Practical Automated Machine Learning for the AutoML Challenge 2018

Matthias Feurer FEURERM@CS.UNI-FREIBURG.DE
Katharina Eggensperger EGGENSPK@CS.UNI-FREIBURG.DE
Stefan Falkner SFALKNER@CS.UNI-FREIBURG.DE
Marius Lindauer LINDAUER@CS.UNI-FREIBURG.DE
Frank Hutter FH@CS.UNI-FREIBURG.DE
University of Freiburg

Abstract

Despite great successes in many fields, machine learning typically requires substantial human resources to determine a good machine learning pipeline (including various types of preprocessing, and the choice of classifiers and hyperparameters). AutoML aims to free human practitioners and researchers from these menial tasks. The current state-of-the-art in AutoML has been evaluated in the *AutoML challenge 2018*. Here, we describe our winning entry to this challenge, dubbed *PoSH Auto-sklearn*, which combines an automatically preselected portfolio, ensemble building and Bayesian optimization with successive halving. Finally, we share insights in the importance of different parts of our approach.

Keywords: automated machine learning, Bayesian optimization, successive halving, hyperparameter tuning, competitions, meta-learning

1. Introduction

It is widely acknowledged these days that finding a well-performing machine learning pipeline (including pre-processing, machine learning algorithm selection and hyperparameter optimization) is a tedious and error-prone task for humans. In the light of growing interest in AutoML methods, both from industry and academia, the ChaLearn team organized several challenges on the topic. The first AutoML challenge (Guyon et al., 2015, 2016) attracted over 100 teams and aimed at evaluating AutoML systems in a fair and systematic way. The results demonstrated that automated machine learning can solve this task quite efficiently and often even performs better than human experts. The second AutoML challenge (Guyon et al., 2018.) just finished, and with this paper we describe our winning entry to this challenge, *PoSH Auto-sklearn*. Our contributions are as follows:

- For the first time, we give details of our winning entry to the previous AutoML challenge (a variant of *Auto-sklearn* (Feurer et al., 2015a)), which inspired our new submission (Section 2).
- We describe our new submission, *PoSH Auto-sklearn* and share our insights that gave rise to its design (Section 3).

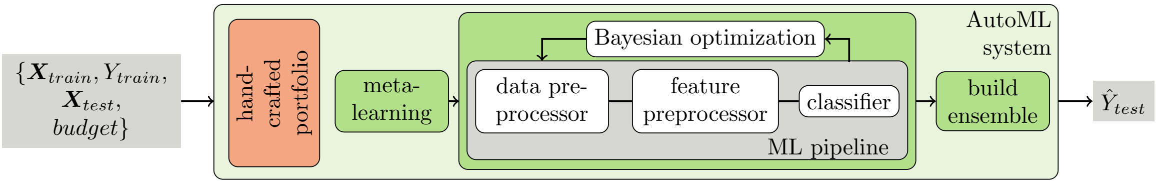

Figure 1: Our pipeline submitted to the ChaLearn Automatic Machine Learning Challenge (2014-2016)

- Although the datasets of the competition are not yet publicly available, we share insights in the importance of the different parts of our approach based on the feedback we received from the competition platform (Section 4).

2. ChaLearn Automatic Machine Learning Challenge (2014-2016)

The first AutoML challenge consisted of five rounds of increasing difficulty. In this section, we briefly describe the setting of this challenge and our submitted AutoML system (which also formed the basis of our submission to the AutoML challenge 2018).

2.1. Competition format

The challenge focused on supervised learning with featurized data. The competition had five rounds, each of them featuring five new, featurized datasets from unknown domains. Each of its rounds consisted of an *auto*-phase and a *tweakathon*-phase. For the *tweakathon*-phase, participants had to submit results (gathered by training a model offline without resource limitation), while for the *auto*-phase participants had to submit code that was run on new unknown datasets under strict time and memory constraints. In both types of phases, participants were ranked on each dataset and the average rank across all datasets was used as the final rank.

2.2. Our Approach: *Auto-sklearn*

The challenge sparked the development of a general purpose AutoML tool which we dubbed *Auto-sklearn* (Feurer et al., 2015a) due to the underlying machine learning framework, scikit-learn (Pedregosa et al., 2011) and following Auto-WEKA (Thornton et al., 2013). Our approach automatically constructed machine learning pipelines suggested by the Bayesian optimization method SMAC (Hutter et al., 2011), warm-started with meta-learning (Feurer et al., 2015b) and combined with post-hoc ensemble building to achieve robust performance (see Figure 1).

Starting with a simple tool in the first phases of the challenge, we quickly extended it to keep up with the increasing difficulty in later phases. As the dataset size increased, our warmstarting approach (Feurer et al., 2015b) became too slow. For this reason, we included a manual strategy to produce predictions that are better than random as fast as possible: Before starting Bayesian optimization, we ran a manually defined set of three diverse, simple pipelines on a subset of the data (one third of the data, up to a maximum of 10 000 data points) for a short budget. If one of these failed, we further reduced the amount of data twice, moving on if the configuration failed three times. Since the challenge featured large datasets in the later stages, this strategy was vital to our success. We also believe that our strict resource management, rigorous testing and robust error handling contributed to a robust submission.

