A COMPARISON BETWEEN FLAML AND A SIMPLE AUTOML SOLUTION

Michał Tomczyk, Mikołaj Roguski, Michał Komorowski
Faculty of Mathematics and Information Science
Warsaw University of Technology

June 8, 2022

ABSTRACT

Machine learning is becoming more popular with each day. While this branch of IT is useful in many areas, there is a need to create machine learning frameworks that can be used quickly, easily and do not require any advanced knowledge of IT. An answer to that need is Auto Machine Learning.

There are many AutoML frameworks and FLAML is one of them. It is easy to use and highly customizable. We tested this Python package on binary classification datasets to check its performance, we examined how it works and how it can be used by someone with a limited machine learning knowledge.

We also tried to create our own AutoML package - to see how difficult it is to create a tool for auto machine learning and to compare performance of a relatively simple solution with FLAML's advanced search algorithms and vast space of models and hyperparameters.

1 Introduction

There is no debate that machine learning is an important branch of IT. Even though it can be a powerful tool, the classical approach to machine learning is not always right. It requires manually performing each step - EDA, preprocessing of data, choosing the best model, fitting it and finding the best hyperparmeters for it. This list of tasks can overwhelm a person with lacking experience in statistics or machine learning itself. On the other hand, quite often we use the same steps to prepare the data and choose the optimal model - even though each dataset is slightly different, the same preprocessing techniques can be applied as well as hyperparameter selection can be implemented in the same way for each dataset with a specific type of problem to solve.

This creates the need for another approach - automatic one, where the user can be relieved from the necessity of manually applying the same techniques for each dataset. Auto Machine Learning (also referred to as AutoML) is a rapidly growing subcategory of machine learning. Presumably it is supposed to work quickly, efficiently and be easy to use. Multiple approaches to AutoML have been implemented.

One of the popular AutoML frameworks for Python is FLAML. In the first part of our work we describe this package. We try to make the reader understand how to use FLAML, what are its main components and what philosophy motivated its developers. We relied on an article introducing FLAML [1] and on its documentation for technical insight into the package. We also created our own AutoML tool that can be used for binary classification. For that purpose, we relied on different functions from the `scikit-learn` package. We wanted to check how a sophisticated software like FLAML compares to a simplified solution that uses only few models. We benchmarked both softwares on 11 datasets from the OpenML package [2]

2 Description of FLAML framework

FLAML (Fast and Lightweight AutoML) is a Python library, developed by Microsoft under the MIT 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, regression, time series forecasting and deep learning.

Most of the frameworks usability is provided in a single class, an estimator, which complies with sklearn's API, so that it may be used in pipelines. This class trains all of the provided estimators, optimizes their hyperparameters and finally selects the best one.

One of the biggest advantages of FLAML is its customizability - the metric, estimator selection, search spaces and time limit for training can be specified by the user.

Module `tune` of this library enables acquiring the right hyperparameters for a given estimator. 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

Thanks to `tune` the user can quickly and easily optimize a custom function, even if it not a machine learning function.

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 and a single sentence saying that it transforms data), which led us to do our own preprocessing.

3 Actions necessary 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 ourselves, so that the user does not have to handle this matter. Actions that we decided to include are:

- removing columns with zero variance
- 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 behaviuors 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 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.

4.1 Preprocessing

We used the same preprocessing as for Flaml as we decided it would make the comparison fairer.

4.2 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.

4.2.1 Used estimators 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

4.3 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 training time or provide low computational cost default values to use instead of searching the whole space with each training.

5 Evaluation results

To evaluate the performance of both functions we tested them on eleven binary classification datasets from OpenML, included in a task available under the link (we have not used each of the data sets included due to time and resources consumption needed for testing all datasets). We relied on an article [3] about benchmark in OpenML, which included datasets from the above-mentioned task. We examined the ROC AUC Score achieved for each dataset by best model and hyperparameter configuration found by both FLAML (using the created by us function) and OurML by executing the `fit()` method. For `OurML` we also examined the time of executing `fit()` function on each dataset 3, as the time needed to find the best configuration varies depending on the dataset. FLAML has a time budget set by the user - independently from the dataset with a default value of 60 seconds, therefore examining the execution time in that case is pointless. We plotted results of benchmark for both `FLAML` 1 and `OurML` 2 using matplotlib package in Python.

