Auto-PyTorch - article

Adam Frej, Łukasz Tomaszewski, Marcel Witas

1 Introduction

1.1 Bibliography

- github.com/automl/Auto-PyTorch - official Auto-PyTorch repository containing source code and information from authors.
- Auto-PyTorch Tabular: Multi-Fidelity MetaLearning for Efficient and Robust AutoDL - article introducing Auto-PyTorch. It describes framework architecture and provides information about benchmarks that authors conducted to provide evaluation.
- Documentation - official Auto-PyTorch documentation. Very helpful when running framework.
- OpenML - open platform sharing datasets and benchmarks. We use it to evaluate Auto-PyTorch and recreate experiments provided by framework authors.
- An Open Source AutoML Benchmark - open-source AutoML benchmark using public datasets from OpenML. It has 39 datasets (22 binary) and cross-validation splits.

2 Framework

Auto-PyTorch is an Auto Deep Learning framework, which offers classification and regression on tabular data and image classification. It optimizes Neural Network architectural parameters and training hyperparameters, by using multi-fidelity optimization. It relies on PyTorch as the DL framework. Auto-PyTorch implements and optimizes DL pipeline, which includes data preprocessing, neaural architecture, network training techniques and regularization techniques. This framework offers warmstarting by sampling configurations from portfolio. It also ensures an automated ensemble selection.