3. AutoML 2018 challenge

The second AutoML challenge ran from November 2017 until March 2018. In the following we highlight the differences to the first challenge and describe our winning entry.

3.1. Competition format

Notable differences to the first challenge were (1) there was only a single *auto*-phase, (2) the datasets provided were comparably homogeneous: dense, binary classification and normalized area under the ROC curve as the target metric (however, they could have missing values, categorical features and be imbalanced), and (3) datasets could be marked as being recorded sequentially. Table 1 provides characteristics of the datasets provided for development (left) and the datasets used for the final evaluation (right).

As in the previous challenge, the goal was to submit an AutoML-system to be executed without human intervention on the Codalab platform. The resources available were 2 CPU Cores with 16GB memory[1] and 40GB SSD running Ubuntu OS under rigid time limits (see column 'time (sec)' in Table 1).

Name	#Sampl.	#Feat.	Seq.	time (sec)
Gina	3 153	970		600
Ada	4 147	48		600
Arcene	10 000	100		600
Guillermo	20 000	4 296		1200
Rl	31 406	22	✓	1200

Name	#Sampl.	#Feat.	Seq.	time (sec)
Riccardo	20 000	4 296		1200
Rm	28 278	89	✓	1200
Pm	29 964	89	✓	1200
Rh	31 498	76	✓	1200
Ri	30 562	113	✓	1200

Table 1: Properties of the datasets provided for developing the submission (left) and of the datasets used for the final evaluation (right).

3.2. Our Approach: *PoSH Auto-sklearn*

Particularly in the light of the tighter time constraints, we revisited the previous version of *Auto-sklearn* and made substantial changes which we describe in the following. Figure 2 illustrates our new AutoML pipeline, which includes the following elements.

[1] While the website states the resources were 8GB, a private email of the organizers to the participants states that the amount of memory was doubled for the final execution.

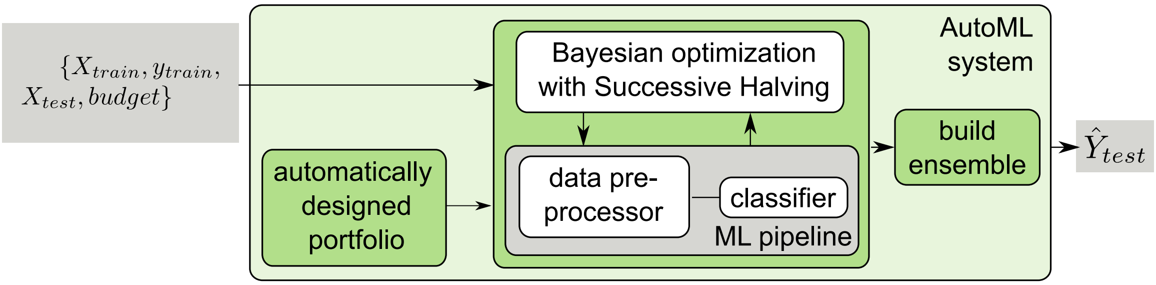
$\{X_{train}, y_{train},$
$X_{test}, budget\}$
AutoML system
Bayesian optimization with Successive Halving
data pre-processor
classifier
ML pipeline
automatically designed portfolio
build ensemble
$\hat{Y}_{test}$

Figure 2: Our pipeline submitted to the AutoML challenge 2018

Successive Halving. A key issue we identified during the last AutoML challenge was that training expensive configurations on the complete training set, combined with a low time budget, does not scale well to large datasets. At the same time, we noticed that our (then manual) strategy to run predefined pipelines on subsets of the data already yielded predictions good enough for ensemble building. For our new submission, we therefore made use of the recent bandit strategy *Successive Halving* (SH, Jamieson and Talwalkar (2016)) and also adapted our configuration space to consider classifiers which can be leveraged by SH's budget allocation. SH allows to specify a minimum and maximum budget for each configuration (e.g. in terms of number of training iterations or dataset size); it starts with the minimum budget and iteratively increases the budget for the pipelines that perform best with the current budget. A nice aspect of SH is its simplicity: it can be implemented with only a few lines of code.

