autoMLk Documentation

Release 0.0.1a

pierre-chaville

Apr 08, 2018

Contents

This toolkit is designed to be integrated within a python project, but also independently through the interface of the app.

The framework is built with principles from auto-sklearn, with the following improvements:

- web interface (flask) to review the datasets, the search results and graphs
- include sklearn models, but also Xgboost, LightGBM, CatBoost and keras Neural Networks
- 2nd level ensembling with model selection and stacking
- can be used in competition mode (to generate a submit file from a test set), on benchmark mode (separate train set and public set) and standard mode.

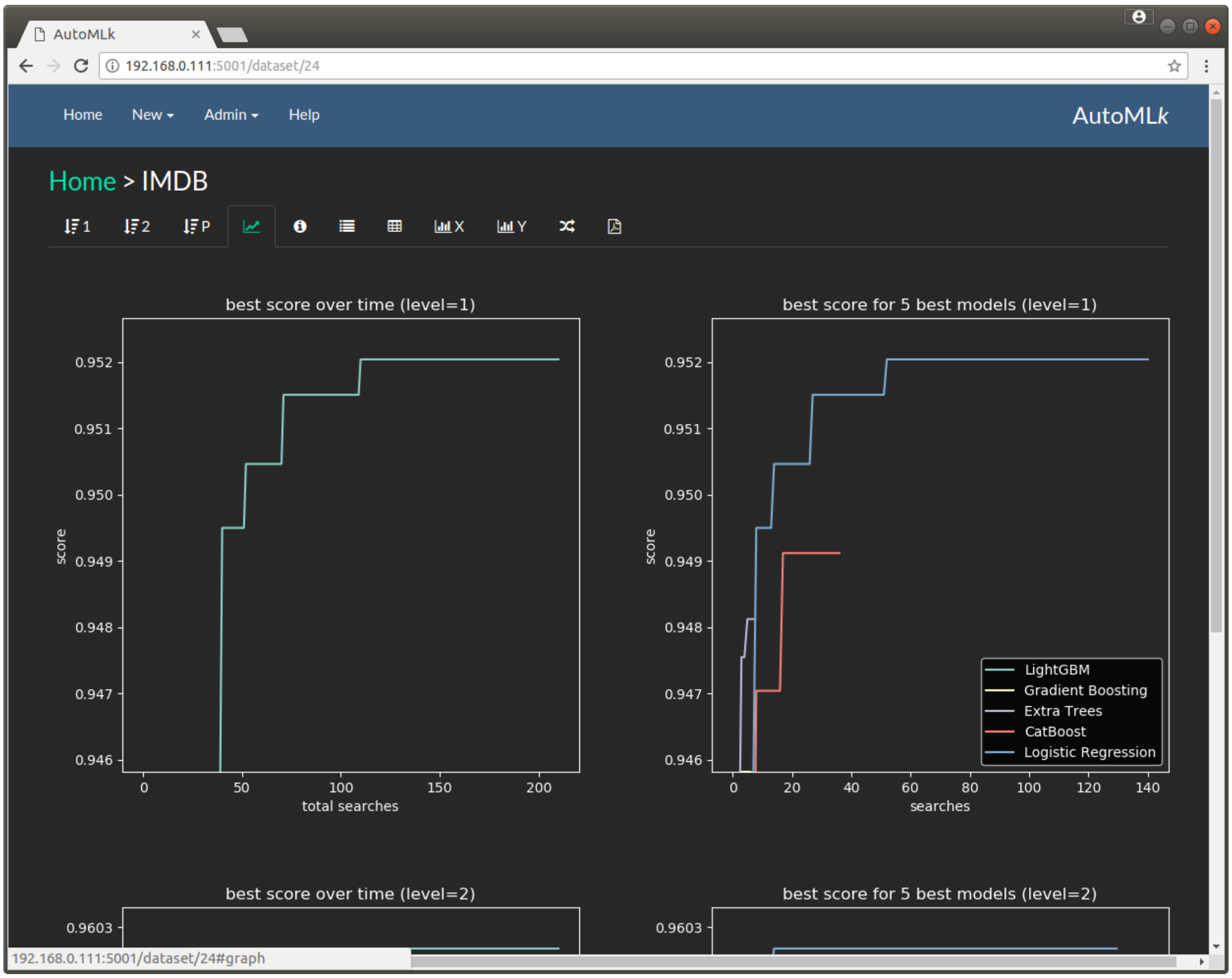


Fig. 1: Best models by eval score

We have provided some public datasets to initialize the framework and compare results with best scores.

CHAPTER 1

Content

1.1 User guide

The dataset and the results of the search are best viewed with the web app through a standard browers.

to start the app, please go in the web folder and run the app server:

```
python run.py
```

then access the app in a browser with the follwoing url:

```
http://localhost:5001
```

or from another machine with the ip address of the machine where the server is running:

```
http://192.168.0.10:5001
```

(in this example, we suppose the address of the server is 192.168.0.10)

1.1.1 Home

The home page shows the list of datasets:

You can select a list of datasets from a specific domain, with the selector at the top right:

1.1.2 Dataset

To import the list of preloaded datasets (or your own list), you can select the option 'Import' in the menu 'New':

You may create directly a dataset by using the 'Dataset' option in the menu 'New':

You may afterwards update *some fields* of a dataset by using the edit icon in the list of datasets in the home page:

