OUR PROJECT

A PREPRINT

KTR Group
Faculty of Mathematics and Information Science
Warsaw University of Technology

June 2, 2022

ABSTRACT

TODO

1 Outline of introduction

Sources that we will be quoting:

- https://microsoft.github.io/FLAML/docs/reference/model – everything about the model
- https://new.openml.org/search?type=benchmarksort=tasks_includedstudy_type=task&id=218 – data sets for evaluation

2 Description of FLAML framework

FLAML (Fast and Lightweight AutoML) is a Python library, developed by Microsoft under Open Source license, used for Auto Machine Learning (AutoML). According to its creators, FLAML is a lightweight package that finds the right machine learning model for given task in an automatic, effective and economic way. Hence it should free a user from the necessity of choosing an optimal machine learning model and its hyperparameters. This framework supports both conventional machine learning methods - quickly finds qualitative models while using low amount of computational resources for common tasks, such as classification or regression - and deep learning.

The main class of the package is `flaml.AutoML`. Its constructor takes kwarg `settings`, which includes time budget, metrics, task (classification, regression, etc.), list of models to be used and others. The most important method, just like in scikit-learn package, is `fit()`, which takes data frame of features X and label y (it can also overwrite settings). While calling this method suitable models are fitted with data and optimization of hyperparameters is performed on them. With help of `best_estimator` and `best_config` attributes it is possible to obtain the best model and its configuration of hyperparameters acquired by FLAML.

Apart from `fit()` we can call `predict()` and `predict_proba()` methods, which allows compatibility with scikit-learn package.

Another advantage of FLAML is availability of personalising software - the package can handle an user defined function (most conveniently by using functions compatible with scikit-learn), so ones including `fit()` and `predict()` methods) and use an exterior metrics.

Module `tune` of this library enables acquiring the right parameters for given model. It is used internally by `flaml.AutoML`, but it can also be used directly by the user, meeting one of the conditions:

- user's task is not one of the built-in `flaml.AutoML` tasks
- user's input data cannot be presented as features data frame and label column
- the user wants to tune a function that does not qualify as a machine learning procedure

A considerable disadvantage of the package is its incomplete documentation. For example, preprocessing done by the package is not described in any place (there was only a mention of `DataTransformer` class in SDK), which led us to do our own preprocessing.

3 Actions necessary to engage to make the package work properly

Due to poor documentation, we reckoned that the package does not perform preprocessing of data. Hence to eliminate undefined behaviour we had to perform preprocessing, so that the user does not have to handle this matter. Actions that we decided to include are:

- removing columns that are IDs
- imputation by mode
- replacing outliers with ceiling value
- normalization with `QuantileTransformer`
- scaling with `StandardScaler`
- encoding of categorical data with `WoEEncoder`

We chose that kind of preprocessing not only to guarantee the lack of undefined actions in the case of binary classification, but also to avoid overfitting and to acquire better models. It turns out however that FLAML includes necessary preprocessing for the package to work. Similarly to ours, it removes columns having zero variance and imputes missing data either with a new category in case of categorical features, or with a median in case of numerical ones. It also fragments dates to smaller components.

4 Evaluation results

As an initial evaluation we checked the performance of a function created by us on twenty data sets from OpenML, included in a task available under the link (we have not used each of the data sets included). 13 out of these data sets represent binary classification. On each of them we performed the `fit()` method from a `OurFlaml` class created by us (for that to happen we had to download each of the data sets split it into features data frame and label column). We appended the best ROC AUC score obtained by the function, the time to train the best configuration and the name of the data set to appropriate lists. Then we visualized the results:

5 OurML

We created an AutoML pipeline designed to solve binary classification tasks and compared its performance to FLAML. Our solution is using scikit-learn API and consists of applying simple preprocessing methods and training a few different learners while tuning their hyperparameters using cross-validated (number of folds is passed to the fit() method, by default it uses 5-fold cross-validation) randomized search. The best model is chosen using ROC AUC metric.

5.1 Preprocessing

To ensure that all the models trained and tested in the process of fitting our estimator work properly, we created a preprocessing pipeline, which should work for most datasets and produce great results. It consists of several steps:

- removal of features with zero variance
- imputation of missing data with the most frequent value strategy
- substitution of values beyond .975 quantile and .025 with the threshold value
- quantile transform to make the distribution of values resemble uniform distribution
- scaling the data with mean and variance

For categorical variables:

- grouping rare labels to form bigger groups
- encoding labels with Word of Evidence encoder.

5.2 Model selection

Same as FLAML, we used a predefined selection of estimators and corresponding hyperparameter search spaces over which a random search is performed. Best model is selected using predefined metric (by default ROC AUC). This is a very crude and simplified copy of FLAMLs algorithm which performs a much more complicated search and provides a way to significantly reduce training time.

