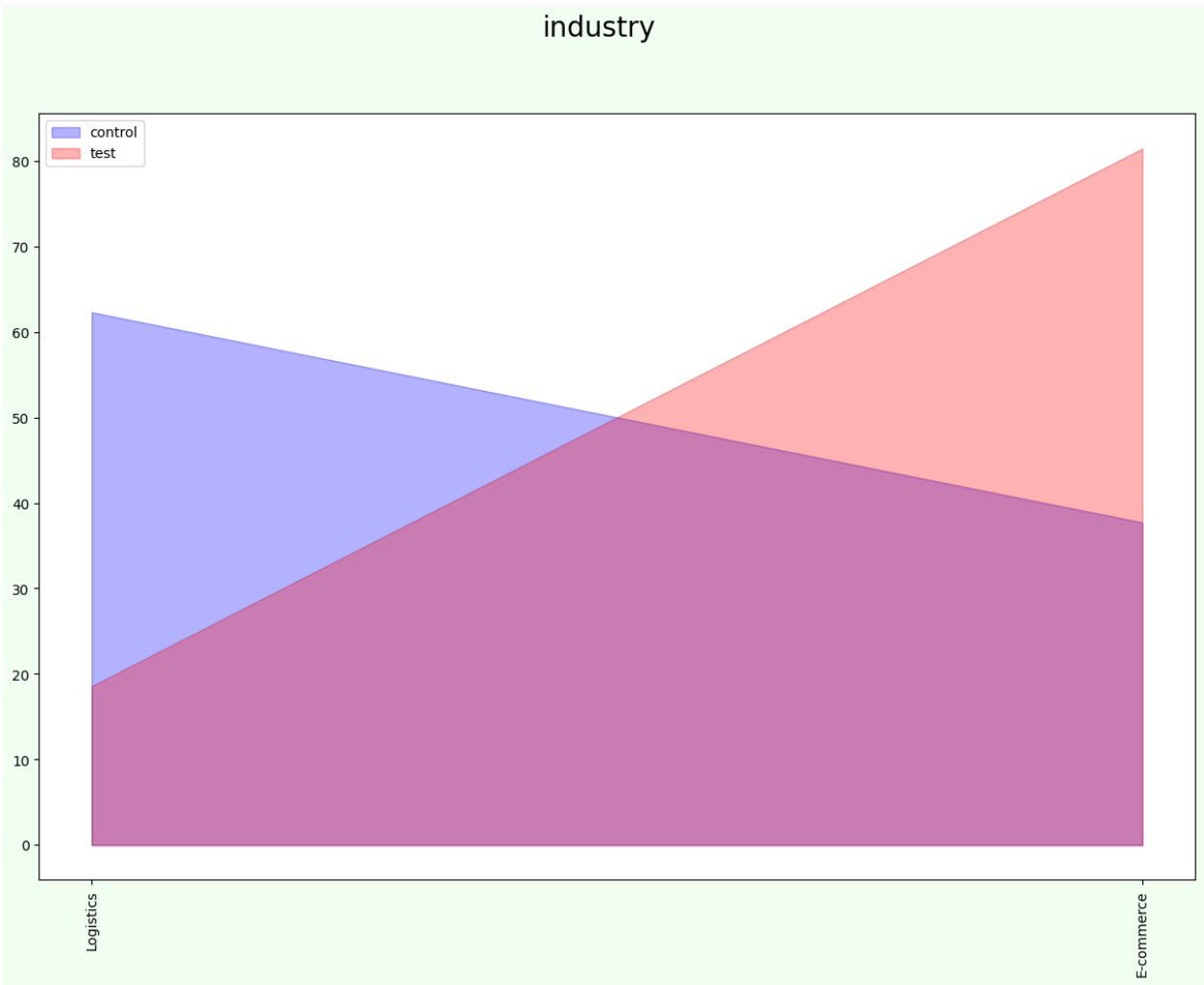
[31]:
```
result = experiment.process(data=data, iterations=2000)
```

```
  0%|          | 0/2000 [00:00<?, ?it/s]
```




post_spends
t-test p-value: 0.639
ks-test p-value: 0.915
Histogram
control
test
Percent
post_spends
Cumulative distribution
Percentile distribution


pre_spends
t-test p-value: 0.093
ks-test p-value: 0.319
Histogram
control
test
Percent
pre_spends
Cumulative distribution
control
test
Percent
pre_spends
Percentile distribution
control
test



[32]:
```
result['split'].groupby(['gender', 'industry', 'group'])['user_id'].count()
```

[32]:
```
gender  industry    group
F       E-commerce  test       2261
        Logistics   control    2305
M       E-commerce  control    2240
        Logistics   control    2194
Name: user_id, dtype: int64
```

[33]:
```
show_result(result)
```

```
experiments
```

```
     random_state  post_spends a mean  post_spends b mean  \
0               0              452.05              452.46
1               2              452.05              452.46
2               7              452.05              452.46
3               8              452.05              452.46
4              15              452.05              452.46
..            ...                 ...                 ...
```

(continues on next page)

(continued from previous page)

```
643          1981              452.05              452.46
644          1982              452.05              452.46
645          1984              452.05              452.46
646          1988              452.05              452.46
647          1998              452.05              452.46

     post_spends ab delta  post_spends ab delta %  post_spends t-test p-value  \
0                    0.41                    0.09                        0.64
1                    0.41                    0.09                        0.64
2                    0.41                    0.09                        0.64
3                    0.41                    0.09                        0.64
4                    0.41                    0.09                        0.64
..                    ...                     ...                         ...
643                  0.41                    0.09                        0.64
644                  0.41                    0.09                        0.64
645                  0.41                    0.09                        0.64
646                  0.41                    0.09                        0.64
647                  0.41                    0.09                        0.64

     post_spends ks-test p-value  post_spends t-test passed  \
0                           0.91                      False
1                           0.91                      False
2                           0.91                      False
3                           0.91                      False
4                           0.91                      False
..                           ...                       ...
643                         0.91                      False
644                         0.91                      False
645                         0.91                      False
646                         0.91                      False
647                         0.91                      False

     post_spends ks-test passed  pre_spends a mean  ...  \
0                         False             486.90  ...
1                         False             486.90  ...
2                         False             486.90  ...
3                         False             486.90  ...
4                         False             486.90  ...
..                          ...                ...  ...
643                       False             486.90  ...
644                       False             486.90  ...
645                       False             486.90  ...
646                       False             486.90  ...
647                       False             486.90  ...

     pre_spends ks-test passed  control %  test %  control size  test size  \
0                        False      72.23   27.77          7223       2777
1                        False      72.23   27.77          7223       2777
2                        False      72.23   27.77          7223       2777
3                        False      72.23   27.77          7223       2777
4                        False      72.23   27.77          7223       2777
..                         ...        ...     ...           ...        ...
```

(continues on next page)

(continued from previous page)

```
643                      False      72.23   27.77          7223       2777
644                      False      72.23   27.77          7223       2777
645                      False      72.23   27.77          7223       2777
646                      False      72.23   27.77          7223       2777
647                      False      72.23   27.77          7223       2777

     t-test mean p-value  ks-test mean p-value  t-test passed %  \
0                   0.37                  0.62             0.00
1                   0.37                  0.62             0.00
2                   0.37                  0.62             0.00
3                   0.37                  0.62             0.00
4                   0.37                  0.62             0.00
..                   ...                   ...              ...
643                 0.37                  0.62             0.00
644                 0.37                  0.62             0.00
645                 0.37                  0.62             0.00
646                 0.37                  0.62             0.00
647                 0.37                  0.62             0.00

     ks-test passed %  mean_tests_score
0                0.00              0.53
1                0.00              0.53
2                0.00              0.53
3                0.00              0.53
4                0.00              0.53
..                ...               ...
643              0.00              0.53
644              0.00              0.53
645              0.00              0.53
646              0.00              0.53
647              0.00              0.53

[648 rows x 26 columns]
```

```
aa_score
```

```
             t-test passed score  ks-test passed score t-test aa passed  \
post_spends                 0.00                  0.00             0.00
pre_spends                  0.00                  0.00             0.00
mean                        0.00                  0.00             0.00

            ks-test aa passed
post_spends              0.00
pre_spends               0.00
mean                     0.00
```

```
split
```

```
      user_id  signup_month  treat  pre_spends  post_spends   age gender  \
0        8193             0      0      494.50       427.11 40.00      F
1        8195             0      0      494.00       416.22 41.00      F
2           4             1      1      543.00       514.56 18.00      F
```

(continues on next page)

(continued from previous page)

```
3        8203             0      0      472.50       412.67 52.00      F
4        8205             0      0      460.00       408.22 66.00      F
...       ...           ...    ...         ...          ...   ...    ...
9995     9990             0      0      490.00       426.00   NaN      M
9996     9992             0      0      491.50       424.00 29.00      M
9997     9994             0      0      486.00       423.78 69.00      F
9998     9995            10      1      538.50       450.44 42.00      M
9999     9996             0      0      500.50       430.89 26.00      F

        industry    group
0     E-commerce     test
1     E-commerce     test
2     E-commerce     test
3     E-commerce     test
4     E-commerce     test
...          ...      ...
9995   Logistics  control
9996  E-commerce  control
9997   Logistics  control
9998   Logistics  control
9999   Logistics  control

[10000 rows x 9 columns]
```

```
best_experiment_stat
```

```
            a mean b mean ab delta ab delta % t-test p-value ks-test p-value  \
post_spends 452.05 452.46     0.41       0.09           0.64            0.91
pre_spends  486.90 487.60     0.71       0.15           0.09            0.32

            t-test passed ks-test passed
post_spends         False          False
pre_spends          False          False
```

```
split_stat
```

```
control %              72.23
test %                 27.77
control size            7223
test size               2777
t-test mean p-value     0.37
ks-test mean p-value    0.62
t-test passed %         0.00
ks-test passed %        0.00
mean_tests_score        0.53
Name: 0, dtype: object
```

```
resume
```

```
            aa test passed split is uniform
post_spends         not OK               OK
pre_spends          not OK               OK
```

2.6 Unbalanced AA test

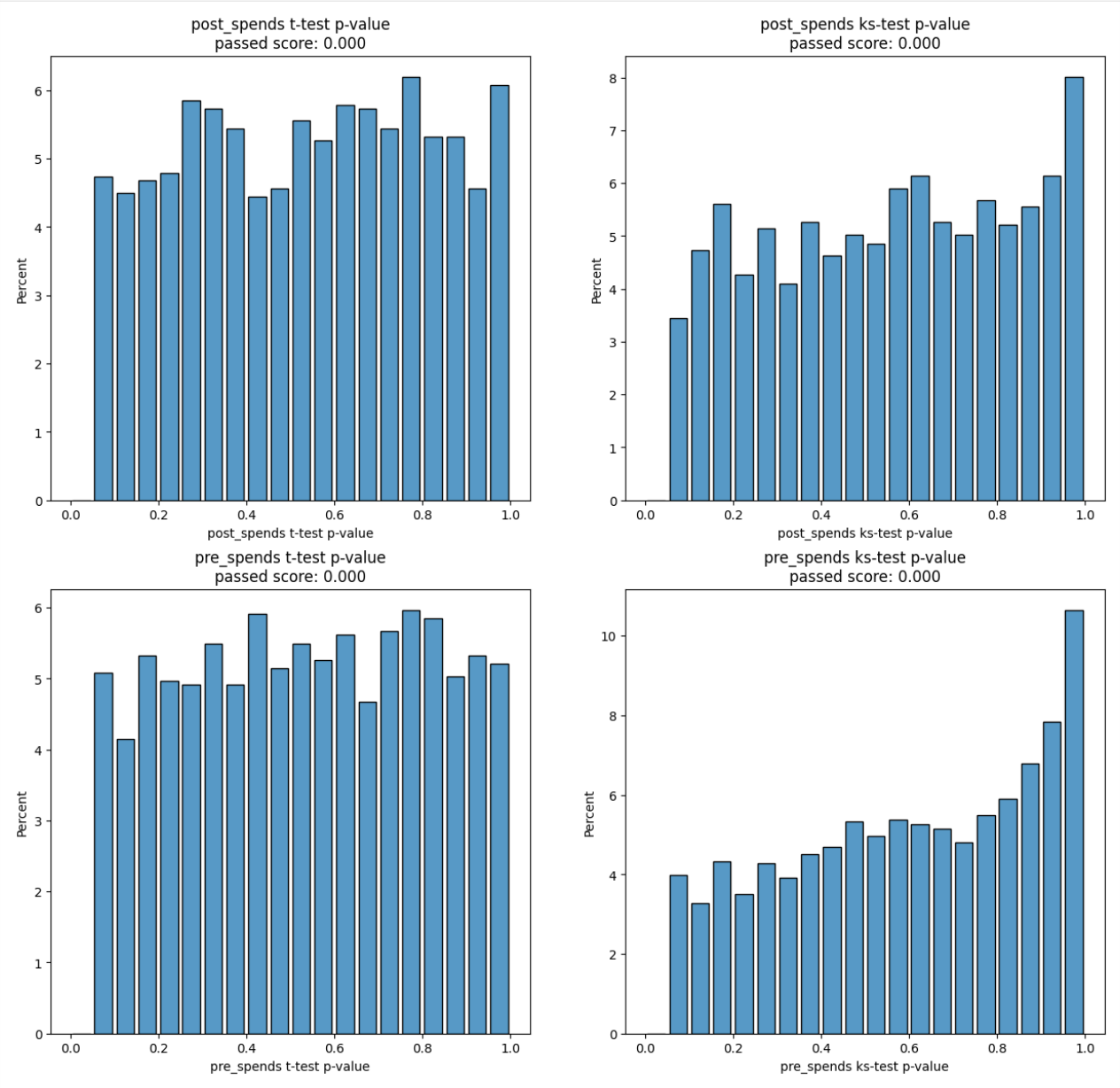
If you want to perform AA test with unbalanced groups, you can use parametr ``test_size`` to define sizes of test group and control group

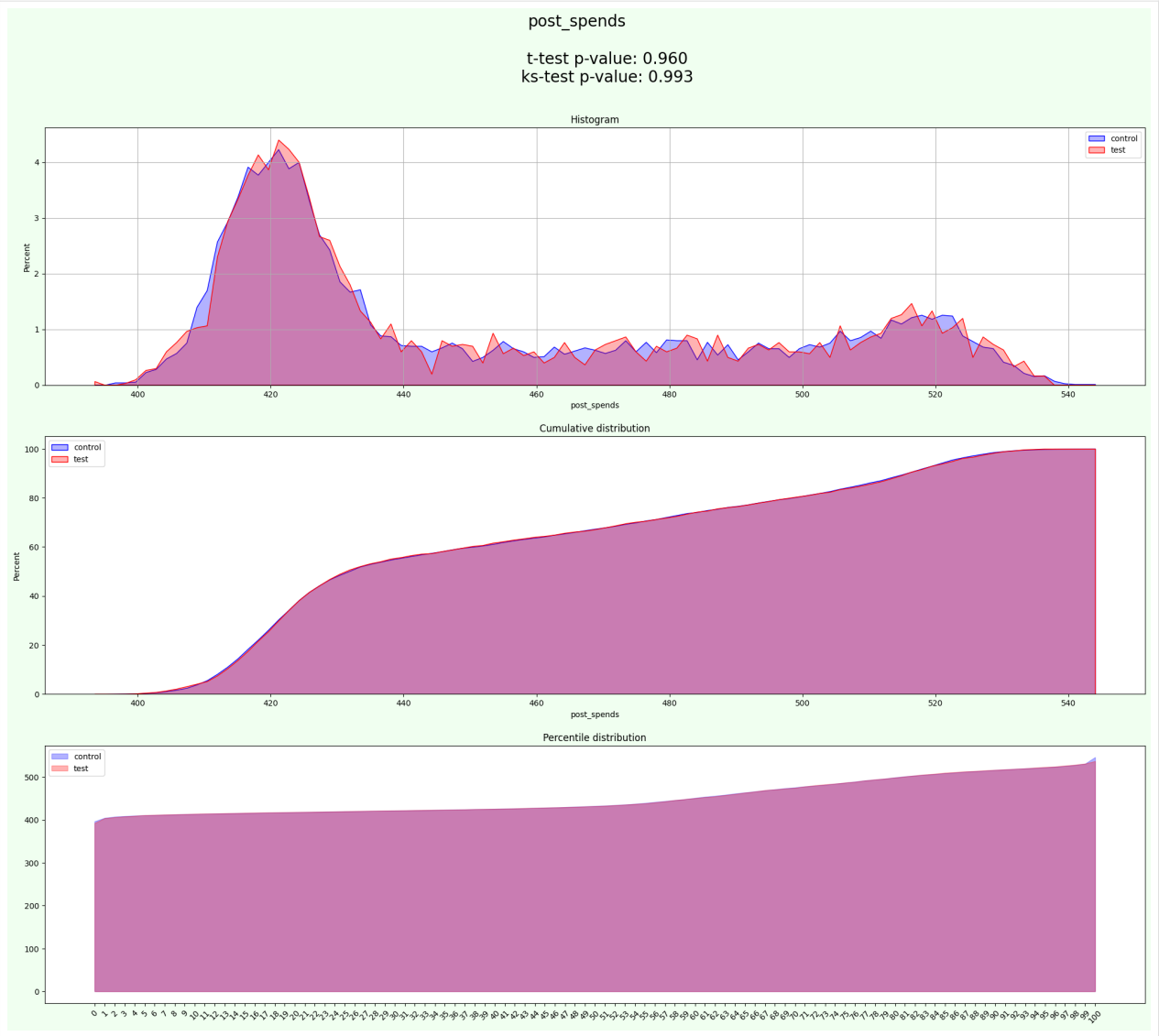
[34]:
```
info_cols = ['user_id', 'signup_month']
target = ['post_spends', 'pre_spends']

group_cols = 'industry'
```

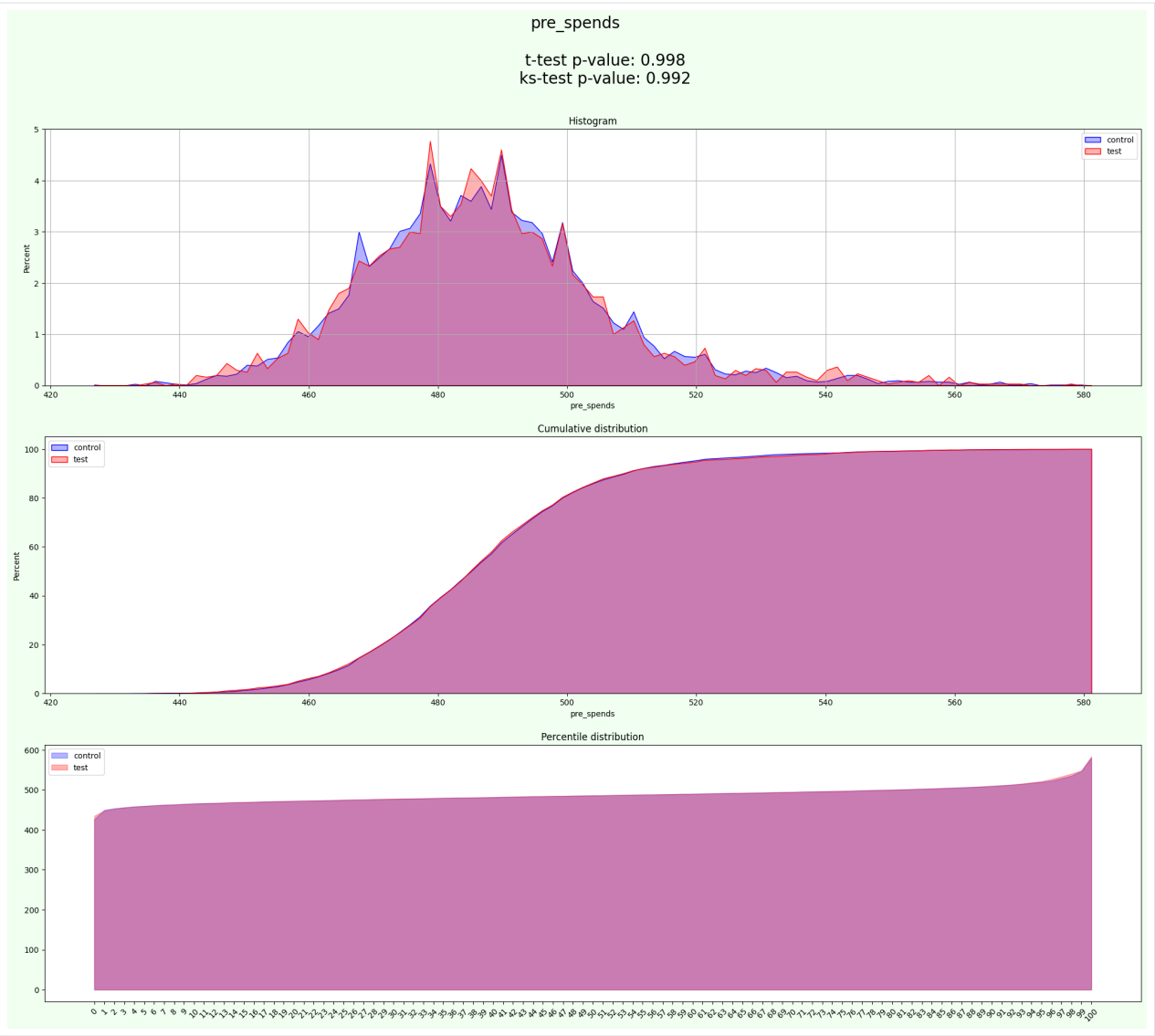
[35]:
```
experiment = AATest(info_cols=info_cols, target_fields=target, group_cols=group_cols)
```