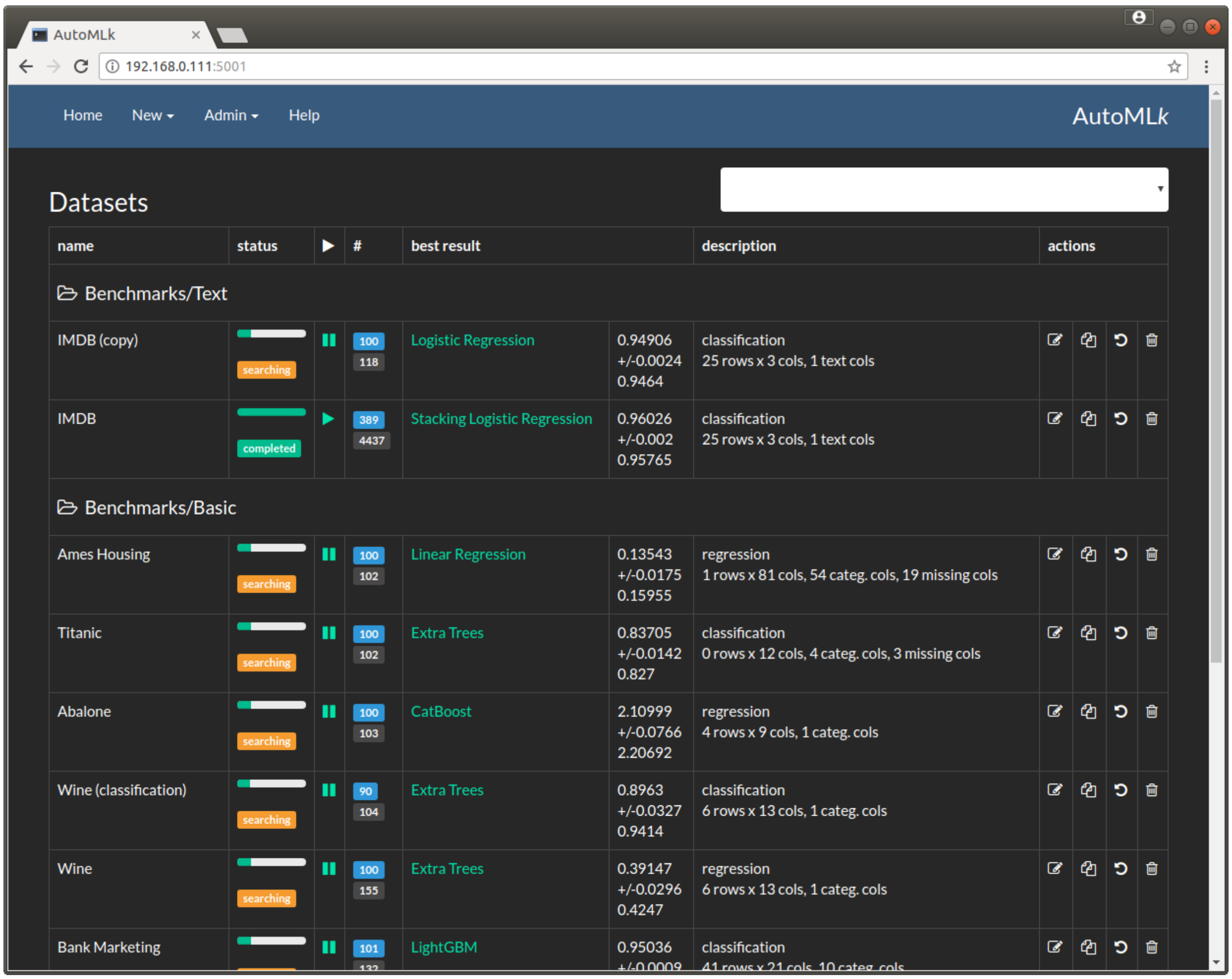


Fig. 1.1: list of datasets in autoMLk

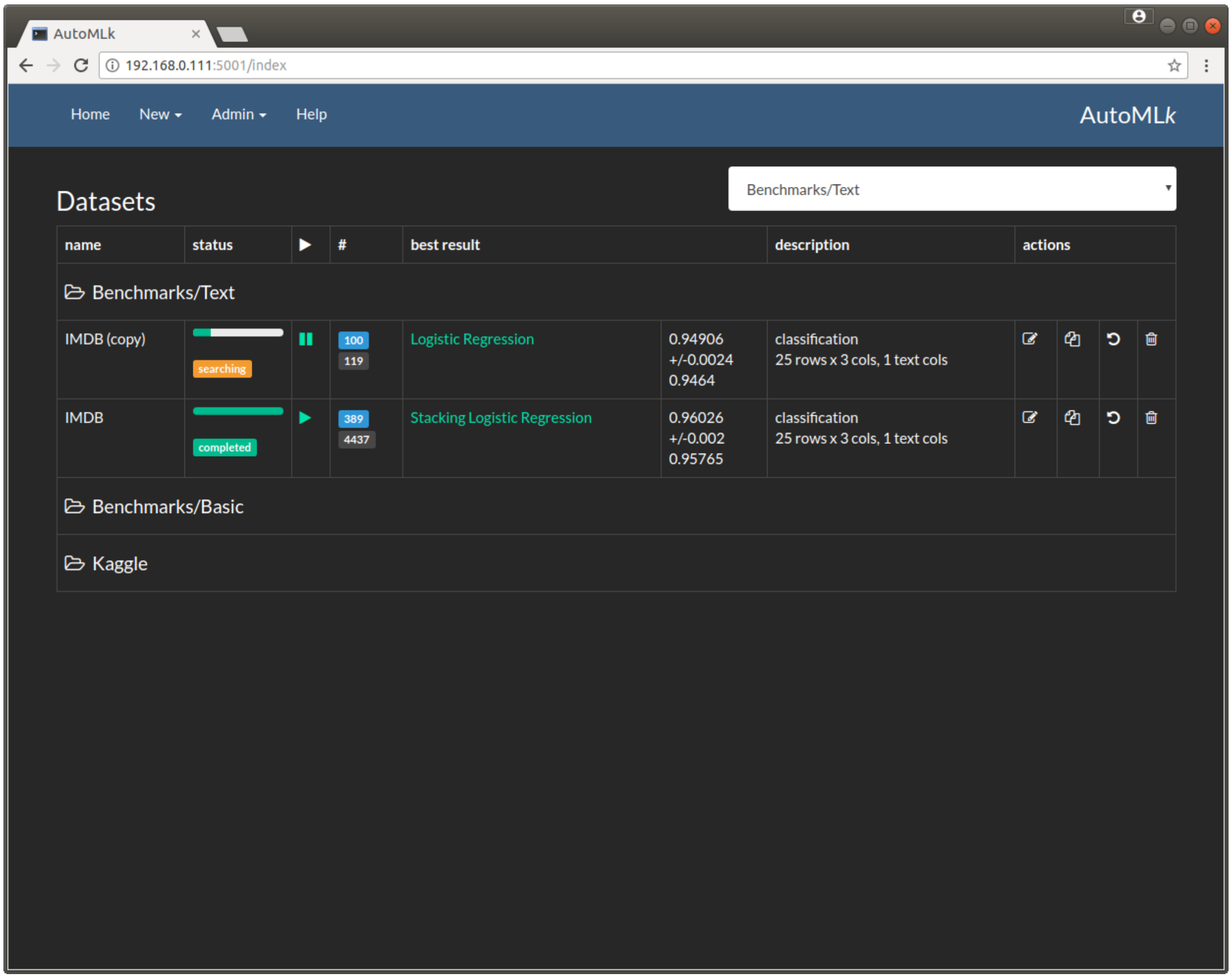


Fig. 1.2: list of datasets per domain

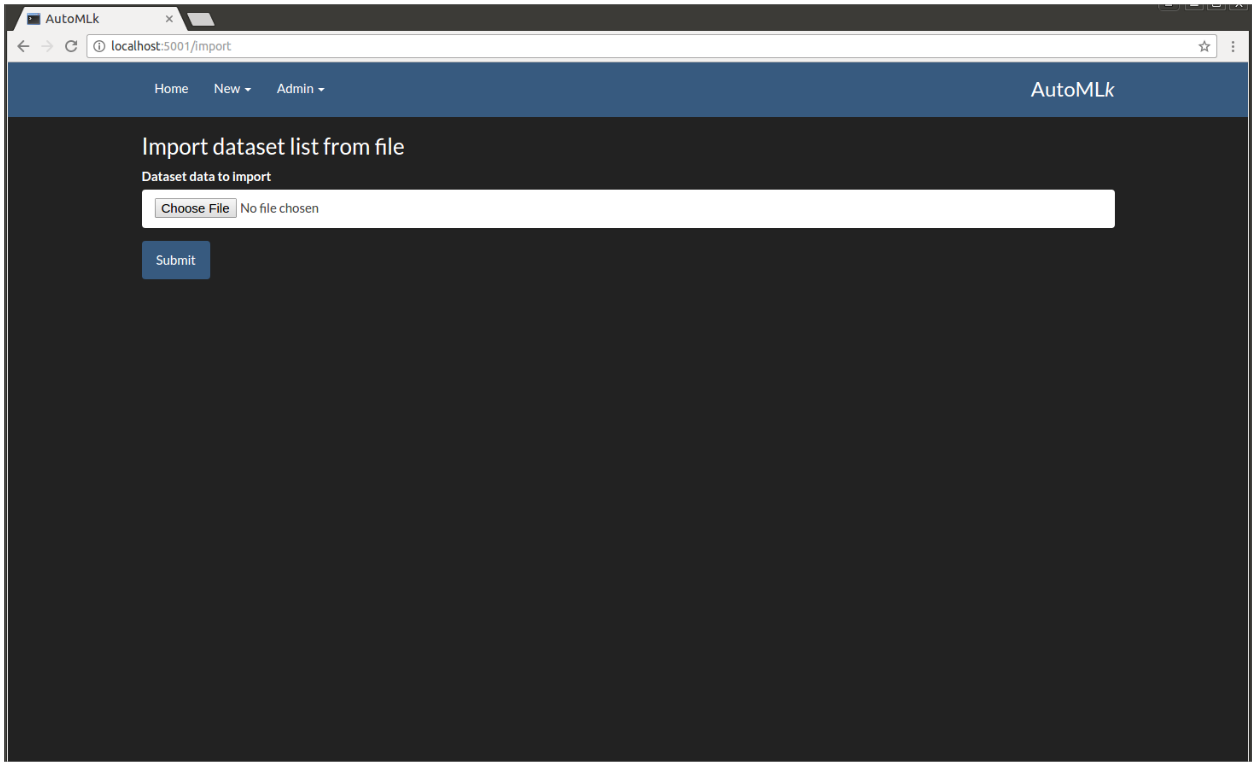


Fig. 1.3: import a list of datasets

We can access to a specific dataset in clicking on the row of the required dataset. When a dataset is created, there is only the features and analysis of the data available:

By clicking on the various tabs, we can view:

We need to launch the search process with various models in order to access to be results

1.1.3 Results and best models

When the search is launched, 3 additional tabs are available:

And per pre-processing steps:

The graph of the best results over time:

And after a while, the best ensembles:

The best ensembles

And then by clicking on a specific model access to the details

And then on a specific round:

Where we can view the performance and the predictions:

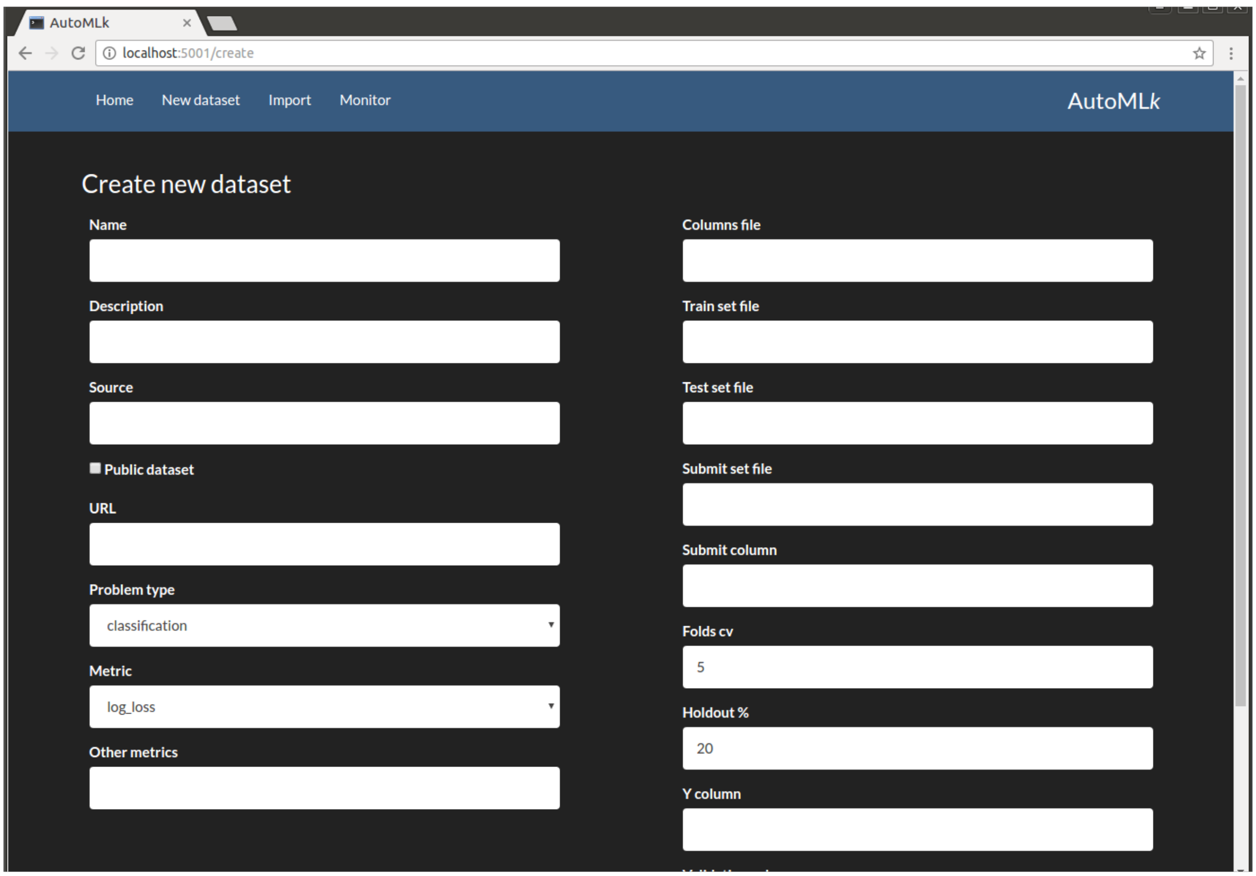


Fig. 1.4: create a new dataset

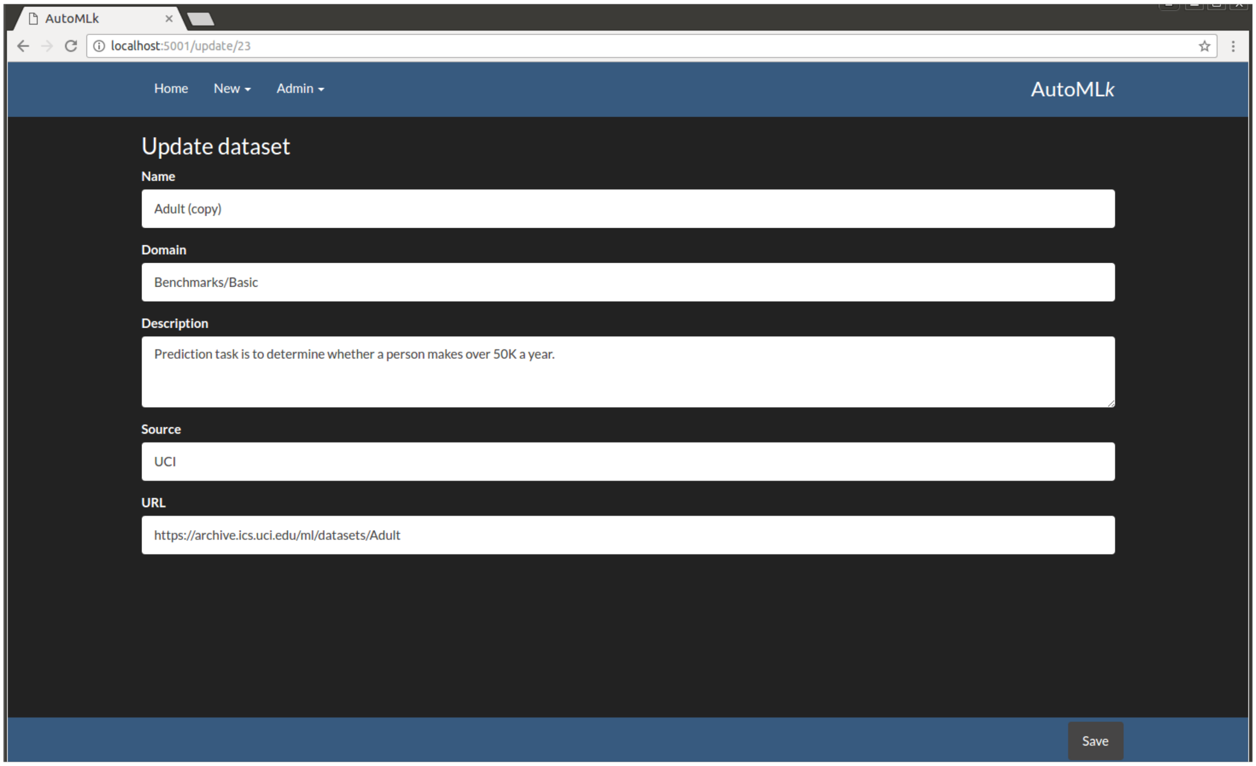


Fig. 1.5: update a dataset

img/info.png

Fig. 1.6: parameters of the dataset

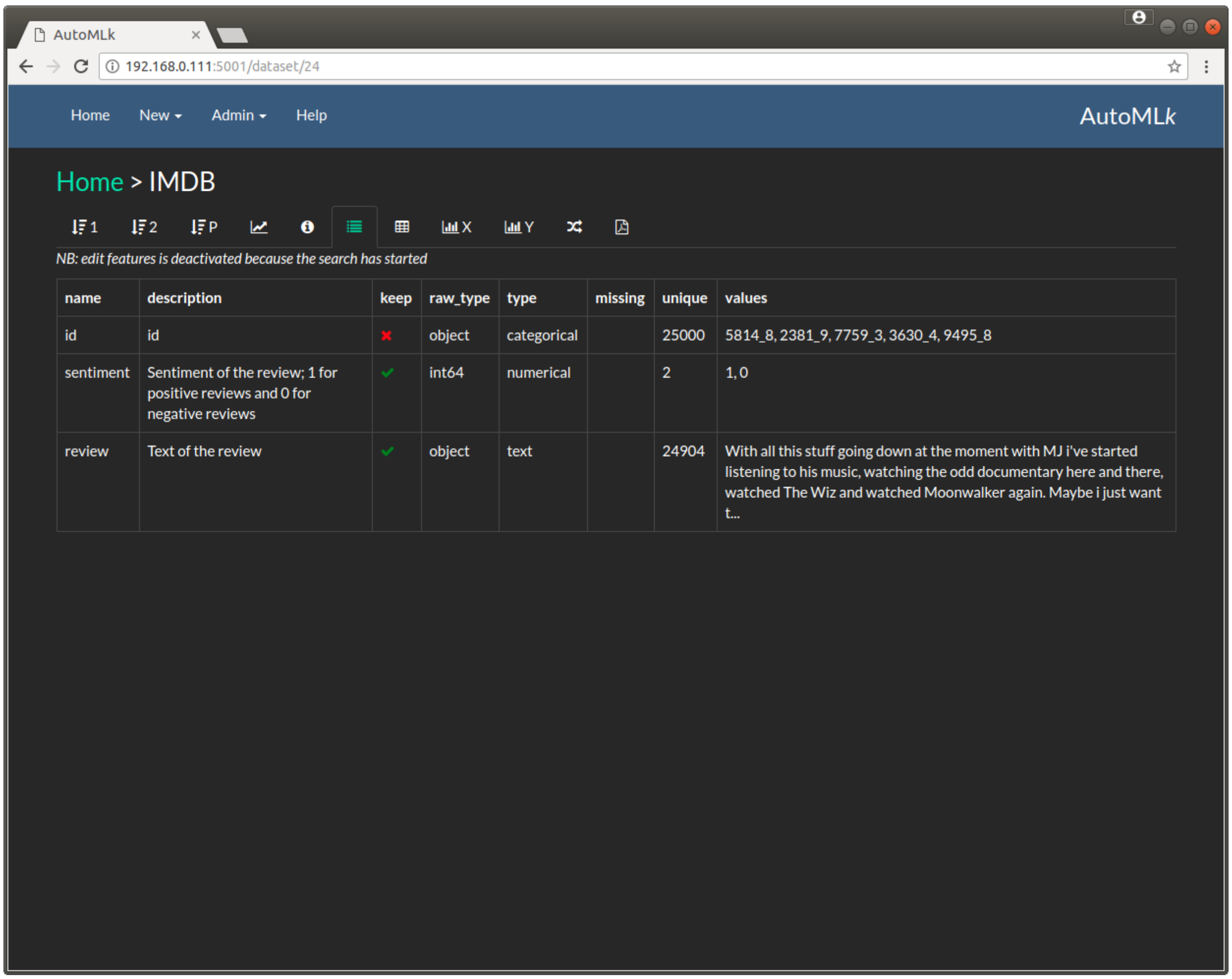


Fig. 1.7: the list of features of the dataset

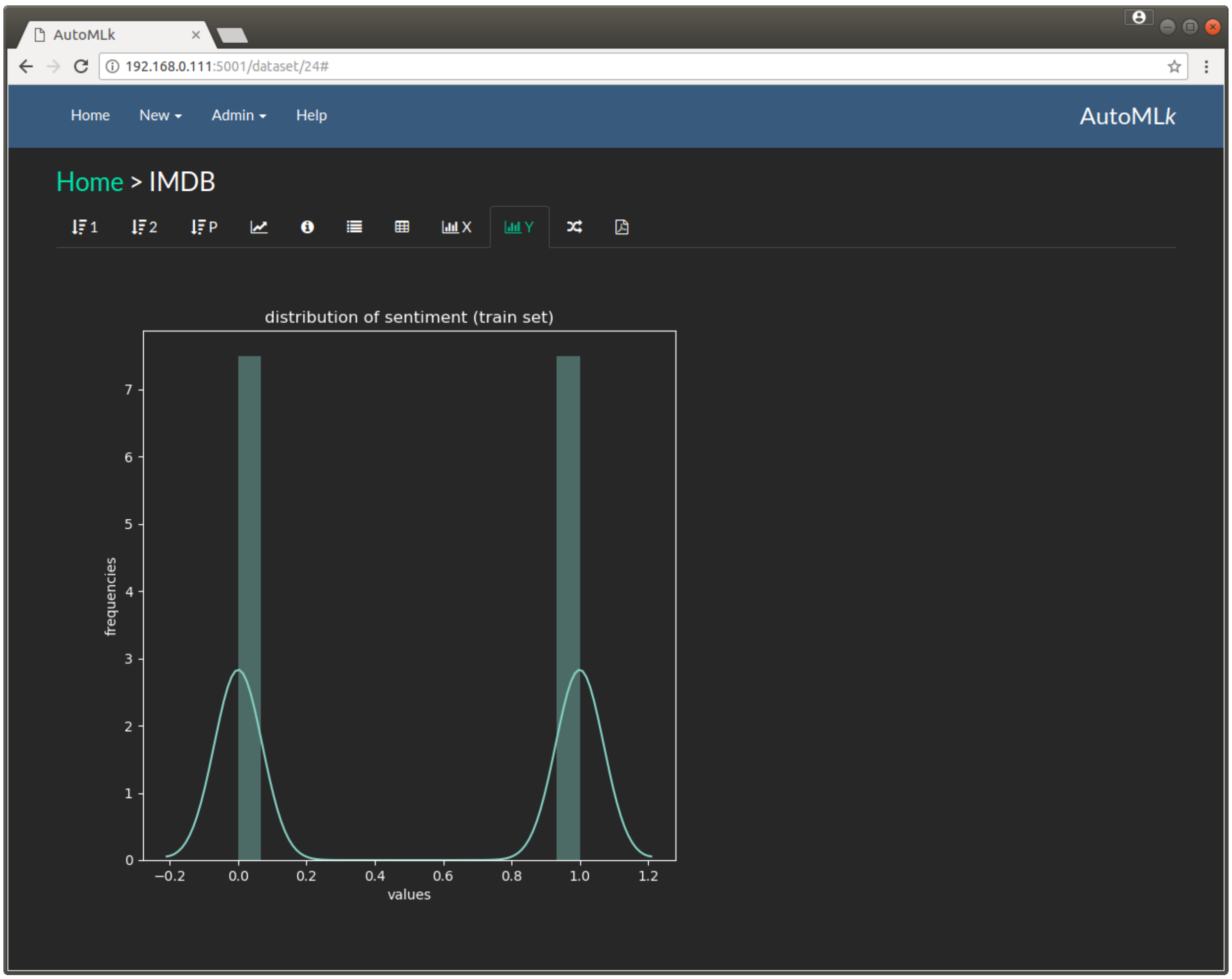


