MLBox Documentation

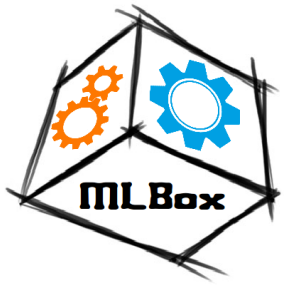
Axel ARONIO DE ROMBLAY

Aug 25, 2020

Tutorials

1 Links 3

Index 29



MLBox,
Machine Learning Box

MLBox is a powerful Automated Machine Learning python library. It provides the following features:

- Fast reading and distributed data preprocessing/cleaning/formatting.
- Highly robust feature selection and leak detection.
- Accurate hyper-parameter optimization in high-dimensional space.
- State-of-the art predictive models for classification and regression (Deep Learning, Stacking, LightGBM,. . .).
- Prediction with models interpretation.

CHAPTER 1

Links

- **Performance experiments:**
  - Kaggle competition "Two Sigma Connect: Rental Listing Inquiries" (rank: **85/2488**)
  - Kaggle competition "Sberbank Russian Housing Market" (rank: **190/3274**)
- **Examples & demos:**
  - Kaggle kernel on "Titanic" dataset (classification)
  - Kaggle kernel on "House Prices" dataset (regression)
- **Articles, books & tutorials from users:**
  - Tutorial on Automated Machine Learning using MLBox (Analytics Vidhya article)
  - MLBox: a short regression tutorial (user blog)
  - Implementing Auto-ML Systems with Open Source Tools (KDnuggets article)
  - Hands-On Automated Machine Learning (O'Reilly book)
  - Automatic Machine Learning (Youtube tutorial)
  - Automated Machine Learning with MLBox (user blog)
  - Introduction to AutoML with MLBox (user blog)
- **Webinars & conferences:**
  - Paris ML Hors Série #13: Automated Machine Learning
  - Analytics Vidhya: Automated Machine Learning using MLBox python package
  - DataHack Summit 2017 by Analytics Vidhya
- **References:**
  - AutoML.org
  - Skymind AI Wiki
  - TPOT github

  - Towards Data Science

1.1 Installation guide

1.1.1 Compatibilities

- *Operating systems:* **Linux**, **MacOS** & **Windows**.
- *Python versions:* **3.5** - **3.7**. & **64-bit version** only (32-bit python is not supported)

1.1.2 Basic requirements

We suppose that pip is already installed.

Also, please make sure you have setuptools and wheel installed, which is usually the case if pip is installed. If not, you can install both by running the following commands respectively: `pip install setuptools` and `pip install wheel`.

1.1.3 Preparation (MacOS only)

For **MacOS** users only, **OpenMP** is required. You can install it by the following command: `brew install libomp`.

1.1.4 Installation

You can choose to install MLBox either from pip or from the Github.

Install from pip

Official releases of MLBox are available on **PyPI**, so you only need to run the following command:

```
$ pip install mlbox
```

Install from the Github

If you want to get the latest features, you can also install MLBox from the Github.

- **The sources for MLBox can be downloaded** from the Github repo.
  - You can either clone the public repository:

```
$ git clone git://github.com/AxeldeRomblay/mlbox
```

  - Or download the tarball:

```
$ curl  -OL https://github.com/AxeldeRomblay/mlbox/tarball/master
```

- Once you have a copy of the source, **you can install it**:

```
$ cd MLBox
$ python setup.py install
```

1.1.5 Issues

If you get any troubles during installation, you can refer to the issues.

Please first check that there are no similar issues opened before opening one.

1.2 Getting started: 30 seconds to MLBox

MLBox main package contains 3 sub-packages : **preprocessing**, **optimisation** and **prediction**. Each one of them are respectively aimed at reading and preprocessing data, testing or optimising a wide range of learners and predicting the target on a test dataset.

Here are a few lines to import the MLBox:

```
from mlbox.preprocessing import *
from mlbox.optimisation import *
from mlbox.prediction import *
```

Then, all you need to give is :

- the list of paths to your train datasets and test datasets
- the name of the target you try to predict (classification or regression)

```
paths = ["<file_1>.csv", "<file_2>.csv", ..., "<file_n>.csv"] #to modify
target_name = "<my_target>" #to modify
```

Now, let the MLBox do the job !

... to read and preprocess your files :

```
data = Reader(sep=",").train_test_split(paths, target_name)  #reading
data = Drift_thresholder().fit_transform(data)  #deleting non-stable variables
```

... to evaluate models (here default configuration):

```
Optimiser().evaluate(None, data)
```

... or to test and optimize the whole Pipeline [**OPTIONAL**]:

- missing data encoder, aka 'ne'
- categorical variables encoder, aka 'ce'
- feature selector, aka 'fs'
- meta-features stacker, aka 'stck'
- final estimator, aka 'est'

NB : please have a look at all the possibilities you have to configure the Pipeline (steps, parameters and values...)

```
space = {

        'ne__numerical_strategy' : {"space" : [0, 'mean']},

        'ce__strategy' : {"space" : ["label_encoding", "random_projection", "entity_
↪embedding"]},

        'fs__strategy' : {"space" : ["variance", "rf_feature_importance"]},
        'fs__threshold': {"search" : "choice", "space" : [0.1, 0.2, 0.3]},

        'est__strategy' : {"space" : ["LightGBM"]},
        'est__max_depth' : {"search" : "choice", "space" : [5,6]},
        'est__subsample' : {"search" : "uniform", "space" : [0.6,0.9]}

        }

best = opt.optimise(space, data, max_evals = 5)
```

