AutoML - Autogluon

Mikołaj Gałkowski, Hubert Bujakowski, Kacper Kurowski

May 11, 2022

1 Introduction

In recent years, we have seen an exponential growth in the field of machine learning. This is due to people who prepare the data, select the right variables and models, then optimize the models and critically analyze the results. Because the complexity of these tasks is often beyond the reach of non-experts in machine learning, the rapid growth of machine learning applications has created a demand for off-the-shelf machine learning methods that can be used without expertise. The emerging research area aimed at the progressive automation of machine learning is called AutoML.

AutoML provides methods that overcome the barrier posed by machine learning to non-professionals/scientists, so that they are able to use ML with little input. It also helps accelerate the research done by researchers through methods i.e. ensembling, hyperparameter tuning, feature engineering, data preprocessing or data splitting.

In this article, we will focus on binary classification in tabular data using the AutoML framework Autogluon [Erickson et al. 2020]. We will test the capabilities offered by the framework and create a pipeline with which we can throw in raw data and get satisfactory results.

2 Related literature

- (Gijsbers et al. 2019) — The benchmark we conducted was based on datasets used earlier in this paper. Therefore, it is the rationale for using these particular datasets
- (Erickson et al. 2020) — This article describes how the tabular data part of the autogluon package works. It is therefore a source of information for the description of the framework in the next section.

3 Framework description

We believe the design of an AutoML framework should adhere to the following principles:

- Simplicity – A user can train a model on the raw data directly without knowing the details about the data and ML models.
- Predictable Timing – It returns the results within the timebudget specified by users.
- Fault Tolerance – The training can be stopped and resumed at any time.
- Robustness – The framework can handle a large variety of structured datasets and ensures training succeeds even when some of the individual ML models fail.

3.1 Tabular

3.1.1 The `fit` API

Consider a structured dataset of raw values stored in a CSV file, `train.csv`, with the label values to predict stored in a column named — `class`. Three lines of code are all that's needed to train and test a model with AutoGluon:

```
from autogluon import TabularPrediction as task
predictor = task.fit("train.csv", label="class")
predictions = predictor.predict("test.csv")
```

Within the call `.fit()`, AutoGluon automatically:

- Determines the type of each variable
- Handles common preprocessing problems such as missing data and rescaling the values of individual variables
- Identifies what type of prediction problem this is (binary, multi-class classification or regression)
- Partitions the data into various folds for model-training vs. validation
- Individually fits various models, and finally creates an optimized model ensemble that outperforms any of the individual trained model

Autogluon also automates the process of hyperparameterization of individual model (which the user would have to define in advance).

3.1.2 Initial settings

AutoGluon comes with a variety of presets that can be specified in the call to `.fit` via the presets argument.

- `best_quality` – when accuracy is what matters.
- `high_quality` – better than good_quality.

- `good_quality` – significantly better than medium_quality.
- `medium_quality` – initial prototyping, establishing a performance baseline.

The last preset mentioned is the default one. The rest of them require more memory and take much longer time to develop. Therefore, it is recommended to gradually change these settings in order to improve the evaluation and to determine if the model is able to learn within a certain time frame.

3.2 Preprocessing

Autogluon does almost all[1] the necessary preprocessing. The following types of columns are allowed: `boolean`, `numerical`, `categorical`, `datetime`, `object`.

4 Benchmark

4.1 Description

In order to do the benchmark, we have used 21 datasets present in (Gijsbers et al. 2019) for the binary prediction task which was carried out via a pipeline. While most of the preprocessing was automatically done by Autogluon, we had to transform columns which, as per the framework's claim, contained bytes. We have fixed the issue by decoding any instances of **b'string'** into the usual **string**, so that the previously problematic columns were properly detected as having `object` type.

We have performed 10-fold cross-validation using the function `StratifiedKFold` from `sklearn` package so that the results of the benchmark could be compared to the ones from (Gijsbers et al. 2019).