[36]:
```
result = experiment.process(data=data, test_size=0.3, iterations=2000)
```

```
  0%|          | 0/2000 [00:00<?, ?it/s]
```


post_spends t-test p-value
passed score: 0.000
Percent
post_spends t-test p-value
post_spends ks-test p-value
passed score: 0.000
Percent
post_spends ks-test p-value
pre_spends t-test p-value
passed score: 0.000
Percent
pre_spends t-test p-value
pre_spends ks-test p-value
passed score: 0.000
Percent
pre_spends ks-test p-value


post_spends
t-test p-value: 0.960
ks-test p-value: 0.993
Histogram
control
test
Percent
post_spends
Cumulative distribution
control
test
Percent
post_spends
Percentile distribution
control
test


pre_spends
t-test p-value: 0.998
ks-test p-value: 0.992
Histogram
control
test
Percent
pre_spends
Cumulative distribution
control
test
Percent
pre_spends
Percentile distribution
control
test

[37]:
```
result['split']['group'].value_counts(normalize=True)
```

[37]:
```
group
control   0.70
test      0.30
Name: proportion, dtype: float64
```

[38]:
```
show_result(result)
```

```
experiments
```

```
      random_state  post_spends a mean  post_spends b mean  \
0                1              451.99              452.57
1                2              452.16              452.17
2                3              452.22              452.03
3                4              451.90              452.78
4                5              452.64              451.05
...            ...                 ...                 ...
1705          1995              452.12              452.28
1706          1996              452.00              452.54
1707          1997              452.29              451.88
```

(continues on next page)

(continued from previous page)

```
1708          1998              451.78              453.07
1709          1999              452.25              451.96

      post_spends ab delta  post_spends ab delta %  \
0                     0.57                    0.13
1                     0.01                    0.00
2                    -0.20                   -0.04
3                     0.88                    0.19
4                    -1.59                   -0.35
...                    ...                     ...
1705                  0.16                    0.04
1706                  0.54                    0.12
1707                 -0.40                   -0.09
1708                  1.29                    0.28
1709                 -0.29                   -0.06

      post_spends t-test p-value  post_spends ks-test p-value  \
0                            0.50                         0.76
1                            0.99                         0.64
2                            0.82                         0.59
3                            0.31                         0.24
4                            0.06                         0.14
...                           ...                          ...
1705                         0.85                         0.63
1706                         0.53                         0.85
1707                         0.64                         0.60
1708                         0.13                         0.33
1709                         0.74                         0.77

      post_spends t-test passed  post_spends ks-test passed  \
0                         False                       False
1                         False                       False
2                         False                       False
3                         False                       False
4                         False                       False
...                         ...                         ...
1705                      False                       False
1706                      False                       False
1707                      False                       False
1708                      False                       False
1709                      False                       False

      pre_spends a mean  ...  pre_spends ks-test passed  control %  test %  \
0                487.11  ...                      False      70.01   29.99
1                487.00  ...                      False      70.01   29.99
2                487.18  ...                      False      70.01   29.99
3                487.13  ...                      False      70.01   29.99
4                487.20  ...                      False      70.01   29.99
...                 ...  ...                        ...        ...     ...
1705             487.10  ...                      False      70.01   29.99
1706             486.94  ...                      False      70.01   29.99
1707             487.04  ...                      False      70.01   29.99
```

(continues on next page)

(continued from previous page)

```
1708             487.12  ...                      False      70.01   29.99
1709             487.17  ...                      False      70.01   29.99

      control size  test size  t-test mean p-value  ks-test mean p-value  \
0             7001       2999                 0.69                  0.66
1             7001       2999                 0.73                  0.72
2             7001       2999                 0.66                  0.48
3             7001       2999                 0.54                  0.60
4             7001       2999                 0.23                  0.14
...            ...        ...                  ...                   ...
1705          7001       2999                 0.89                  0.64
1706          7001       2999                 0.37                  0.72
1707          7001       2999                 0.65                  0.44
1708          7001       2999                 0.47                  0.66
1709          7001       2999                 0.64                  0.71

      t-test passed %  ks-test passed %  mean_tests_score
0                0.00              0.00              0.67
1                0.00              0.00              0.72
2                0.00              0.00              0.54
3                0.00              0.00              0.58
4                0.00              0.00              0.17
...               ...               ...               ...
1705             0.00              0.00              0.73
1706             0.00              0.00              0.60
1707             0.00              0.00              0.51
1708             0.00              0.00              0.60
1709             0.00              0.00              0.69

[1710 rows x 26 columns]
```

```
aa_score
```

```
             t-test passed score  ks-test passed score t-test aa passed  \
post_spends                 0.00                  0.00             0.00
pre_spends                  0.00                  0.00             0.00
mean                        0.00                  0.00             0.00

            ks-test aa passed
post_spends              0.00
pre_spends               0.00
mean                     0.00
```

```
split
```

```
      user_id  signup_month  treat  pre_spends  post_spends   age gender  \
0           0             0      0      488.00       414.44   NaN      M
1        8193             0      0      494.50       427.11 40.00      F
2        8192             0      0      487.50       436.78 20.00      M
3           2             7      1      483.00       479.44 25.00      M
4        8200             5      1      486.00       495.00   NaN      M
...       ...           ...    ...         ...          ...   ...    ...
```

(continues on next page)

(continued from previous page)

```
9995     9994             0      0      486.00       423.78 69.00      F
9996     9995            10      1      538.50       450.44 42.00      M
9997     9997             3      1      473.00       534.11 22.00      F
9998     9998             2      1      495.00       523.22 67.00      F
9999     9999             7      1      508.00       475.89 38.00      F

        industry    group
0     E-commerce     test
1     E-commerce     test
2     E-commerce     test
3      Logistics     test
4      Logistics     test
...          ...      ...
9995   Logistics  control
9996   Logistics  control
9997  E-commerce  control
9998  E-commerce  control
9999  E-commerce  control

[10000 rows x 9 columns]
```

```
best_experiment_stat
```

```
            a mean b mean ab delta ab delta % t-test p-value ks-test p-value  \
post_spends 452.15 452.19     0.04       0.01           0.96            0.99
pre_spends  487.09 487.09    -0.00      -0.00           1.00            0.99

            t-test passed ks-test passed
post_spends         False          False
pre_spends          False          False
```

```
split_stat
```

```
control %              70.01
test %                 29.99
control size            7001
test size               2999
t-test mean p-value     0.98
ks-test mean p-value    0.99
t-test passed %         0.00
ks-test passed %        0.00
mean_tests_score        0.99
Name: 1472, dtype: object
```

```
resume
```

```
            aa test passed split is uniform
post_spends         not OK               OK
pre_spends          not OK               OK
```

MDE

this is the boundary value of the effect, for which it makes sense to introduce some changes.

[39]:
```
info_cols = ['user_id', 'signup_month']
target = ['post_spends', 'pre_spends']

group_cols = 'industry'
mde_target = 'post_spends'
```

[40]:
```
experiment = AATest(info_cols=info_cols, target_fields=target, group_cols=group_cols)
```

Single experiment of data splitting for MDE calculation.

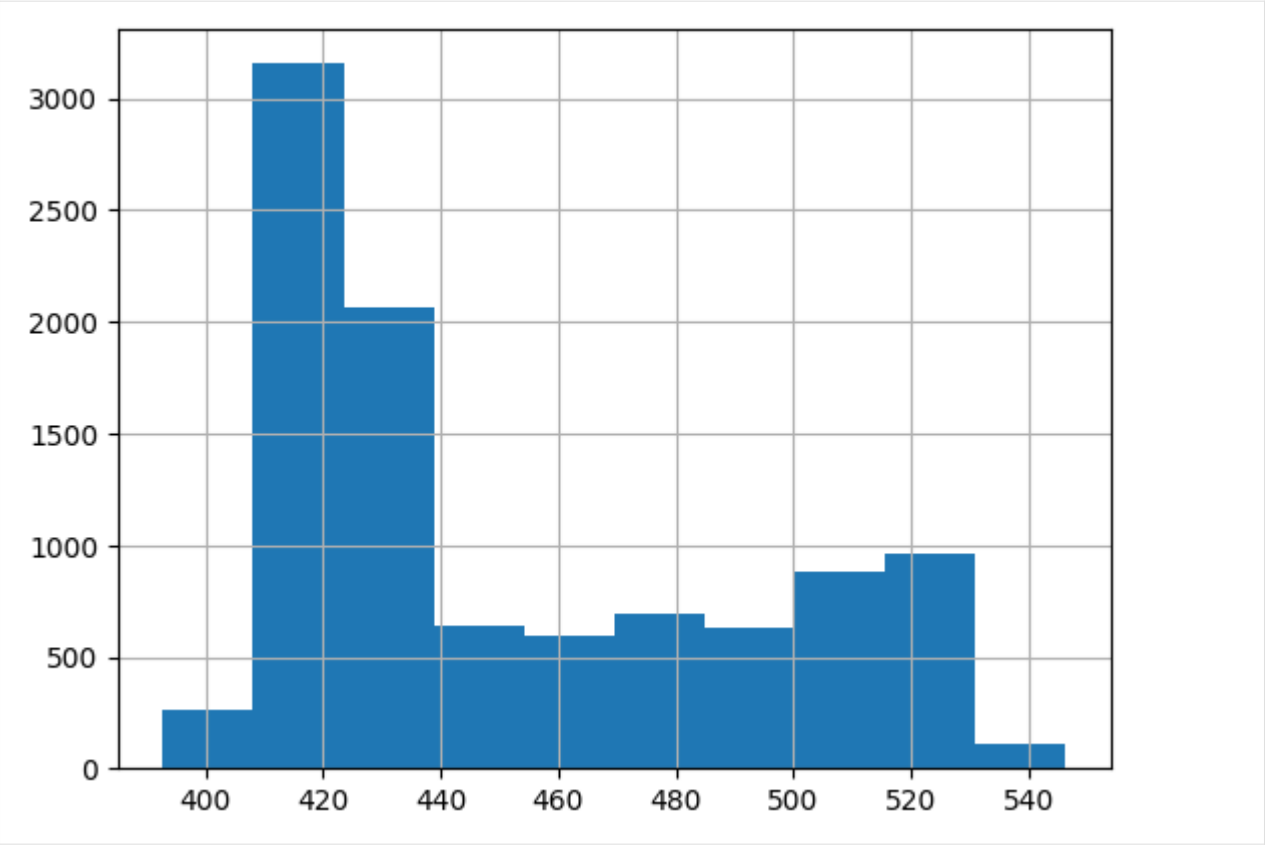
P.s. [None] is the number of random state. You can change it like sampling_metrics(data, random_state=42) and get result with [42] instead of [None]

[41]:
```
splitted_data = experiment.sampling_metrics(data)['data_from_experiment'][None]
splitted_data
```

[41]:
```
      user_id  signup_month  treat  pre_spends  post_spends   age gender  \
0           0             0      0      488.00       414.44   NaN      M
1           2             7      1      483.00       479.44 25.00      M
2           5             6      1      486.50       486.56 44.00      M
3           7            11      1      496.00       432.89 57.00      M
4           9             4      1      470.00       512.11 54.00      M
...       ...           ...    ...         ...          ...   ...    ...
9995     9994             0      0      486.00       423.78 69.00      F
9996     9995            10      1      538.50       450.44 42.00      M
9997     9996             0      0      500.50       430.89 26.00      F
9998     9997             3      1      473.00       534.11 22.00      F
9999     9999             7      1      508.00       475.89 38.00      F

        industry    group
0     E-commerce     test
1      Logistics     test
2     E-commerce     test
3     E-commerce     test
4      Logistics     test
...          ...      ...
9995   Logistics  control
9996   Logistics  control
9997   Logistics  control
9998  E-commerce  control
9999  E-commerce  control

[10000 rows x 9 columns]
```

[42]:
```
splitted_data[mde_target].hist()
```

[42]:
```
<Axes: >
```



You can evaluate minimum detectable effect for your data. This will be the smallest true effect obtained from the changes, which the statistical criterion will be able to detect with confidence

[43]:
```
mde = experiment.calc_mde(data=splitted_data, group_field="group", target_field=mde_
↪target)
mde
```

[43]:
```
(0.88, 0.02)
```

You can also calculate the amount of data you need to have in order to determine the minimum effect of the test.

[44]:
```
experiment.calc_sample_size(data=splitted_data, target_field=mde_target, mde=5)
```

[44]:
```
1949.4372012485414
```

Chi2 Test

[45]:
```
target = ['post_spends', 'pre_spends']
treated_field = 'treat'
```

[46]:
```
experiment = AATest(target_fields=target)
```

[47]:
```
experiment.calc_chi2(data, treated_field)
```

[47]:
```
{'post_spends': 4.2708618195357307e-129, 'pre_spends': 0.39046261817671 34}
```

2.2.2 AA test multigroup split simple

Provides simple approach to split into 2 and more groups.
Uses memory more efficiently than old method “process”

[1]:
```
import numpy as np
import pandas as pd

from lightautoml.addons.hypex import AATest
from lightautoml.addons.hypex.utils.tutorial_data_creation import create_test_data

pd.options.display.float_format = '{:,.2f}'.format

np.random.seed(42)  # needed to create example data
```

Prepare some large data

[2]:
```
some_large_dataframe = create_test_data(rs=52, na_step=10, nan_cols=['age', 'gender'],␣
↪num_users=100_000)
some_large_dataframe
```

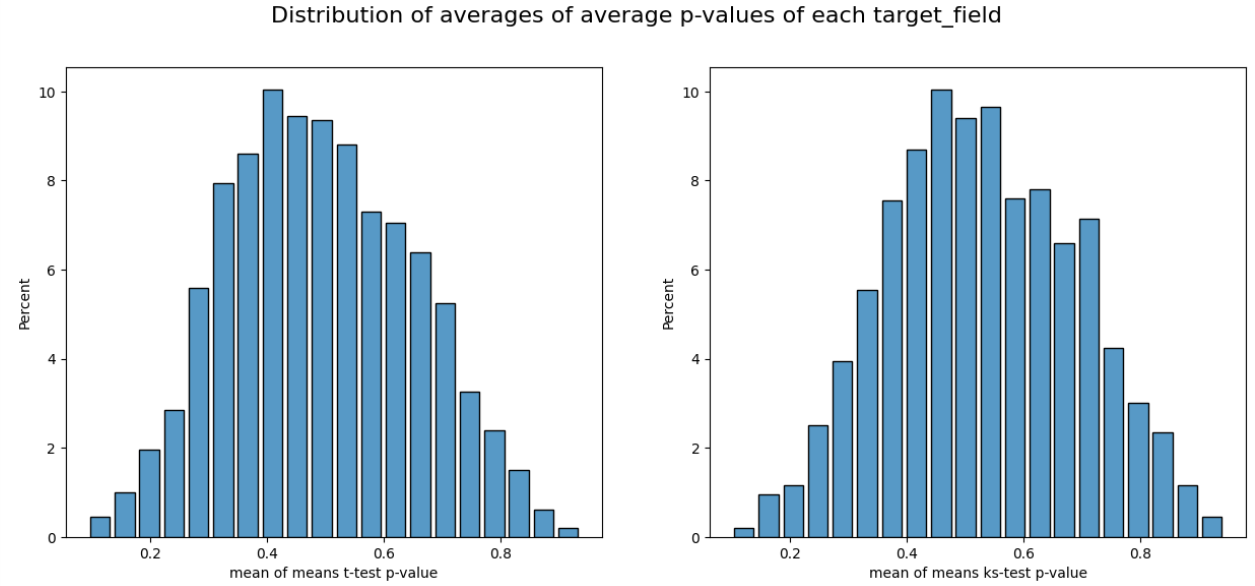
[2]:
```
       user_id  signup_month  treat  pre_spends  post_spends   age gender  \
0            0             0      0      478.00       422.89   NaN      M
1            1             0      0      472.50       407.22 68.00    NaN
2            2             0      0      485.00       426.11 44.00      F
3            3             8      1      485.00       466.11 59.00      F
4            4             1      1      539.00       522.78 54.00      M
...        ...           ...    ...         ...          ...   ...    ...
99995    99995             0      0      473.50       428.56 22.00      F
99996    99996             0      0      497.00       421.89 65.00      M
99997    99997             3      1      485.00       517.56 20.00      M
99998    99998             0      0      458.50       410.78 69.00      M
99999    99999             0      0      494.00       416.89 44.00      F

         industry
0       Logistics
1      E-commerce
2       Logistics
3      E-commerce
4      E-commerce
...           ...
99995   Logistics
99996  E-commerce
99997   Logistics
99998  E-commerce
99999  E-commerce

[100000 rows x 8 columns]
```