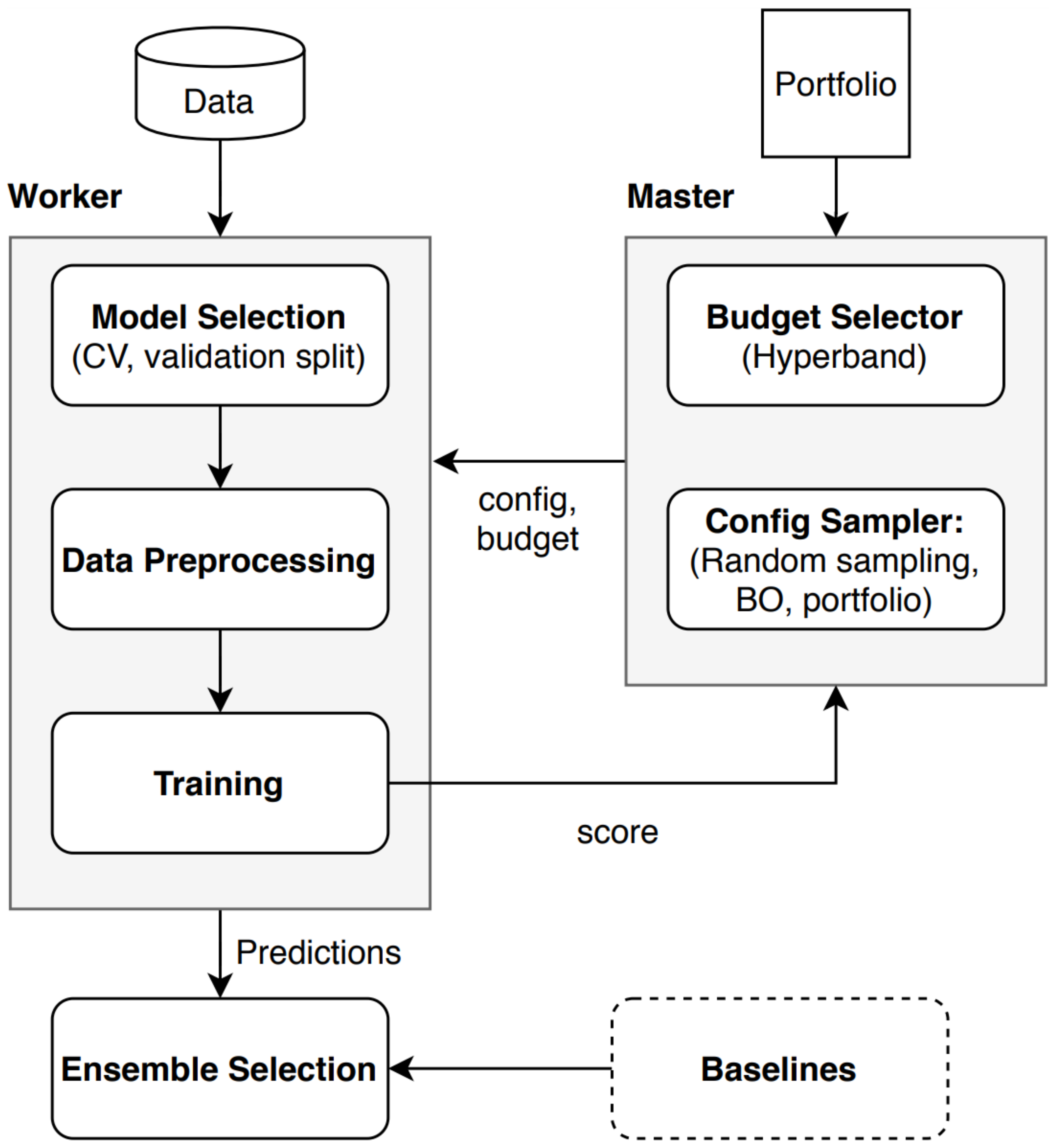

Figure 1: Auto-PyTorch workflow

2.1 Configuration Space

Auto-PyTorch contains many hyperparameters. User can change preprocessing options, architectural hyperparameters and training hyperparameters. Hyperparameters have a hierarchy, top-level hyperparameters can change sub-level hyperparameters.

2.2 Multi-Fidelity Optimalization

To achieve good anytime performance Auto-PyTorch uses BOHB, which is a combination of Bayesian Optimalization and Hyperband. It has been shown, that BOHB outperforms BO and HB on many tasks and it is also up to 55 times faster than Random Search. BOHB uses an outer and an inner loop like HB. In the inner loop configurations are evaluated on lower budget. Then well-performing configurations are being promoted to higher budgets via SuccessiveHalving. Budgets are being calculated in the outer loop. BOHB fits a kernel density estimator as a probabilistic model to the observed performance data. This allows it to find promising areas in the configuration space. Since the configurations on smaller budgets are evaluated first, the KDE's on smaller budgets are available earlier and can guide search until better models on higher budgets are available.

2.3 Parallel Optimization

Auto-PyTorch uses a master-worker architecture. [3] Thanks to this, it can use additional compute resources.

2.4 Model Selection

Auto-PyTorch uses user-defined splits or automated splitting or cross-validation with any iterator from scikit-learn. The score of a model is evaluated via a predefined or user specified metric.

2.5 Ensembles

Auto-PyTorch trains many models and then ensembles them. Ensembling is inspired by auto-sklearn. It starts on an empty set and adds to it, the model, which gives the biggest performance improvement. This process lasts until the set reaches a predefined size. Because one model can be added to the set multiple times, every model gets a weight.

2.6 Portfolios

BOHB start from zero for each new task. To improve performance, Auto-PyTorch uses a warmstart. BOHB first iteration is started with a set of configurations, which cover a set of meta-training datasets.

3 Preprocessing

Since Auto-PyTorch implements and automatically tunes the full Deep Learning pipeline, including data preprocessing, there is no need to do a lot of preprocessing. However, framework accepts only numerical and categorical types of data. Therefore, we changed columns with the object type them to categorical. Such preprocessing turned out to be enough for Auto-PyTorch to work.

