ARCANE KNOWLEDGE OF AUTOMATED MACHINE LEARNING

A PREPRINT

Tomasz Siudalski **Grzegorz Zbrzeżny** **Piotr Marciniak**

ABSTRACT

The rapid growth of machine learning applications has created an increasing demand for ML methods that can be used easily and without deep knowledge. Automated Machine Learning aims to resolve this problem by automating time-consuming tasks e.g. data preprocessing, model selection and hyperparameter tuning. In recent years many AutoML tools arised on the market. We explore two of the existing AutoML frameworks, rating their usage and performance. Then with collected experience we propose our own AutoML tool that achieves similar performance on tested data and is easy to use. Our results prove that AutoML is suitable for many problems and yields mostly satisfying results.

1 Introduction

In the introduction we would like to present some essential thoughts about Auto Machine Learning. It is important to know why the whole idea of automating machine learning processes was born and how this approach varies from the original. Firstly Auto ML could be very beneficial in solving some ML problems. Moreover we can spare huge amount of time using automatic algorithms for creating models instead of hand coding it, because it allows to automate time-consuming, iterative tasks. Traditional approach requires some domain knowledge to create and compare dozens of models, Auto ML does not really need it at all producing models with high scale, efficiency while maintaining quality. But unfortunately there are also some serious drawbacks. For example, you don't get as much insights on data, so it might be harder to interpret ML models. Apart from that, sometimes quality of models would not be as high as we wished.

There are different approaches to Auto ML represented by multiple implementations. One of them is GAMA ([Gijsbers and Vanschoren, 2020]), which uses different methods to optimize scores. There are three optimization algorithms, which search for optimal machine learning pipelines: random search, an asynchronous successive halving algorithm and an asynchronous multi-objective evolutionary algorithm. It uses genetic algorithms to optimize score.

Another one is Autosklearn ([Feurer et al., 2015]), which uses meta-learning for initializing the Bayesian optimizer and automated ensemble construction. It also uses meta-features (i.e characteristics of the dataset), which can be computed efficiently, to choose which algorithm to use on a new dataset. These meta-features are complimentary to Bayesian optimization. They can quickly suggest some ML algorithms that are likely to perform well. But they don't work well with the high dimensional configuration spaces.

There is a newer version of the framework called Autosklearn2.0 ([Feurer et al., 2020]). It presents two new parts, which improved performance of package. One of them was Portfolio Successive Halving. Another one was proposal of model-base policy selector to automatically choose the best optimization policy for an AutoML system for a given dataset. That is the implementation chosen by us to perform various machine learning tasks.

2 Autosklearn 2.0 description

Auto-sklearn is an automated machine learning toolkit and a drop-in replacement for a scikit-learn estimator. It allows its users to automate:

- data preprocessing
- feature preprocessing
- hyperparameter optimization
- model selection
- model evaluation

Auto-sklearn 2.0 is currently suitable only for classification. It chooses from sklearn-based models. The space of possible models currently spans 16 classifiers. Auto-sklearn performs various preprocessing techniques such as balancing the dataset using class weights, imputing missing values using mean, median or mode, rescaling the data, by default using standardization. Specifically for categorical features it chooses between One Hot Encoding and no preprocessing. It also coerces rarely occurring categories, by default these smaller than 0.01 of the observations. It also performs quantile Transformation by default with 1000 quantiles and uniform output distribution. When it finds out there are outliers it handles them by removing the median and scaling the data according to the quantile range.

After data pre-processing, features may be optionally pre-processed with one or more of the categories of feature pre-processors. Autosklearn 2.0 might perform matrix decomposition using PCA, truncated SCV, kernel PCA or ICA, univariate feature selection or feature selection, feature clustering or embeddings, kernel approximations, polynomial feature expansion or sparse representation and transformation. For Auto-sklearn to find the best performing pipeline in given time from vast searching space the authors constructed a portfolio consisting of high-performing and complementary ML pipelines to perform well on as many datasets as possible offline. Then for a dataset at hand all pipelines in this portfolio are simply evaluated one after the other and if there is time left afterwards, the algorithm continues with pipelines suggested by Bayesian Optimization warmstarted with the evaluated portfolio pipelines.

Auto-sklearn uses use an early-stopping strategy inside the whole search space which performs well on large datasets, but it's mostly useful for tree-based classifiers. To improve model selection the framework uses a multi-fidelity optimization method such as BOHB and several different strategies, including holdout and cross-validation. It also builds an automated policy selection on top of the previous improvements to select the best strategy.