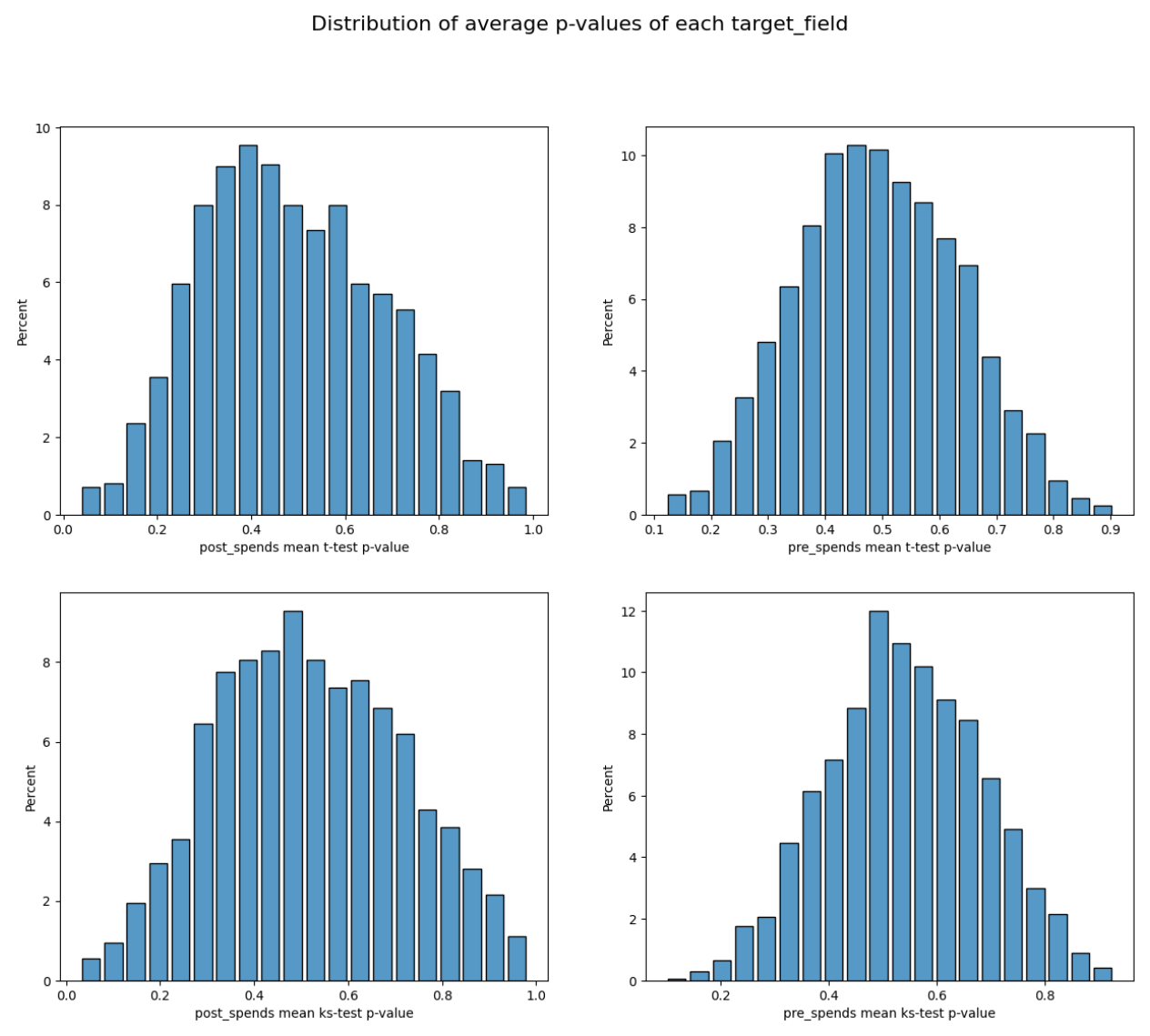
Process split

[3]:
```
target = ['post_spends', 'pre_spends']
# Dict
#  key - group name
#  value - share
#  The sum of the shares is 1
groups = {'test1': 0.3, 'test2': 0.2, 'control': 0.5 }

aa_test = AATest()
results = aa_test.process_split(df=some_large_dataframe, target_fields=target,␣
↪groups=groups)
```

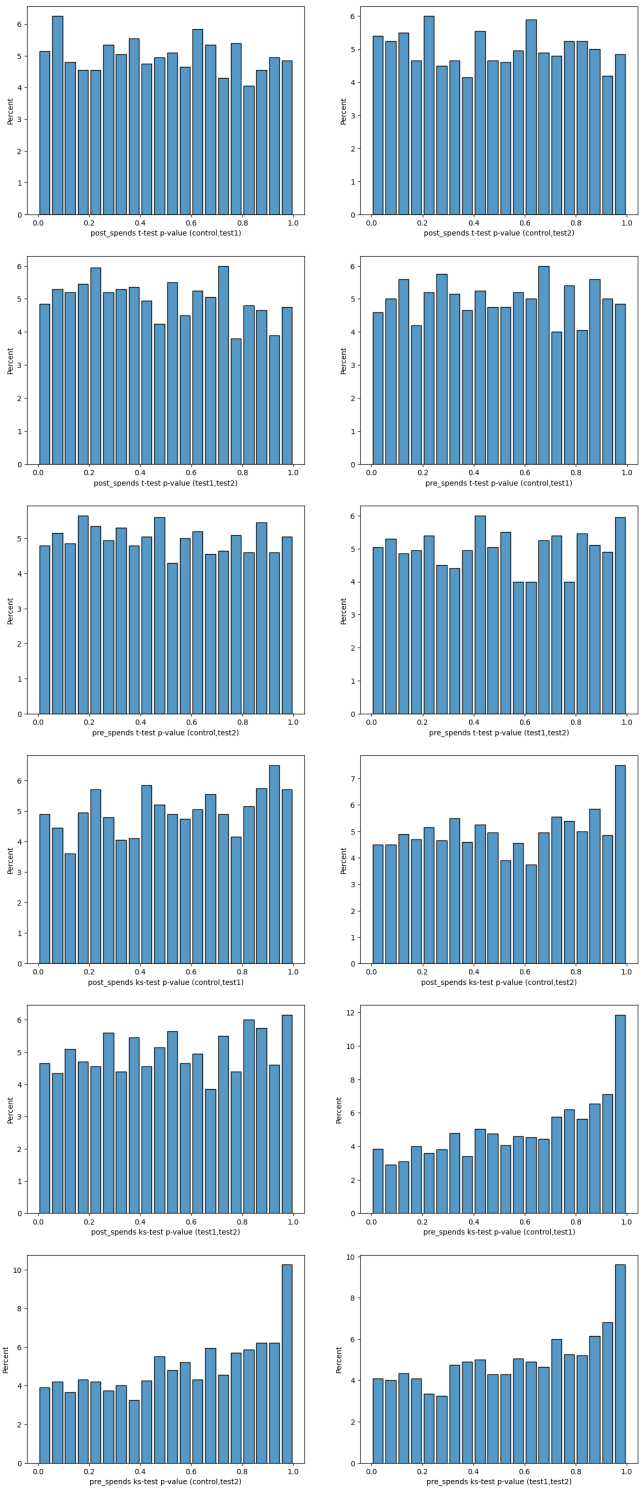
```
Generating random divisions:   0%|          | 0/2000 [00:00<?, ?it/s]
```

```
Metrics calculations:   0%|          | 0/2000 [00:00<?, ?it/s]
```




Distribution of average p-values of each target_field
Percent
post_spends mean t-test p-value
pre_spends mean t-test p-value
post_spends mean ks-test p-value
pre_spends mean ks-test p-value

Distribution of p-value for each group combination for each target_field



[4]:
```
results.keys()
```

[4]:
```
dict_keys(['best metric', 'best split', 'all metrics', 'all splits', 'best split␣
↪DataFrame', 'get_resume'])
```

`results` is a dictionary with dataframes as values. * 'best split' - result of best separation * 'best metric' - metrics of best split

* 'all metrics' - metrics of all experiments (i.e. of random splits)

* 'all splists' - results of all random splits

* 'best split DataFrame' - pandas DataFrame with column 'group' (or defined by 'group_column_name' parameter) contains values defined as keys in 'groups' parameter

* 'get resume' - function that plots p-values distribution of all experiments

[5]:
```
results['best metric']
```

[5]:
```
control post_spends mean                            451.86
test1 post_spends mean                              451.93
test2 post_spends mean                              451.90
post_spends mean delta (control - test1)             -0.06
post_spends mean delta% (control - test1)/test1      -0.01
post_spends t-test p-value (control,test1)            0.83
post_spends ks-test p-value (control,test1)           0.98
post_spends mean delta (control - test2)             -0.04
post_spends mean delta% (control - test2)/test2      -0.01
post_spends t-test p-value (control,test2)            0.91
post_spends ks-test p-value (control,test2)           1.00
post_spends mean delta (test1 - test2)                0.02
post_spends mean delta% (test1 - test2)/test2         0.01
post_spends t-test p-value (test1,test2)              0.95
post_spends ks-test p-value (test1,test2)             0.97
post_spends mean t-test p-value                       0.89
post_spends mean ks-test p-value                      0.98
post_spends t-test passed                             True
post_spends ks-test passed                            True
control pre_spends mean                             487.31
test1 pre_spends mean                               487.35
test2 pre_spends mean                               487.27
pre_spends mean delta (control - test1)              -0.04
pre_spends mean delta% (control - test1)/test1       -0.01
pre_spends t-test p-value (control,test1)             0.75
pre_spends ks-test p-value (control,test1)            0.77
pre_spends mean delta (control - test2)               0.03
pre_spends mean delta% (control - test2)/test2        0.01
pre_spends t-test p-value (control,test2)             0.83
pre_spends ks-test p-value (control,test2)            0.94
pre_spends mean delta (test1 - test2)                 0.08
pre_spends mean delta% (test1 - test2)/test2          0.02
pre_spends t-test p-value (test1,test2)               0.66
pre_spends ks-test p-value (test1,test2)              0.51
pre_spends mean t-test p-value                        0.83
```

(continues on next page)

(continued from previous page)

```
pre_spends mean ks-test p-value                    0.88
pre_spends t-test passed                           True
pre_spends ks-test passed                          True
mean of means t-test p-value                       0.86
mean of means ks-test p-value                      0.93
t-test passed %                                  100.00
ks-test passed %                                 100.00
mean_test_score                                    0.91
experiment_index                                    909
Name: 908, dtype: object
```

[ ]:

2.2.3 AB test

0. Import libraries

[1]:
```
import numpy as np
import pandas as pd

from lightautoml.addons.hypex import ABTest
from lightautoml.addons.hypex.utils.tutorial_data_creation import create_test_data

pd.options.display.float_format = '{:,.2f}'.format

np.random.seed(42)  # needed to create example data
```

1. Create or upload your dataset

In this case we will create random dataset with known effect size
If you have your own dataset, go to the part 2

[2]:
```
data = create_test_data(num_users=10000, rs=52, na_step=10, nan_cols=['age', 'gender'])
data
```

[2]:
```
      user_id  signup_month  treat  pre_spends  post_spends   age gender  \
0           0             0      0      488.00       414.44   NaN      M
1           1             8      1      512.50       462.22 26.00    NaN
2           2             7      1      483.00       479.44 25.00      M
3           3             0      0      501.50       424.33 39.00      M
4           4             1      1      543.00       514.56 18.00      F
...       ...           ...    ...         ...          ...   ...    ...
9995     9995            10      1      538.50       450.44 42.00      M
9996     9996             0      0      500.50       430.89 26.00      F
9997     9997             3      1      473.00       534.11 22.00      F
9998     9998             2      1      495.00       523.22 67.00      F
9999     9999             7      1      508.00       475.89 38.00      F

        industry
0     E-commerce
```

(continues on next page)

(continued from previous page)

```
1     E-commerce
2      Logistics
3     E-commerce
4     E-commerce
...          ...
9995   Logistics
9996   Logistics
9997  E-commerce
9998  E-commerce
9999  E-commerce

[10000 rows x 8 columns]
```

2. AB-test

2.0 Data

Let's correct data to see how AB-test works

[3]:
```
data_ab = data.copy()

half_data = int(data.shape[0] / 2)
data_ab['group'] = ['test'] * half_data + ['control'] * half_data
data_ab.head(3)
```

[3]:
```
   user_id  signup_month  treat  pre_spends  post_spends   age gender  \
0        0             0      0      488.00       414.44   NaN      M
1        1             8      1      512.50       462.22 26.00    NaN
2        2             7      1      483.00       479.44 25.00      M

     industry group
0  E-commerce  test
1  E-commerce  test
2   Logistics  test
```

3.1 Full AB-test

Full (basic) version of test includes calculation of all available metrics, which are: "diff in means", "diff in diff" and "cuped" Pay attention, that for "cuped" and "diff in diff" metrics required target before pilot.

[4]:
```
model = ABTest()
results = model.execute(
    data=data_ab,
    target_field='post_spends',
    target_field_before='pre_spends',
    group_field='group'
)
results
```

[4]:
```
{'size': {'test': 5000, 'control': 5000},
 'difference': {'ate': 1.108044444444488,
  'medain_diff': 0.16666666666668561,
  'cuped': 0.897496915890514,
```

(continues on next page)

(continued from previous page)

```
 'diff_in_diff': 0.610344444444479},
'p-value': {'t-test': 0.15973563889393272,
 'mann_whitney': 0.11494755666097989}}
```

2.2 Simple AB-test

To estimate effect without target data before pilot `calc_difference_method='ate'` can be used - effect will be estimated with "diff in means" method

[5]:
```
model = ABTest(calc_difference_method='ate')
model.execute(data=data_ab, target_field='post_spends', group_field='group')
```

[5]:
```
{'size': {'test': 5000, 'control': 5000},
 'difference': {'ate': 1.108044444444488},
 'p-value': {'t-test': 0.15973563889393272,
  'mann_whitney': 0.11494755666097989}}
```

2.2.4 Matching

0. Import libraries

[1]:
```
from lightautoml.addons.hypex import Matcher
```

1. Create or upload your dataset

In this case we will create random dataset with known effect size
If you have your own dataset, go to the part 2

[2]:
```
from lightautoml.addons.hypex.utils.tutorial_data_creation import create_test_data
```

[3]:
```
df = create_test_data(num_users=10000, rs=42, na_step=45, nan_cols=['age', 'gender'])
df
```

[3]:
```
      user_id  signup_month  treat  pre_spends  post_spends   age gender  \
0           0             0      0       504.5   422.777778   NaN      F
1           1             4      1       500.0   506.333333  51.0    NaN
2           2             0      0       485.0   434.000000  56.0      F
3           3             8      1       452.0   468.111111  46.0      M
4           4             0      0       488.5   420.111111  56.0      M
...       ...           ...    ...         ...          ...   ...    ...
9995     9995             2      1       482.0   501.666667  31.0      M
9996     9996             0      0       453.0   406.888889  53.0      M
9997     9997             0      0       461.0   415.111111  52.0      F
9998     9998            10      1       491.5   439.222222  22.0      M
9999     9999             2      1       481.0   517.222222  53.0      M

        industry
0      Logistics
1     E-commerce
2      Logistics
```

(continues on next page)

(continued from previous page)

```
3     E-commerce
4      Logistics
...          ...
9995   Logistics
9996   Logistics
9997  E-commerce
9998  E-commerce
9999  E-commerce

[10000 rows x 8 columns]
```

[4]:
```
df.columns
```

[4]:
```
Index(['user_id', 'signup_month', 'treat', 'pre_spends', 'post_spends', 'age',
       'gender', 'industry'],
      dtype='object')
```

[5]:
```
df['treat'].value_counts()
```

[5]:
```
treat
0    5002
1    4998
Name: count, dtype: int64
```

[6]:
```
df['gender'].isna().sum()
```

[6]:
```
223
```

2. Matching

2.0 Init params

info_col used to define informative attributes that should not be part of matching, such as user_id
But to explicitly store this column in the table, so that you can compare directly after computation

[7]:
```
info_col = ['user_id']

outcome = 'post_spends'
treatment = 'treat'
```

2.1 Simple matching

This is the easiest way to initialize and calculate metrics on a Matching task
Use it when you are clear about each attribute or if you don't have any additional task conditions (Strict equality for certain features)

[8]:
```
# Standard model with base parameters
model = Matcher(input_data=df, outcome=outcome, treatment=treatment, info_col=info_col,
                algo='fast')
```

```
[18.12.2024 22:05:15 | hypex | INFO]: Number of NaN values filled with zeros: 446
```

Feature selection models the significance of features for the accuracy of target approximation. However, it does not rule out the possibility of overlooked features, the complex impact of features on target description, or the significance of features from a business logic perspective. The algorithm will not function correctly if there are data leaks. Points to consider when selecting features:

- Data leaks - these should not be present.
- Influence on treatment distribution - features should not affect the treatment distribution.
- The target should be describable by features.
- All features significantly affecting the target should be included.
- The business rationale of features.
- The feature selection function can be useful for addressing these tasks, but it does not solve them nor does it absolve the user of the responsibility for their selection, nor does it justify it.

Link to ReadTheDocs

[9]:
```
selected_features = model.feature_select()
selected_features
```

```
/Users/tikhomirov/PycharmProjects/Sber_New/LightAutoML/.venv/lib/python3.10/site-
↪packages/hypex/selectors/feature_selector.py:42: UserWarning: FeatureSelector does not␣
↪rule out the possibility of overlooked features, the complex impact of features on␣
↪target description, or the significance of features from a business logic perspective.
  warnings.warn(
```

[9]:

	rank
signup_month	1
pre_spends	2
age	3
gender_F	4
gender_M	5
industry_Logistics	6

[10]:
```
chosen_features = selected_features[:4].index
chosen_features
```

[10]:
```
Index(['signup_month', 'pre_spends', 'age', 'gender_F'], dtype='object')
```

[11]:
```
results, quality_results, df_matched = model.estimate(features=chosen_features)
```

```
  0%|          | 0/10000 [00:00<?, ?it/s]
```

```
[18.12.2024 22:05:18 | Faiss hypex | INFO]: The entry of bias into the ATT is 0.1%
```

What is necessary to check in this table?¶

- effect size in ATT - it is effect in treated group
- standart error shows how accurately the parameter estimate corresponds to the true value in the total population
- p-value shows the measure of randomness of the sample (in this example: A p-value of 0.0 means that there is a 0% (percentage probability) that the result is due to randomness)

- ci (confidence interval) - the interval that covers the estimated parameter (ATT, ATC, ATE) with a given probability

[12]:
```
results
```

[12]:

	effect_size	std_err	p-val	ci_lower	ci_upper	outcome
ATE	82.636764	2.339283	0.0	78.051770	87.221757	post_spends
ATC	101.687566	4.571439	0.0	92.727546	110.647585	post_spends
ATT	63.570714	0.694726	0.0	62.209051	64.932378	post_spends

Variable quality_results contains:

- results of psi test
- resulnt of Kolmogorov-Smirnov test
- results of smd
- number of repeats

PSI (Population Stability Index)
shows the difference between treated and untreated populations

Rules:

- PSI < 0.1 - No change. You can continue using existing model.
- PSI >=0.1 but less than 0.2 - Slight change is required.
- PSI >=0.2 - Significant change is required. Ideally, you should not use this model any more.

SMD (Standardized Mean Differences)
helps to check if the balance of the groups has been reached

Rules:

- Smaller than 0.1. For a randomized trial, the smd between all the covariates should typically fall into this bucket.
- 0.1 - 0.2. Not necessarily balanced, but small enough that people are usually not too worried about them.
- 0.2. Values that are greater than this threshold are considered seriously imbalanced.

Repeats
shows the fraction of duplicated indexes

[13]:
```
quality_results.keys()
```

[13]:
```
dict_keys(['psi', 'ks_test', 'smd', 'repeats'])
```

Kolmogorov-Smirnov test¶ the distribution of one sample is compared with the distribution of the second sample and it is decided whether the samples have the same or different distribution.

Table shows the p-value results of the test. If p-value < 0.05 we reject the null hypothesis and we have enough evidence to say that the sample data do not have the same distribution.