... finally to predict on the test set with the best parameters (or None for default configuration):

```
Predictor().fit_predict(best, data)
```

That's all ! You can have a look at the folder "save" where you can find :

- your predictions
- feature importances
- drift coefficients of your variables (0.5 = very stable, 1. = not stable at all)

1.3 Preprocessing

1.3.1 Reading

class mlbox.preprocessing.**Reader**(*sep=None*, *header=0*, *to_hdf5=False*, *to_path='save'*, *verbose=True*)

Reads and cleans data

Parameters

- **sep** (*str, defaut = None*) – Delimiter to use when reading a csv file.
- **header** (*int or None, default = 0.*) – If header=0, the first line is considered as a header. Otherwise, there is no header. Useful for csv and xls files.
- **to_hdf5** (*bool, default = True*) – If True, dumps each file to hdf5 format.
- **to_path** (*str, default = "save"*) – Name of the folder where files and encoders are saved.
- **verbose** (*bool, defaut = True*) – Verbose mode

clean(*path*, *drop_duplicate=False*)

Reads and cleans data (accepted formats : csv, xls, json and h5):

- del Unnamed columns
- casts lists into variables
- try to cast variables into float

- cleans dates and extracts timestamp from 01/01/2017, year, month, day, day_of_week and hour
- drop duplicates (if drop_duplicate=True)

Parameters

- **path** (*str*) – The path to the dataset.
- **drop_duplicate** (*bool, default = False*) – If True, drop duplicates when reading each file.

Returns Cleaned dataset.

Return type pandas dataframe

train_test_split(*Lpath*, *target_name*)

Creates train and test datasets

Given a list of several paths and a target name, automatically creates and cleans train and test datasets. IMPORTANT: a dataset is considered as a test set if it does not contain the target value. Otherwise it is considered as part of a train set. Also determines the task and encodes the target (classification problem only).

Finally dumps the datasets to hdf5, and eventually the target encoder.

Parameters

- **Lpath** (*list, defaut = None*) – List of str paths to load the data
- **target_name** (*str, default = None*) – The name of the target. Works for both classification (multiclass or not) and regression.

Returns

Dictionnary containing :

- 'train' : pandas dataframe for train dataset
- 'test' : pandas dataframe for test dataset
- 'target' : encoded pandas Serie for the target on train set (with dtype='float' for a regression or dtype='int' for a classification)

Return type dict

1.3.2 Drift thresholding

class mlbox.preprocessing.**Drift_thresholder**(*threshold=0.6*, *inplace=False*, *verbose=True*, *to_path='save'*)

Automatically drops ids and drifting variables between train and test datasets.

Drops on train and test datasets. The list of drift coefficients is available and saved as "drifts.txt". To get familiar with drift: https://github.com/AxeldeRomblay/MLBox/blob/master/docs/webinars/features.pdf

Parameters

- **threshold**(*float, defaut = 0.6*) – Drift threshold under which features are kept. Must be between 0. and 1. The lower the more you keep non-drifting/stable variables: a feature with a drift measure of 0. is very stable and a one with 1. is highly unstable.
- **inplace** (*bool, default = False*) – If True, train and test datasets are transformed. Returns self. Otherwise, train and test datasets are not transformed. Returns a new dictionnary with cleaned datasets.

- **verbose** (*bool, default = True*) – Verbose mode
- **to_path** (*str, default = "save"*) – Name of the folder where the list of drift coefficients is saved.

drifts()
Returns the univariate drifts for all variables.

Returns Dictionnary containing the drifts for each feature

Return type dict

fit_transform(*df*)
Fits and transforms train and test datasets

Automatically drops ids and drifting variables between train and test datasets. The list of drift coefficients is available and saved as "drifts.txt"

Parameters **df** (*dict, defaut = None*) – Dictionnary containing :

- 'train' : pandas dataframe for train dataset
- 'test' : pandas dataframe for test dataset
- 'target' : pandas serie for the target on train set

Returns

Dictionnary containing :

- 'train' : transformed pandas dataframe for train dataset
- 'test' : transformed pandas dataframe for test dataset
- 'target' : pandas serie for the target on train set

Return type dict

1.4 Encoding

1.4.1 Missing values

class mlbox.encoding.**NA_encoder**(*numerical_strategy='mean'*, *categorical_strategy='<NULL>'*)
Encodes missing values for both numerical and categorical features.

Several strategies are possible in each case.

Parameters

- **numerical_strategy** (*str or float or int. default = "mean"*) – The strategy to encode NA for numerical features. Available strategies = "mean", "median", "most_frequent" or a float/int value
- **categorical_strategy** (*str, default = '<NULL>'*) – The strategy to encode NA for categorical features. Available strategies = a string or "most_frequent"

fit(*df_train*, *y_train=None*)
Fits NA Encoder.

Parameters

- **df_train** (*pandas dataframe of shape = (n_train, n_features)*) – The train dataset with numerical and categorical features.

- **y_train** (*pandas series of shape = (n_train,), default = None*) – The target for classification or regression tasks.

Returns self

Return type object

fit_transform(*df_train*, *y_train=None*)
Fits NA Encoder and transforms the dataset.