3 Our preprocessing made for Auto-sklearn 2.0

We made some functions responsible for the whole preprocessing. It is worth mentioning that we assume every column including data connected with dates in given set is in Datetime type. First step of preprocess is to make some changes in Datetime columns. We can not leave them without a change, because Auto sklearn 2.0 would treat them as a categorical column, thus it would be One Hot Encoded. This kind of operation creates many columns, which does not really contribute to any predictions. That is why we divide values in each row into three new columns, first of them is a day, second one is month and last of them is the year of the Datetime object. Then the original column is removed from data set. Second step is to change numerical columns which contains no more than 10 unique values to categorical type.

Our framework does not know how to operate on columns with type object, so we decided to convert every object type column into categorical type. Sometimes we can come across a column in a data set which contains only NA or NaN values. It does not give us any information, that is why we remove them before giving the data to our framework.

4 Methodology

To test our framework we decided to use data sets from OpenML ([Gijsbers et al., 2019]). We chose twenty-two sets related to binary classification issue. We used ten fold cross-validation on each set to test performance. To rate the scores received for each set we decided to use two metrics, accuracy score and roc auc score.

5 Results

Our first results was surely not the best one, because we gave our framework only sixty second on each fold due to an *No space left on device* error. After we managed to solve a problem with this error we run a benchmark with 5 minutes per fold. With this settings on 8 out of 22 sets we get Dummy Classifier, so we tried to run an algorithm on these datasets using a time budget of 10 minutes per fold without any success. In lack of improvements can be involved the platform, which we used to run algorithms, because many times we encountered the situation in which a smac folder was not created. The reason behind this can be a starvation of a process involved in creating it. When we tried to run an benchmark on another computer, the case with smac folder did not occur. The results for 14 sets can be seen below.

dataset	ROC AUC	Accuracy
Amazon_employee_access	0.872223 ± 0.011111	0.942476 ± 0.000885
Australian	0.936038 ± 0.021857	0.855072 ± 0.050204
adult	0.922539 ± 0.004531	0.842594 ± 0.035044
bank-marketing	0.933456 ± 0.007257	0.889385 ± 0.010957
blood-transfusion-service-center	0.749903 ± 0.041646	0.718703 ± 0.173707
credit-g	0.79319 ± 0.03691	0.771 ± 0.033149
higgs	0.800174 ± 0.006334	0.721601 ± 0.004806
jasmine	0.87574 ± 0.016639	0.815672 ± 0.019221
kc1	0.837224 ± 0.03377	0.867233 ± 0.01534
kr-vs-kp	0.99989 ± 0.000215	0.996245 ± 0.003848
nomao	0.993504 ± 0.000931	0.946295 ± 0.03
numerai28.6	0.530174 ± 0.00468	0.520577 ± 0.004589
phoneme	0.958366 ± 0.008585	0.872146 ± 0.041322
sylvine	0.983737 ± 0.003638	0.939112 ± 0.007332

Table 1: Performance on 14 out of 22 sets. The value presented is average with standard deviation across the 10 folds.