[14]:
```
quality_results['ks_test']
```

[14]:
```
              match_control_to_treat  match_treat_to_control
age                     1.418144e-01            2.209790e-07
pre_spends             2.348715e-264            3.829212e-19
signup_month            0.000000e+00            0.000000e+00
```

[15]:
```
df_matched
```

[15]:
```
      index  signup_month  pre_spends   age  gender_F  gender_M  \
0       956             8       487.0  22.0         0         1
1      7966             5       471.5  69.0         0         0
2      7231             4       487.0  62.0         1         0
3      1443             1       517.0  36.0         1         0
4      7973            10       501.0  65.0         1         0
...     ...           ...         ...   ...       ...       ...
4997   2926             0       486.0  45.0         0         0
4998   1656             0       465.5  39.0         1         0
4999   2812             0       451.0  53.0         1         0
5000   2813             0       504.5  69.0         0         1
5001   1297             0       495.5  43.0         1         0

      industry_Logistics  signup_month_matched  pre_spends_matched  \
0                      0                   0.0               506.0
1                      1                   0.0               483.5
2                      1                   0.0               496.5
3                      1                   0.0               520.5
4                      0                   0.0               525.0
...                  ...                   ...                 ...
4997                   0                   2.0               479.5
4998                   1                   2.0               460.0
4999                   1                   2.0               441.5
5000                   0                   1.0               514.0
5001                   1                   1.0               496.5

      age_matched  gender_F_matched  gender_M_matched  \
0            23.0               0.0               1.0
1            69.0               0.0               1.0
2            62.0               1.0               0.0
3            36.0               1.0               0.0
4            65.0               1.0               0.0
...           ...               ...               ...
4997         46.0               0.0               1.0
4998         40.0               1.0               0.0
4999         57.0               1.0               0.0
5000         65.0               0.0               1.0
5001         38.0               1.0               0.0

      industry_Logistics_matched index_matched  post_spends  \
0                            1.0        [6352]   462.222222
1                            1.0        [5349]   505.222222
```

(continues on next page)

(continued from previous page)

```
2                           1.0        [8654]   503.555556
3                           1.0        [7881]   526.111111
4                           0.0        [7333]   440.444444
...                         ...           ...          ...
4997                        1.0         [971]   421.777778
4998                        1.0        [6730]   417.666667
4999                        1.0        [6610]   417.333333
5000                        0.0        [9319]   422.888889
5001                        0.0        [3004]   415.111111

      post_spends_matched  post_spends_matched_bias  treat  treat_matched
0              408.333333                 54.004165      1              0
1              415.222222                 90.068667      1              0
2              428.777778                 74.832139      1              0
3              423.111111                103.020028      1              0
4              416.666667                 23.915112      1              0
...                   ...                       ...    ...            ...
4997           509.888889                 89.948512      0              1
4998           502.222222                 86.443231      0              1
4999           521.555556                106.002291      0              1
5000           528.555556                107.086127      0              1
5001           532.555556                118.405412      0              1

[10000 rows x 19 columns]
```

[16]:
```
df_matched[df_matched['industry_Logistics'] != df_matched['industry_Logistics_matched']]
```

[16]:
```
      index  signup_month  pre_spends   age  gender_F  gender_M  \
0       956             8       487.0  22.0         0         1
5      8108             7       482.0  54.0         0         1
6      7228             7       493.0  35.0         1         0
7      7968            10       512.5  64.0         1         0
8      1396             6       479.0  27.0         0         0
...     ...           ...         ...   ...       ...       ...
4990   2924             0       493.0  32.0         1         0
4991    313             0       484.0  35.0         1         0
4992   2919             0       490.5  20.0         0         1
4997   2926             0       486.0  45.0         0         0
5001   1297             0       495.5  43.0         1         0

      industry_Logistics  signup_month_matched  pre_spends_matched  \
0                      0                   0.0               506.0
5                      1                   0.0               499.0
6                      1                   0.0               511.0
7                      0                   0.0               540.0
8                      0                   0.0               493.5
...                  ...                   ...                 ...
4990                   1                   1.0               496.5
4991                   0                   2.0               480.5
4992                   1                   2.0               483.5
4997                   0                   2.0               479.5
5001                   1                   1.0               496.5
```

(continues on next page)

(continued from previous page)

```
      age_matched  gender_F_matched  gender_M_matched  \
0            23.0               0.0               1.0
5            55.0               0.0               1.0
6            36.0               1.0               0.0
7            63.0               1.0               0.0
8            27.0               0.0               1.0
...           ...               ...               ...
4990         38.0               1.0               0.0
4991         34.0               1.0               0.0
4992         18.0               0.0               1.0
4997         46.0               0.0               1.0
5001         38.0               1.0               0.0

      industry_Logistics_matched index_matched  post_spends  \
0                            1.0        [6352]   462.222222
5                            0.0        [1742]   478.333333
6                            0.0        [5653]   492.555556
7                            1.0        [4470]   436.777778
8                            1.0        [6721]   485.666667
...                          ...           ...          ...
4990                         0.0        [3004]   425.000000
4991                         1.0        [7816]   426.000000
4992                         0.0        [2781]   420.333333
4997                         1.0         [971]   421.777778
5001                         0.0        [3004]   415.111111

      post_spends_matched  post_spends_matched_bias  treat  treat_matched
0              408.333333                 54.004165      1              0
5              411.222222                 67.214942      1              0
6              419.444444                 73.220665      1              0
7              418.666667                 18.261920      1              0
8              408.888889                 76.860750      1              0
...                   ...                       ...    ...            ...
4990           532.555556                108.985004      0              1
4991           510.777778                 86.703675      0              1
4992           517.555556                 98.940997      0              1
4997           509.888889                 89.948512      0              1
5001           532.555556                118.405412      0              1

[4964 rows x 19 columns]
```

2.2 Matching with a fixed variable

Used when you have categorical feature(s) that you want to compare by strict equality
group_col is used for strict comparison of categorical features.
In our case there is only one attribute
If there are several such attributes, you should make one of them and use it

[17]:
```
group_col = "industry"
```

Also group_col might be the list.

[18]:
```
model = Matcher(input_data=df, outcome=outcome, treatment=treatment,
                info_col=info_col, group_col=group_col)
```

```
[18.12.2024 22:05:18 | hypex | INFO]: Number of NaN values filled with zeros: 446
```

[19]:
```
selected_features = model.feature_select()
selected_features
```

```
/Users/tikhomirov/PycharmProjects/Sber_New/LightAutoML/.venv/lib/python3.10/site-
↪packages/hypex/selectors/feature_selector.py:42: UserWarning: FeatureSelector does not␣
↪rule out the possibility of overlooked features, the complex impact of features on␣
↪target description, or the significance of features from a business logic perspective.
  warnings.warn(
```

[19]:

	rank
signup_month	1
pre_spends	2
age	3
gender_F	4
gender_M	5

[20]:
```
chosen_features = selected_features[:4].index
chosen_features
```

[20]:
```
Index(['signup_month', 'pre_spends', 'age', 'gender_F'], dtype='object')
```

[21]:
```
results, quality_results, df_matched = model.estimate(features=chosen_features)
```

```
  0%|          | 0/4 [00:00<?, ?it/s]
```

```
[18.12.2024 22:05:22 | Faiss hypex | INFO]: The entry of bias into the ATT is 0.2%
```

[22]:
```
results
```

[22]:

	effect_size	std_err	p-val	ci_lower	ci_upper	outcome
ATE	81.192392	1.631028	0.0	77.995577	84.389208	post_spends
ATC	98.945415	3.116605	0.0	92.836869	105.053961	post_spends
ATT	63.425162	0.652349	0.0	62.146559	64.703765	post_spends

[23]:
```
df_matched[df_matched['industry'] != df_matched['industry_matched']]
```

[23]:
```
Empty DataFrame
Columns: [index, signup_month, pre_spends, age, gender_F, gender_M, industry, signup_
↪month_matched, pre_spends_matched, age_matched, gender_F_matched, gender_M_matched,␣
↪industry_matched, index_matched, post_spends, post_spends_matched, post_spends_matched_
↪bias, treat, treat_matched]
Index: []
```

3. Results

3.1 ATE, ATT, ATC

[24]:
```
results
```

[24]:
```
     effect_size   std_err  p-val   ci_lower    ci_upper      outcome
ATE    81.192392  1.631028    0.0  77.995577   84.389208  post_spends
ATC    98.945415  3.116605    0.0  92.836869  105.053961  post_spends
ATT    63.425162  0.652349    0.0  62.146559   64.703765  post_spends
```

3.2 SMD, PSI, KS-test, repeats

[25]:
```
quality_results.keys()
```

[25]:
```
dict_keys(['psi', 'ks_test', 'smd', 'repeats'])
```

[26]:
```
quality_results['psi']
```

[26]:
```
         column_treated  anomaly_score_treated check_result_treated  \
0           age_treated                   0.01                   OK
1      gender_F_treated                   0.00                   OK
2      industry_treated                   0.00                   OK
3    pre_spends_treated                   0.61                  NOK
4  signup_month_treated                  16.14                  NOK

         column_untreated  anomaly_score_untreated check_result_untreated
0           age_untreated                     0.08                     OK
1      gender_F_untreated                     0.00                     OK
2      industry_untreated                     0.00                     OK
3    pre_spends_untreated                     0.16                     OK
4  signup_month_untreated                     0.00                     OK
```

[27]:
```
quality_results['ks_test']
```

[27]:
```
              match_control_to_treat  match_treat_to_control
age                     1.292517e-01            2.721265e-04
pre_spends             4.164710e-263            1.725086e-24
signup_month            0.000000e+00            0.000000e+00
```

[28]:
```
quality_results['repeats']
```

[28]:
```
{'match_control_to_treat': 0.42, 'match_treat_to_control': 0.07}
```

4. Save model

[29]:
```
model.save("test_model.pickle")
```

[30]:
```
model2 = Matcher.load("test_model.pickle")
```

[31]:
```
model2.results
```

[31]:
```
     effect_size   std_err  p-val   ci_lower    ci_upper      outcome
ATE    81.192392  1.631028    0.0  77.995577   84.389208  post_spends
```

(continues on next page)

(continued from previous page)

```
ATC    98.945415  3.116605    0.0  92.836869  105.053961  post_spends
ATT    63.425162  0.652349    0.0  62.146559   64.703765  post_spends
```

[32]:
```
model.results
```

[32]:
```
     effect_size   std_err  p-val   ci_lower    ci_upper      outcome
ATE    81.192392  1.631028    0.0  77.995577   84.389208  post_spends
ATC    98.945415  3.116605    0.0  92.836869  105.053961  post_spends
ATT    63.425162  0.652349    0.0  62.146559   64.703765  post_spends
```

[ ]:

2.2.5 Matching without replacement

0. Import libraries

[1]:
```
from lightautoml.addons.hypex import Matcher
```

1. Create or upload your dataset

In this case we will create random dataset with known effect size
If you have your own dataset, go to the part 2

[2]:
```
from lightautoml.addons.hypex.utils.tutorial_data_creation import create_test_data
```

[3]:
```
df = create_test_data(num_users=10000, rs=42, na_step=45, nan_cols=['age', 'gender'])
df
```

[3]:
```
      user_id  signup_month  treat  pre_spends  post_spends   age gender  \
0           0             0      0       504.5   422.777778   NaN      F
1           1             4      1       500.0   506.333333  51.0    NaN
2           2             0      0       485.0   434.000000  56.0      F
3           3             8      1       452.0   468.111111  46.0      M
4           4             0      0       488.5   420.111111  56.0      M
...       ...           ...    ...         ...          ...   ...    ...
9995     9995             2      1       482.0   501.666667  31.0      M
9996     9996             0      0       453.0   406.888889  53.0      M
9997     9997             0      0       461.0   415.111111  52.0      F
9998     9998            10      1       491.5   439.222222  22.0      M
9999     9999             2      1       481.0   517.222222  53.0      M

        industry
0      Logistics
1     E-commerce
2      Logistics
3     E-commerce
4      Logistics
...          ...
9995   Logistics
9996   Logistics
```

(continues on next page)

(continued from previous page)

```
9997  E-commerce
9998  E-commerce
9999  E-commerce

[10000 rows x 8 columns]
```

[4]:
```
df.columns
```

[4]:
```
Index(['user_id', 'signup_month', 'treat', 'pre_spends', 'post_spends', 'age',
       'gender', 'industry'],
      dtype='object')
```

[5]:
```
df['treat'].value_counts()
```

[5]:
```
treat
0    5002
1    4998
Name: count, dtype: int64
```

[6]:
```
df['gender'].isna().sum()
```

[6]:
```
223
```

2. Matching without replacement

2.0 Init params

info_col used to define informative attributes that should not be part of matching, such as user_id
But to explicitly store this column in the table, so that you can compare directly after computation

[7]:
```
info_col = ['user_id']

outcome = 'post_spends'
treatment = 'treat'
```

2.1 Matching

This is the easiest way to initialize and calculate metrics on a Matching task
Use it when you are clear about each attribute or if you don't have any additional task conditions (Strict equality for certain features)

[8]:
```
# Standard model with base parameters
model = Matcher(input_data=df, outcome=outcome, treatment=treatment, info_col=info_col)
```

```
[18.12.2024 22:04:52 | hypex | INFO]: Number of NaN values filled with zeros: 446
```

[9]:
```
df_matched = model.match_no_rep()
```

[10]:
```
df_matched
```

[10]:

```
      signup_month  treat  pre_spends  post_spends   age  gender_F  gender_M  \
0                0      0       504.5   422.777778   0.0         1         0
1                4      1       500.0   506.333333  51.0         0         0
2                0      0       485.0   434.000000  56.0         1         0
3                8      1       452.0   468.111111  46.0         0         1
4                0      0       488.5   420.111111  56.0         0         1
...            ...    ...         ...          ...   ...       ...       ...
9991             2      1       482.0   501.666667  31.0         0         1
9992             0      0       453.0   406.888889  53.0         0         1
9993             0      0       461.0   415.111111  52.0         1         0
9994            10      1       491.5   439.222222  22.0         0         1
9995             2      1       481.0   517.222222  53.0         0         1

      industry_Logistics  user_id  signup_month_matched  treat_matched  \
0                      1        0                     4              1
1                      0        1                     0              0
2                      1        2                     2              1
3                      0        3                     0              0
4                      1        4                     8              1
...                  ...      ...                   ...            ...
9991                   1     9995                     0              0
9992                   1     9996                     4              1
9993                   0     9997                     7              1
9994                   0     9998                     0              0
9995                   0     9999                     0              0

      pre_spends_matched  post_spends_matched  age_matched  gender_F_matched  \
0                  522.5           509.777778          0.0                 1
1                  488.0           420.333333         50.0                 0
2                  492.5           510.777778         56.0                 1
3                  453.5           415.222222         42.0                 0
4                  488.0           472.111111         59.0                 0
...                  ...                  ...          ...               ...
9991               474.0           435.111111         28.0                 0
9992               459.0           518.777778         57.0                 0
9993               459.5           472.111111         51.0                 1
9994               492.5           411.555556         25.0                 0
9995               471.5           427.000000         53.0                 0

      gender_M_matched  industry_Logistics_matched  user_id_matched
0                    0                           1             4095
1                    0                           0             3916
2                    0                           1             6057
3                    1                           0             5255
4                    1                           1             6874
...                ...                         ...              ...
9991                 1                           1             8397
9992                 0                           1             8416
9993                 0                           0             5727
9994                 1                           0             5675
9995                 1                           0             6361

[9996 rows x 18 columns]
```

2.2.6 Modeling Limit Distribution

Modeling real data

This part provides the optimal method for tests in which you need to determine a statistically significantly better sample

[1]:
```
import plotly.graph_objects as go
from scipy.stats import bernoulli, norm
from statsmodels.stats.proportion import proportions_ztest

import numpy as np

from lightautoml.addons.hypex.abn_test import min_sample_size
from lightautoml.addons.hypex.abn_test import quantile_of_marginal_distribution
from lightautoml.addons.hypex.abn_test import test_on_marginal_distribution
```

Help functions for easy solution and construction of confidence intervals

[2]:
```
def min_sample_size_2(d, alpha=0.05, beta=0.2, p=1 / 2):
    l = 2 * p * (1 - p) * (((norm.ppf(1 - alpha) - norm.ppf(beta)) / d) ** 2)
    return int(l)


def confidence_interval_bern(p, cnt, c=0.95):
    se = np.sqrt(p * (1 - p) / cnt)
    q = norm.ppf(1 - (1 - c) / 2)
    return p - q * se, p + q * se
```

Comparison of simple and optimal solutions

Consider the size of a single sample depending on the number of samples.

[3]:
```
k_list = [2, 3, 4, 5, 6]  # Number of samples
alpha = 0.05  # Significance level
beta = 0.2  # 1 - power
p_true = 0.3  # Bernoulli distribution parameter
d_relative_pred = 0.1  # I assume that the conversion rate will increase by d_relative_
↪pred * p

n_1 = []  # limit method
n_2 = []  # method 2
```

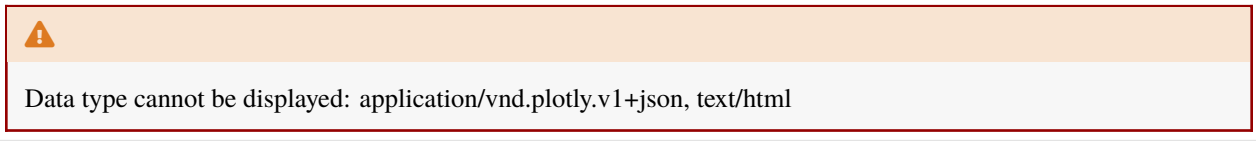
[4]:
```
for k in k_list:
    alpha_ = alpha / k
    beta_ = beta / (k - 1)

    c_1 = quantile_of_marginal_distribution(num_samples=k, quantile_level=1 - alpha_)
    c_2 = quantile_of_marginal_distribution(num_samples=k, quantile_level=beta)
    n_1 += [min_sample_size(number_of_samples=k,
                            minimum_detectable_effect=p_true * d_relative_pred,
                            variances=p_true * (1 - p_true),
                            significance_level=alpha,
                            power_level=beta,
```

(continues on next page)

(continued from previous page)

```
                              quantile_1=c_1,
                              quantile_2=c_2)]
    n_2 += [min_sample_size_2(d=p_true * d_relative_pred, p=p_true,
                              alpha=alpha_, beta=beta_)]
```

[5]:
```
fig = go.Figure()
fig.add_trace(go.Scatter(x=k_list, y=n_1,
                         mode="lines", name=f"Optimal solution"))
fig.add_trace(go.Scatter(x=k_list, y=n_2,
                         mode="lines", name=f"Simple solution"))

fig.update_layout(title={
    "text": "Modeling the minimum sample size",
    "font": {
        "size": 18
    }
},
    xaxis={
        "title_text": "Number of samples"
    },
    yaxis={
        "title_text": "The size of a single sample",
        "range": [0, 10500]
    },
    legend={
        "yanchor": "top",
        "y": 0.99,
        "xanchor": "left",
        "x": 0.005
    },
    width=725,
    height=500,
    autosize=True,
    font={
        "size": 16
    })
fig.show()
```



Data type cannot be displayed: application/vnd.plotly.v1+json, text/html

Consider the probability of an error of the first kind - this is the probability of rejecting a hypothesis if it is valid. In order not to make mistakes too often, we want to control this value.

The value is theoretical, so we will model the frequency of incorrect rejection of the hypothesis, close to the true probability of error of the first kind with a sufficient number of simulations.

[6]:
```
d_relative_pred = 0.05  # Assume that the conversion rate will increase by d_relative_
↪pred * p
```

(continues on next page)

(continued from previous page)

```
iter_size = 1000  # Number of test iterations

reject_stat_1 = []  # limit method
reject_stat_2 = []  # pairwise comparisons
```

[7]:
```
for k in k_list:
    alpha_ = alpha / k
    beta_ = beta / (k - 1)

    c_1 = quantile_of_marginal_distribution(num_samples=k, quantile_level=1 - alpha_)
    c_2 = quantile_of_marginal_distribution(num_samples=k, quantile_level=beta)
    n_1 = min_sample_size(number_of_samples=k,
                          minimum_detectable_effect=p_true * d_relative_pred,
                          variances=p_true * (1 - p_true),
                          significance_level=alpha,
                          power_level=beta,
                          quantile_1=c_1,
                          quantile_2=c_2)
    n_2 = min_sample_size_2(d=p_true * d_relative_pred, p=p_true,
                            alpha=alpha_, beta=beta_)

    success_cnt_1 = bernoulli.rvs(p_true, size=[iter_size, k, n_1])
    success_cnt_2 = bernoulli.rvs(p_true, size=[iter_size, k, n_2])
    # limit method
    reject_cnt_0 = 0
    for i in range(iter_size):
        hyp = test_on_marginal_distribution(success_cnt_1[i],
                                            significance_level=alpha,
                                            quantiles=c_1)
        if hyp == 0:
            reject_cnt_0 += 1
    reject_stat_1.append(1 - reject_cnt_0 / iter_size)

    # pairwise comparisons
    reject_cnt_0 = 0
    for i in range(iter_size):
        hyp = 0  # if nothing is rejected, then H(0) is valid
        for s in range(k):
            all_rejected_flg = True
            for t in range(k):
                if s != t:
                    if proportions_ztest([np.sum(success_cnt_2[i][s]), np.sum(success_
↪cnt_2[i][t])], [n_2, n_2],
                                         alternative="larger")[1] > alpha_:
                        all_rejected_flg = False
                        break
            if all_rejected_flg:
                hyp = s + 1
                break
        if hyp == 0:
            reject_cnt_0 += 1
    reject_stat_2.append(1 - reject_cnt_0 / iter_size)
```

(continues on next page)

(continued from previous page)

```
reject_stat_1 = np.array(reject_stat_1)
left_side_1, right_side_1 = confidence_interval_bern(reject_stat_1, iter_size, c=1 -␣
↪alpha)

reject_stat_2 = np.array(reject_stat_2)
left_side_2, right_side_2 = confidence_interval_bern(reject_stat_2, iter_size, c=1 -␣
↪alpha)
```

[8]:
```
fig = go.Figure()
fig.add_trace(go.Scatter(x=k_list, y=len(k_list) * [alpha],
                         mode="lines", name="Significance level"))
fig.add_trace(go.Scatter(x=k_list,
                         y=reject_stat_1,
                         mode="lines",
                         name=f"Optimal solution",
                         error_y={
                             "type": "data",
                             "symmetric": False,
                             "array": reject_stat_1 - left_side_1,
                             "arrayminus": right_side_1 - reject_stat_1,
                             "visible": True
                         }))

fig.add_trace(go.Scatter(x=k_list,
                         y=reject_stat_2,
                         mode="lines",
                         name=f"Simple solution",
                         error_y={
                             "type": "data",
                             "symmetric": False,
                             "array": reject_stat_2 - left_side_2,
                             "arrayminus": right_side_2 - reject_stat_2,
                             "visible": True
                         }))

fig.update_layout(title="Modeling the probability of a first-kind error",
                  xaxis={
                      "title_text": "Number of samples"
                  },
                  yaxis={
                      "tickformat": ".0%",
                      "title_text": "The frequency of incorrect rejection of the␣
↪hypothesis",
                      "range": [-0.005, 0.07]
                  },
                  legend={
                      "yanchor": "top",
                      "y": 0.99,
                      "xanchor": "right",
                      "x": 0.995
                  },
                  width=725,
```

(continues on next page)

(continued from previous page)

```
                  height=500)
fig.show()
```

Data type cannot be displayed: application/vnd.plotly.v1+json, text/html

Consider the probability of a second kind of error - this is the probability of accepting a hypothesis if it is unfair.