[1] See 4.1

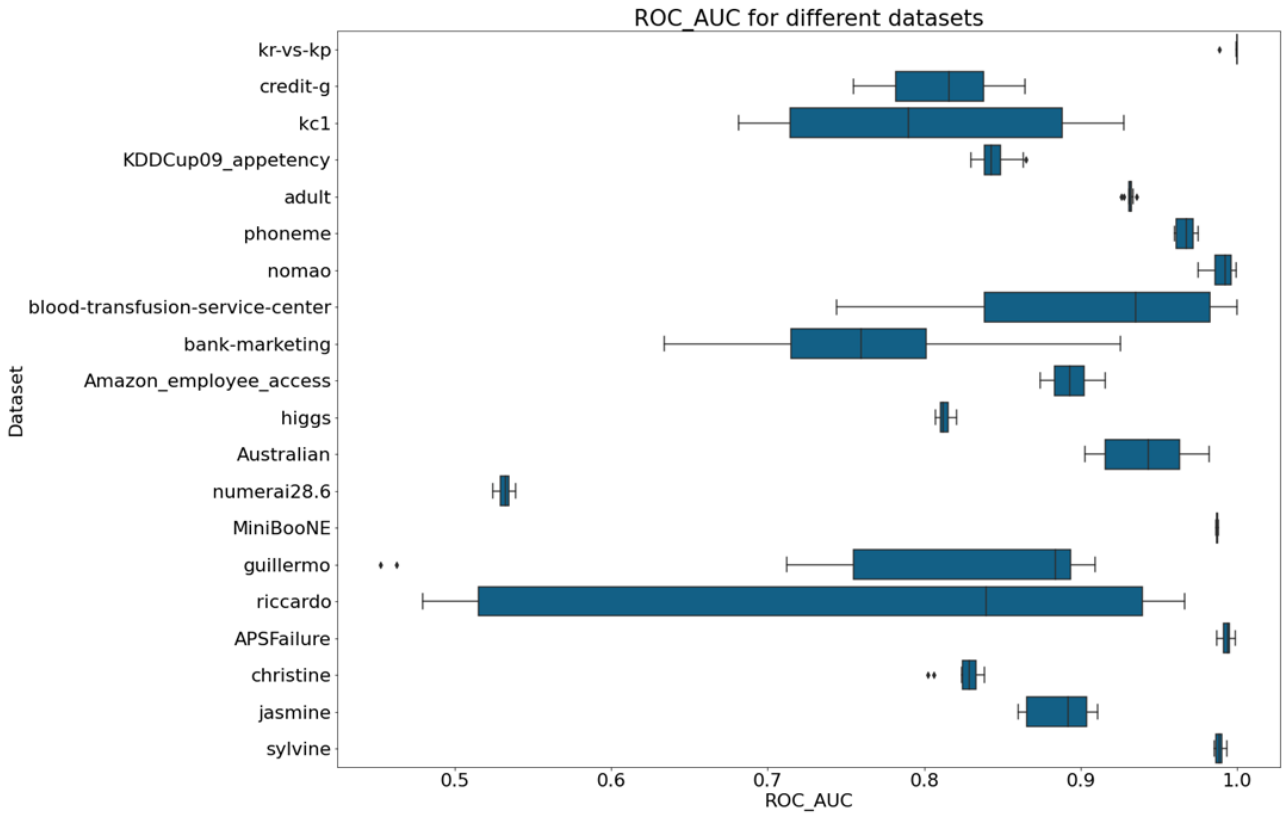

Figure 1: Values of ROC_AUC metric for the binary prediction task achieved using 10-fold cross-validation

4.2 Results

For comparison, let us consider datasets `riccardo` and `sylvine`. For the first one, we have large variance in results, which, most likely, can be attributed to large number (1000) of predictive variables. The other one, however, has given satisfying results, which could be due to the fact that the dataset contains only 21 predictive columns. We conclude that one of the main factors influencing the correctness of prediction for the AutoGluon framework is the size of used data.

5 Conclusion

TO DO

References

Erickson, Nick et al. (Mar. 2020). *AutoGluon-Tabular: Robust and Accurate AutoML for Structured Data*. DOI: 10.48550/ARXIV.2003.06505. URL: https://arxiv.org/abs/2003.06505.

Gijsbers, Pieter et al. (July 2019). "An Open Source AutoML Benchmark". In: *arXiv e-prints*, arXiv:1907.00909, arXiv:1907.00909. arXiv: 1907.00909 [cs.LG].