4 Evaluation results

Zarys:

- Przedstawienie uzyskanych wyników na wykresie i porównanie ich z wynikami z benchmarku

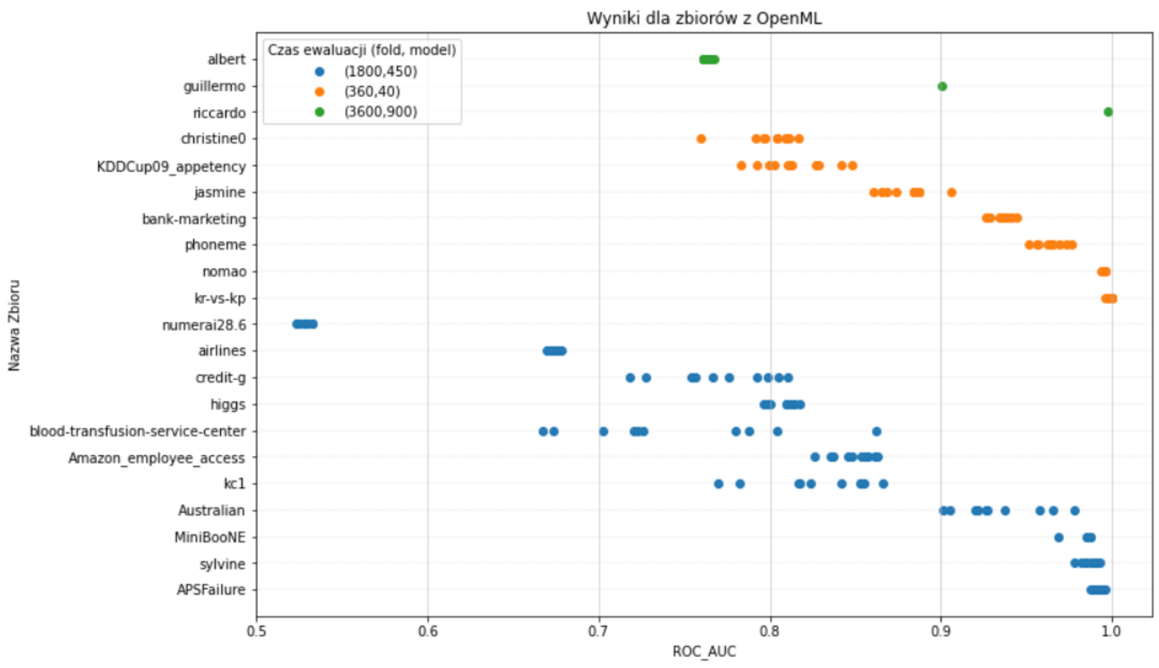

Figure 2: Auto-PyTorch results

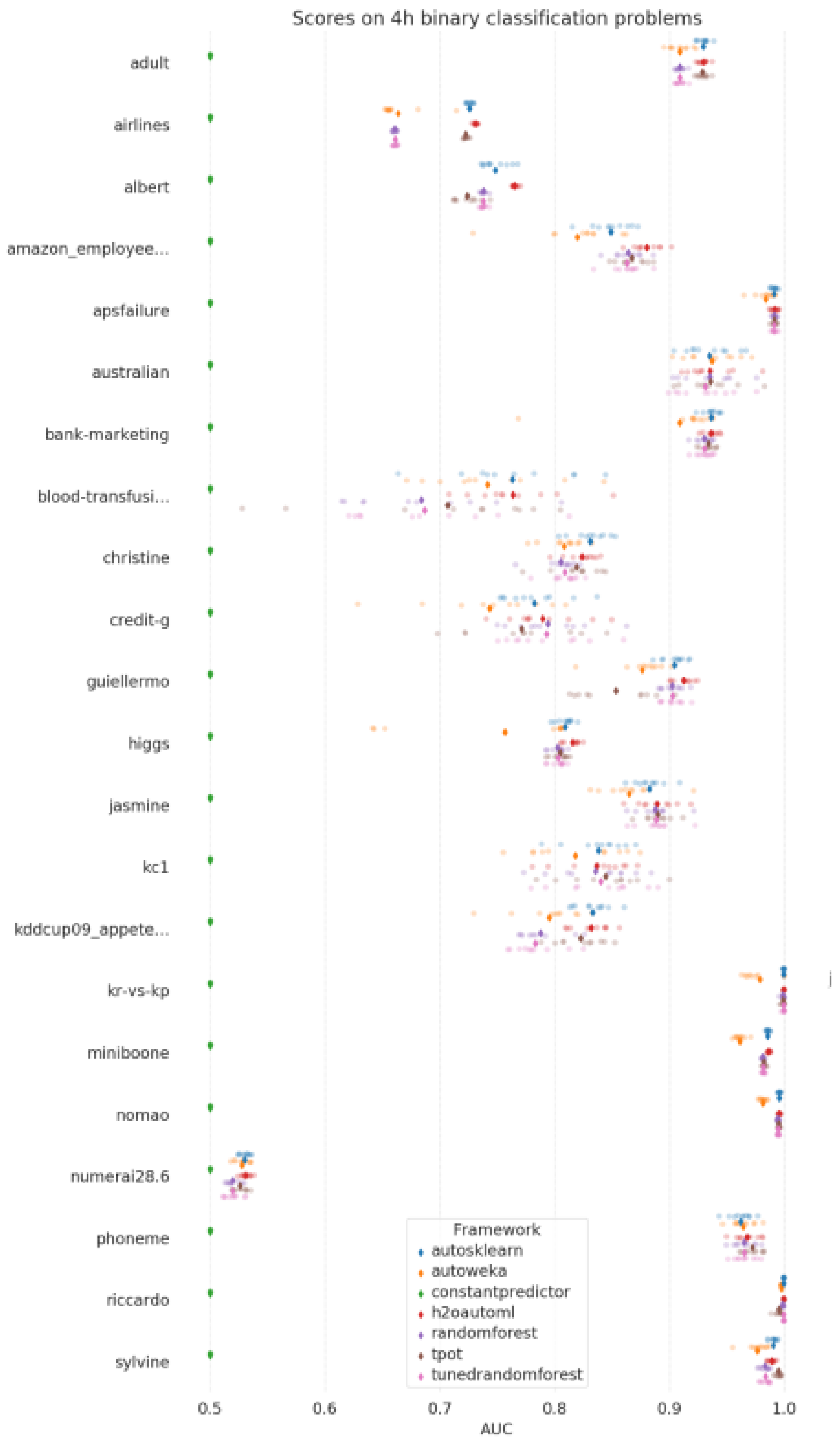

Figure 3: Results of other frameworks - benchmark