Parameters

- **df_train** (*pandas.Dataframe of shape = (n_train, n_features)*) – The train dataset with numerical and categorical features.
- **y_train** (*pandas.Series of shape = (n_train,), default = None*) – The target for classification or regression tasks.

Returns The train dataset with no missing values.

Return type pandas.Dataframe of shape = (n_train, n_features)

get_params(*deep=True*)
Get parameters of a NA_encoder object.

set_params(***params*)
Set parameters for a NA_encoder object.

Set numerical strategy and categorical strategy.

Parameters

- **numerical_strategy** (*str or float or int. default = "mean"*) – The strategy to encode NA for numerical features.
- **categorical_strategy** (*str, default = '<NULL>'*) – The strategy to encode NA for categorical features.

transform(*df*)
Transform the dataset.

Parameters **df** (*pandas.Dataframe of shape = (n, n_features)*) – The dataset with numerical and categorical features.

Returns The dataset with no missing values.

Return type pandas.Dataframe of shape = (n, n_features)

1.4.2 Categorical features

class mlbox.encoding.**Categorical_encoder**(*strategy='label_encoding'*, *verbose=False*)
Encodes categorical features.

Several strategies are possible (supervised or not). Works for both classification and regression tasks.

Parameters

- **strategy** (*str, default = "label_encoding"*) – The strategy to encode categorical features. Available strategies = {"label_encoding", "dummification", "random_projection", entity_embedding"}
- **verbose** (*bool, default = False*) – Verbose mode. Useful for entity embedding strategy.

fit(*df_train*, *y_train*)
Fit Categorical Encoder.

Encode categorical variable of a dataframe following strategy parameters.

Parameters

- **df_train** (*pandas.Dataframe of shape = (n_train, n_features)*) – The training dataset with numerical and categorical features. NA values are allowed.
- **y_train** (*pandas.Series of shape = (n_train,)*) – The target for classification or regression tasks.

Returns self

Return type object

fit_transform(*df_train*, *y_train*)
Fits Categorical Encoder and transforms the dataset.

Fit categorical encoder following strategy parameter and transform the dataset df_train.

Parameters

- **df_train** (*pandas.Dataframe of shape = (n_train, n_features)*) – The training dataset with numerical and categorical features. NA values are allowed.
- **y_train** (*pandas.Series of shape = (n_train,)*) – The target for classification or regression tasks.

Returns Training dataset with numerical and encoded categorical features.

Return type pandas.Dataframe of shape = (n_train, n_features)

get_params(*deep=True*)
Get param that can be defined by the user.

Get strategy parameters and verbose parameters

Parameters

- **strategy** (*str, default = "label_encoding"*) – The strategy to encode categorical features. Available strategies = {"label_encoding", "dummification", "random_projection", entity_embedding"}
- **verbose** (*bool, default = False*) – Verbose mode. Useful for entity embedding strategy.

Returns **dict** – Dictionary that contains strategy and verbose parameters.

Return type dictionary

set_params(***params*)
Set param method for Categorical encoder.

Set strategy parameters and verbose parameters

Parameters

- **strategy** (*str, default = "label_encoding"*) – The strategy to encode categorical features. Available strategies = {"label_encoding", "dummification", "random_projection", entity_embedding"}
- **verbose** (*bool, default = False*) – Verbose mode. Useful for entity embedding strategy.

transform(*df*)
Transform categorical variable of df dataset.

Transform df DataFrame encoding categorical features with the strategy parameter if self.__fitOK is set to True.

Parameters **df** (*pandas.Dataframe of shape = (n_train, n_features)*) – The training dataset with numerical and categorical features. NA values are allowed.

Returns The dataset with numerical and encoded categorical features.

Return type pandas.Dataframe of shape = (n_train, n_features)

1.5 Model

1.5.1 Classification

Feature selection

class mlbox.model.classification.**Clf_feature_selector**(*strategy='l1', threshold=0.3*)
Selects useful features.

Several strategies are possible (filter and wrapper methods). Works for classification problems only (multiclass or binary).

Parameters

- **strategy** (*str, defaut = "l1"*) – The strategy to select features. Available strategies = {"variance", "l1", "rf_feature_importance"}
- **threshold** (*float, defaut = 0.3*) – The percentage of variable to discard according to the strategy. Must be between 0. and 1.

fit(*df_train*, *y_train*)
Fits Clf_feature_selector

Parameters

- **df_train** (*pandas dataframe of shape = (n_train, n_features)*) – The train dataset with numerical features and no NA
- **y_train** (*pandas series of shape = (n_train,)*) – The target for classification task. Must be encoded.

Returns self

Return type object

fit_transform(*df_train*, *y_train*)
Fits Clf_feature_selector and transforms the dataset

Parameters

- **df_train** (*pandas dataframe of shape = (n_train, n_features)*) – The train dataset with numerical features and no NA
- **y_train** (*pandas series of shape = (n_train,)*) – The target for classification task. Must be encoded.

Returns The train dataset with relevant features

Return type pandas dataframe of shape = (n_train, n_features*(1-threshold))

transform(*df*)
Transforms the dataset

Parameters **df** (*pandas dataframe of shape = (n, n_features)*) – The dataset with numerical features and no NA

Returns The train dataset with relevant features

Return type pandas dataframe of shape = (n_train, n_features*(1-threshold))

Classification

class mlbox.model.classification.**Classifier**(***params*)
Wraps scikitlearn classifiers.