Fig. 1.8: the histogram of the target column

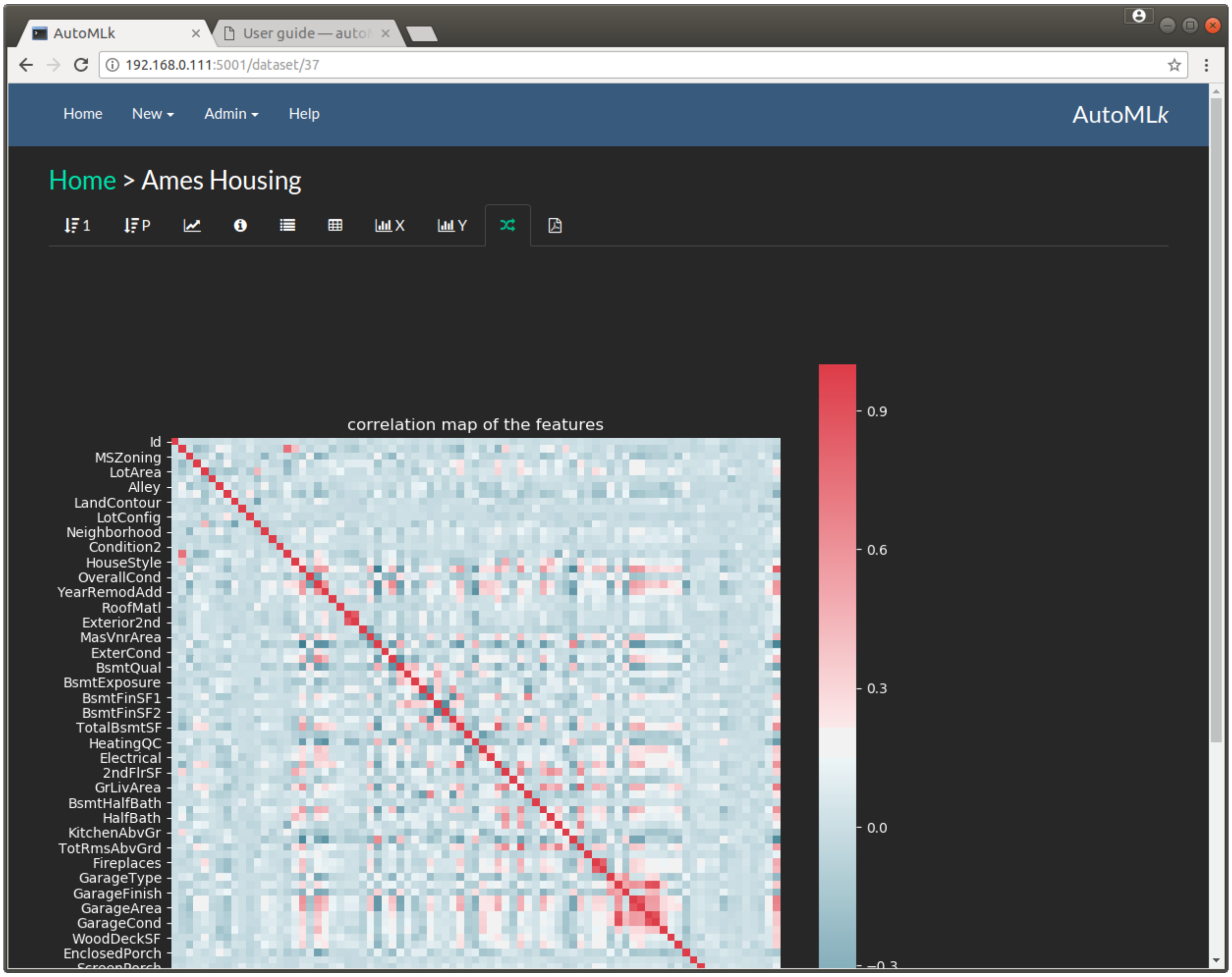


Fig. 1.9: the correlation matrix of the features

img/models.png

Fig. 1.10: Best models by eval score

img/processes.png

Fig. 1.11: pre-processing steps by eval score

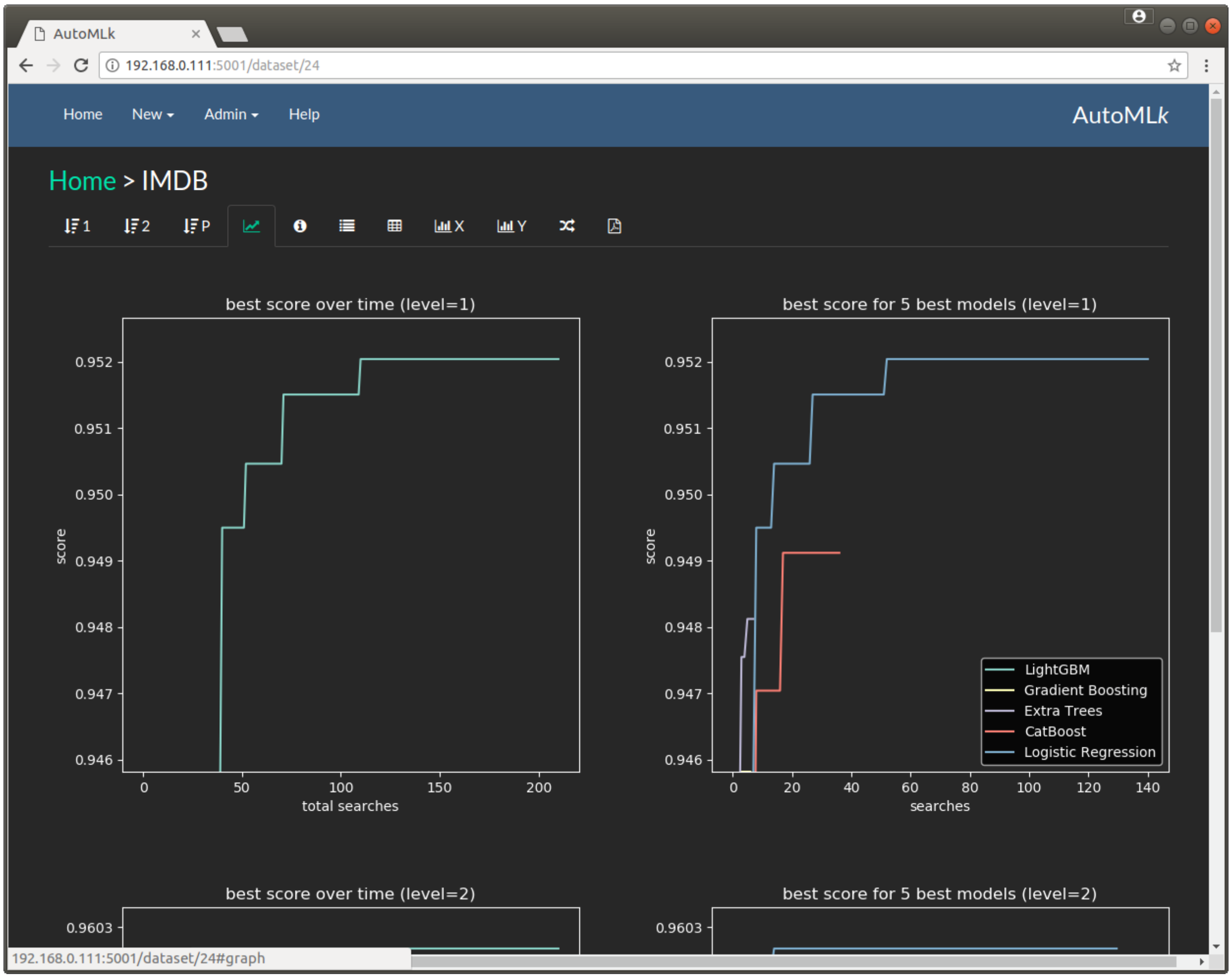
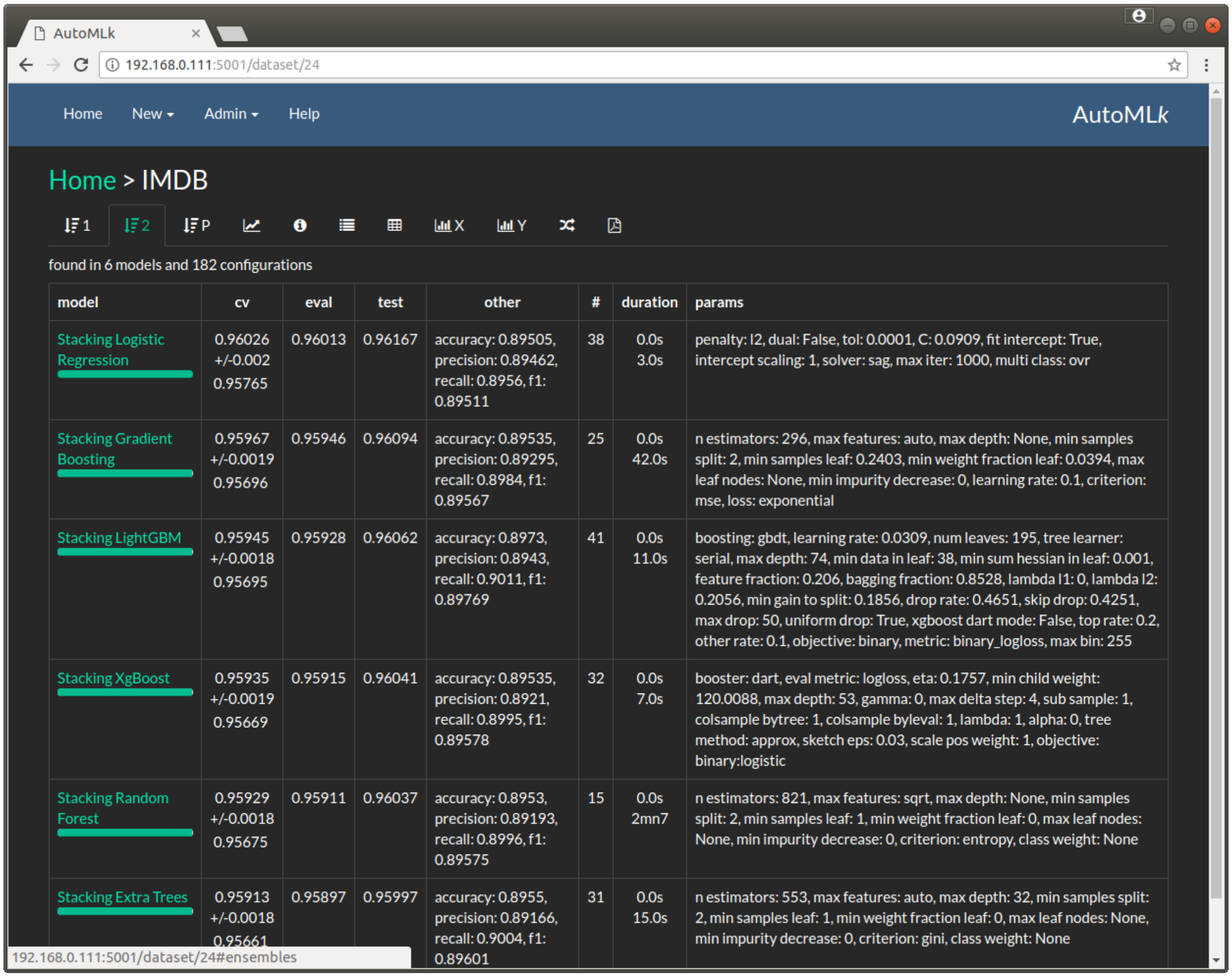