5.2.1 Used learners and their search spaces

We selected three learners:

- Logistic regression
- Random forest
- Gradient boosting

Hyperparameter	Search space
C	0.001, 0.01, 0.1, 1, 10, 100, 1000
penalty	L1, L2
solver	saga

Table 1: Logistic regression hyperparameter search space

Hyperparameter	Search space
bootstrap	True, False
max_depth	10, 25, 50, 75, 100, None
max_features	log2, sqrt
min_samples_leaf	1, 2, 4
min_samples_split	2, 5, 10
n_estimators	250, 500, 750, 1000, 1250, 1500, 1750, 2000

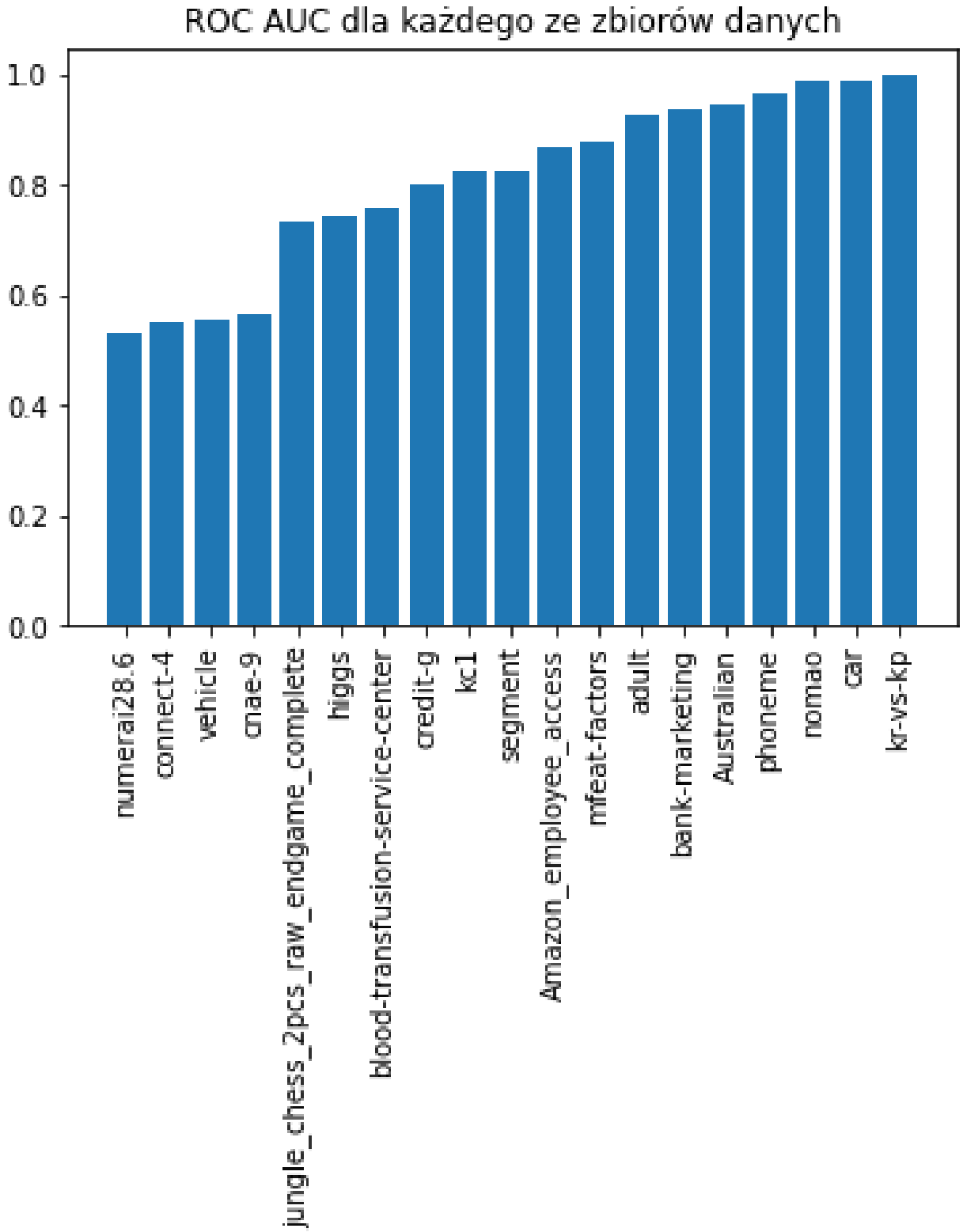
Table 2: Random forest hyperparameter search space