Parameters

- **strategy** (*str, default = "LightGBM"*) – The choice for the classifier. Available strategies = {"LightGBM", "RandomForest", "ExtraTrees", "Tree", "Bagging", "AdaBoost" or "Linear"}.
- ****params** (*default = None*) – Parameters of the corresponding classifier. Examples : n_estimators, max_depth...

feature_importances()
Compute feature importances.

Classifier must be fitted before.

Returns Dictionnary containing a measure of feature importance (value) for each feature (key).

Return type dict

fit(*df_train*, *y_train*)
Fits Classifier.

Parameters

- **df_train** (*pandas dataframe of shape = (n_train, n_features)*) – The train dataset with numerical features.
- **y_train** (*pandas series of shape = (n_train,)*) – The numerical encoded target for classification tasks.

Returns self

Return type object

get_estimator()
Return classfier.

get_params(*deep=True*)
Get strategy parameters of Classifier object.

predict(*df*)
Predicts the target.

Parameters **df** (*pandas dataframe of shape = (n, n_features)*) – The dataset with numerical features.

Returns The encoded classes to be predicted.

Return type array of shape = (n,)

predict_log_proba(*df*)
Predicts class log-probabilities for df.

Parameters **df** (*pandas dataframe of shape = (n, n_features)*) – The dataset with numerical features.

Returns **y** – The log-probabilities for each class

Return type array of shape = (n, n_classes)

predict_proba(*df*)
Predicts class probabilities for df.

Parameters **df** (*pandas dataframe of shape = (n, n_features)*) – The dataset with numerical features.

Returns The probabilities for each class

Return type array of shape = (n, n_classes)

score(*df*, *y*, *sample_weight=None*)
Return the mean accuracy.

Parameters

- **df** (*pandas dataframe of shape = (n, n_features)*) – The dataset with numerical features.
- **y** (*pandas series of shape = (n,)*) – The numerical encoded target for classification tasks.

Returns Mean accuracy of self.predict(df) wrt. y.

Return type float

set_params(***params*)
Set strategy parameters of Classifier object.

Stacking

class mlbox.model.classification.**StackingClassifier**(*base_estimators=[<mlbox.model.classification.classifier.Classifier object>, <mlbox.model.classification.classifier.Classifier object>, <mlbox.model.classification.classifier.Classifier object>], level_estimator=<Mock name='mock()' id='140140225005944'>, n_folds=5, copy=False, drop_first=True, random_state=1, verbose=True*)
A stacking classifier.

A stacking classifier is a classifier that uses the predictions of several first layer estimators (generated with a cross validation method) for a second layer estimator.

Parameters

- **base_estimators**(*list, default = [Classifier(strategy="LightGBM"), Classifier(strategy="RandomForest"),Classifier(strategy="ExtraTrees")]*) – List of estimators to fit in the first level using a cross validation.

- **level_estimator** (*object, default = LogisticRegression()*) – The estimator used in second and last level.
- **n_folds** (*int, default = 5*) – Number of folds used to generate the meta features for the training set
- **copy** (*bool, default = False*) – If true, meta features are added to the original dataset
- **drop_first** (*bool, default = True*) – If True, each estimator output n_classes-1 probabilities
- **random_state** (*None or int or RandomState. default = 1*) – Pseudo-random number generator state used for shuffling. If None, use default numpy RNG for shuffling.
- **verbose** (*bool, default = True*) – Verbose mode.

fit (*df_train*, *y_train*)
Fits the first level estimators and the second level estimator on X.

Parameters

- **df_train** (*pandas dataframe of shape (n_samples, n_features)*) – Input data
- **y_train** (*pandas series of shape = (n_samples,)*) – The target

Returns self.

Return type object

fit_transform (*df_train*, *y_train*)
Creates meta-features for the training dataset.

Parameters

- **df_train** (*pandas dataframe of shape = (n_samples, n_features)*) – The training dataset.
- **y_train** (*pandas series of shape = (n_samples,)*) – The target.

Returns The transformed training dataset.

Return type pandas dataframe of shape = (n_samples, n_features*int(copy)+n_metafeatures)

predict (*df_test*)
Predicts class for the test set using the meta-features.

Parameters **df_test** (*pandas DataFrame of shape = (n_samples_test, n_features)*) – The testing samples

Returns The predicted classes.

Return type array of shape = (n_samples_test,)

predict_proba (*df_test*)
Predicts class probabilities for the test set using the meta-features.

Parameters **df_test** (*pandas DataFrame of shape = (n_samples_test, n_features)*) – The testing samples

Returns The class probabilities of the testing samples.

Return type array of shape = (n_samples_test, n_classes)

transform(*df_test*)
Creates meta-features for the test dataset.

Parameters **df_test** (*pandas dataframe of shape = (n_samples_test, n_features)*) – The test dataset.

Returns The transformed test dataset.

Return type pandas dataframe of shape = (n_samples_test, n_features*int(copy)+n_metafeatures)

1.5.2 Regression

Feature selection

class mlbox.model.regression.**Reg_feature_selector**(*strategy='l1'*, *threshold=0.3*)
Selects useful features.

Several strategies are possible (filter and wrapper methods). Works for regression problems only.

Parameters

- **strategy** (*str, defaut = "l1"*) – The strategy to select features. Available strategies = {"variance", "l1", "rf_feature_importance"}
- **threshold** (*float, defaut = 0.3*) – The percentage of variable to discard according the strategy. Must be between 0. and 1.

fit(*df_train*, *y_train*)
Fits Reg_feature_selector.