Fig. 1.12: The evolution of the best scores in time



img/list_model.png

Fig. 1.13: details of the search by model

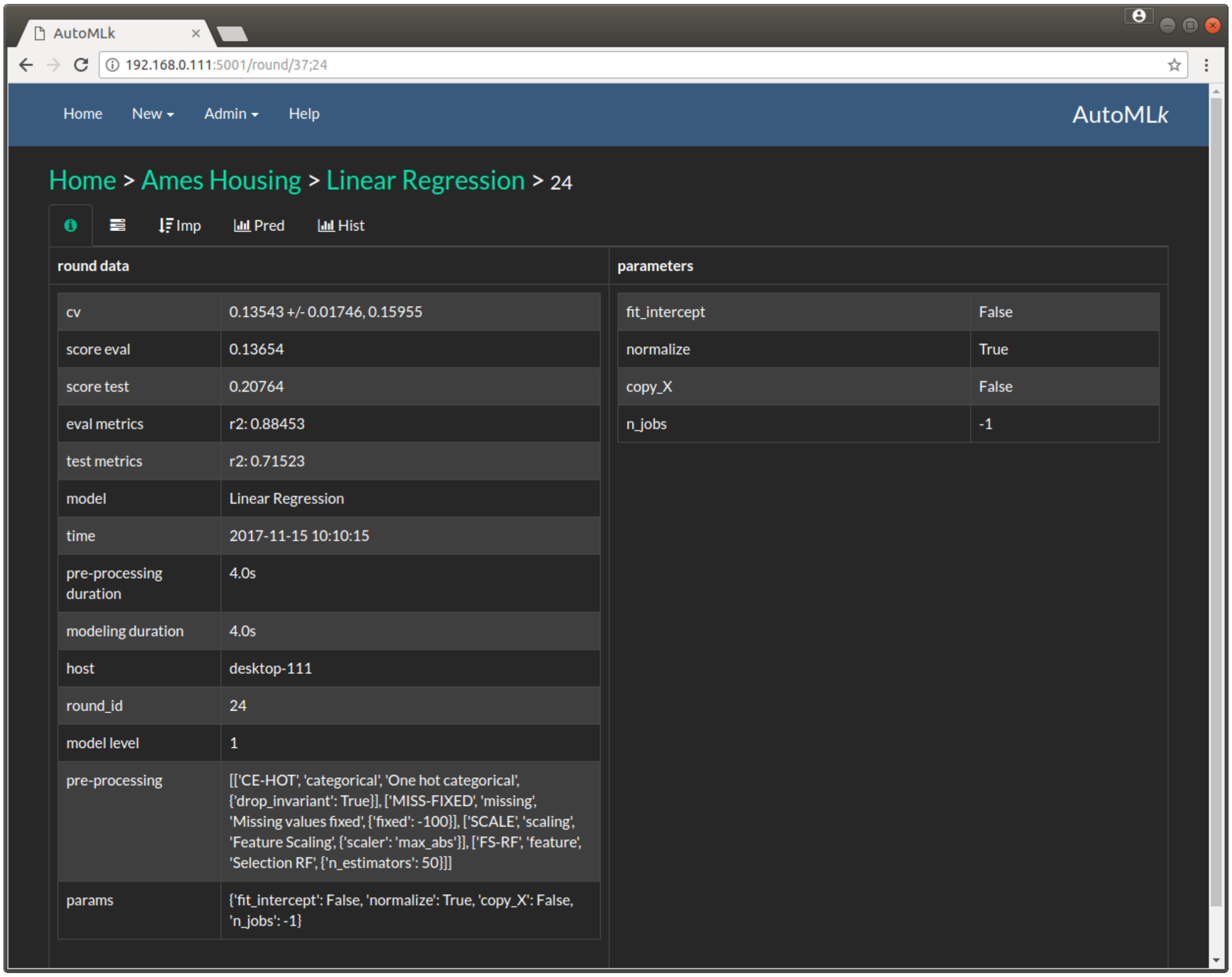


Fig. 1.14: a round with a se of model parameters and pre-processing

img/steps.png

Fig. 1.15: details of the re-processing steps

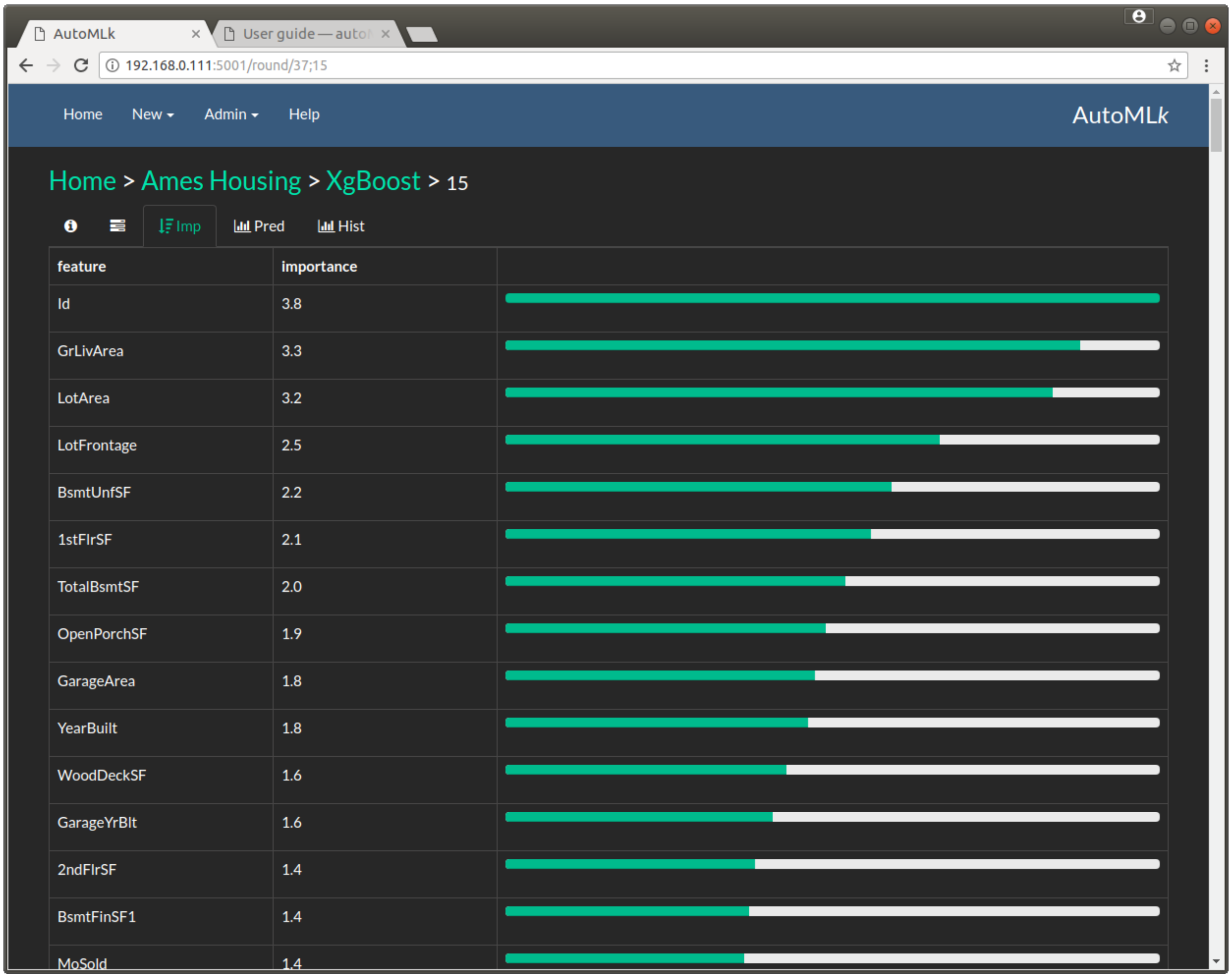


Fig. 1.16: feature importance scored by the model

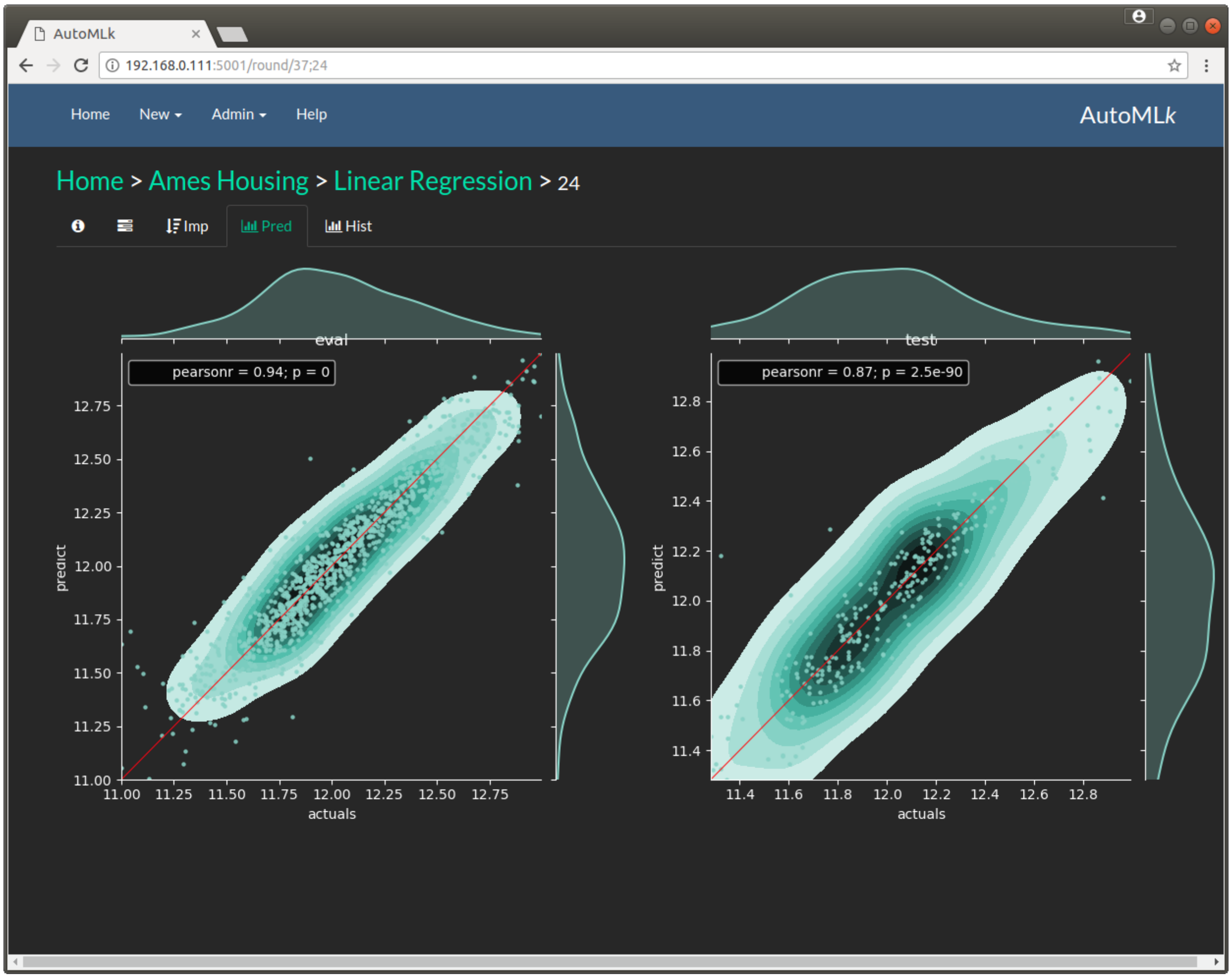


Fig. 1.17: predictions versus actuals (in regression)

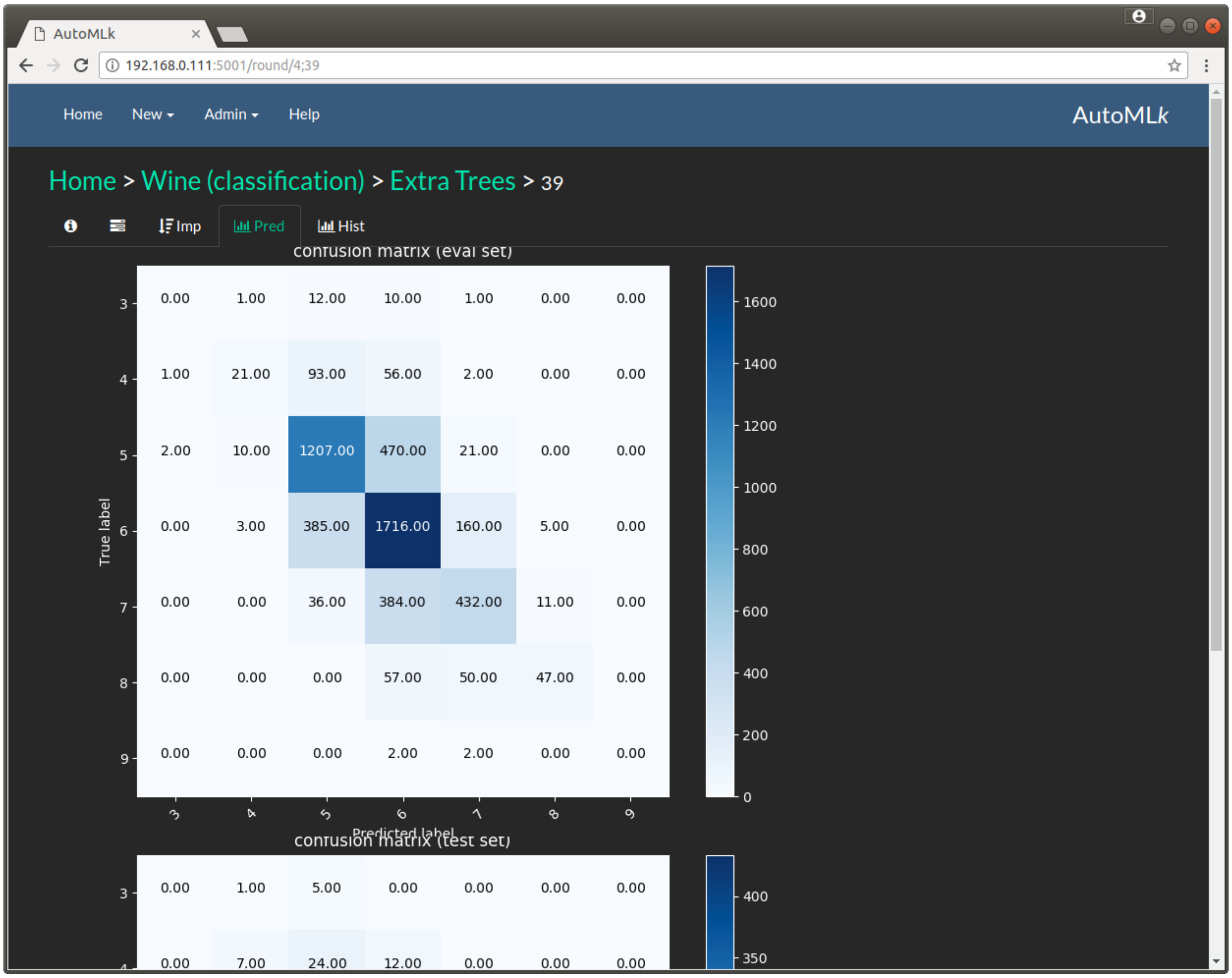


Fig. 1.18: and a confusion matrix (in classification)