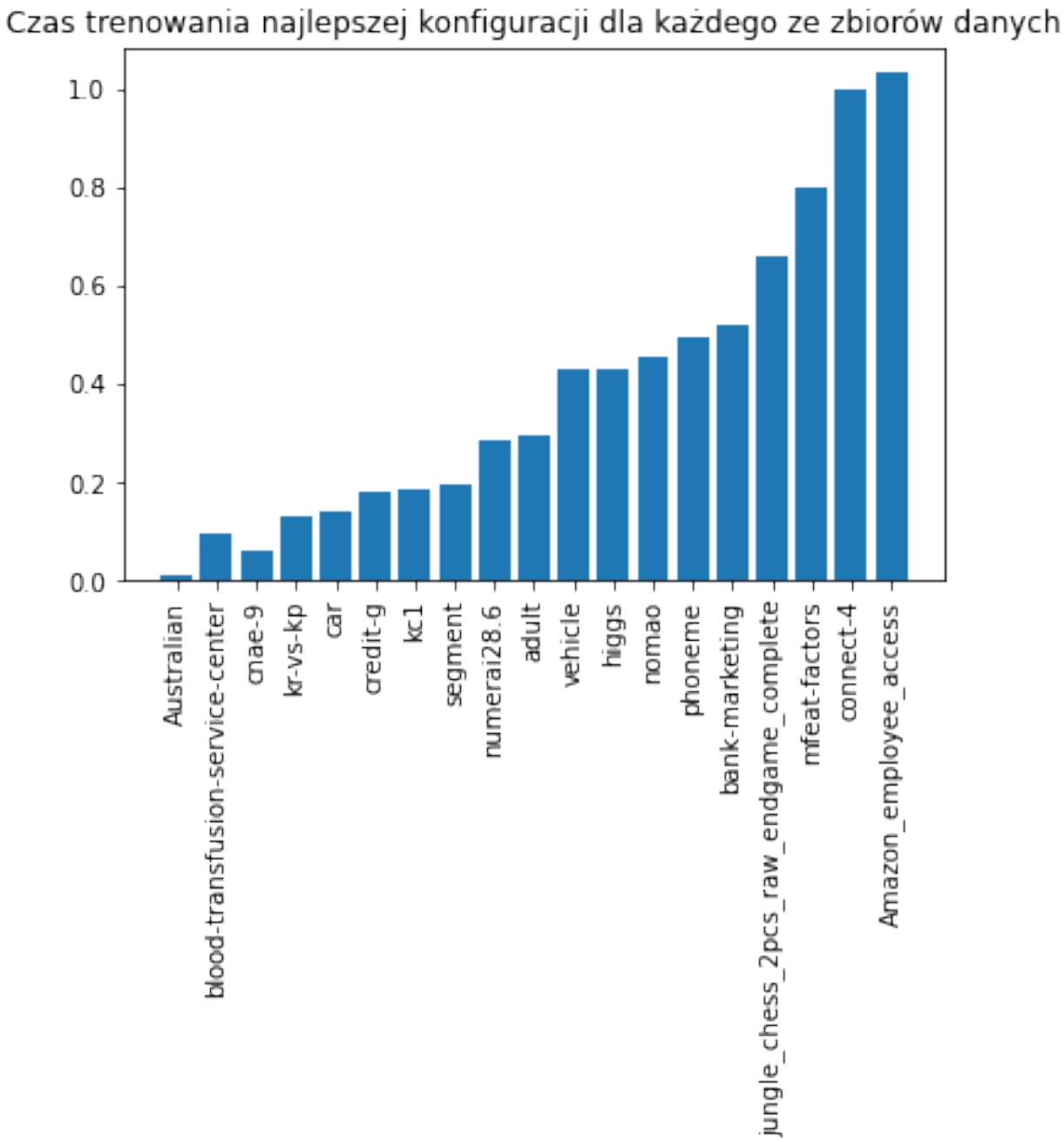
Hyperparameter	Search space
learning_rate	0.01, 0.05, 0.1, 0.2
min_samples_split	0.1, 0.136, 0.178, 0.209, 0.245, 0.281, 0.318, 0.354, 0.390, 0.427, 0.463, 0.5
min_samples_leaf	0.1, 0.136, 0.178, 0.209, 0.245, 0.281, 0.318, 0.354, 0.390, 0.427, 0.463, 0.5
max_depth	3, 5, 8
min_samples_leaf	1, 2, 4
min_samples_split	2, 5, 10
subsample	0.5, 0.8, 0.9, 1.0

Table 3: Gradient boosting hyperparameter search space

5.3 Performance

Due to the way the search is performed and number of models the fit time of our estimator is not optimal. A way to improve on that would be to further refine the way hyperparameter tuning is performed, use models with quicker learning time or provide low computational cost default values to use instead of searching the whole space with each training.


ROC AUC dla każdego ze zbiorów danych
1.0
0.8
0.6
0.4
0.2
0.0
numerai28.6
connect-4
vehicle
cnae-9
jungle_chess_2pcs_raw_endgame_complete
higgs
blood-transfusion-service-center
credit-g
kc1
segment
Amazon_employee_access
mfeat-factors
adult
bank-marketing
Australian
phoneme
nomao
car
kr-vs-kp


Czas trenowania najlepszej konfiguracji dla każdego ze zbiorów danych
1.0
0.8
0.6
0.4
0.2
0.0
Australian
blood-transfusion-service-center
cnae-9
kr-vs-kp
car
credit-g
kc1
segment
numerai28.6
adult
vehicle
higgs
nomao
phoneme
bank-marketing
jungle_chess_2pcs_raw_endgame_complete
mfeat-factors
connect-4
Amazon_employee_access