Parameters

- **df_train** (*pandas dataframe of shape = (n_train, n_features)*) – The train dataset with numerical features and no NA
- **y_train** (*pandas series of shape = (n_train,)*) – The target for regression task.

Returns self

Return type sobject

fit_transform(*df_train*, *y_train*)
Fits Reg_feature_selector and transforms the dataset

Parameters

- **df_train** (*pandas dataframe of shape = (n_train, n_features)*) – The train dataset with numerical features and no NA
- **y_train** (*pandas series of shape = (n_train,)*) – The target for regression task.

Returns The train dataset with relevant features

Return type pandas dataframe of shape = (n_train, n_features*(1-threshold))

transform(*df*)
Transforms the dataset

Parameters **df** (*pandas dataframe of shape = (n, n_features)*) – The dataset with numerical features and no NA

Returns The train dataset with relevant features

Return type pandas dataframe of shape = (n_train, n_features*(1-threshold))

Regression

class mlbox.model.regression.**Regressor**(***params*)

Wrap scikitlearn regressors.

Parameters

- **strategy** (*str, default = "LightGBM"*) – The choice for the regressor. Available strategies = {"LightGBM", "RandomForest", "ExtraTrees", "Tree", "Bagging", "AdaBoost" or "Linear"}
- ****params** (*default = None*) – Parameters of the corresponding regressor. Examples : n_estimators, max_depth...

feature_importances()

Computes feature importances.

Regressor must be fitted before.

Returns Dictionnary containing a measure of feature importance (value) for each feature (key).

Return type dict

fit(*df_train*, *y_train*)

Fits Regressor.

Parameters

- **df_train** (*pandas dataframe of shape = (n_train, n_features)*) – The train dataset with numerical features.
- **y_train** (*pandas series of shape = (n_train,)*) – The target for regression tasks.

Returns self

Return type object

get_estimator()

Return classfier.

get_params(*deep=True*)

Get parameters of Regressor object.

predict(*df*)

Predicts the target.

Parameters **df** (*pandas dataframe of shape = (n, n_features)*) – The dataset with numerical features.

Returns The target to be predicted.

Return type array of shape = (n,)

score(*df*, *y*, *sample_weight=None*)

Return R^2 coefficient of determination of the prediction.

Parameters

- **df** (*pandas dataframe of shape = (n, n_features)*) – The dataset with numerical features.

- **y** (*pandas series of shape = (n,)*) – The numerical encoded target for classification tasks.

Returns R^2 of self.predict(df) wrt. y.

Return type float

set_params(***params*)
Set parameters of Regressor object.

transform(*df*)
Transform dataframe df.

Parameters **df** (*pandas dataframe of shape = (n, n_features)*) – The dataset with numerical features.

Returns The transformed dataset with its most important features.

Return type pandas dataframe of shape = (n, n_selected_features)

Stacking

class mlbox.model.regression.**StackingRegressor**(*base_estimators=[<mlbox.model.regression.regressor.Regressor object>, <mlbox.model.regression.regressor.Regressor object>, <mlbox.model.regression.regressor.Regressor object>], level_estimator=<Mock name='mock()' id='140140224548648'>, n_folds=5, copy=False, random_state=1, verbose=True*)

A Stacking regressor.

A stacking regressor is a regressor that uses the predictions of

several first layer estimators (generated with a cross validation method) for a second layer estimator.

Parameters

- **base_estimators**(*list, default = [*Regressor*(strategy="LightGBM"),*) –

  Regressor(strategy="RandomForest"), Regressor(strategy="ExtraTrees")]

  List of estimators to fit in the first level using a cross validation.

- **level_estimator** (*object, default = LinearRegression()*) – The estimator used in second and last level
- **n_folds** (*int, default = 5*) – Number of folds used to generate the meta features for the training set
- **copy** (*bool, default = False*) – If true, meta features are added to the original dataset
- **random_state** (*None, int or RandomState. default = 1*) – Pseudo-random number generator state used for shuffling. If None, use default numpy RNG for shuffling.
- **verbose** (*bool, default = True*) – Verbose mode.

fit(*df_train*, *y_train*)
Fit the first level estimators and the second level estimator on X.

Parameters

- **df_train** (*pandas DataFrame of shape (n_samples, n_features)*) – Input data
- **y_train** (*pandas series of shape = (n_samples,)*) – The target

Returns self

Return type object

fit_transform(*df_train*, *y_train*)
Create meta-features for the training dataset.

Parameters

- **df_train** (*pandas DataFrame of shape = (n_samples, n_features)*) – The training dataset.
- **y_train** (*pandas series of shape = (n_samples,)*) – The target

Returns n_features*int(copy)+n_metafeatures) The transformed training dataset.

Return type pandas DataFrame of shape = (n_samples,

get_params(*deep=True*)
Get parameters of a StackingRegressor object.

predict(*df_test*)
Predict regression target for X_test using the meta-features.

Parameters **df_test** (*pandas DataFrame of shape = (n_samples_test, n_features)*) – The testing samples

Returns The predicted values.

Return type array of shape = (n_samples_test,)

set_params(***params*)
Set parameters of a StackingRegressor object.

transform(*df_test*)
Create meta-features for the test dataset.

Parameters **df_test** (*pandas DataFrame of shape = (n_samples_test, n_features)*) – The test dataset.