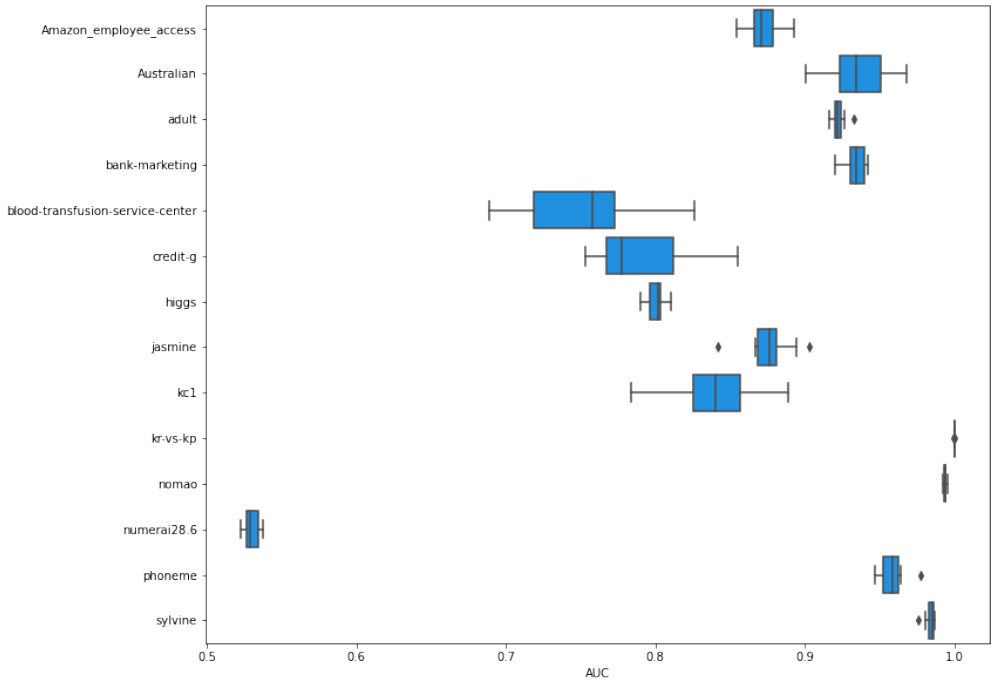

Figure 1: Scores obtained on each dataset with bigger insight than in table. The middle line indicates the mean score. The whiskers extend to points that lie within 1.5 IQR of the lower and upper quartile. The outliers are presented as diamonds.

As we can see on the graph 1, even though that autosklearn 2.0 has much more less time than other frameworks presented in this work ([Gijsbers et al., 2019]), it does not underperform. The results are comparable on this datasets, the differences are quite small. It is quite unexpected that, for example it achieves better scores on *kr-vs-kp* than autoweka. Especially when we take into a consideration fact, that autosklearn 2.0 has only 10 minutes in comparison to autoweka's 4 hours.

5.1 Closer looks into selected models

We tried to take a closer look into selected models to ensembles. It was not the most pleasant experience, because autosklearn 2.0 is in an experimental phase. Many features, which are available in autosklearn, are not implemented in autosklearn 2.0 or rather are implemented very poorly. This poor implementation carried to many errors mostly concerning *Key Error*. As we can see in figure 2, even though framework can use 14 models, in our cases it used only 6 types of models. The most preferred models were tree-based models as Gradient Boosting Classifier, Extra Trees Classifier, Random Forest. It used neural networks quite often too. The least preferable was linear models as a SGD and a Passive Agressive.

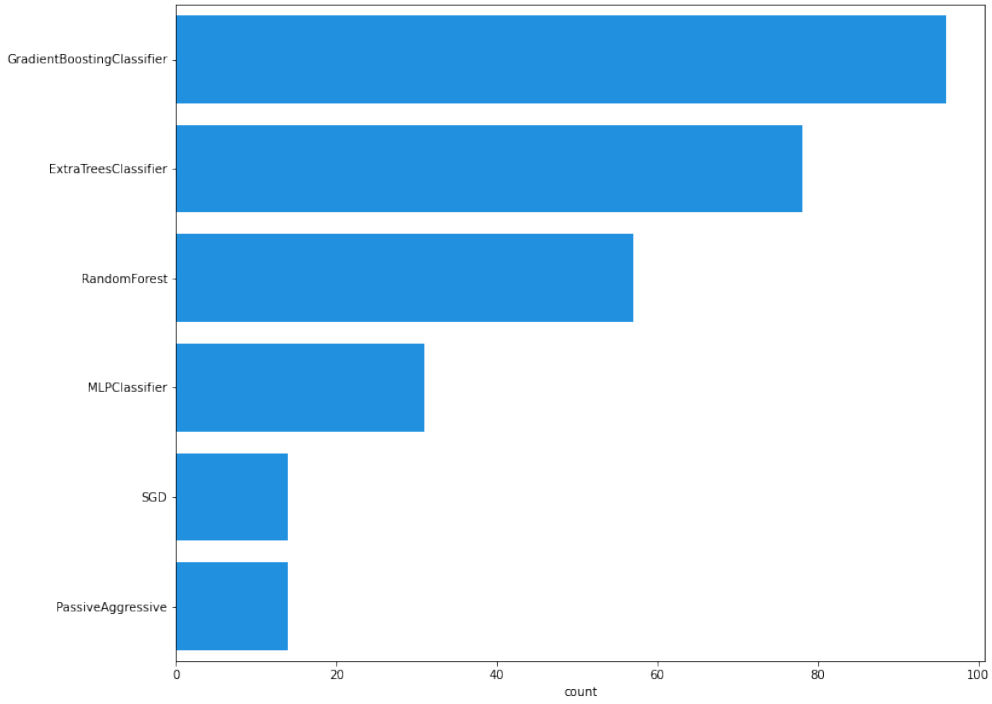

Figure 2: Distribution of models in ensembles.