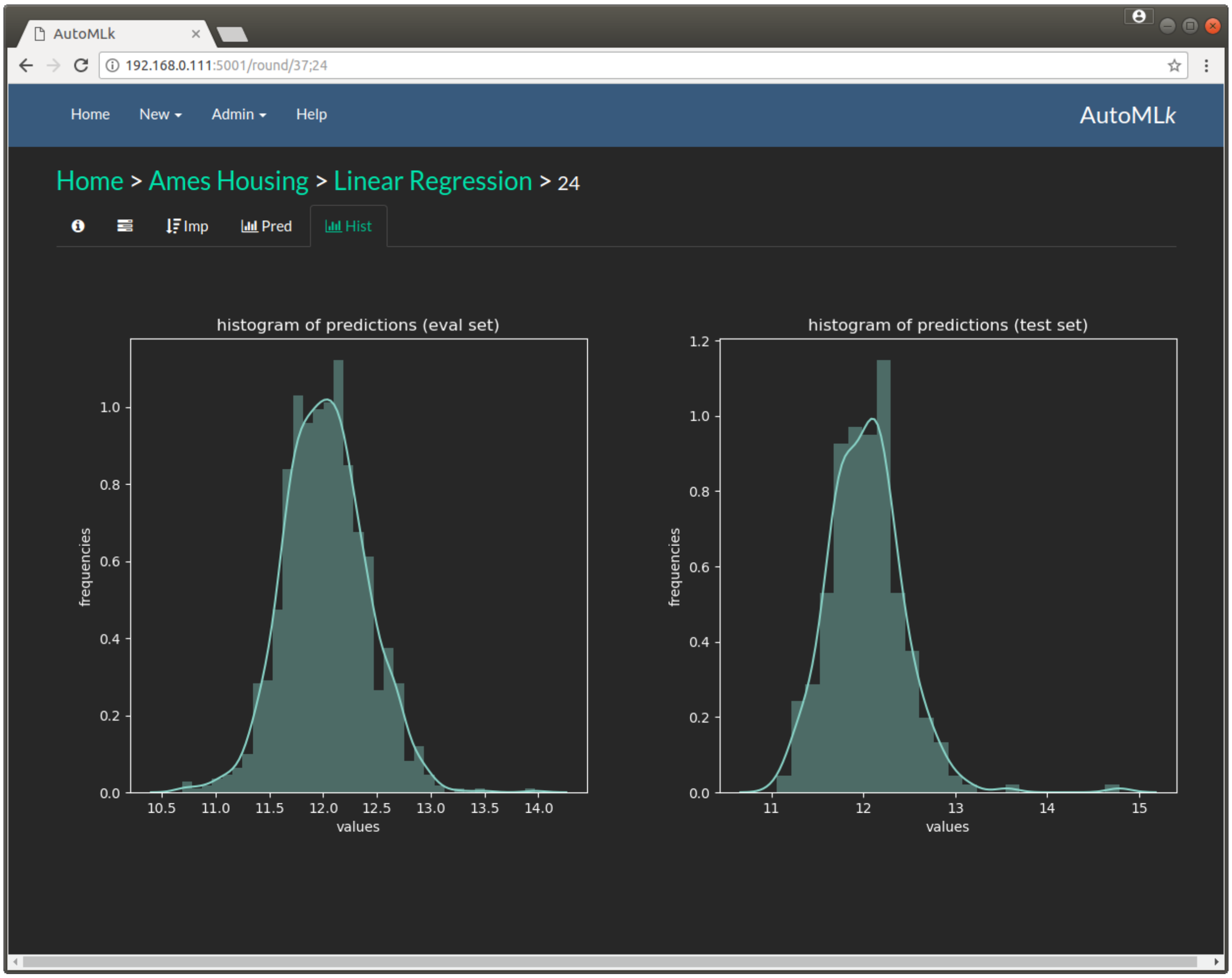


Fig. 1.19: and the histogram of the predictions

Admin

Monitoring

The monitoring screen displays the different status of the different components in the architecture: controller and workers

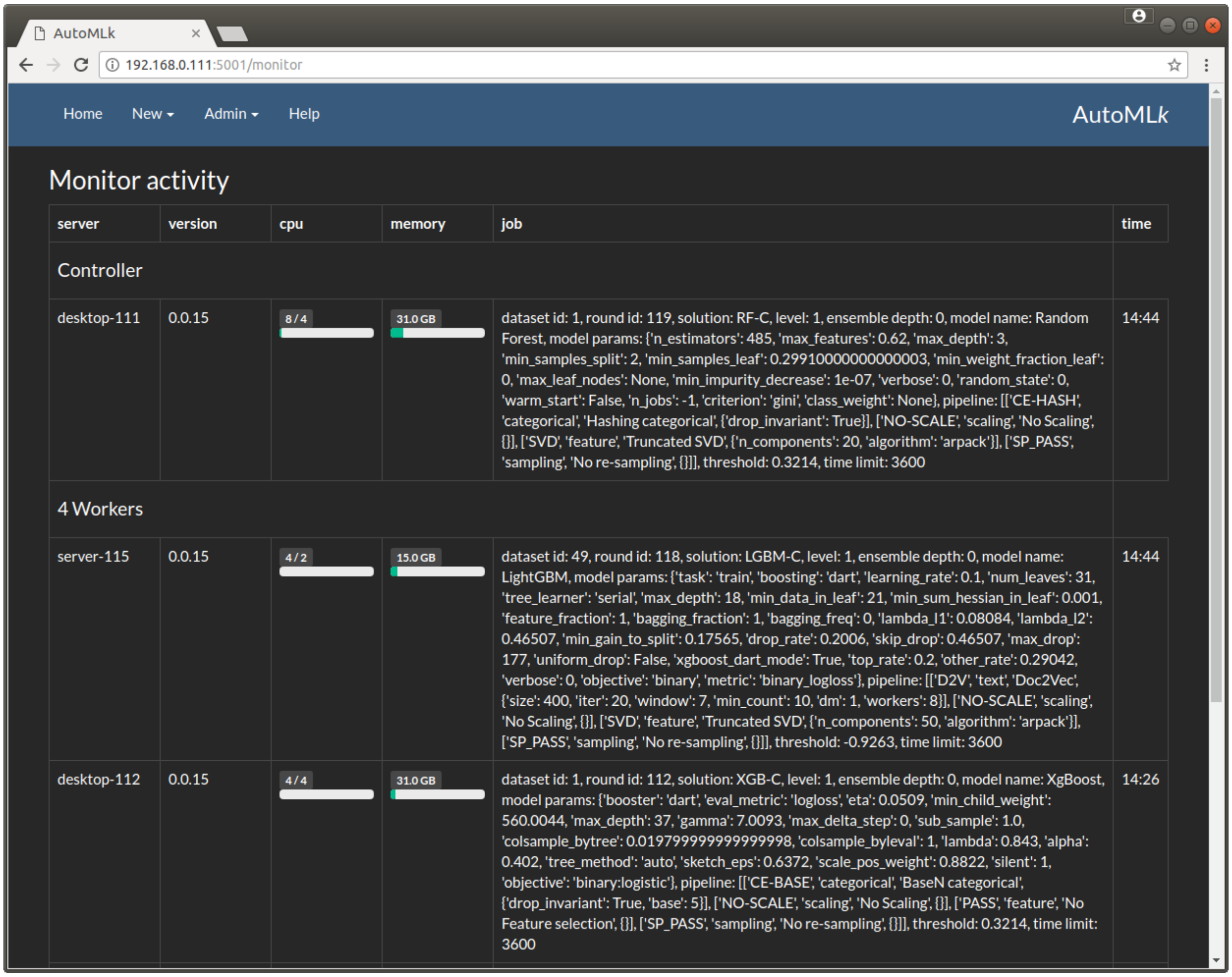


Fig. 1.20: monitoring panel

Config

It is also possible to modify the theme of the user interface directly from the config panel:

1.2 Installation

1.2.1 Pre-requisites

Sklearn version must be > 0.19, otherwise there will be several blocking issues.

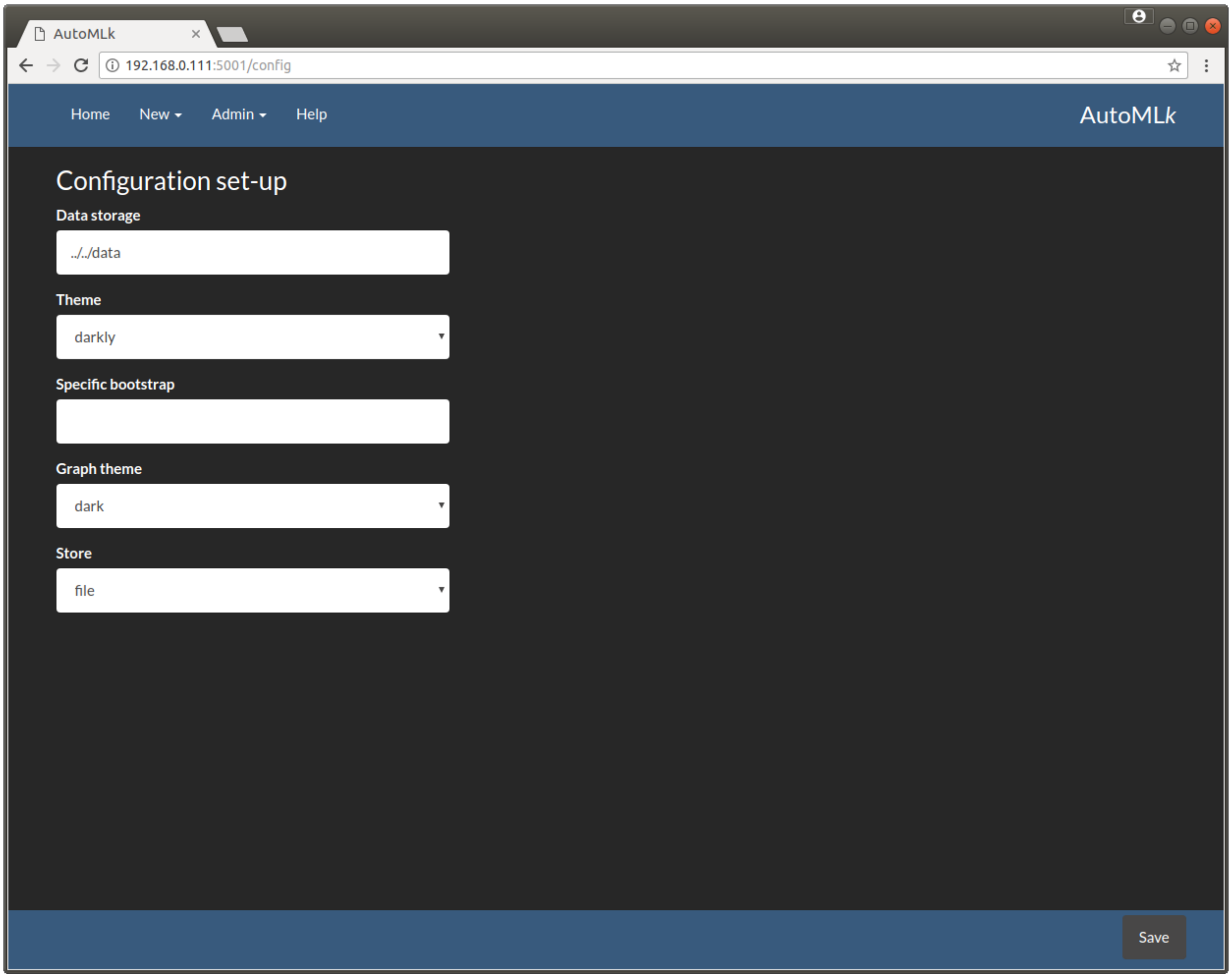


Fig. 1.21: configuration panel

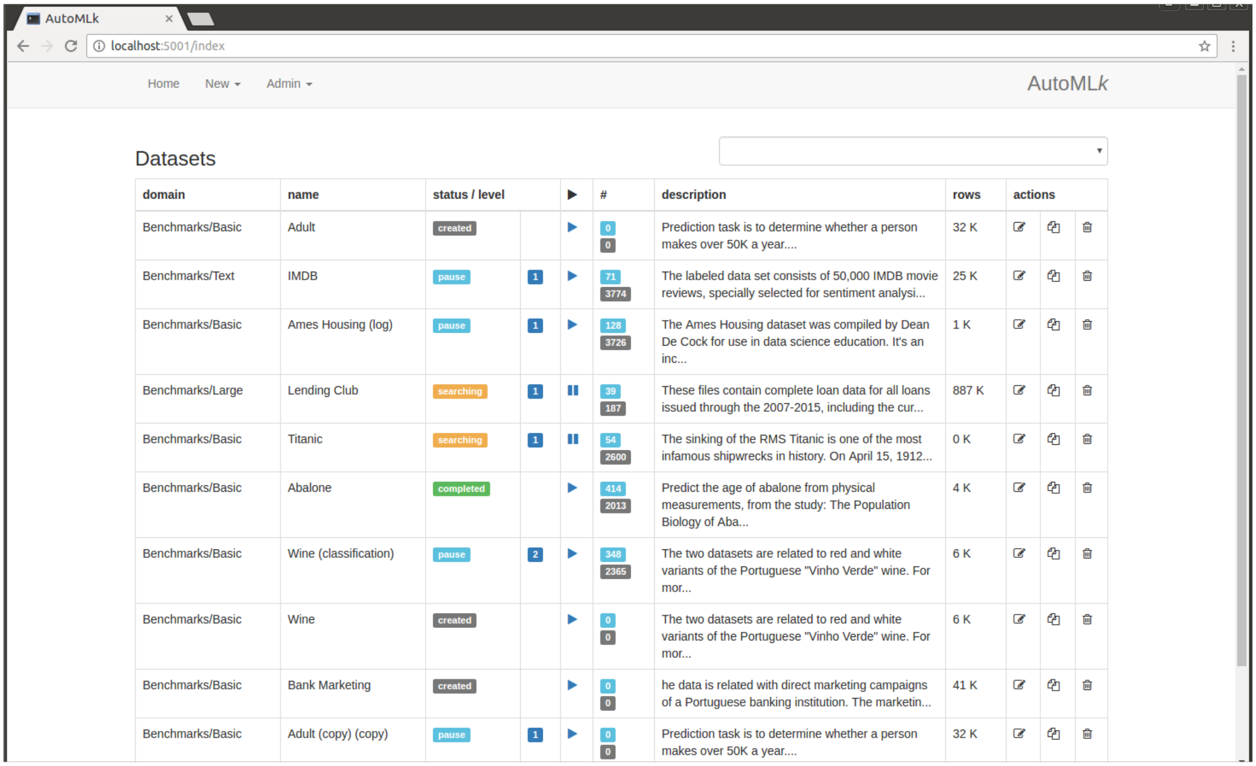


Fig. 1.22: configuration panel

to upgrade scikit-learn:

On conda:

```
conda update conda

conda update scikit-learn
```

If you do not use conda, update with pip:

```
pip install scikit-learn --update
```

Warning: if you use conda, you must absolutely update sklearn with conda

Additionally, you must also install category_encoders and imbalanced-learn:

```
pip install category_encoders
pip install imbalanced-learn
```

Optionally, you may install the following models:

- LightGBM (highly recommended, because it is very quick and efficient):

```
pip install lightgbm
```

- Xgboost (highly recommended, because it is also state of the art):

See Xgboost documentation for installation

- Catboost:

```
pip install catboost
```

- keras with theano or tensorflow:

See keras, theano or tensorflow documentation for installation

1.2.2 Installation

Download the module from github and extract the zip file in a folder (by default automlk-master)

Install as:

```
cd automlk-master

python setup.py install
```

1.2.3 Basic installation

The simplest installation runs on a single machine, with at least the following processes: 1. the web app 2. the controller, grapher and text worker 3. a single worker

These 3 components are run in a console (Windows) or Terminal (Linux).

The basic installation will use a data folder on the same machine. By default, the data folder should be created at one level upper the automlk-master folder.

For example, let's assume that autoMLk is created in the $HOME (Linux) level or Documents (windows):

- **home**
  - **pierre**
    - **automlk-master**
      - automlk
      - run
      - web
    - data

If you want to use a data folder in another location, you can define this in the config screen.

To run the web app:

```
cd automlk-master/web

python run.py
```

This will launch the web app, which can be accessed from a web browser, at the following address:

```
http://localhost:5001
```

From the web app, you can now define the set-up and then import the example of datasets.

You can launch the search in a dataset simply by clicking on the start/pause button in the home screen, and view the results through with the web interface. The search will continue automatically until the search is completed.

To run the controller, grapher et text manager:

```
cd automlk-master/run

python run_controller.py
python run_grapher.py
python run_worker_text.py
```

To run the workers on one or multiple machines:

On Linux:

```
cd automlk-master/run

sh worker.sh
```

On Windows:

```
cd automlk-master/run

worker
```

Note: This will run the python module ru_worker.py in an infinite loop, in order to catch the potential crashes from the worker.

1.2.4 Advanced configuration

Data server

The data are stored in a specific folder. In the default configuration, it is supposed to be on the same machine, and in the folder data. You may specify a different machine and location. The configuration is stored in the config.json file

{“data”: “../../data”, “theme”: “bootswatch/3.3.7/darkly”, “store”: “file”, “store_url”: “192.168.0.18”}

The data folder must be accessible by all the machines with the following components: - web server - controller - worker

Web server

The web server should be on a separate machine than the workers, in order to guarantee the response times for the user inferface.

If you want to use a data folder in another location, you can define this in the config screen.

To run the web app:

```
cd automlk-master/web

python run.py
```

This will launch the web app, which can be accessed from a web browser, at the following address:

```
http://localhost:5001
```

From the web app, you can now define the set-up and then import the example of datasets.

You can launch the search in a dataset simply by clicking on the start/pause button in the home screen, and view the results through with the web interface. The search will continue automatically until the search is completed.

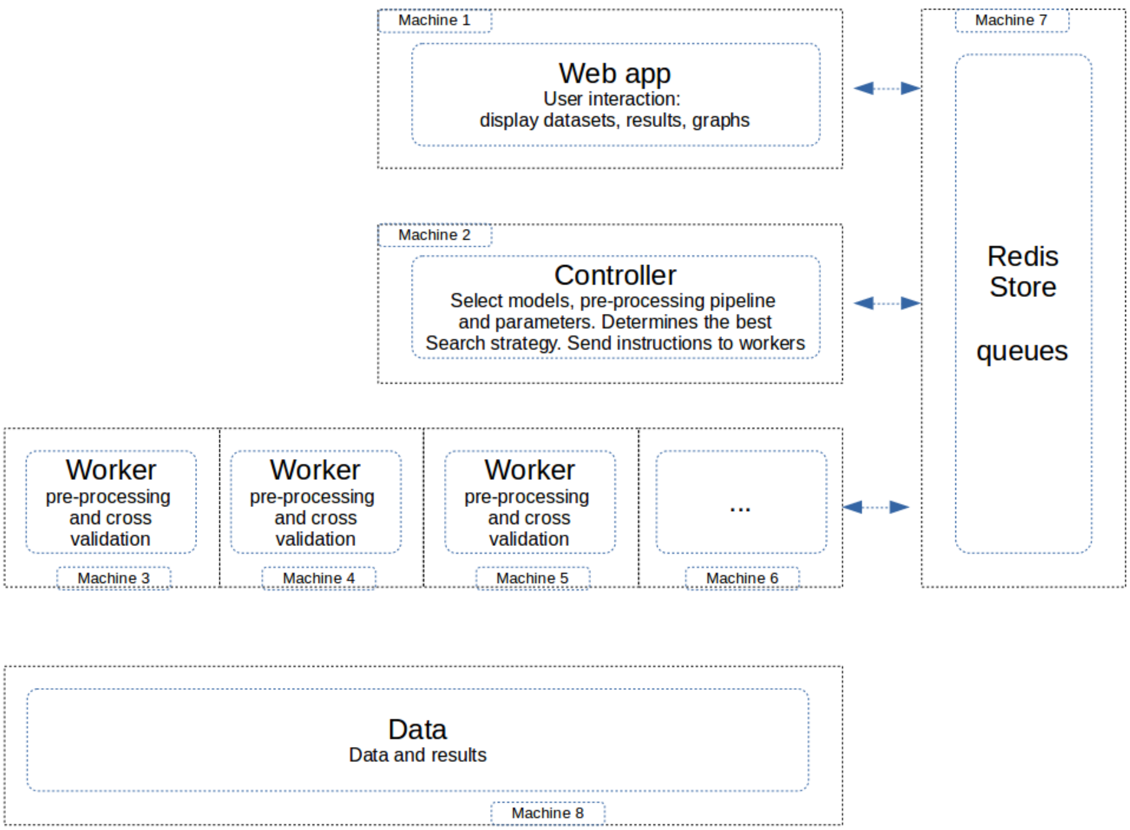


Fig. 1.23: independent components of the architecture

Store

The store by default is implemented using the file system, in he folder data/store, where 'data' is the folder defined for data storage.

The recommended mode is Redis, with the following advantages: - faster user experience of the web app, thanks to the in-memory storage of Redis which is very fast - more robust queuing and communication mecanism between controller and workers.

It is then highly recommended to use Redis for the store, when you have a cluster of multiple workers.

The installation of Redis is simple on Linux machines, and there is also a windows version available. Please see the Redis documentation directly to install and configure your Redis store.

The Redis server can be installed on the same machine as the web server.

Controller, grapher and text worker

The controller can be executed on the machine of the web server. It can also be installed if required on a specific machine.

It must be run in a standalone process, and we recommend that you install this process in a service (windows server) or a permanent process (Linux).

To run the controller:

```
cd automlk-master/run

python run_controller.py
python run_grapher.py
python run_worker_text.py
```

Workers

The workers are the components in the architecture with the most significant impact: the speed of search is directly proportional to the number of workers. We recommend to run at least 4 workers, and with multiple datasets to be searched simultaneously, a cluster of 10 to 20 machines should deliver great performance and speed.

To run the worker:

On Linux:

```
cd automlk-master/run

sh worker.sh
```

On Windows:

```
cd automlk-master/run

worker
```

Note: This will run the python module ru_worker.py in an infinite loop, in order to catch the potential crashes from the worker.

1.3 Architecture

The architecture is distributed and can be installed on multiple machines

- the web app for user interaction and display results
- the controller manages the search between models and parameters
- the grapher generates graphs on a dataset asynchronously
- the texter generates unsupervised models for text sets
- the workers execute the pre-processing steps and cross validation (cpu intensive): the more workers are run in parallel, the quicker the results
- the Redis store is an in-memory database and queue manager

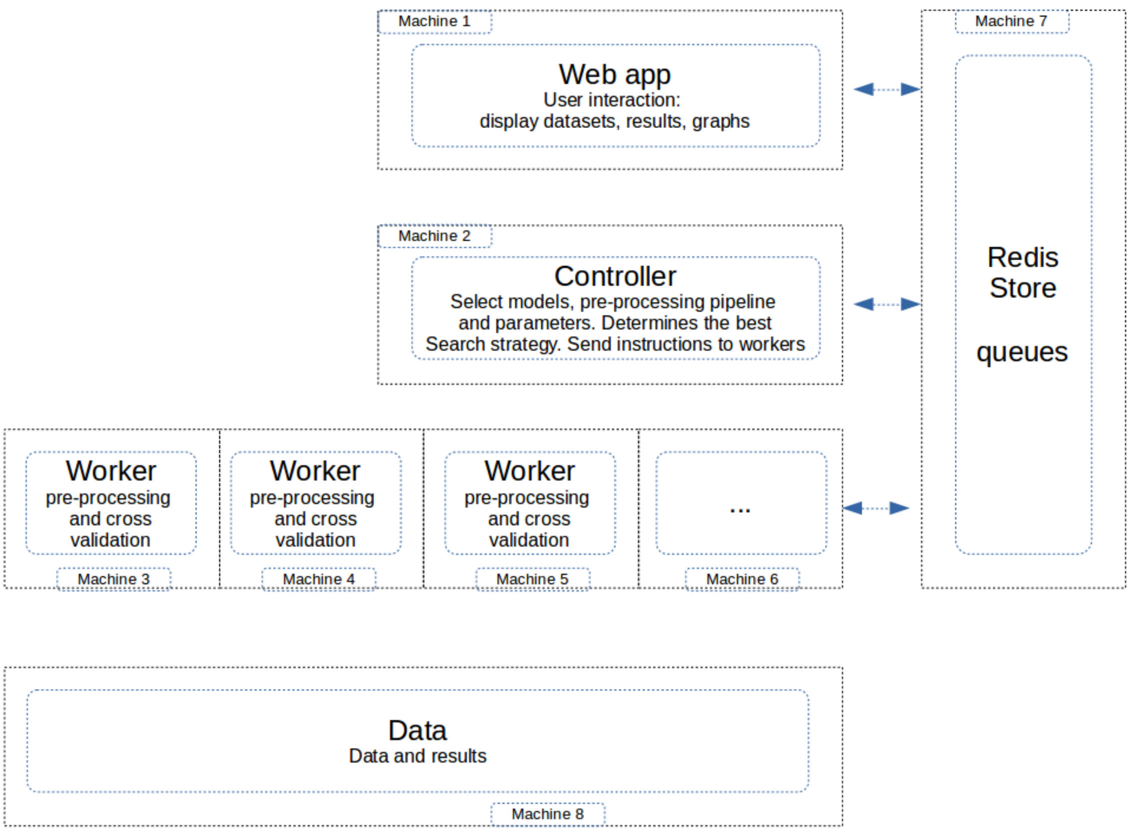


Fig. 1.24: independent components of the architecture

The software architecture is organized in concentric layers:

1.4 DataSet

The features of the automated machine learning are defined and stored in the DataSet object. All features and data of a DataSet object can be viewed with the web app.