Returns n_features*int(copy)+n_metafeatures) The transformed test dataset.

Return type pandas DataFrame of shape = (n_samples_test,

1.6 Optimisation

class mlbox.optimisation.**Optimiser**(*scoring=None*, *n_folds=2*, *random_state=1*, *to_path='save'*, *verbose=True*)
Optimises hyper-parameters of the whole Pipeline.

- NA encoder (missing values encoder)
- CA encoder (categorical features encoder)

- Feature selector (OPTIONAL)
- Stacking estimator - feature engineer (OPTIONAL)
- Estimator (classifier or regressor)

Works for both regression and classification (multiclass or binary) tasks.

Parameters

- **scoring** (*str, callable or None. default: None*) – A string or a scorer callable object.

  If None, "neg_log_loss" is used for classification and "neg_mean_squared_error" for regression

  Available scorings can be found in the module sklearn.metrics: https://scikit-learn.org/stable/modules/model_evaluation.html#the-scoring-parameter-defining-model-evaluation-rules
- **n_folds** (*int, default = 2*) – The number of folds for cross validation (stratified for classification)
- **random_state** (*int, default = 1*) – Pseudo-random number generator state used for shuffling
- **to_path** (*str, default = "save"*) – Name of the folder where models are saved
- **verbose** (*bool, default = True*) – Verbose mode

evaluate (*params*, *df*)

Evaluates the data.

Evaluates the data with a given scoring function and given hyper-parameters of the whole pipeline. If no parameters are set, default configuration for each step is evaluated : no feature selection is applied and no meta features are created.

Parameters

- **params** (*dict, default = None.*) – Hyper-parameters dictionary for the whole pipeline.
  - The keys must respect the following syntax : "enc__param".
    - "enc" = "ne" for na encoder
    - "enc" = "ce" for categorical encoder
    - "enc" = "fs" for feature selector [OPTIONAL]
    - "enc" = "stck"+str(i) to add layer n°i of meta-features [OPTIONAL]
    - "enc" = "est" for the final estimator
    - "param" : a correct associated parameter for each step. Ex: "max_depth" for "enc"="est", ...
  - The values are those of the parameters. Ex: 4 for key = "est__max_depth", ...
- **df** (*dict, default = None*) – Dataset dictionary. Must contain keys and values:
  - "train": pandas DataFrame for the train set.
  - "target" : encoded pandas Serie for the target on train set (with dtype='float' for a regression or dtype='int' for a classification). Indexes should match the train set.

Returns The score. The higher the better. Positive for a score and negative for a loss.

Return type float.

Examples

```
>>> from mlbox.optimisation import *
>>> from sklearn.datasets import load_boston
>>> #load data
>>> dataset = load_boston()
>>> #evaluating the pipeline
>>> opt = Optimiser()
>>> params = {
...     "ne__numerical_strategy" : 0,
...     "ce__strategy" : "label_encoding",
...     "fs__threshold" : 0.1,
...     "stck__base_estimators" : [Regressor(strategy="RandomForest"),␣
↪Regressor(strategy="ExtraTrees")],
...     "est__strategy" : "Linear"
... }
>>> df = {"train" : pd.DataFrame(dataset.data), "target" : pd.Series(dataset.
↪target)}
>>> opt.evaluate(params, df)
```

optimise (*space*, *df*, *max_evals=40*)

Optimises the Pipeline.

Optimises hyper-parameters of the whole Pipeline with a given scoring function. Algorithm used to optimize : Tree Parzen Estimator.

IMPORTANT : Try to avoid dependent parameters and to set one feature selection strategy and one estimator strategy at a time.

Parameters

- **space** (*dict, default = None.*) – Hyper-parameters space:
  - The keys must respect the following syntax : “enc__param”.
    - ”enc” = “ne” for na encoder
    - ”enc” = “ce” for categorical encoder
    - ”enc” = “fs” for feature selector [OPTIONAL]
    - ”enc” = “stck”+str(i) to add layer n°i of meta-features [OPTIONAL]
    - ”enc” = “est” for the final estimator
    - ”param” : a correct associated parameter for each step. Ex: “max_depth” for “enc”=”est”, . . .
  - The values must respect the syntax: {“search”:strategy,”space”:list}
    - ”strategy” = “choice” or “uniform”. Default = “choice”
    - list : a list of values to be tested if strategy=”choice”. Else, list = [value_min, value_max].
- **df** (*dict, default = None*) – Dataset dictionary. Must contain keys and values:
  - ”train”: pandas DataFrame for the train set.

- ”target” : encoded pandas Serie for the target on train set (with dtype=’float’ for a regression or dtype=’int’ for a classification). Indexes should match the train set.

- **max_evals** (*int, default = 40.*) – Number of iterations. For an accurate optimal hyper-parameter, max_evals = 40.

Returns The optimal hyper-parameter dictionary.

Return type dict.

Examples

```
>>> from mlbox.optimisation import *
>>> from sklearn.datasets import load_boston
>>> #loading data
>>> dataset = load_boston()
>>> #optimising the pipeline
>>> opt = Optimiser()
>>> space = {
...     'fs__strategy':{"search":"choice","space":["variance","rf_feature_
↪importance"]},
...     'est__colsample_bytree':{"search":"uniform", "space":[0.3,0.7]}
... }
>>> df = {"train" : pd.DataFrame(dataset.data), "target" : pd.Series(dataset.
↪target)}
>>> best = opt.optimise(space, df, 3)
```

1.7 Prediction

class mlbox.prediction.**Predictor**(*to_path='save'*, *verbose=True*)

Fits and predicts the target on the test dataset.