We will model the frequency of incorrect hypothesis acceptance, close to the true probability of a second-kind error with a sufficient number of simulations.

[9]:
```
d_relative_pred = 0.1  # Assume that the conversion rate will increase by d_relative_
↪pred * p
iter_size = 1000  # Number of test iterations

reject_stat_1 = []
reject_stat_2 = []
```

[10]:
```
for k in k_list:
    alpha_ = alpha / k
    beta_ = beta / (k - 1)

    c_1 = quantile_of_marginal_distribution(num_samples=k, quantile_level=1 - alpha_)
    c_2 = quantile_of_marginal_distribution(num_samples=k, quantile_level=beta)
    n_1 = min_sample_size(number_of_samples=k,
                          minimum_detectable_effect=p_true * d_relative_pred,
                          variances=p_true * (1 - p_true),
                          significance_level=alpha,
                          power_level=beta,
                          quantile_1=c_1,
                          quantile_2=c_2)
    n_2 = min_sample_size_2(d=p_true * d_relative_pred, p=p_true,
                            alpha=alpha_, beta=beta_)

    success_cnt_2 = []
    for i in range(k - 1):
        success_cnt_2 += [bernoulli.rvs(p_true, size=[iter_size, n_2])]
    success_cnt_2 += [bernoulli.rvs(p_true * (1 + d_relative_pred), size=[iter_size, n_
↪2])]

    # limit method
    reject_cnt_0 = 0
    for i in range(iter_size):
        success_cnt_1 = []
        for _ in range(k - 1):
            success_cnt_1 += [bernoulli.rvs(p_true, size=n_1)]
        success_cnt_1 += [bernoulli.rvs(p_true * (1 + d_relative_pred), size=n_1)]
        hyp = test_on_marginal_distribution(success_cnt_1,
                                            significance_level=alpha,
                                            quantiles=c_1)
```

(continues on next page)

(continued from previous page)

```
        if hyp == k:
            reject_cnt_0 += 1
    reject_stat_1.append(1 - reject_cnt_0 / iter_size)

    # pairwise comparisons
    reject_cnt_0 = 0
    for i in range(iter_size):
        hyp = 0
        for s in range(k):
            all_rejected_flg = True
            for t in range(k):
                if s != t:
                    if proportions_ztest([np.sum(success_cnt_2[s][i]), np.sum(success_
↪cnt_2[t][i])], [n_2, n_2],
                                         alternative="larger")[1] > alpha_:
                        all_rejected_flg = False
                        break
            if all_rejected_flg:
                hyp = s + 1
                break
        if hyp == k:
            reject_cnt_0 += 1
    reject_stat_2.append(1 - reject_cnt_0 / iter_size)

reject_stat_1 = np.array(reject_stat_1)
left_side_1, right_side_1 = confidence_interval_bern(reject_stat_1, iter_size, c=1 -␣
↪alpha)
reject_stat_2 = np.array(reject_stat_2)
left_side_2, right_side_2 = confidence_interval_bern(reject_stat_2, iter_size, c=1 -␣
↪alpha)
```

[11]:
```
fig = go.Figure()
fig.add_trace(go.Scatter(x=k_list, y=len(k_list) * [beta],
                         mode="lines", name="The probability of an error of the 2nd kind
↪"))
fig.add_trace(go.Scatter(x=k_list,
                         y=reject_stat_1,
                         mode="lines",
                         name=f"Optimal solution",
                         error_y={
                             "type": "data",
                             "symmetric": False,
                             "array": reject_stat_1 - left_side_1,
                             "arrayminus": right_side_1 - reject_stat_1,
                             "visible": True
                         }))

fig.add_trace(go.Scatter(x=k_list,
                         y=reject_stat_2,
                         mode="lines",
                         name=f"Simple solution",
                         error_y={
```

(continues on next page)

(continued from previous page)

```
                            "type": "data",
                            "symmetric": False,
                            "array": reject_stat_2 - left_side_2,
                            "arrayminus": right_side_2 - reject_stat_2,
                            "visible": True
                        }))

fig.update_layout(title="Modeling the probability of a second kind of error",
                  xaxis={
                      "title_text": "Number of samples"
                  },
                  yaxis={
                      "tickformat": ".0%",
                      "title_text": "The frequency of hypothesis acceptance",
                      "range": [0, 0.25]
                  },
                  legend={
                      "yanchor": "top",
                      "y": 0.99,
                      "xanchor": "right",
                      "x": 0.995
                  },
                  width=725,
                  height=500)
fig.show()
```

[ ]:

2.2.7 How to do experiment via Limit Distribution? (ABn Test)

[1]:
```
from scipy.stats import bernoulli

from lightautoml.addons.hypex.abn_test import min_sample_size
```

(continues on next page)

(continued from previous page)

```
from lightautoml.addons.hypex.abn_test import test_on_marginal_distribution
import numpy as np
```

Initialize random state

[2]:
```
seed = 42  # You can choose any number as the seed
random_state = np.random.RandomState(seed)
```

Multiple testing for best sample selection

Number of samples and parameters

[3]:
```
num_samples = 10  # Number of samples
minimum_detectable_effect = 0.05  # MDE
assumed_conversion = 0.3  # Assumed conversion rate
significance_level = 0.05  # Significance level
power_level = 0.2  # Power level (1 - beta)
```

Calculate the minimum sample size

[4]:
```
sample_size = min_sample_size(
    num_samples,
    minimum_detectable_effect,
    variances=assumed_conversion * (1 - assumed_conversion),
    significance_level=significance_level,
    power_level=power_level,
    equal_variance=True,
)
print(f"Sample size = {sample_size}")
```

```
Sample size = 1313
```

Testing samples with equal conversion rate

[5]:
```
print("\nSamples with equal conversion rate")
for _ in range(5):
    samples = bernoulli.rvs(
        assumed_conversion, size=[num_samples, sample_size], random_state=random_state
    )
    hypothesis = test_on_marginal_distribution(
        samples, significance_level=significance_level
    )
    print(f"\tAccepted hypothesis H({hypothesis})")
```

```

Samples with equal conversion rate
        Accepted hypothesis H(0)
        Accepted hypothesis H(0)
        Accepted hypothesis H(0)
        Accepted hypothesis H(0)
        Accepted hypothesis H(0)
```

Testing where the last sample has a higher conversion rate by MDE

[6]:
```
print("\nLast sample has higher conversion by MDE")
for _ in range(5):
    samples = [
        bernoulli.rvs(assumed_conversion, size=sample_size, random_state=random_state)
        for _ in range(num_samples - 1)
    ]
    samples.append(
        bernoulli.rvs(
            assumed_conversion + minimum_detectable_effect,
            size=sample_size,
            random_state=random_state,
        )
    )
    hypothesis = test_on_marginal_distribution(
        samples, significance_level=significance_level
    )
    print(f"\tAccepted hypothesis H({hypothesis})")
```

```
Last sample has higher conversion by MDE
        Accepted hypothesis H(10)
        Accepted hypothesis H(10)
        Accepted hypothesis H(10)
        Accepted hypothesis H(10)
        Accepted hypothesis H(10)
```

Multiple testing for best client income sample (conversion * price)

Parameters for different samples

[7]:
```
num_samples = 5  # Number of samples
minimum_detectable_effect = 2.5  # MDE
prices = [100, 150, 150, 200, 250]  # Tariff prices
conversions = [0.15, 0.1, 0.1, 0.075, 0.06]  # Tariff conversions
significance_level = 0.05
power_level = 0.2
variances = [
    price ** 2 * conversion * (1 - conversion)
    for price, conversion in zip(prices, conversions)
]
```

Calculate minimum sample size for unequal variances

[8]:
```
sample_size = min_sample_size(
    num_samples,
    minimum_detectable_effect,
    variances=variances,
    significance_level=significance_level,
    power_level=power_level,
    equal_variance=False,
)
```

(continues on next page)

(continued from previous page)

```
print(f"Sample size = {sample_size}")
```

```
Sample size = 7200
```

Testing samples with equal ARPU (Average Revenue Per User)

[9]:
```
print("\nSamples with equal ARPU")
for _ in range(5):
    samples = [
        price * bernoulli.rvs(conversion, size=sample_size)
        for price, conversion in zip(prices, conversions)
    ]
    hypothesis = test_on_marginal_distribution(
        samples, significance_level=significance_level
    )
    print(f"\tAccepted hypothesis H({hypothesis})")
```

```
Samples with equal ARPU
        Accepted hypothesis H(0)
        Accepted hypothesis H(0)
        Accepted hypothesis H(4)
        Accepted hypothesis H(0)
        Accepted hypothesis H(0)
```

Testing where the last sample has higher ARPU by MDE

[10]:
```
print("\nLast sample has higher ARPU by MDE")
for _ in range(5):
    samples = [
        price * bernoulli.rvs(conversion, size=sample_size)
        for price, conversion in zip(prices, conversions[:-1])
    ]
    samples.append(
        prices[-1]
        * bernoulli.rvs(
            conversions[-1] + minimum_detectable_effect / prices[-1], size=sample_size
        )
    )
    hypothesis = test_on_marginal_distribution(
        samples, significance_level=significance_level
    )
    print(f"\tAccepted hypothesis H({hypothesis})")
```

```
Last sample has higher ARPU by MDE
        Accepted hypothesis H(5)
        Accepted hypothesis H(5)
        Accepted hypothesis H(5)
        Accepted hypothesis H(0)
        Accepted hypothesis H(5)
```

[ ]:

CHAPTER

THREE

KAGGLE KERNELS

- Tabular Playground Series April 2021 competition solution
- Titanic competition solution (80% accuracy)
- Titanic **12-code-lines** competition solution (78% accuracy)
- House prices competition solution
- Natural Language Processing with Disaster Tweets solution
- Tabular Playground Series March 2021 competition solution
- Tabular Playground Series February 2021 competition solution
- Interpretable WhiteBox solution
- Custom ML pipeline elements inside existing ones
- Tabular Playground Series November 2022 competition solution with Neural Networks

CHAPTER

FOUR

OTHERS

4.1 LightAutoML crash courses

(Russian) AutoML course for OpenDataScience community

4.2 Video guides

- (Russian) LightAutoML webinar for Sberloga community (Alexander Ryzhkov), Dmitry Simakov)
- (Russian) LightAutoML hands-on tutorial in Kaggle Kernels (Alexander Ryzhkov)
- (English) Automated Machine Learning with LightAutoML: theory and practice (Alexander Ryzhkov)
- (English) LightAutoML framework general overview, benchmarks and advantages for business (Alexander Ryzhkov)
- (English) LightAutoML practical guide - ML pipeline presets overview (Dmitry Simakov)

4.3 Papers

Anton Vakhrushev, Alexander Ryzhkov, Dmitry Simakov, Rinchin Damdinov, Maxim Savchenko, Alexander Tuzhilin "LightAutoML: AutoML Solution for a Large Financial Services Ecosystem". arXiv:2109.01528, 2021.

4.4 Articles about LightAutoML

- (English) LightAutoML vs Titanic: 80% accuracy in several lines of code (Medium)
- (English) Hands-On Python Guide to LightAutoML – An Automatic ML Model Creation Framework (Analytic Indian Mag)

CHAPTER

FIVE

PYTHON-API

5.1 lightautoml.automl

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

AutoML	Class for compile full pipeline of AutoML task.

5.1.1 AutoML

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

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*, *path_to_save=None*)

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*.
- **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.
- **path_to_save** (Optional[str]) – The path that joblib will use to save the model after fit stage is completed. Use *.joblib format.

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.

5.1.2 Presets

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

base.AutoMLPreset	Basic class for automl preset.
tabular_presets.TabularAutoML	Classic preset - work with tabular data.
tabular_presets.TabularUtilizedAutoML	Template to make TimeUtilization from TabularAutoML.
text_presets.TabularNLPAutoML	Classic preset - work with tabular and text data.
whitebox_presets.WhiteBoxPreset	Preset for AutoWoE - logistic regression over binned features (scorecard).

AutoMLPreset

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

Bases: *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 useful case - end-to-end model development.

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')`.

Example

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

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** (Any) – 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*, *path_to_save=None*)

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.
- **path_to_save** (Optional[str]) – The path that joblib will use to save the model after fit stage is completed. Use *.joblib format.

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;

TabularAutoML

class lightautoml.automl.presets.tabular_presets.**TabularAutoML**(*task*, *timeout=3600*, *memory_limit=16*, *cpu_limit=4*, *gpu_ids='all'*, *debug=False*, *timing_params=None*, *config_path=None*, *general_params=None*, *reader_params=None*, *read_csv_params=None*, *nested_cv_params=None*, *tuning_params=None*, *selection_params=None*, *lgb_params=None*, *cb_params=None*, *xgb_params=None*, *rf_params=None*, *linear_l2_params=None*, *nn_params=None*, *gbm_pipeline_params=None*, *linear_pipeline_params=None*, *nn_pipeline_params=None*, *time_series_pipeline_params=None*, *is_time_series=False*)

Bases: *AutoMLPreset*

Classic preset - work with tabular data.

Supported data roles - numbers, dates, categories. Limitations:

- No memory management
- No text support

GPU support in catboost/lightgbm (if installed for GPU) training.

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 `TabularAutoML.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.
- **debug** (bool) – To catch running model exceptions or not.
- **timing_params** (Optional[dict]) – Timing param dict. Optional.
- **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).

- **nested_cv_params** (Optional[dict]) – Param dict for nested cross-validation.
- **tuning_params** (Optional[dict]) – Params of Optuna tuner.
- **selection_params** (Optional[dict]) – Params of feature selection.
- **lgb_params** (Optional[dict]) – Params of lightgbm model.
- **cb_params** (Optional[dict]) – Params of catboost model.
- **xgb_params** (Optional[dict]) – Params of xgboost model.
- **rf_params** (Optional[dict]) – Params of Sklearn Random Forest model.
- **linear_l2_params** (Optional[dict]) – Params of linear model.
- **nn_params** (Optional[dict]) – Params of neural network model.
- **gbm_pipeline_params** (Optional[dict]) – Params of feature generation for boosting models.
- **linear_pipeline_params** (Optional[dict]) – Params of feature generation for linear models.
- **nn_pipeline_params** (Optional[dict]) – Params of feature generation for neural network models.

get_feature_pipeline(*model*, ***kwargs*)

Get LGBSeqSimpleFeatures pipeline if task is the time series prediction.

Parameters

- **model** – one from ["gbm", "linear_l2",, "rf", "nn"].
- **kwargs** – Arbitrary keyword arguments.

Returns

appropriate features pipeline.

create_automl(***fit_args*)

Create basic automl instance.

Parameters

****fit_args** – Contain all information needed for creating automl.

fit_predict(*train_data*, *roles=None*, *train_features=None*, *cv_iter=None*, *valid_data=None*, *valid_features=None*, *log_file=None*, *verbose=0*, *path_to_save=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** (Union[str, ndarray, DataFrame, Dict[str, ndarray], *Batch*]) – Dataset to train.
- **roles** (Optional[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** (Union[str, ndarray, DataFrame, Dict[str, ndarray], *Batch*, None]) – 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;
- **log_file** (Optional[str]) – Filename for writing logging messages. If log_file is specified, the messages will be saved in a the file. If the file exists, it will be overwritten.
- **path_to_save** (Optional[str]) – The path that joblib will use to save the model after fit stage is completed. Use *.joblib format.

Return type
NumpyDataset

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

predict(*data*, *features_names=None*, *batch_size=None*, *n_jobs=1*, *return_all_predictions=None*)

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.

Parallel inference - you can pass `n_jobs` to speedup prediction (requires more RAM). Batch_inference - you can pass `batch_size` to decrease RAM usage (may be longer).

Parameters

- **data** (Union[str, ndarray, DataFrame, Dict[str, ndarray], *Batch*]) – Dataset to perform inference.
- **features_names** (Optional[Sequence[str]]) – Optional features names, if cannot be inferred from *train_data*.
- **batch_size** (Optional[int]) – Batch size or None.
- **n_jobs** (Optional[int]) – Number of jobs.
- **return_all_predictions** (Optional[bool]) – if True, returns all model predictions from last level

Return type
NumpyDataset

Returns
Dataset with predictions.

TabularUtilizedAutoML

class lightautoml.automl.presets.tabular_presets.**TabularUtilizedAutoML**(*task*, *timeout=3600*, *memory_limit=16*, *cpu_limit=4*, *gpu_ids=None*, *timing_params=None*, *configs_list=None*, *drop_last=True*, *return_all_predictions=False*, *max_runs_per_config=5*, *random_state=42*, *outer_blender_max_nonzero_coef=0.05*, ***kwargs*)

Bases: *TimeUtilization*

Template to make TimeUtilization from TabularAutoML.

Simplifies using `TimeUtilization` module for `TabularAutoMLPreset`.

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 params level that are passed to each automl.
- **configs_list** (`Optional`[`Sequence`[`str`]]) – List of str path to configs files.
- **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 blending phase
- **max_runs_per_config** (`int`) – Maximum number of multistart loops.
- **random_state** (`int`) – Initial random seed that will be set in case of search in config.

TabularNLPAutoML

class lightautoml.automl.presets.text_presets.**TabularNLPAutoML**(*task*, *timeout=3600*, *memory_limit=16*, *cpu_limit=4*, *gpu_ids='all'*, *debug=False*, *timing_params=None*, *config_path=None*, *general_params=None*, *reader_params=None*, *read_csv_params=None*, *nested_cv_params=None*, *tuning_params=None*, *selection_params=None*, *nn_params=None*, *lgb_params=None*, *cb_params=None*, *rf_params=None*, *linear_l2_params=None*, *nn_pipeline_params=None*, *gbm_pipeline_params=None*, *linear_pipeline_params=None*, *text_params=None*, *tfidf_params=None*, *autonlp_params=None*)

Bases: *TabularAutoML*

Classic preset - work with tabular and text data.

Supported data roles - numbers, dates, categories, text Limitations - no memory management.

GPU support in catboost/lightgbm (if installed for GPU), NN models training.