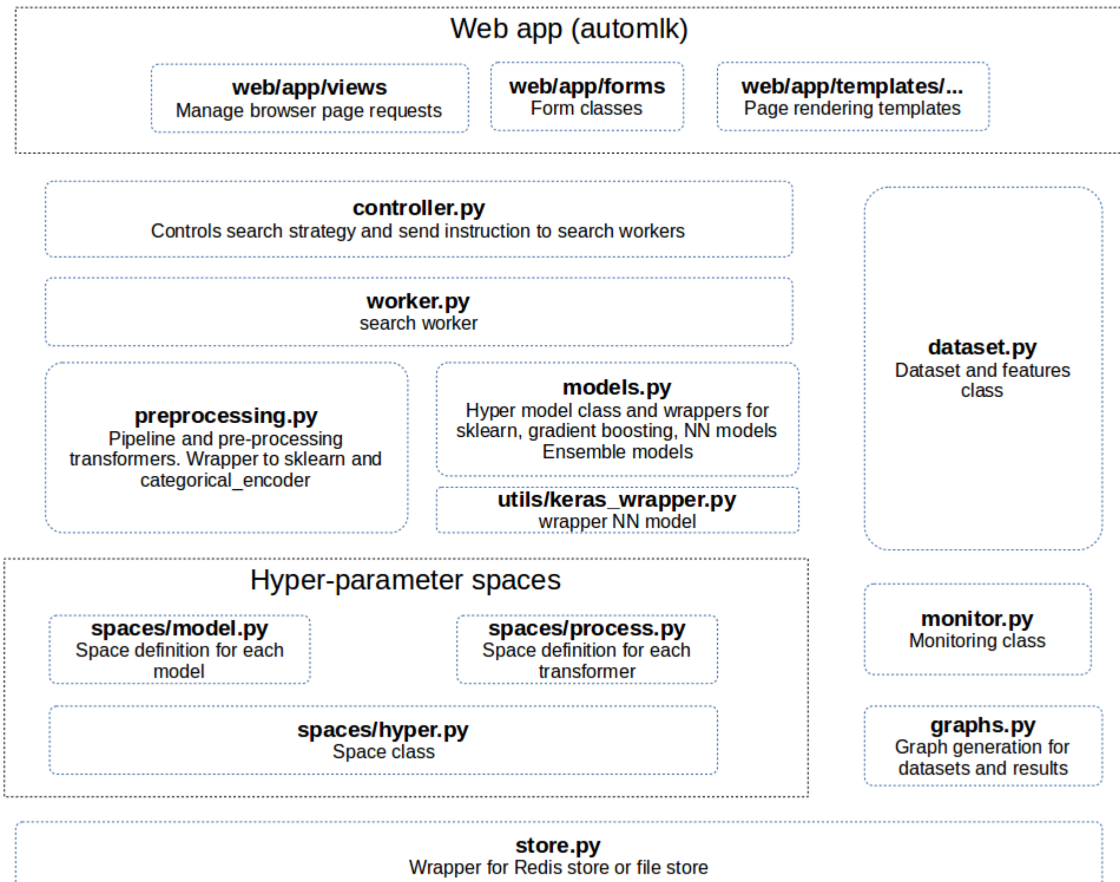


Fig. 1.25: software components of the architecture

We have included a sample of public datasets to start with autoMLk.

To use these datasets, upload the list of datasets or create a dataset in the New dataset from the menu.

the data describing these datasets are located in the csv file 'dataset.csv' in the automlk/datasets folder. You may use the same format to create your own datasets.

1.5 Searching

The automated search will test preprocessing steps and models.

1.6 List of models

The following models are included in autoMLk, with their respective hyper-parameters:

1.6.1 Models level 1

regression:

LightGBM *boosting_type, num_leaves, max_depth, learning_rate, n_estimators, min_split_gain, min_child_weight, min_child_samples, subsample, subsample_freq, colsample_bytree, reg_alpha, reg_lambda, verbose, objective, metric*

XgBoost *max_depth, learning_rate, n_estimators, booster, gamma, min_child_weight, max_delta_step, subsample, colsample_bytree, colsample_bylevel, reg_alpha, reg_lambda, scale_pos_weight, tree_method, sketch_eps, n_jobs, silent, objective, eval_metric*

CatBoost *learning_rate, depth, verbose*

Neural Networks *units, batch_size, batch_normalization, activation, optimizer, learning_rate, number_layers, dropout*

Extra Trees *n_estimators, max_features, max_depth, min_samples_split, min_samples_leaf, min_weight_fraction_leaf, max_leaf_nodes, min_impurity_decrease, verbose, random_state, warm_start, criterion*

Random Forest *n_estimators, max_features, max_depth, min_samples_split, min_samples_leaf, min_weight_fraction_leaf, max_leaf_nodes, min_impurity_decrease, verbose, random_state, warm_start, n_jobs, criterion*

Gradient Boosting *n_estimators, max_features, max_depth, min_samples_split, min_samples_leaf, min_weight_fraction_leaf, max_leaf_nodes, min_impurity_decrease, verbose, random_state, warm_start, learning_rate, loss*

AdaBoost *n_estimators, learning_rate, random_state, loss*

Knn *n_neighbors, weights, algorithm, leaf_size, p, n_jobs*

SVM *C, epsilon, kernel, degree, gamma, coef0, shrinking, tol, max_iter, verbose*

Linear SVR *C, loss, epsilon, dual, tol, fit_intercept, intercept_scaling, max_iter, verbose*

Linear Regression *fit_intercept, normalize, copy_X, n_jobs*

Ridge Regression *alpha, fit_intercept, normalize, copy_X, tol, solver*

Lasso Regression *alpha, fit_intercept, normalize, precompute, copy_X, tol, positive, selection*

Huber Regression *epsilon, alpha, fit_intercept, tol*

classification:

LightGBM *boosting_type, num_leaves, max_depth, learning_rate, n_estimators, min_split_gain, min_child_weight, min_child_samples, subsample, subsample_freq, colsample_bytree, reg_alpha, reg_lambda, verbose, objective, metric*

XgBoost *max_depth, learning_rate, n_estimators, booster, gamma, min_child_weight, max_delta_step, subsample, colsample_bytree, colsample_bylevel, reg_alpha, reg_lambda, scale_pos_weight, tree_method, sketch_eps, n_jobs, silent, objective, eval_metric*

CatBoost *learning_rate, depth, verbose*

Extra Trees *n_estimators, max_features, max_depth, min_samples_split, min_samples_leaf, min_weight_fraction_leaf, max_leaf_nodes, min_impurity_decrease, verbose, random_state, warm_start, n_jobs, criterion, class_weight*

Random Forest *n_estimators, max_features, max_depth, min_samples_split, min_samples_leaf, min_weight_fraction_leaf, max_leaf_nodes, min_impurity_decrease, verbose, random_state, warm_start, n_jobs, criterion, class_weight*

Gradient Boosting *n_estimators, max_features, max_depth, min_samples_split, min_samples_leaf, min_weight_fraction_leaf, max_leaf_nodes, min_impurity_decrease, verbose, random_state, warm_start, learning_rate, criterion, loss*

AdaBoost *n_estimators, learning_rate, random_state, algorithm*

Knn *n_neighbors, weights, algorithm, leaf_size, p, n_jobs*

SVM *C, kernel, degree, gamma, coef0, shrinking, tol, max_iter, verbose, probability*

Logistic Regression *penalty, dual, tol, C, fit_intercept, intercept_scaling, solver, max_iter, multi_class, n_jobs*

Naive Bayes Gaussian **

Naive Bayes Bernoulli *alpha, binarize, fit_prior*

Neural Networks *units, batch_size, batch_normalization, activation, optimizer, learning_rate, number_layers, dropout*

1.6.2 Ensembles

regression:

Stacking LightGBM *task, boosting, learning_rate, num_leaves, tree_learner, max_depth, min_data_in_leaf, min_sum_hessian_in_leaf, feature_fraction, bagging_fraction, bagging_freq, lambda_l1, lambda_l2, min_gain_to_split, drop_rate, skip_drop, max_drop, uniform_drop, xgboost_dart_mode, top_rate, other_rate, verbose, objective, metric*

Stacking XgBoost *booster, eval_metric, eta, min_child_weight, max_depth, gamma, max_delta_step, sub_sample, colsample_bytree, colsample_bylevel, lambda, alpha, tree_method, sketch_eps, scale_pos_weight, silent, objective*

Stacking Extra Trees *n_estimators, max_features, max_depth, min_samples_split, min_samples_leaf, min_weight_fraction_leaf, max_leaf_nodes, min_impurity_decrease, verbose, random_state, warm_start, criterion*

Stacking Random Forest *n_estimators, max_features, max_depth, min_samples_split, min_samples_leaf, min_weight_fraction_leaf, max_leaf_nodes, min_impurity_decrease, verbose, random_state, warm_start, n_jobs, criterion*

Stacking Gradient Boosting *n_estimators, max_features, max_depth, min_samples_split, min_samples_leaf, min_weight_fraction_leaf, max_leaf_nodes, min_impurity_decrease, verbose, random_state, warm_start, learning_rate, loss*

Stacking Linear Regression *fit_intercept, normalize, copy_X, n_jobs*

classification:

Stacking LightGBM *task, boosting, learning_rate, num_leaves, tree_learner, max_depth, min_data_in_leaf, min_sum_hessian_in_leaf, feature_fraction, bagging_fraction, bagging_freq, lambda_l1, lambda_l2, min_gain_to_split, drop_rate, skip_drop, max_drop, uniform_drop, xgboost_dart_mode, top_rate, other_rate, verbose, objective, metric*

Stacking XgBoost *booster, eval_metric, eta, min_child_weight, max_depth, gamma, max_delta_step, sub_sample, colsample_bytree, colsample_bylevel, lambda, alpha, tree_method, sketch_eps, scale_pos_weight, silent, objective*

Stacking Neural Networks *units, batch_size, batch_normalization, activation, optimizer, learning_rate, number_layers, dropout*

Stacking Extra Trees *n_estimators, max_features, max_depth, min_samples_split, min_samples_leaf, min_weight_fraction_leaf, max_leaf_nodes, min_impurity_decrease, verbose, random_state, warm_start, n_jobs, criterion, class_weight*

Stacking Random Forest *n_estimators, max_features, max_depth, min_samples_split, min_samples_leaf, min_weight_fraction_leaf, max_leaf_nodes, min_impurity_decrease, verbose, random_state, warm_start, n_jobs, criterion, class_weight*

Stacking Gradient Boosting *n_estimators, max_features, max_depth, min_samples_split, min_samples_leaf, min_weight_fraction_leaf, max_leaf_nodes, min_impurity_decrease, verbose, random_state, warm_start, learning_rate, criterion, loss*

Stacking Logistic Regression *penalty, dual, tol, C, fit_intercept, intercept_scaling, solver, max_iter, multi_class, n_jobs*

Stacking Neural Networks *units, batch_size, batch_normalization, activation, optimizer, learning_rate, number_layers, dropout*

1.7 Pre-processing steps

The following pre-processing methods are included in autoMLk, with their respective hyper-parameters:

1.7.1 categorical encoding:

No encoding **

Label Encoder **

One hot categorical *drop_invariant*

BaseN categorical *drop_invariant, base*

Hashing categorical *drop_invariant*

1.7.2 text encoding:

Bag of words

Word2Vec

Doc2Vec

1.7.3 imputing missing values:

No missing **

Missing values fixed *fixed*

Missing values frequencies *frequency*

1.7.4 feature scaling:

No scaling **

Scaling Standard **

Scaling MinMax **

Scaling MaxAbs **

Scaling Robust *quantile_range*

1.7.5 feature selection:

No Feature selection **

Truncated SVD *n_components, algorithm*

Fast ICA *n_components, algorithm*

PCA *n_components*

Selection RF *n_estimators*

Selection RF *n_estimators*

Selection LSVR **

CHAPTER 2

Indices

- genindex