What is more we checked the average weight of models in ensembles. We noticed that average weights are quite similar. Extra Trees Classifier distinguishes itself as the most impactful from all models. It is noticeable that Passive Agressive is insignificant due to fact it has the lowest average weight and occurs in only small fraction of ensembles from chosen models.

Model	Average weight
RandomForest	0.045965
ExtraTreesClassifier	0.061538
GradientBoostingClassifier	0.053750
SGD	0.051429
MLPClassifier	0.054194
PassiveAggressive	0.035714

Table 2: Average weight of models used in ensembles

6 Own implementation

Based on experiences we collected, we have committed ourselves to the task of creating our own AutoML framework. Our goal was to create simple and universal tool that would achieve relatively good results compared to much more advanced frameworks we learned during our classes. In sections below we describe our solution in detail.

6.1 Preprocessing

To perform preprocessing needed in our implementation of Auto ML we created another few classes and functions responsible for whole task. We made different preprocessing for different column types. Missing values in numerical columns were imputed with mean and then whole columns were scaled with *StandardScaler* and then *MinMaxScaler*. DateTime columns were encoded by our custom class *DateTransformer* which splits the date into 3 values: year, month and day. Majority of categorical values had their missing values filled with most frequent value and there whole columns were *OneHotEncoded*. We treated numerical columns with no more than 10 unique values as categorical ones. Moreover some categorical columns were transformed in other way: if categorical column had more than 20 unique values we used our woe pipe on it, which firstly impute all missing values in the same manner as before and then transform whole column using *WoEEncoder*, which replaces categories by the weight of evidence. Pipes for each categories were stored together in *ColumnTransformer*.

6.2 Model selection

To find solution that would work quite well in many cases and wouldn't take a lot of time we decided to create seven stacking ensembles of sklearn [Pedregosa et al., 2012] models. The ensembles differed in size and models used. The models used by as are: *LogisticRegression*, *RandomForestClassifier*, *AdaBoostClassifier*, *SVC*, *GaussianNB* and *KNeighborsClassifier*. We selected these models specifically because they are known to perform well and differ in structure. For the final estimator in each ensembling we used *LogisticRegression*. For each model we created spaces of hyper-parameters. Our framework tests all ensembles on data and performs Bayesian Optimization on them tuning these hyper-parameters. To sum up, you can see a diagram of our solution below.

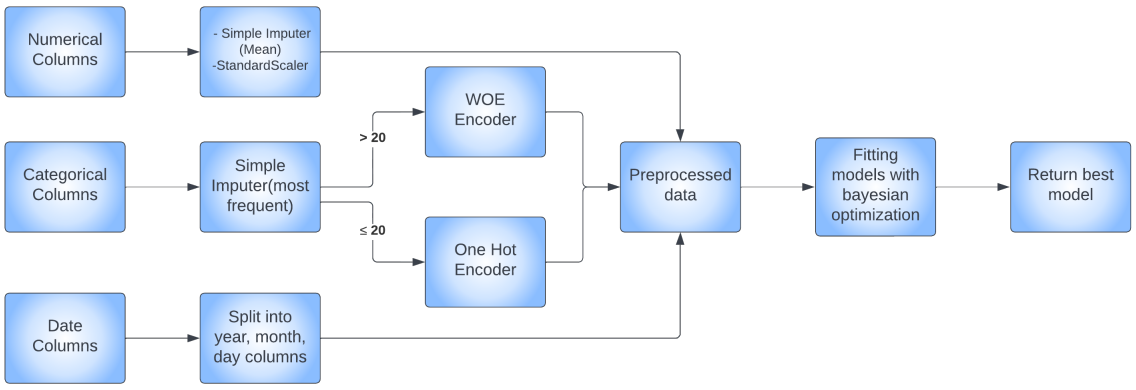

Figure 3: Diagram of our Auto ML solution

We also present you our ensembles used in selection process in table 3. If you want to check configuration of models' search spaces, you should see a table 5 in appendix.