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 `TabularNLPAutoML.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.
- **debug** (bool) – To catch running model exceptions or not.
- **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).
- **nested_cv_params** (Optional[dict]) – Param dict for nested cross-validation.
- **tuning_params** (Optional[dict]) – Params of Optuna tuner.

- **selection_params** (Optional[dict]) – Params of feature selection.
- **nn_params** (Optional[dict]) – Params of neural network model.
- **lgb_params** (Optional[dict]) – Params of lightgbm model.
- **cb_params** (Optional[dict]) – Params of catboost model.
- **linear_l2_params** (Optional[dict]) – Params of linear model.
- **nn_pipeline_params** (Optional[dict]) – Params of feature generation for neural network models.
- **gbm_pipeline_params** (Optional[dict]) – Params of feature generation for boosting models.
- **linear_pipeline_params** (Optional[dict]) – Params of feature generation for linear models.
- **text_params** (Optional[dict]) – General params of text features.
- **tfidf_params** (Optional[dict]) – Params of tfidf features.
- **autonlp_params** (Optional[dict]) – Params of text embeddings features.

create_automl(***fit_args*)

Create basic automl instance.

Parameters
****fit_args** – Contain all information needed for creating automl.

predict(*data*, *features_names=None*, *batch_size=None*, *n_jobs=1*)

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.

Parallel inference - you can pass `n_jobs` to speedup prediction (requires more RAM). Batch_inference - you can pass `batch_size` to decrease RAM usage (may be longer).

Parameters

- **data** (Union[str, ndarray, DataFrame, Dict[str, ndarray], *Batch*]) – Dataset to perform inference.
- **features_names** (Optional[Sequence[str]]) – Optional features names, if cannot be inferred from *train_data*.
- **batch_size** (Optional[int]) – Batch size or `None`.
- **n_jobs** (Optional[int]) – Number of jobs.

Return type
NumpyDataset

Returns
Dataset with predictions.

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: *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.

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).

property whitebox

Get wrapped AutoWoE object.

Returns
Model.

infer_auto_params(***kwargs*)

Infer automatic parameters.

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;
- **fit_params** – Others.

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.

create_model_str_desc()

String description of model.

Return type
str

5.1.3 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.

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*, *class_mapping*)

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.
- **class_mapping** (Dict) – Mapping for target classes.

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.

BestModelSelector

class lightautoml.automl.blend.**BestModelSelector**

Bases: *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.

MeanBlender

class lightautoml.automl.blend.**MeanBlender**

Bases: *Blender*

Simple average level predictions.

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

WeightedBlender

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

Bases: *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.

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.

5.2 lightautoml.addons

Extensions of core functionality.

5.2.1 Utilization

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

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 a list of config files as input and runs it until a time limit is 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.

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*, *path_to_save=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*.
- **verbose** (int) – Verbose.
- **log_file** (Optional[str]) – Log filename.
- **path_to_save** (Optional[str]) – The path that joblib will use to save the model after fit stage is completed. Use *.joblib format.

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
- ****kwargs** – Other params.

Return type
LAMLDataset

Returns
Dataset with predictions.

5.2.2 HypEx – Hypothesises and Experiments

The official HypEx documentation can be found at:

HypEx Documentation

For a detailed reference, visit the HypEx API documentation.

5.3 lightautoml.dataset

Provides an internal interface for working with data.

5.3.1 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.

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.

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** (Any) – Special named array of attributes (target, group etc.).

property features

Define how to get features names list.

Returns
Features names.

property data

Get data attribute.

Returns
Any, array like or None.

property roles

Get roles dict.

Returns
Dict of feature roles.

property inverse_roles

Get inverse dict of feature roles.

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.

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: # noqa DAR202
Converted type dataset.

property dataset_type

Get type of dataset.

NumpyDataset

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

Bases: *LAMLDataset*

Dataset that contains info in np.ndarray format.

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** (ndarray) – 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.

property features

Features list.

property roles

Roles 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.

Parameters
dataset (TypeVar(Dataset, bound= *LAMLDataset*)) – Dataset.

Return type
NumpyDataset

Returns
Numpy dataset.

PandasDataset

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

Bases: *LAMLDataset*

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

Parameters

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

`property features`

Get list of features.

Returns
list of features.

`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.

Parameters
dataset (TypeVar(Dataset, bound= *LAMLDataset*)) – Dataset.

Return type
PandasDataset

Returns
Converted to pandas dataset.

`nan_rate()`

Counts overall number of nans in dataset.

Returns
Number of nans.

CSRSparseDataset

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

Bases: *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.

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** (ndarray) – 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.

Parameters
dataset (TypeVar(Dataset, bound= *LAMLDataset*)) – Dataset.

Return type
CSRSparseDataset

Returns
Dataset in sparse form.

5.3.2 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.

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.

Returns
str role name.

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

Create default params role from string.

Parameters

- **name** (str) – Role name.

- **kwargs** (Any) – Other parameters.

Return type
ColumnRole

Returns
Corresponding role object.

NumericRole

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

Bases: *ColumnRole*

Numeric role.

Parameters

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

CategoryRole

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

Bases: *ColumnRole*

Category role.

Parameters

- **dtype** (Union[Callable, type, 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.
- **ordinal** (bool) – Ordinal category.

Note

Valid encoding_type:

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

TextRole

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

Bases: *ColumnRole*

Text role.

Parameters

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

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: *ColumnRole*

Datetime role.

Parameters

- **dtype** (Union[Callable, type, 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.

TargetRole

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

Bases: *ColumnRole*

Target role.

Parameters

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

GroupRole

class lightautoml.dataset.roles.**GroupRole**

Bases: *ColumnRole*

Group role.

DropRole

class lightautoml.dataset.roles.**DropRole**

Bases: *ColumnRole*

Drop role.

WeightsRole

class lightautoml.dataset.roles.**WeightsRole**

Bases: *ColumnRole*

Weights role.

FoldsRole

class lightautoml.dataset.roles.**FoldsRole**

Bases: *ColumnRole*

Folds role.

PathRole

class lightautoml.dataset.roles.**PathRole**

Bases: *ColumnRole*

Path role.

5.3.3 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.

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.

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.

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.

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.

5.4 lightautoml.image

Provides an internal interface for working with image features.

5.4.1 Image Feature Extractors

Image feature extractors based on color histograms and CNN embeddings.

CreateImageFeatures	Class for parallel histogram computation.
TimmModelEmbedder	Class to compute TimmModels embeddings.

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.

TimmModelEmbedder

class lightautoml.image.image.**TimmModelEmbedder**(*model_name='efficientnet_b0.ra_in1k'*, *weights_path=None*, *device=torch.device*)

Bases: Module

Class to compute TimmModels embeddings.

__init__(*model_name='efficientnet_b0.ra_in1k'*, *weights_path=None*, *device=torch.device*)

Pytorch module for image embeddings based on timm models.

Parameters

- **model_name** (str) – Name of effnet model.
- **weights_path** (Optional[str]) – Path to saved weights.
- **device** – Device to use.

get_shape()

Calculate output embedding shape.

Return type
int

Returns
Shape of embedding.

forward(*x*)

Forward pass.

Return type
Tensor

5.4.2 PyTorch Image Datasets

ImageTimmDataset	Image for Timm Dataset Class.
DeepTimmImageEmbedder	Timm Transformer for image embeddings.

ImageTimmDataset

class lightautoml.image.image.**ImageTimmDataset**(*model*, *data*, *loader=<function pil_loader>*)

Bases: object

Image for Timm Dataset Class.

__init__(*model*, *data*, *loader=<function pil_loader>*)

Pytorch Dataset for TimmModelEmbedder.

Parameters

- **model** (*TimmModelEmbedder*) – model which we train.
- **data** (Sequence[str]) – Sequence of paths.
- **loader** (Callable) – Callable for reading image from path.

DeepTimmImageEmbedder

class lightautoml.image.image.**DeepTimmImageEmbedder**(*device=torch.device*, *n_jobs=4*, *random_state=42*, *model_name='efficientnet_b0.ra_in1k'*, *weights_path=None*, *batch_size=128*, *verbose=True*)

Bases: TransformerMixin

Timm Transformer for image embeddings.

__init__(*device=torch.device*, *n_jobs=4*, *random_state=42*, *model_name='efficientnet_b0.ra_in1k'*, *weights_path=None*, *batch_size=128*, *verbose=True*)

Pytorch Dataset for `TimmModelEmbedder`.

Parameters

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

fit(*data=None*)

Train model.

transform(*data*)

Calculate image embeddings from paths.

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

Return type
`ndarray`

Returns
Array of embeddings.

5.4.3 Utils

pil_loader	Load image from paths.

pil_loader

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

Load image from paths.

Parameters
path (`str`) – Image path.

Return type
<module 'PIL.Image' from '/home/docs/checkouts/readthedocs.org/user_builds/lightautoml/envs/latest/lib/python3.10/site-packages/PIL/Image.py'>

Returns
Loaded PIL Image in rgb.

5.5 lightautoml.ml_algo

Models used for machine learning pipelines.

5.5.1 Base Classes

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

MLAlgo

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

Bases: ABC

Abstract class for machine learning algorithm.

Assume that features are already selected, but parameters my be tuned and set before training.

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.
- **timer** (Optional[*TaskTimer*]) – Timer for Algo.

property name

Get model name.

property features

Get list of features.

property is_fitted

Get flag is the model fitted or not.

property params

Get model's params 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.

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

TabularMLAlgo

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

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

Return type
Tuple[Any, ndarray]

Returns: # noqa: DAR202
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*, *CSRSparseDataset*, *PandasDataset*]) – Dataset used for prediction.

Return type
ndarray

Returns: # noqa: DAR202
Predictions for input dataset.

predict(*dataset*)

Mean prediction for all fitted models.

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

Return type
NumpyDataset

Returns
Dataset with predicted values.

5.5.2 Linear Models

LinearLBFGS	LBFGS L2 regression based on torch.
LinearL1CD	Coordinate descent based on sklearn implementation.
TorchModel	Neural net for tabular datasets.

LinearLBFGS

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

Bases: *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 can be rewritten depending on dataset.
- `False`: params can 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*, *CSRSparseDataset*, *PandasDataset*]) – Train Dataset.
- **valid** (Union[*NumpyDataset*, *CSRSparseDataset*, *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*, *CSRSparseDataset*, *PandasDataset*]) – `NumpyDataset` used for prediction.

Return type
ndarray

Returns
Predictions for input dataset.

LinearL1CD

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

Bases: *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*, *CSRSparseDataset*, *PandasDataset*]) – Train Dataset.
- **valid** (Union[*NumpyDataset*, *CSRSparseDataset*, *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*, *CSRSparseDataset*, *PandasDataset*]) – Dataset used for prediction.

Return type
ndarray

Returns
Predictions for input dataset.

TorchModel

`class lightautoml.ml_algo.dl_model.TorchModel`(*default_params=None*, *freeze_defaults=True*, *timer=None*, *optimization_search_space={}*)

Bases: *TabularMLAlgo*

Neural net for tabular datasets.

default_params:

- bs: Batch size.
- num_workers: Number of threads for multiprocessing.
- max_length: Max sequence length.
- opt_params: Dict with optim params.
- scheduler_params: Dict with scheduler params.
- is_snap: Use snapshots.
- snap_params: Dict with SE parameters.
- init_bias: Init last linear bias by mean target values.
- n_epochs: Number of training epochs.
- input_bn: Use 1d batch norm for input data.
- emb_dropout: Dropout probability.

- emb_ratio: Ratio for embedding size = (x + 1) // emb_ratio.
- max_emb_size: Max embedding size.
- device: Torch device or str.
- use_cont: Use numeric data.
- use_cat: Use category data.
- use_text: Use text data.
- lang: Text language.
- bert_name: Name of HuggingFace transformer model.
- pooling: Type of pooling strategy for bert model.
- deterministic: CUDNN backend.
- multigpu: Use Data Parallel.
- model_with_emb: Use model with custom embeddings.
- loss: Torch loss or str or func with (y_pred, y_true) args.
- loss_params: Dict with loss params.
- loss_on_logits: Calculate loss on logits or on predictions of model for classification tasks.
- clip_grad: Clip gradient before loss backprop.
- clip_grad_params: Dict with clip_grad params.
- dataset: Class for data retrieval
- tuned: Tune custom model
- num_init_features: Scale input dimension to another one
- use_noise: Use Noise
- use_bn: Use BatchNorm
- path_to_save: Path to save model checkpoints, `None` - stay in memory.
- random_state: Random state to take subsample.
- verbose_inside: Number of steps between verbose inside epoch or `None`.
- verbose: Verbose every N epochs.

freeze_defaults:

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

timer: *Timer* instance or `None`.

static get_mean_target(*target*, *task_name*)

Get target mean / inverse sigmoid transformation to init bias in last layer of network.

Parameters

- **target** – Target values.
- **task_name** (str) – One of the available task names

Returns

Array with bias values.

init_params_on_input(*train_valid_iterator*)

Get model parameters depending on dataset parameters.

Parameters
train_valid_iterator – Classic cv-iterator.

Return type
dict

Returns
Parameters of model.

get_dataloaders_from_dicts(*data_dict*)

Construct dataloaders depending on stage.

Parameters
data_dict (Dict) – Dict with (stage_name, data) (key, value).

Returns
Dataloaders.

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.

fit_predict_single_fold(*train*, *valid*)

Implements training and prediction on single fold.

Parameters

- **train** – Train Dataset.
- **valid** – Validation Dataset.

Returns
Tuple (model, predicted_values).

predict_single_fold(*model*, *dataset*)

Predict target values for dataset.

Parameters

- **model** (any) – Neural net object or dict or str.
- **dataset** (Union[*NumpyDataset*, *CSRSparseDataset*, *PandasDataset*]) – Test dataset.

Return type
ndarray

Returns
Predicted target values.

5.5.3 Boosted Trees

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

BoostLGBM

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

Bases: *TabularMLAlgo*, *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*, *CSRSparseDataset*, *PandasDataset*]) – Train Dataset.
- **valid** (Union[*NumpyDataset*, *CSRSparseDataset*, *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*, *CSRSparseDataset*, *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.

BoostCB

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

Bases: *TabularMLAlgo*, *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*, *CSRSparseDataset*, *PandasDataset*]) – Train Dataset.

- **valid** (Union[*NumpyDataset*, *CSRSparseDataset*, *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*, *CSRSparseDataset*, *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.

5.5.4 Neural Networks

MLP	Realisation of *'mlp'* model.
DenseLightModel	Realisation of *'denselight'* model.
DenseModel	Realisation of *'dense'* model.
ResNetModel	The ResNet model from https://github.com/Yura52/rtdl.
SNN	Realisation of *'snn'* model.

MLP

class lightautoml.ml_algo.torch_based.nn_models.**MLP**(**args*, ***kwargs*)

Bases: *DenseLightModel*

Realisation of *'mlp'* model.

Parameters

- **n_in** – Input dim.
- **n_out** – Output dim.
- **hidden_size** – List of hidden dims.
- **drop_rate** – Dropout rate for each layer separately or altogether.
- **act_fun** – Activation function.
- **noise_std** – Std of noise.
- **num_init_features** – If not none add fc layer before model with certain dim.
- **use_bn** – Use BatchNorm.
- **use_noise** – Use noise.
- **dropout_first** – Use dropout in the first layer or not.
- **bn_momentum** – BatchNorm momentum
- **ghost_batch** – If not none use GhoastNorm with ghost_batch.
- **leaky_gate** – Use LeakyGate or not.
- **use_skip** – Use another Linear model to blend them after.
- **weighted_sum** – Use weighted blender or half-half.
- **device** – Device to compute on.

DenseLightModel

class lightautoml.ml_algo.torch_based.nn_models.**DenseLightModel**(*n_in*, *n_out=1*, *hidden_size=[512, 750]*, *drop_rate=0.1*, *act_fun=torch.nn.LeakyReLU*, *noise_std=0.05*, *num_init_features=None*, *use_bn=True*, *use_noise=False*, *concat_input=True*, *dropout_first=True*, *bn_momentum=0.1*, *ghost_batch=None*, *use_skip=False*, *leaky_gate=False*, *weighted_sum=True*, *device=torch.device*, ***kwargs*)

Bases: `Module`

Realisation of *'denselight'* model.

Parameters

- **n_in** (`int`) – Input dim.
- **n_out** (`int`) – Output dim.
- **hidden_size** (`List[int]`) – List of hidden dims.
- **drop_rate** (`Union[float, List[float]]`) – Dropout rate for each layer separately or altogether.
- **act_fun** (`Module`) – Activation function.

- **noise_std** (float) – Std of noise.
- **num_init_features** (Optional[int]) – If not none add fc layer before model with certain dim.
- **use_bn** (bool) – Use BatchNorm.
- **use_noise** (bool) – Use noise.
- **concat_input** (bool) – Concatenate input to all hidden layers. # MLP False
- **dropout_first** (bool) – Use dropout in the first layer or not.
- **bn_momentum** (float) – BatchNorm momentum
- **ghost_batch** (Optional[int]) – If not none use GhoastNorm with ghost_batch.
- **leaky_gate** (bool) – Use LeakyGate or not.
- **use_skip** (bool) – Use another Linear model to blend them after.
- **weighted_sum** (bool) – Use weighted blender or half-half.
- **device** (device) – Device to compute on.

forward(*X*)

Forward-pass.

Return type
Tensor

DenseModel

class lightautoml.ml_algo.torch_based.nn_models.**DenseModel**(*n_in*, *n_out=1*, *block_config=[2, 2]*, *drop_rate=0.1*, *num_init_features=None*, *compression=0.5*, *growth_size=256*, *bn_factor=2*, *act_fun=torch.nn.ReLU*, *use_bn=True*, ***kwargs*)

Bases: Module

Realisation of *'dense'* model.

Parameters

- **n_in** (int) – Input dim.
- **n_out** (int) – Output dim.
- **block_config** (List[int]) – List of number of layers within each block
- **drop_rate** (Union[float, List[float]]) – Dropout rate for each layer separately or altogether.
- **num_init_features** (Optional[int]) – If not none add fc layer before model with certain dim.
- **compression** (float) – portion of neuron to drop after block.
- **growth_size** (int) – Output dim of every layer.
- **bn_factor** (float) – Dim of intermediate fc is increased times *bn_factor* in DenseModel layer.
- **act_fun** (Module) – Activation function.

- **use_bn** (bool) – Use BatchNorm.

forward(*x*)

Forward-pass.

Return type
Tensor

ResNetModel

class lightautoml.ml_algo.torch_based.nn_models.**ResNetModel**(*n_in*, *n_out=1*, *hid_factor=[2, 2]*, *drop_rate=0.1*, *noise_std=0.05*, *act_fun=torch.nn.ReLU*, *num_init_features=None*, *use_bn=True*, *use_noise=False*, *device=torch.device*, ***kwargs*)

Bases: Module

The ResNet model from https://github.com/Yura52/rtdl.

Parameters

- **n_in** (int) – Input dim.
- **n_out** (int) – Output dim.
- **hid_factor** (List[float]) – Dim of intermediate fc is increased times this factor in ResnetModel layer.
- **drop_rate** (Union[float, List[float], List[List[float]]]) – Dropout rate for each layer separately or altogether.
- **noise_std** (float) – Std of noise.
- **act_fun** (Module) – Activation function.
- **num_init_features** (Optional[int]) – If not none add fc layer before model with certain dim.
- **use_bn** (bool) – Use BatchNorm.
- **use_noise** (bool) – Use noise.
- **device** (device) – Device to compute on.

forward(*x*)

Forward-pass.

Return type
Tensor

SNN

class lightautoml.ml_algo.torch_based.nn_models.**SNN**(*n_in*, *n_out*, *hidden_size=[512, 512, 512]*, *num_init_features=None*, *drop_rate=0.1*, ***kwargs*)

Bases: Module

Realisation of *'snn'* model.

Parameters

- **n_in** (int) – Input dim.