The test dataset must not contain the target values.

Parameters

- **to_path** (*str, default = "save"*) – Name of the folder where feature importances and predictions are saved (.png and .csv formats). Must contain target encoder object (for classification task only).

- **verbose** (*bool, default = True*) – Verbose mode

fit_predict(*params*, *df*)

Fits the model and predicts on the test set.

Also outputs feature importances and the submission file (.png and .csv format).

Parameters

- **params** (*dict, default = None.*) – Hyper-parameters dictionary for the whole pipeline.

  – The keys must respect the following syntax : “enc__param”.

    * ”enc” = “ne” for na encoder

    * ”enc” = “ce” for categorical encoder

    * ”enc” = “fs” for feature selector [OPTIONAL]

* ”enc” = “stck”+str(i) to add layer n°i of meta-features [OPTIONAL]
* ”enc” = “est” for the final estimator
* ”param” : a correct associated parameter for each step. Ex: “max_depth” for “enc”=”est”, . . .

– The values are those of the parameters. Ex: 4 for key = “est__max_depth”, . . .

- **df** (*dict, default = None*) – Dataset dictionary. Must contain keys and values:
  - ”train”: pandas DataFrame for the train set.
  - ”test” : pandas DataFrame for the test set.
  - ”target” : encoded pandas Serie for the target on train set (with dtype=’float’ for a regression or dtype=’int’ for a classification). Indexes should match the train set.

Returns self.

Return type object

1.8 Authors

1.8.1 Development Lead

- Axel ARONIO DE ROMBLAY
  - email: <axelderomblay@gmail.com>
  - linkedin: <https://www.linkedin.com/in/axel-de-romblay-6444a990/>

1.8.2 Contributors

- Nicolas CHEREL <nicolas.cherel@telecom-paristech.fr>
- Mohamed MASKANI <maskani.mohamed@gmail.com>
- Henri GERARD <hgerard.pro@gmail.com>

1.9 History

1.9.1 0.1.0 (2017-02-09)

- First non-official release.

1.9.2 0.1.1 (2017-02-23)

- add of several estimators : Random Forest, Extra Trees, Logistic Regression, . . .
- improvement in verbose mode for reader.

1.9.3 0.1.2 (2017-03-02)

- add of dropout for entity embeddings.
- improvement in optimiser.

1.9.4 0.2.0 (2017-03-22)

- add of feature importances for base learners.
- add of leak detection.
- add of stacking meta-model.
- improvement in verbose mode for optimiser (folds variance).

1.9.5 0.2.1 (2017-04-26)

- add of feature importances for bagging and boosting meta-models.

1.9.6 0.2.2 (first official release : 2017-06-13)

- update of dependencies (Keras 2.0,...).
- add of LightGBM model.

1.9.7 0.3.0 (2017-07-11)

- Python 2.7 & Python 3.4-3.6 compatibilities

1.9.8 0.3.1 (2017-07-12)

- Availability on PyPI.

1.9.9 0.4.0 (2017-07-18)

- add of pipeline memory.

1.9.10 0.4.1 (2017-07-21)

- improvement in verbose mode for reader (display missing values)

1.9.11 0.4.2 (2017-07-25)

- update of dependencies

1.9.12 0.4.3 (2017-07-26)

- improvement in verbose mode for predictor (display feature importances)
- wait until modules and engines are imported

1.9.13 0.4.4 (2017-08-04)

- pep8 style
- normalization of drift coefficients
- warning size of folder 'save'

1.9.14 0.5.0 (2017-08-24)

- improvement in verbose mode
- add of new dates features
- add of a new strategy for missing categorical values
- new parallel computing

1.9.15 0.5.1 (2017-08-25)

- improvement in verbose mode for reader (display target quantiles for regression)

1.9.16 0.6.0 (2019-04-26)

- remove xgboost installation

1.9.17 0.7.0 (2019-06-26)

- add support for Mac OS & Windows
- update support for python versions
- improve setup
- add tests
- improve documentation & examples
- minor changes in the package architecture

1.9.18 0.8.0 (2019-07-29)

- remove support for python 2.7 version

1.9.19 0.8.1 (2019-08-29)

- add python 3.7 version
- update package dependencies

1.9.20 0.8.4 (2020-04-13)

- update package dependencies

1.9.21 0.8.5 (2020-08-25)

- minor fix (package dependencies)

1.10 Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

1.10.1 Types of Contributions

Report Bugs

Report bugs at https://github.com/AxeldeRomblay/mlbox/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- The smallest possible example to reproduce the bug.

Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" and "help wanted" is open to whoever wants to implement it.

Implement Features

Look through the GitHub issues for features. Anything tagged with "enhancement" and "help wanted" is open to whoever wants to implement it.

Write Documentation

MLBox could always use more documentation, whether as part of the official MLBox docs, in docstrings, or even on the web in blog posts, articles, and such.

Submit Feedback

The best way to send feedback is to file an issue at https://github.com/AxeldeRomblay/mlbox/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

1.10.2 Get Started!

Ready to contribute? Here's how to set up *mlbox* for local development.