- Jakie modele były najczęściej używane (1 fold inny niż pozostałe)

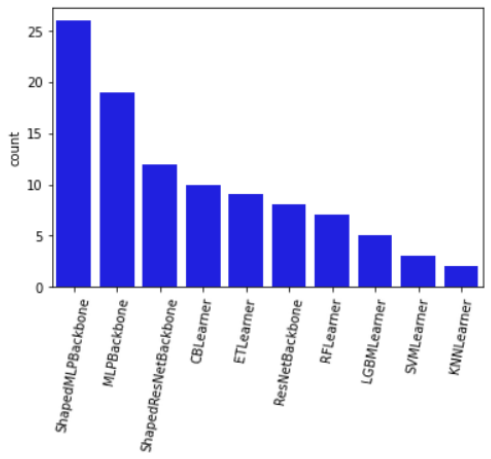

Figure 4: Most common models in fold 1.

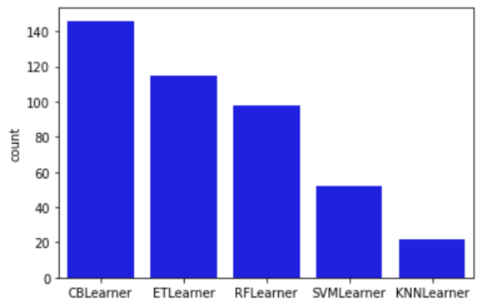

Figure 5: Most common models in folds 2-10.

- Problemy z ewaluacją (problemy z kernelem, dobór czasów)