- **n_out** (int) – Output dim.
- **hidden_size** (List[int]) – List of hidden dims.
- **drop_rate** (Union[float, List[float]]) – Dropout rate for each layer separately or altogether.
- **num_init_features** (Optional[int]) – If not none add fc layer before model with certain dim.

forward(*x*)

Forward-pass.

Return type
Tensor

reset_parameters()

Init weights.

5.5.5 WhiteBox

WbMLAlgo	WhiteBox - scorecard model.

WbMLAlgo

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

Bases: *TabularMLAlgo*

WhiteBox - scorecard model.

https://github.com/AILab-MLTools/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.
- **metric_th: .5 < metric_th < 1**
  Threshold for feature selection by one-dimensional AUC. WoE with AUC < metric_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.
- **metric_tol: 1e-5 <= metric_tol <=1e-2**
  Metric tolerance. You can lower the metric_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 can be rewritten depending on dataset.
- `False`: params can 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.

5.6 lightautoml.ml_algo.tuning

Bunch of classes for hyperparameters tuning.

5.6.1 Base Classes

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

ParamsTuner

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

Bases: ABC

Base abstract class for hyperparameters tuners.

property best_params

Get best params.

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.

DefaultTuner

class lightautoml.ml_algo.tuning.base.**DefaultTuner**

Bases: *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.

Return type
Tuple[None, None]

Returns
Tuple (None, None).

5.6.2 Tuning with Optuna

OptunaTuner	Wrapper for optuna tuner.

OptunaTuner

class lightautoml.ml_algo.tuning.optuna.**OptunaTuner**(*timeout=1000*, *n_trials=100*, *direction='maximize'*, *fit_on_holdout=True*, *random_state=42*, *fail_tolerance=0.5*)

Bases: *ParamsTuner*

Wrapper for optuna tuner.

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.
- **fail_tolerance** (float) – the maximum allowed percentage of failed tuner trials. Exception will be thrown after crossing the threshold value.

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

Tune model.

Parameters

- **ml_algo** (TypeVar(TunableAlgo, bound= *MLAlgo*)) – Algo that is tuned.
- **train_valid_iterator** (Optional[*TrainValidIterator*]) – Classic cv-iterator.

Return type
Tuple[Optional[TypeVar(TunableAlgo, bound= *MLAlgo*)], 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.

5.7 lightautoml.ml_algo

Torch utils.

5.7.1 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.

SequenceAbstractPooler

class lightautoml.ml_algo.torch_based.nn_models.**SequenceAbstractPooler**(**args*, ***kwargs*)
Bases: Module

Abstract pooling class.

forward(*x*, *x_mask*)
Forward-pass.

Return type
Tensor

SequenceClsPooler

class lightautoml.ml_algo.torch_based.nn_models.**SequenceClsPooler**(**args*, ***kwargs*)
Bases: *SequenceAbstractPooler*

CLS token pooling.

forward(*x*, *x_mask*)
Forward-pass.

Return type
Tensor

SequenceMaxPooler

class lightautoml.ml_algo.torch_based.nn_models.**SequenceMaxPooler**(**args*, ***kwargs*)
Bases: *SequenceAbstractPooler*

Max value pooling.

forward(*x*, *x_mask*)

Forward-pass.

Return type
Tensor

SequenceSumPooler

class lightautoml.ml_algo.torch_based.nn_models.**SequenceSumPooler**(**args*, ***kwargs*)

Bases: *SequenceAbstractPooler*

Sum value pooling.

forward(*x*, *x_mask*)

Forward-pass.

Return type
Tensor

SequenceAvgPooler

class lightautoml.ml_algo.torch_based.nn_models.**SequenceAvgPooler**(**args*, ***kwargs*)

Bases: *SequenceAbstractPooler*

Mean value pooling.

forward(*x*, *x_mask*)

Forward-pass.

Return type
Tensor

SequenceIndentityPooler

class lightautoml.ml_algo.torch_based.nn_models.**SequenceIndentityPooler**(**args*, ***kwargs*)

Bases: *SequenceAbstractPooler*

Identity pooling.

forward(*x*, *x_mask*)

Forward-pass.

Return type
Tensor

5.8 lightautoml.pipelines

Pipelines for solving different tasks.

5.8.1 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.

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.

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** (Any) – Specific parameters values to search. Example: search for categories with OHE processing only.

Return type
List[str]

Returns
List of str features names.

5.9 lightautoml.pipelines.selection

Feature selection module for ML pipelines.

5.9.1 Base Classes

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

ImportanceEstimator

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

Bases: object

Abstract class, that estimates feature importances.

fit(**args*, ***kwargs*)

Calculate feature importance.

`get_features_score()`

Get raw features importances.

Return type
Series

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

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.

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** (Any) – Not used.

`property is_fitted`

Check if selection pipeline is already fitted.

Returns
True for fitted pipeline and False for not fitted.

`property selected_features`

Get selected features.

Returns
List of selected feature names.

`property in_features`

Input features to the selector.

Raises exception if not fitted beforehand.

Returns
List of input features.

`property dropped_features`

Features that were dropped.

Returns
list of dropped features.

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.

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

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.

noqa: DAR201

get_features_score()

Get input feature importances.

Returns
Series with importances in not ascending order.

5.9.2 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.

ModelBasedImportanceEstimator

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

Bases: *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[TypeVar(ImportanceEstimatedAlgo, bound=*ImportanceEstimator*)]) – ML algorithm used for importance estimation.
- **preds** (Optional[*LAMLDataset*]) – predicted target values.

ImportanceCutoffSelector

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

Bases: *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.

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.

NpPermutationImportanceEstimator

class lightautoml.pipelines.selection.permutation_importance_based.**NpPermutationImportanceEstimator**(*rand*

Bases: *ImportanceEstimator*

Permutation importance based estimator.

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

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.

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: *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.

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.

5.9.3 Other Selectors

HighCorrRemoval	Selector to remove highly correlated features.

HighCorrRemoval

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

Bases: *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.

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** – Additional parameters. Used for initialization 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.

5.10 lightautoml.pipelines.features

Pipelines for features generation.

5.10.1 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.

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.

property output_features

List of feature names that produces _pipeline.

property used_features

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

create_pipeline(*train*)

Analyse dataset and create composite transformer.

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

Return type
LAMLTransformer

Returns: # noqa DAR202
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.

EmptyFeaturePipeline

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

Bases: *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.

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** (Any) – 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 handle. 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*, *mode*, *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.
- **mode** (str) – What feature generation mode is used. Can be “cat_intersections” or “groupby”.
- **top_n** (int) – Number of top categories.

Return type
List[str]

Returns
List.

get_top_numeric(*train*, *top_n=5*)

Get top numeric features 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 numeric features.

Return type
List[str]

Returns
List.

get_groupby(*train*)

Get transformer that calculates group by features.

Amount of features is limited to `self.top_group_by_categorical` and `self.top_group_by_numerical` fields.

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

Return type
Optional[*LAMLTransformer*]

Returns
Transformer.

5.10.2 Feature Pipelines for Boosting Models

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

LGBSimpleFeatures

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

Bases: *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.

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*, *fill_na=False*, *ascending_by_cardinality=False*, *use_groupby=False*, *groupby_types=['delta_median', 'delta_mean', 'min', 'max', 'std', 'mode', 'is_mode']*, *groupby_triplets=[]*, *groupby_top_based_on='cardinality'*, *groupby_top_categorical=3*, *groupby_top_numerical=3*, ***kwargs*)

Bases: *FeaturesPipeline*, *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.

Parameters

- **feats_imp** (Union[*ImportanceEstimator*, *SelectionPipeline*, None]) – 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.

5.10.3 Feature Pipelines for Linear Models

LinearFeatures	Creates pipeline for linear models and nnets.

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*, *use_groupby=False*, *groupby_types=['delta_median', 'delta_mean', 'min', 'max', 'std', 'mode', 'is_mode']*, *groupby_triplets=[]*, *groupby_top_based_on='cardinality'*, *groupby_top_categorical=3*, *groupby_top_numerical=3*, ***kwargs*)

Bases: *FeaturesPipeline*, *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.
- Standardization and nan handling for numbers.
- Numbers discretization if needed.
- Dates handling.
- Handling probs (output of lower level models).

Parameters

- **feats_imp** (Union[*ImportanceEstimator*, *SelectionPipeline*, None]) – 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.

5.10.4 Feature Pipelines for WhiteBox

WBFeatures	Simple WhiteBox pipeline.

WBFeatures

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

Bases: *FeaturesPipeline*, *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.

5.10.5 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.

ImageDataFeatures

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

Bases: object

Class contains basic features transformations for image data.

Parameters
****kwargs** (Any) – Default parameters.

ImageSimpleFeatures

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

Bases: *FeaturesPipeline*, *ImageDataFeatures*

Class contains simple color histogram features for image data.

create_pipeline(*train*)

Create pipeline for images data.

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

Return type
LAMLTransformer

Returns
Transformer.

ImageAutoFeatures

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

Bases: *FeaturesPipeline*, *ImageDataFeatures*

Class contains efficient-net embeddings features for image data.

create_pipeline(*train*)

Create pipeline for images data.

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

Return type
LAMLTransformer

Returns
Transformer.

5.10.6 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.

NLPDataFeatures

class lightautoml.pipelines.features.text_pipeline.**NLPDataFeatures**(***kwargs*)

Bases: object

Class contains basic features transformations for text data.

TextAutoFeatures

class lightautoml.pipelines.features.text_pipeline.**TextAutoFeatures**(***kwargs*)

Bases: *FeaturesPipeline*, *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.

NLPTFiDFFeatures

class lightautoml.pipelines.features.text_pipeline.**NLPTFiDFFeatures**(***kwargs*)

Bases: *FeaturesPipeline*, *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.

TextBertFeatures

class lightautoml.pipelines.features.text_pipeline.**TextBertFeatures**(***kwargs*)

Bases: *FeaturesPipeline*, *NLPDataFeatures*

Features pipeline for BERT.

create_pipeline(*train*)

Create pipeline for BERT.

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

Return type
LAMLTransformer

Returns
Transformer.

5.10.7 Feature Pipelines for Neural Networks Models

TorchSimpleFeatures	Creates simple pipeline for neural network models.

TorchSimpleFeatures

class lightautoml.pipelines.features.torch_pipeline.**TorchSimpleFeatures**(*use_te=False*, *top_intersections=5*, *max_bin_count=10*, *max_intersection_depth=3*, *te_subsample=None*, *sparse_ohe='auto'*, *auto_unique_co=50*, *output_categories=True*, *multiclass_te_co=3*, *use_qnt=True*, *n_quantiles=None*, *subsample=1000000000*, *output_distribution='normal'*, *noise=0.001*, *qnt_factor=30*, ***kwargs*)

Bases: *FeaturesPipeline*, *TabularDataFeatures*

Creates simple pipeline for neural network models.

__init__(*use_te=False*, *top_intersections=5*, *max_bin_count=10*, *max_intersection_depth=3*, *te_subsample=None*, *sparse_ohe='auto'*, *auto_unique_co=50*, *output_categories=True*, *multiclass_te_co=3*, *use_qnt=True*, *n_quantiles=None*, *subsample=1000000000*, *output_distribution='normal'*, *noise=0.001*, *qnt_factor=30*, ***kwargs*)

TorchSimpleFeatures.

Parameters

- **use_qnt** (bool) – Use quantile transformer for numerical columns.
- **n_quantiles** (Optional[int]) – Number of quantiles to be computed.
- **subsample** (int) – Maximum number of samples used to estimate the quantiles for computational efficiency.
- **output_distribution** (str) – Marginal distribution for the transformed data. The choices are ‘uniform’ or ‘normal’.
- **noise** (float) – Add noise with certain std to dataset before quantile transformation to make data more smooth.
- **qnt_factor** (int) – If number of quantiles is none then it equals dataset size / factor
- **use_te** (bool) – Use target encoding for categorical columns.
- **top_intersections** (int) – Max number of categories to generate intersections.

- **max_bin_count** (int) – Max number of bins for cat columns.
- **max_intersection_depth** (int) – Max depth of cat intersection.
- **te_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.
- **kwargs** – Other params.

create_pipeline(*train*)

Create tree pipeline.

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

Return type
LAMLTransformer

Returns
Composite datetime, categorical, numeric transformer.

5.11 lightautoml.pipelines.ml

Pipelines that merge together single model training steps.

5.11.1 Base Classes

MLPipeline	Single ML pipeline.

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.

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.

5.11.2 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.

NestedTabularMLAlgo

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

Bases: *TabularMLAlgo*, *ImportanceEstimator*

Wrapper for MLAlgo to make it trainable over nested folds.

Limitations - only for `TabularMLAlgo`.

property params

Parameters of ml_algo.

init_params_on_input(*train_valid_iterator*)

Init params depending on input data.

Parameters
train_valid_iterator (*TrainValidIterator*) – Iterator over 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*, *CSRSparseDataset*, *PandasDataset*]) – Tabular-Dataset to train.
- **valid** (Union[*NumpyDataset*, *CSRSparseDataset*, *PandasDataset*]) – Tabular-Dataset to validate.

Return type
Tuple[Any, ndarray]

Returns
Tuple (model, predicted_values).

predict_single_fold(*model*, *dataset*)

Model prediction on a dataset.

Parameters

- **model** (Any) – Model.
- **dataset** (Union[*NumpyDataset*, *CSRSparseDataset*, *PandasDataset*]) – Dataset.

Return type
ndarray

Returns
Predictions.

get_features_score()

Score of each features.

Return type
Series

fit(*train_valid*)

Just to be compatible with `ImportanceEstimator`.

Parameters

train_valid (*TrainValidIterator*) – Classic cv iterator.

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: *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.

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`.

5.11.3 Pipeline for WhiteBox

WBPipeline	Special pipeline to handle WhiteBox model.

WBPipeline

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

Bases: *MLPipeline*

Special pipeline to handle WhiteBox model.

property whitebox

Return first 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.

5.12 lightautoml.reader

Utils for reading, training and analysing data.

5.12.1 Readers

Reader	Abstract Reader class.
PandasToPandasReader	Pandas Reader.

Reader

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

Bases: object

Abstract Reader class.

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 available size and returns dataset.

Parameters

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

property **roles**

Roles dict.

property **dropped_features**

List of dropped features.

property **used_features**

List of used features.

property **used_array_attrs**

Dict of used array attributes.

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.

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: *Reader*

Pandas 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.

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_unique_rate** (float) – 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** (Any) – 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, TypeVar(RoleType, bound= *ColumnRole*), None], Sequence[str]]]) – Dict of features roles in format `{RoleX: ['feat0', 'feat1', ...], RoleY: 'TARGET', ....}`.
- ****kwargs** (Any) – Can be used for target/group/weights.

Return type
PandasDataset

Returns
Dataset with selected features.

check_class_target(*target*)

Validate target values.

Return type
Tuple[Series, Union[Mapping, Dict[str, Mapping], None]]

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, TypeVar(RoleType, bound= *ColumnRole*)]]) – Dict of user defined roles.

Return type
Dict[str, TypeVar(RoleType, bound= *ColumnRole*)]

Returns
Dict.

5.12.2 Tabular Batch Generators

Batch Handler Classes

Batch	Class to wraps batch of data in different formats.
FileBatch	Batch of csv file.
BatchGenerator	Abstract - generator of batches from data.
DfBatchGenerator	Batch generator from DataFrame.
FileBatchGenerator	Generator of batches from file.

Batch

class lightautoml.reader.tabular_batch_generator.**Batch**(*data*)

Bases: object

Class to wraps batch of data in different formats.

Default - batch of DataFrame.

property data

Get data from Batch object.

Returns
Data.

FileBatch

class lightautoml.reader.tabular_batch_generator.**FileBatch**(*file*, *offset*, *cnt*, *read_csv_params*)

Bases: *Batch*

Batch of csv file.

Parameters

- **file** – File path.
- **offset** – File start.
- **cnt** – Number of rows to read.
- **read_csv_params** – Additional params to pandas.read_csv.

property data

Get data from Batch object.

Returns
Read data.

BatchGenerator

class lightautoml.reader.tabular_batch_generator.**BatchGenerator**(*batch_size*, *n_jobs*)

Bases: object

Abstract - generator of batches from data.

Parameters

- **batch_size** – Batch size. Default is None, split by *n_jobs*.
- **n_jobs** – Number of processes to handle.

DfBatchGenerator

class lightautoml.reader.tabular_batch_generator.**DfBatchGenerator**(*data*, *n_jobs=1*, *batch_size=None*)

Bases: *BatchGenerator*

Batch generator from DataFrame.

Parameters

- **data** (DataFrame) – Data used for generator.
- **n_jobs** (int) – Number of processes to handle.
- **batch_size** (Optional[int]) – Batch size. Default is None, split by *n_jobs*.

FileBatchGenerator

class lightautoml.reader.tabular_batch_generator.**FileBatchGenerator**(*file*, *n_jobs=1*, *batch_size=None*, *read_csv_params=None*)

Bases: *BatchGenerator*

Generator of batches from file.

Parameters

- **file** – File path.
- **n_jobs** (int) – Number of processes to handle.
- **batch_size** (Optional[int]) – Batch size. Default is None, split by *n_jobs*.
- **read_csv_params** (Optional[dict]) – Params of reading csv file. Look for pandas.read_csv params.

Data Read Functions

read_batch	Read data for inference by batches for simple tabular data.
read_data	Get DataFrame from different data formats.

read_batch

lightautoml.reader.tabular_batch_generator.**read_batch**(*data*, *features_names=None*, *n_jobs=1*, *batch_size=None*, *read_csv_params=None*)

Read data for inference by batches for simple tabular data.

> **Note**
>
> Supported now data formats:
>
> - 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** (Union[str, ndarray, DataFrame, Dict[str, ndarray], *Batch*]) – Readable to DataFrame data.
- **features_names** (Optional[Sequence[str]]) – Optional features names if `numpy.ndarray`.
- **n_jobs** (int) – Number of processes to read file.
- **batch_size** (Optional[int]) – Batch size.
- **read_csv_params** (Optional[dict]) – Params to read csv file.

Return type
Iterable[*BatchGenerator*]

Returns
Generator of batches.

read_data

lightautoml.reader.tabular_batch_generator.**read_data**(*data*, *features_names=None*, *n_jobs=1*, *read_csv_params=None*)

Get `DataFrame` from different data formats.

> **Note**
>
> Supported now data formats:
>
> - 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** (Union[str, ndarray, DataFrame, Dict[str, ndarray], *Batch*]) – Readable to DataFrame data.

- **features_names** (Optional[Sequence[str]]) – Optional features names if numpy.ndarray.
- **n_jobs** (int) – Number of processes to read file.
- **read_csv_params** (Optional[dict]) – Params to read csv file.

Return type
Tuple[DataFrame, Optional[dict]]

Returns
Tuple with read data and new roles mapping.

5.13 lightautoml.report

Report generators and templates.

ReportDeco	Decorator to wrap *AutoML* class to generate html report on fit_predict and predict.
ReportDecoWhitebox	Special report wrapper for *WhiteBoxPreset*.

5.13.1 ReportDeco

class lightautoml.report.report_deco.**ReportDeco**(**args*, ***kwargs*)

Bases: object

Decorator to wrap *AutoML* class to generate html report on fit_predict and predict.

Example

```
>>> report_automl = ReportDeco(output_path="output_path", report_file_name=
↪"report_file_name")(automl).
>>> report_automl.fit_predict(train_data)
>>> report_automl.predict(test_data)
```

Report will be generated at output_path/report_file_name automatically.

Warning

Do not use it just to inference (if you don"t need report), because:

- It needs target variable to calc performance metrics.
- It takes additional time to generate report.
- Dump of decorated automl takes more memory to store.

To get unwrapped fitted instance to pickle and inferecne access report_automl.model attribute.

property model

Get unwrapped model.

Returns
model.

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

> **Note**
>
> Valid kwargs are:
>
> - output_path: Folder with report files.
> - report_file_name: Name of main report file.

Parameters

- ***args** – Arguments.
- ****kwargs** – Additional parameters.

fit_predict(**args*, ***kwargs*)

Wrapped `automl.fit_predict` method.