Figure 1: ROC AUC scores for each dataset in benchmark for OurML

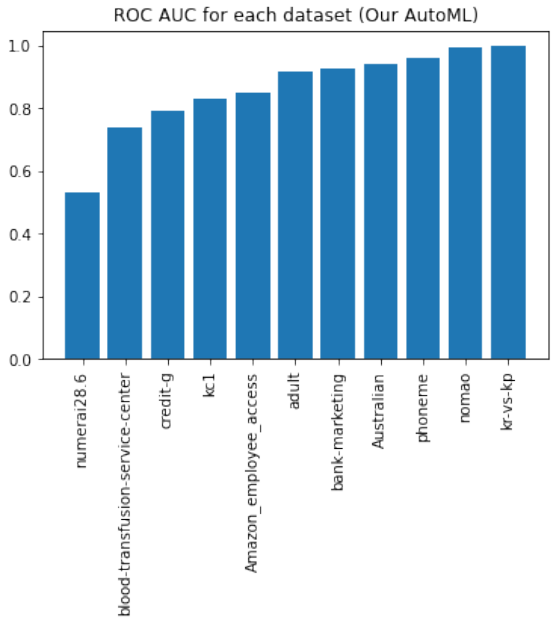
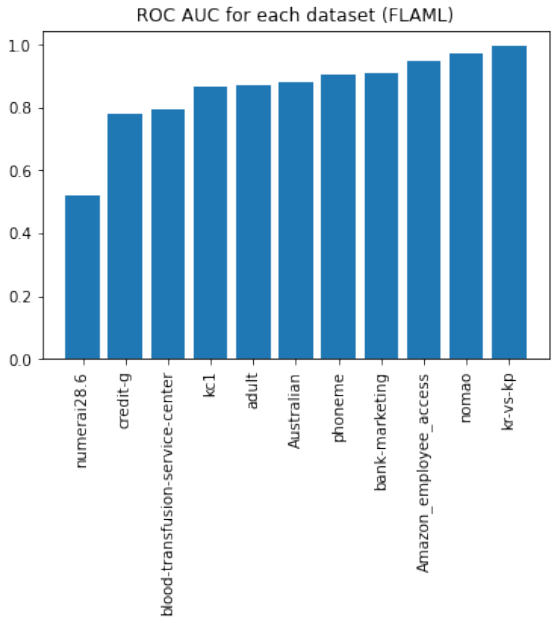


Figure 2: ROC AUC scores for each dataset in benchmark for FLAML package



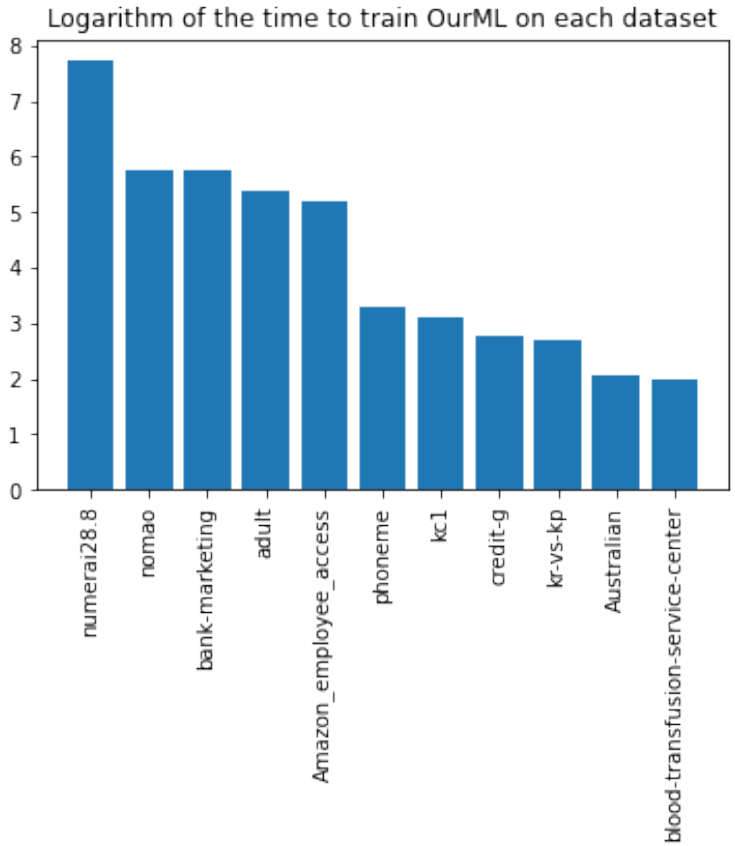
To examine more closely the score achieved on each dataset, we created a table of scores:

As we can see, the results vary - for some datasets FLAML perform better, for others OurML. Looking at the plot of the time of execution of OurML 3 we can see that often OurML performs better then FLAML when its time of execution is high. That is a reason of time budget for FLAML, set in this benchmark for 60 seconds. If Microsoft's package had received as much time as OurML, it would have outperformed OurML more often.

Dataset	ROC AUC for FLAML	ROC AUC for OurML
credit-g	0.7780	0.7898
kr-vs-kp	0.9956	0.9990
blood-transfusion-service-center	0.7954	0.7410
kc1	0.8663	0.8304
phoneme	0.9029	0.9586
bank-marketing	0.9089	0.9295
nomao	0.9723	0.9946
Amazon_employee_access	0.9493	0.8484
adult	0.8702	0.9160
Australian	0.8797	0.9405
numerai28.6	0.5206	0.5304

Table 4: A comparison of ROC AUC scores for FLAML and OurML

Figure 3: Time consumption of executing the `fit()` method of OurML for each dataset



References

[1] Chi Wang and Qingyun Wu. FLAML: A Fast and Lightweight AutoML Library. *CoRR*, abs/1911.04706, 2019.

[2] Joaquin Vanschoren, Jan N. van Rijn, Bernd Bischl, and Luís Torgo. OpenML: networked science in machine learning. *CoRR*, abs/1407.7722, 2014.

[3] Pieter Gijsbers, Erin LeDell, Janek Thomas, Sébastien Poirier, Bernd Bischl, and Joaquin Vanschoren. An open source automl benchmark. *CoRR*, abs/1907.00909, 2019.