1. Fork the *mlbox* repo on GitHub.
2. Clone your fork:

```
$ git clone git@github.com:your_name_here/mlbox.git
```

3. If you have virtualenv installed, skip this step. Either, run the following:

```
$ pip install virtualenv
```

4. Install your local copy into a virtualenv following this commands to set up your fork for local development:

```
$ cd MLBox
$ virtualenv env
$ source env/bin/activate
$ python setup.py develop
```

If you have any troubles with the setup, please refer to the installation guide

5. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

Now you're set, you can make your changes locally.

NOTE : each time you work on your branch, you will need to activate the virtualenv: `$ source env/bin/activate`. To deactivate it, simply run: `$ deactivate`.

6. When you're done making changes, check that your changes pass the tests.

NOTE : you need to install **pytest** before running the tests:

```
$ cd tests
$ pytest
```

7. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

8. Submit a pull request through the GitHub website.

1.10.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring.
3. The pull request should work for all supported Python versions and for PyPy. Check https://travis-ci.org/AxeldeRomblay/MLBox/pull_requests and make sure that the tests pass for all supported Python versions.

Index

C

Categorical_encoder (*class in mlbox.encoding*), 9
Classifier (*class in mlbox.model.classification*), 12
clean() (*mlbox.preprocessing.Reader method*), 6
Clf_feature_selector (*class in mlbox.model.classification*), 11

D

Drift_thresholder (*class in mlbox.preprocessing*), 7
drifts() (*mlbox.preprocessing.Drift_thresholder method*), 8

E

evaluate() (*mlbox.optimisation.Optimiser method*), 19

F

feature_importances() (*mlbox.model.classification.Classifier method*), 12
feature_importances() (*mlbox.model.regression.Regressor method*), 16
fit() (*mlbox.encoding.Categorical_encoder method*), 9
fit() (*mlbox.encoding.NA_encoder method*), 8
fit() (*mlbox.model.classification.Classifier method*), 12
fit() (*mlbox.model.classification.Clf_feature_selector method*), 11
fit() (*mlbox.model.classification.StackingClassifier method*), 14
fit() (*mlbox.model.regression.Reg_feature_selector method*), 15
fit() (*mlbox.model.regression.Regressor method*), 16
fit() (*mlbox.model.regression.StackingRegressor method*), 17
fit_predict() (*mlbox.prediction.Predictor method*), 21
fit_transform() (*mlbox.encoding.Categorical_encoder method*), 10
fit_transform() (*mlbox.encoding.NA_encoder method*), 9
fit_transform() (*mlbox.model.classification.Clf_feature_selector method*), 11
fit_transform() (*mlbox.model.classification.StackingClassifier method*), 14
fit_transform() (*mlbox.model.regression.Reg_feature_selector method*), 15
fit_transform() (*mlbox.model.regression.StackingRegressor method*), 18
fit_transform() (*mlbox.preprocessing.Drift_thresholder method*), 8

G

get_estimator() (*mlbox.model.classification.Classifier method*), 12
get_estimator() (*mlbox.model.regression.Regressor method*), 16
get_params() (*mlbox.encoding.Categorical_encoder method*), 10
get_params() (*mlbox.encoding.NA_encoder method*), 9
get_params() (*mlbox.model.classification.Classifier method*), 12
get_params() (*mlbox.model.regression.Regressor method*), 16
get_params() (*mlbox.model.regression.StackingRegressor method*), 18

N

NA_encoder (*class in mlbox.encoding*), 8

O

optimise() (*mlbox.optimisation.Optimiser method*), 20
Optimiser (*class in mlbox.optimisation*), 18

P

predict() (*mlbox.model.classification.Classifier method*), 12
predict() (*mlbox.model.classification.StackingClassifier method*), 14
predict() (*mlbox.model.regression.Regressor method*), 16
predict() (*mlbox.model.regression.StackingRegressor method*), 18
predict_log_proba() (*mlbox.model.classification.Classifier method*), 12
predict_proba() (*mlbox.model.classification.Classifier method*), 13
predict_proba() (*mlbox.model.classification.StackingClassifier method*), 14
Predictor (*class in mlbox.prediction*), 21

R

Reader (*class in mlbox.preprocessing*), 6
Reg_feature_selector (*class in mlbox.model.regression*), 15
Regressor (*class in mlbox.model.regression*), 16

S

score() (*mlbox.model.classification.Classifier method*), 13
score() (*mlbox.model.regression.Regressor method*), 16
set_params() (*mlbox.encoding.Categorical_encoder method*), 10
set_params() (*mlbox.encoding.NA_encoder method*), 9
set_params() (*mlbox.model.classification.Classifier method*), 13
set_params() (*mlbox.model.regression.Regressor method*), 17
set_params() (*mlbox.model.regression.StackingRegressor method*), 18
StackingClassifier (*class in mlbox.model.classification*), 13
StackingRegressor (*class in mlbox.model.regression*), 17

T

train_test_split() (*mlbox.preprocessing.Reader method*), 7
transform() (*mlbox.encoding.Categorical_encoder method*), 10
transform() (*mlbox.encoding.NA_encoder method*), 9
transform() (*mlbox.model.classification.Clf_feature_selector method*), 11
transform() (*mlbox.model.classification.StackingClassifier method*), 14
transform() (*mlbox.model.regression.Reg_feature_selector method*), 15
transform() (*mlbox.model.regression.Regressor method*), 17
transform() (*mlbox.model.regression.StackingRegressor method*), 18