Valid args, kwargs are the same as wrapped automl.

Parameters

- ***args** – Arguments.
- ****kwargs** – Additional parameters.

Returns

OOF predictions.

predict(**args*, ***kwargs*)

Wrapped automl.predict method.

Valid args, kwargs are the same as wrapped automl.

Parameters

- ***args** – arguments.
- ****kwargs** – additional parameters.

Returns

predictions.

5.13.2 ReportDecoWhitebox

class lightautoml.report.report_deco.**ReportDecoWhitebox**(***kwargs*)

Bases: *ReportDeco*

Special report wrapper for *WhiteBoxPreset*. Usage case is the same as main *ReportDeco* class. It generates same report as *ReportDeco* , but with additional whitebox report part.

Difference:

- report_automl.predict gets additional report argument. It stands for updating whitebox report part. Calling `report_automl.predict(test_data, report=True)` will update test part of whitebox report. Calling `report_automl.predict(test_data, report=False)` will extend general report with. New data and keeps whitebox part as is (much more faster).
- *WhiteBoxPreset* should be created with parameter `general_params={"report": True}` to get white box report part. If `general_params` set to `{"report": False}`, only standard ReportDeco part will be created (much faster).

property model

Get unwrapped WhiteBox.

Returns
model.

fit_predict(**args*, ***kwargs*)

Wrapped `AutoML.fit_predict` method.

Valid args, kwargs are the same as wrapped automl.

Parameters

- ***args** – Arguments.
- ****kwargs** – Additional parameters.

Returns
OOF predictions.

predict(**args*, ***kwargs*)

Wrapped `AutoML.predict` method.

Valid args, kwargs are the same as wrapped automl.

Parameters

- ***args** – Arguments.
- ****kwargs** – Additional parameters.

Returns
Predictions.

5.14 lightautoml.tasks

5.14.1 Task Class

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

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.

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.

Available losses for binary task:

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

Available 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, required params:**
  a - threshold between MAE and MSE losses.
- ***'fair'* - Fair loss, required params:**
  c - sets smoothness.
- ***'quantile'* - Quantile loss, required params:**
  q - sets quantile.

Available losses for multi-classification task:

- *'crossentropy'* - (uses by default) Standard crossentropy function.
- ***'f1'* - Optimizes F1-Macro Score, now available 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.

Available 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, required params:**
  a - threshold between MAE and MSE losses.
- ***'fair'* - Fair loss, required params:**
  c - sets smoothness.

- *'quantile'* **- Quantile loss, required params:**
  `q` - sets quantile.

Available 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')
```

property name

Name of task.

get_dataset_metric()

Create metric for dataset.

Get metric that is called on dataset.

Return type
`LAMLMetric`

Returns
Metric in scikit-learn compatible format.

5.14.2 Common Metrics

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.

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.

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.

Parameters

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

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

5.15 lightautoml.tasks.losses

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

5.15.1 Base Classes

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

MetricFunc

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

Bases: object

Wrapper for metric.

Parameters

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

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'*

5.15.2 Wrappers for LightGBM

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: *Loss*

Loss used for LightGBM.

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'*

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.

5.15.3 Wrappers for CatBoost

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: *Loss*

Loss used for CatBoost.

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.

Parameters

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

evaluate(*approxes*, *target*, *weight*)

Calculate metric.

CBRegressionMetric

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

Bases: *CBCustomMetric*

Regression metric wrapper for CatBoost.

evaluate(*approxes*, *target*, *weight*)

Calculate metric.

CBClassificationMetric

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

Bases: *CBCustomMetric*

Classification metric wrapper for CatBoost.

evaluate(*approxes*, *target*, *weight*)

Calculate metric.

CBMulticlassMetric

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

Bases: *CBCustomMetric*

Multiclassification metric wrapper for CatBoost.

evaluate(*approxes*, *target*, *weight*)

Calculate metric.

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. # noqa D403

Parameters

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

Return type

str

Returns

Wrapped CatBoost loss name.

5.15.4 Wrappers for Sklearn

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: *Loss*

Loss used for scikit-learn.

Parameters

- **loss** (str) – One of default loss function. Valid are: ‘logloss’, ‘mse’, ‘crossentropy’, ‘rmsle’.
- **loss_params** (Optional[Dict]) – Additional 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.

5.15.5 Wrappers for Torch

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: Module

Customize PyTorch-based loss.

Parameters

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

Returns

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

forward(*y_true*, *y_pred*, *sample_weight=None*)

Forward-pass.

TORCHLoss

class lightautoml.tasks.losses.torch.**TORCHLoss**(*loss*, *loss_params=None*)

Bases: *Loss*

Loss used for PyTorch.

Parameters

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

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.

5.16 lightautoml.text

Provides an internal interface for working with text features.

5.16.1 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.

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: TransformerMixin

Deep Learning based sentence embeddings.

Parameters

- **model** – Torch model for aggregation word embeddings into sentence embedding.
- **model_params** (Dict) – Dict with model parameters.
- **dataset** – Torch dataset.
- **dataset_params** (Dict) – Dict with dataset params.
- **loader_params** (Dict) – Dict with params for torch dataloader.
- **device** (str) – String with torch device type or device ids. I.e: '0,2'.
- **random_state** (int) – Determines random number generation.
- **embedding_model** (Optional[Any]) – Torch word embedding model, if dataset do not return embeddings.
- **embedding_model_params** (Optional[Dict[str, Dict]]) – Dict with embedding model params.
- **multigpu** (bool) – Use data parallel for multiple GPU.
- **verbose** (bool) – 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.

transform(*data*)

Embedded sentence.

Return type
ndarray

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: Module

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

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** (Any) – 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.

forward(*inp*)

Forward-pass.

Return type
Tensor

RandomLSTM

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

Bases: Module

Class to compute Random LSTM sentence embeddings from words 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** (Any) – 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.

`forward(inp)`

Forward-pass.

Return type
`Tensor`

BertEmbedder

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

Bases: `Module`

Class to compute HuggingFace transformers words or sentence embeddings.

Bert sentence or word embeddings.

Parameters

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

`forward(inp)`

Forward-pass.

Return type
`Tensor`

`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.

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: TransformerMixin

Weighted average of word embeddings.

Calculate sentence embedding as weighted average of word embeddings.

Parameters

- **embedding_model** (Dict) – word2vec, fasstext, 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) – Subtract projection onto first singular vector.
- **alpha** (int) – Param for sif weights.
- **verbose** (bool) – Add prints.
- ****kwargs** (Any) – 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.

5.16.2 Torch Datasets for Text

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

BertDataset

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

Bases: object

Dataset class with transformers tokenization.

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.
- ****kwargs** (Any) – Other.

EmbedDataset

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

Bases: object

Dataset class for extracting word embeddings.

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.

5.16.3 Tokenizers

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

BaseTokenizer

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

Bases: object

Base class for tokenizer method.

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.
- ****kwargs** (Any) – Ignore.

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.

SimpleRuTokenizer

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

Bases: *BaseTokenizer*

Russian tokenizer.

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.
- **kwargs** (Any) – Ignore.

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.

SimpleEnTokenizer

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

Bases: *BaseTokenizer*

English tokenizer.

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.
- **kwargs** (Any) – Ignore.

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.

5.16.4 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.

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.

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.

custom_collate

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

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

Return type
Tensor

single_text_hash

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

Get text hash.

Parameters
x (str) – Text.

Return type
str

Returns
String text hash.

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.

5.17 lightautoml.transformers

Basic feature generation steps and helper utils.

5.17.1 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).

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.

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.

SequentialTransformer

class lightautoml.transformers.base.**SequentialTransformer**(*transformer_list*)

Bases: *LAMLTransformer*

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

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.

UnionTransformer

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

Bases: *LAMLTransformer*

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

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.

ColumnsSelector

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

Bases: *LAMLTransformer*

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

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 (*LAMLDataset*) – Dataset to transform.

Return type
LAMLDataset

Returns
Dataset with selected features.

ColumnwiseUnion

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

Bases: *UnionTransformer*

Apply 1 columns transformer to all columns.

Example: encode all categories with single category encoders.

Parameters

- **transformer** (*LAMLTransformer*) – Dataset - base transformer.
- **n_jobs** (int) – Number of threads.

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.

BestOfTransformers

`class lightautoml.transformers.base.BestOfTransformers`(*transformer_list*, *criterion*)

Bases: *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.

ConvertDataset

class lightautoml.transformers.base.**ConvertDataset**(*dataset_type*)

Bases: *LAMLTransformer*

Convert dataset to given type.

Parameters
dataset_type (Type[*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.

ChangeRoles

class lightautoml.transformers.base.**ChangeRoles**(*roles*)

Bases: *LAMLTransformer*

Change data roles (include dtypes etc).

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.

5.17.2 Numeric

NaNFlags	Create NaN flags.
FillnaMedian	Fillna with median.
FillnaMean	Fillna with mean.
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.
QuantileTransformer	Transform features using quantiles information.

NaNFlags

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

Bases: *LAMLTransformer*

Create NaN flags.

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.

FillnaMedian

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

Bases: *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.

FillnaMean

class lightautoml.transformers.numeric.**FillnaMean**

Bases: *LAMLTransformer*

Fillna with mean.

fit(*dataset*)

Estimate means.

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

Returns
self.

transform(*dataset*)

Transform - fillna with means.

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

Return type
NumpyDataset

Returns
Numpy dataset with encoded labels.

FillInf

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

Bases: *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.

LogOdds

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

Bases: *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.

StandardScaler

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

Bases: *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.

QuantileBinning

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

Bases: *LAMLTransformer*

Discretization of numeric features by quantiles.

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.

QuantileTransformer

class lightautoml.transformers.numeric.**QuantileTransformer**(*n_quantiles=None*, *subsample=1000000000*, *output_distribution='normal'*, *noise=0.001*, *qnt_factor=30*)

Bases: *LAMLTransformer*

Transform features using quantiles information.

__init__(*n_quantiles=None*, *subsample=1000000000*, *output_distribution='normal'*, *noise=0.001*, *qnt_factor=30*)

QuantileTransformer.

Parameters

- **n_quantiles** (Optional[int]) – Number of quantiles to be computed.
- **subsample** (int) – Maximum number of samples used to estimate the quantiles for computational efficiency.
- **output_distribution** (str) – Marginal distribution for the transformed data. The choices are ‘uniform’ or ‘normal’.
- **noise** (float) – Add noise with certain std to dataset before quantile transformation to make data more smooth.
- **qnt_factor** (int) – If number of quantiles is none then it equals dataset size / factor

fit(*dataset*)

Fit Sklearn QuantileTransformer.

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

Returns
self.

transform(*dataset*)

Apply transformer.

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

Return type
NumpyDataset

Returns
Numpy dataset with encoded labels.

5.17.3 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.

LabelEncoder

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

Bases: *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.

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.

OHEEncoder

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

Bases: *LAMLTransformer*

Simple OneHotEncoder over label encoded categories.

Parameters

- **make_sparse** (Optional[bool]) – Create sparse matrix.

- **total_feats_cnt** (Optional[int]) – Initial features number.
- **dtype** (type) – Dtype of new features.

property features

Features list.

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
Union[*NumpyDataset*, *CSRSparseDataset*]

Returns
Numpy dataset with encoded labels.

FreqEncoder

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

Bases: *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.

OrdinalEncoder

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

Bases: *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.

Returns
Self.

TargetEncoder

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

Bases: *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.

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(*dataset*)

Fit 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
Union[*NumpyDataset*, *CSRSparseDataset*]

Returns
Numpy dataset with encoded labels.

MultiClassTargetEncoder

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

Bases: *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

property features

List of features.

static score_func(*candidates*, *target*)

Choose the best encoder.

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
Union[*NumpyDataset*, *CSRSparseDataset*]

Returns
Numpy dataset with encoded labels.

CatIntersectstions

`class lightautoml.transformers.categorical.CatIntersectstions`(*subs=None*, *random_state=42*, *intersections=None*, *max_depth=2*)

Bases: *LabelEncoder*

Build label encoded intertsections of categorical variables.

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

Return type
NumpyDataset

Returns
Transformed dataset.

5.17.4 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.

TimeToNum

class lightautoml.transformers.datetime.**TimeToNum**

Bases: *LAMLTransformer*

Basic conversion strategy, used in selection one-to-one transformers.

Datetime converted to difference with random date from the corresponding column.

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.

BaseDiff

class lightautoml.transformers.datetime.**BaseDiff**(*base_names*, *diff_names*, *basic_interval='D'*)

Bases: *LAMLTransformer*

Basic conversion strategy, used in selection one-to-one transformers.

Datetime converted to difference with basic_date.

Parameters

- **base_names** (Sequence[str]) – Base date names.
- **diff_names** (Sequence[str]) – Difference date names.
- **basic_interval** (Optional[str]) – Time unit.

property features

List of features.

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.

DateSeasons

class lightautoml.transformers.datetime.**DateSeasons**(*output_role=None*)

Bases: *LAMLTransformer*

Basic conversion strategy, used in selection one-to-one transformers.

Datetime converted to difference with basic_date.

Parameters
output_role (Optional[*ColumnRole*]) – Which role to assign for input features.

property features

List of features names.

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.

5.17.5 Decompositions

PCATransformer	PCA.
SVDTransformer	TruncatedSVD.

PCATransformer

class lightautoml.transformers.decomposition.**PCATransformer**(*subs=None*, *random_state=42*, *n_components=500*)

Bases: *LAMLTransformer*

PCA.

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

property features

Features list.

fit(*dataset*)

Fit algorithm on dataset.

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

Returns
Self.

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.

SVDTransformer

class lightautoml.transformers.decomposition.**SVDTransformer**(*subs=None*, *random_state=42*, *n_components=100*)

Bases: *LAMLTransformer*

TruncatedSVD.

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.

property features

Features list.

fit(*dataset*)

Fit algorithm on dataset.

Parameters
dataset (Union[*NumpyDataset*, *CSRSparseDataset*]) – Sparse or Numpy dataset of text features.

Returns
self.

transform(*dataset*)

Transform input dataset to SVD representation.

Parameters
dataset (Union[*NumpyDataset*, *CSRSparseDataset*]) – Sparse or Numpy dataset of text features.

Return type
NumpyDataset

Returns
Numpy dataset with text embeddings.

5.17.6 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.

TunableTransformer

class lightautoml.transformers.text.**TunableTransformer**(*default_params=None*, *freeze_defaults=True*)

Bases: *LAMLTransformer*

Base class for ML transformers.

Assume that parameters my set before training.

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.

property params

Parameters.

Returns
Dict.

init_params_on_input(*dataset*)

Init params depending on input data.

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

Return type
dict

Returns
Dict with model hyperparameters.

TfidfTextTransformer

class lightautoml.transformers.text.**TfidfTextTransformer**(*default_params=None*, *freeze_defaults=True*, *subs=None*, *random_state=42*)

Bases: *TunableTransformer*

Simple Tfidf vectorizer.

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.

property features

Features list.

init_params_on_input(*dataset*)

Get transformer parameters depending on dataset parameters.

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

TokenizerTransformer

`class` lightautoml.transformers.text.`TokenizerTransformer`(*tokenizer=<lightautoml.text.tokenizer.SimpleEnTokenizer object>*)

Bases: *LAMLTransformer*

Simple tokenizer transformer.

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.

OneToOneTransformer

`class` lightautoml.transformers.text.`OneToOneTransformer`(*default_params=None*, *freeze_defaults=False*)

Bases: *TunableTransformer*

Out-of-fold sgd model prediction to reduce dimension of encoded text data.

`property features`

Features list.

`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.

Returns
self.

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.

ConcatTextTransformer

class lightautoml.transformers.text.**ConcatTextTransformer**(*special_token='[SEP]'*)

Bases: *LAMLTransformer*

Concat text features transformer.

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.

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: *LAMLTransformer*

Calculate text embeddings.

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** (Any) – Unused params.

property features

Features list.

fit(*dataset*)

Fit chosen transformer and create feature names.

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

Returns
self.

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.

5.17.7 Image

ImageFeaturesTransformer	Simple image histogram.
AutoCVWrap	Calculate image embeddings.

ImageFeaturesTransformer

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

Bases: *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.

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.

AutoCVWrap

class lightautoml.transformers.image.**AutoCVWrap**(*model='efficientnet_b0.ra_in1k'*, *weights_path=None*, *cache_dir='./cache_CV'*, *subs=None*, *device=torch.device*, *n_jobs=4*, *random_state=42*, *batch_size=128*, *verbose=True*)

Bases: *LAMLTransformer*

Calculate image embeddings.

Parameters

- **model** – Name of effnet model.
- **weights_path** (Optional[str]) – Path to saved weights.
- **cache_dir** (str) – Path to cache directory or None.
- **subs** (Optional[Any]) – Subsample to fit transformer. If None - full data.
- **device** (device) – Torch device.
- **n_jobs** (int) – Number of threads for dataloader.
- **random_state** (int) – Random state to take subsample and set torch seed.
- **batch_size** (int) – Batch size for embedding model.
- **verbose** (bool) – Verbose data processing.

property features

Features list.

Returns
List of features names.

fit(*dataset*)

Fit chosen transformer and create feature names.

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

Returns
self.

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.

5.18 lightautoml.utils

Common util tools.

5.18.1 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.

Timer

class lightautoml.utils.timer.**Timer**

Bases: object

Timer to limit the duration tasks.

PipelineTimer

class lightautoml.utils.timer.**PipelineTimer**(*timeout=None*, *overhead=0.1*, *mode=1*, *tuning_rate=0.7*)

Bases: *Timer*

Timer is used to control time over full automl run.

It decides how much time spend to each algo

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).

TaskTimer

class lightautoml.utils.timer.**TaskTimer**(*pipe_timer*, *key=None*, *score=1.0*, *overhead=1*, *mode=1*, *default_tuner_time_rate=0.7*)

Bases: *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.

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*.

property in_progress

Check if the task is running.

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.

Parameters
n_folds (int) – Number of folds.

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*]

Returns
Timers for each tasks.

5.19 lightautoml.validation

The module provide classes and functions for model validation.

5.19.1 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.

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.

Parameters

- **train** (TypeVar(Dataset, bound= *LAMLDataset*)) – Train dataset.
- ****kwargs** (Any) – Key-word parameters.

property features

Dataset features names.

Returns
List of features names.

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

DummyIterator

class lightautoml.validation.base.**DummyIterator**(*train*)

Bases: *TrainValidIterator*

Simple Iterator which use train data as validation.

__init__(*train*)

Create iterator. WARNING: validation on train.

Parameters
train (TypeVar(Dataset, bound= *LAMLDataset*)) – Train dataset.

get_validation_data()

Just get validation sample.

Return type
TypeVar(Dataset, bound= *LAMLDataset*)

Returns
Whole train dataset.

convert_to_holdout_iterator()

Convert iterator to hold-out-iterator.

Returns
Holdout iterator with `'train == valid'`.

Return type
iterator

HoldoutIterator

class lightautoml.validation.base.**HoldoutIterator**(*train*, *valid*)

Bases: *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.

CustomIterator

`class lightautoml.validation.base.CustomIterator(train, iterator)`

Bases: *TrainValidIterator*

Iterator that uses function to create folds indexes.

Useful 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.

FoldsIterator

`class lightautoml.validation.np_iterators.FoldsIterator(train, n_folds=None)`

Bases: *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[*CSRSparseDataset*, *NumpyDataset*, *PandasDataset*]) – Dataset for folding.
- **n_folds** (Optional[int]) – Number of folds.

get_validation_data()

Just return train dataset.

Return type
Union[*CSRSparseDataset*, *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.

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.

5.19.2 Iterators Getters and Utils

create_validation_iterator	Creates train-validation iterator.
get_numpy_iterator	Get iterator for np/sparse dataset.

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.

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[*CSRSparseDataset*, *NumpyDataset*, *PandasDataset*]) – `LAMLDataset` to train.
- **valid** (Union[*CSRSparseDataset*, *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

SIX

INDICES AND TABLES

- genindex

INDEX

Symbols

A

B

C

D

E

F

G

H

I

L

M

N

O

P

T

U

W