	Model 1	Model 2	Model 3	Model 4
1	SVC	AdaBoostClassifier	RandomForestClassifier	-
2	RandomForestClassifier	AdaBoostClassifier	AdaBoostClassifier	RandomForestClassifier
3	SVC	GaussianNB	KNeighborsClassifier	-
4	KNeighborsClassifier	AdaBoostClassifier	KNeighborsClassifier	-
5	RandomForestClassifier	AdaBoostClassifier	-	-
6	AdaBoostClassifier	AdaBoostClassifier	-	-
7	AdaBoostClassifier	GaussianNB	RandomForestClassifier	SVC

Table 3: Ensembles used in model selection

7 Inner benchmark

As a group we took part in a inner benchmark, which consisted of one dataset. This set concerned binary classification. We got only access to train dataset, so we were not able to check performance of our models on test set. We sent two solution with autosklearn 2.0. Solution 1 was a simple run of autosklearn 2.0. on train dataset. Solution 2 was more sophisticated. The model consisted of 5 autosklearn 2.0 returned models, which were soft-voting. Each model was trained on different 4 out of 5 folds. We could not refit models because our platform was dropping requests and the deadline was coming.

	Team	AUC	framework
1	KTR	0.7912	FLAML
2	Gakubu	0.7908	AutoGluon
3	Gakubu	0.7852	Own implementation
4	WTF	0.7835	AutoPytorch
5	**Moja grupa**	**0.7815**	**Autosklearn solution 1**
6	**Moja grupa**	**0.7808**	**Own implementation**
7	**Moja grupa**	**0.7789**	**Autosklearn solution 2**
8	WTF	0.7768	Own implementation
9	KTR	0.7617	Own implementation
10	Tojada	0.7469	AutoKeras

Table 4: Comparison with another groups

Scores presented in table 4 shows that our autosklearn took fourth place out of 5 teams considering only frameworks and that first solution achieved better score than the second one. Moreover our own implementation took second place out of 4 teams which prepared own solutions. But the difference between groups are not significant.

8 Conclusions

In this paper, we have described and tested a bunch of Auto ML tasks, which can be surely related to some common problems. Those problems are not always solved in easy way due to lack of time, expert knowledge or resources. We can surely admit that automation of machine learning processes is a key to get more efficient models in problems which do not require great insight into data or the interpretability. It is worth mentioning that creating a simple Auto ML implementation is not really hard. Our solution met the demands achieving almost same scores as Autosklearn 2.0 on created model during the inner benchmark. But there are also some bad news, some of the implementations are still under development which means that we can come across several bugs or errors, which should hopefully be solved in the future.

References

Pieter Gijsbers and Joaquin Vanschoren. GAMA: a general automated machine learning assistant. *CoRR*, abs/2007.04911, 2020. URL `https://arxiv.org/abs/2007.04911`.

Matthias Feurer, Aaron Klein, Jost Eggensperger, Katharina Springenberg, Manuel Blum, and Frank Hutter. Efficient and robust automated machine learning. In *Advances in Neural Information Processing Systems 28 (2015)*, pages 2962–2970, 2015.

Matthias Feurer, Katharina Eggensperger, Stefan Falkner, Marius Lindauer, and Frank Hutter. Auto-sklearn 2.0: Hands-free automl via meta-learning. In *arXiv:2007.04074 [cs.LG]*, 2020.

Pieter Gijsbers, Erin LeDell, Janek Thomas, Sébastien Poirier, Bernd Bischl, and Joaquin Vanschoren. An open source automl benchmark. *CoRR*, abs/1907.00909, 2019. URL `http://arxiv.org/abs/1907.00909`.

Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake VanderPlas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Edouard Duchesnay. Scikit-learn: Machine learning in python. *CoRR*, abs/1201.0490, 2012. URL `http://arxiv.org/abs/1201.0490`.

A Search space

We give the configuration space we used in our implementation of Auto ML.

Model	Parameter	search space
Logistic Regression (penalty = elastic net)	C l1_ratio	(0.01, 4) (0.01, 0.99)
Random Forest	n_estimators max_features max_depth min_samples_split min_samples_leaf bootstrap	[5, 1000] ['auto', 'sqrt'] [3, 20] [2, 10] [1,5] [True, False]
AdaBoost	n_estimators learning_rate	[5, 100] [1e-6, 1]
SVC	C kernel degree gamma	(1e-6, 100.0, 'log-uniform') ['linear', 'poly', 'rbf', 'sigmoid'] (1, 5) (1e-6, 100.0, 'log-uniform')
GaussianNB	var_smoothing	(1e-9, 1, 'log-uniform')
KNeighbors	n_neighbors	(3, 10)

Table 5: Search space for models used in our implementation