Bayesian Optimization with Successive Halving. Recently, SH and its extension *Hyberband* have been successfully applied to hyperparameter optimization problems (Jamieson and Talwalkar, 2016; Li et al., 2017), demonstrating strong performance with small budgets. Combining *Hyberband* with Bayesian optimization yields the state-of-the-art hyperparameter optimization method BOHB (**B**ayesian **O**ptimization **H**yper**B**and) (Falkner et al., 2018). Since BOHB was shown to outperform both traditional Bayesian optimization without budget considerations, as well as *Hyberband* and SH by itself, we decided to use it as the optimization workhorse for our submission.[2]

Portfolio Building. While our submission to the last AutoML challenge used meta-features to select a set of previously-seen datasets D_{prev} that were similar to the new dataset to be tackled, and then started with configurations that were found to perform well on D_{prev} (Feurer et al., 2015b), we decided to simplify this system. In particular, we chose to start with the same static portfolio of candidate configurations for every dataset. We constructed this portfolio as follows: we gathered 421 binary-classification datasets from OpenML (Vanschoren et al., 2014). For each dataset we optimized our machine learning pipeline with SMAC (Hutter et al., 2011) for up to 1000 function evaluations or 2 days, whichever was reached first. Next, we evaluated these pipelines on all 421 datasets to obtain a 421×421 matrix containing the test scores. We then used greedy submodular function

[2] In practice, we did not use *Hyberband*, but only SH, as we did not expect to finish a full iteration of *Hyberband* in the competition's very tight budget and rather wanted to maximize the number of evaluated configurations.

minimization (Krause and Golovin, 2014) w.r.t. the normalized regret across all datasets in this matrix to obtain a portfolio of machine learning pipelines (see also Appendix A.3 for more details).

Configuration space. Our configuration space was a subspace of the *Auto-sklearn* configuration space suitable to work with SH's budgets: dataset preprocessing (feature scaling, imputation of missing value, treatment of categorical values), but no feature preprocessing (as we had no iterative option of this available), and one of four classifiers: support vector machine (SVM), random forest, linear classification (via stochastic gradient descent) or XGBoost (Chen and Guestrin, 2016), leading to a total of 37 hyperparameters (for details, see Appendix A.1). We used the number of iterations as the budget, except for the SVM, for which we used dataset size as the budget. The exact budgets are given in Appendix A.2.

Ensemble building. Although ensemble building was an important component in *Auto-sklearn*, we found that ensemble selection (Caruana et al., 2004) added too many bad models to the final ensemble. Therefore, we introduced a new regularization technique similar to library pruning (Caruana et al., 2006). Specifically, we compared the loss of each potential candidate model to the loss of the single best model and if the relative difference between these losses was larger than 3% we did not consider the model for the ensemble.

PoSH Auto-sklearn. We combined all of the above methods to form our challenge entry, dubbed *PoSH Auto-sklearn*.[3] This name is an abbreviation of **Po**rtfolio **S**uccessive **H**alving combined with *Auto-sklearn*. Specifically, it runs one iteration of SH on our portfolio of machine learning pipelines and then uses Bayesian optimization[4] to obtain new configurations for SH. We started ensemble selection in a separate process and continuously updated our ensemble as new models were evaluated.[5]

Manual design decisions. To build ensembles and compare configurations, we used 2/3 of the available data for training and 1/3 for validation in SH and building the ensemble. We shuffled datasets prior to splitting unless they were marked as sequential. Furthermore, we designed our submission to yield robust results within a short time limit as follows: If a dataset had more than 500 features, we used univariate feature selection on a subset of 1 000 data points to reduce the number of features to 500. Further manual design decisions (which were not relevant for the competition in the end) and a description of our development process are listed in Appendix A.4/A.5.

4. Results and Analysis of the AutoML Challenge 2018

Overall, the final datasets used in the challenge had different properties than we anticipated based on the datasets provided for development (compare left and right side of Table 1). Nevertheless, our robust approach was able to cope with them quite well: *PoSH Auto-sklearn* placed first with an average rank of 2.8, with the second place having a rank of 3.8 and the third of 5.4. To provide an analysis of our submission, we use the logfiles

[3] http://ml.informatik.uni-freiburg.de/downloads/automl_competition_2018.zip

[4] BOHB requires at least as many function evaluations as hyperparameters before building a model; until these are available it uses random configurations to obtain a global sample of the space.

[5] We also considered models that were dropped by SH and were only trained on smaller budgets.

Name	#Configs (m)	Ensemble size	Val. Perf (p)	Val. Perf	Test Perf	Rank
Riccardo	39 (0)	2(2/0/0)/ 2	0.9997	0.9997	0.2635	1
Rm	47 (3)	3(2/1/0)/ 4	0.7629	0.7757	0.6766	5
Pm	28 (0)	2(2/0/0)/ 2	0.8669	0.8669	0.5533	3
Rh	32 (0)	6(4/2/0)/23	0.3658	0.3702	0.2839	4
Ri	47 (7)	9(3/5/1)/28	0.3478	0.3806	0.3932	1

Table 2: Results on the competition datasets. *#Configs* is the number of evaluated distinct configurations (with the number of configurations proposed by the model in BOHB given in brackets). The column *Ensemble Size* gives the amount of ensemble members out of the number of models considered for the ensemble. Values in brackets are the number of ensemble members from the portfolio, from random search and from the model. *Val. Perf (p)* is the validation performance of configurations in the portfolio, *Val. Perf* is the final validation performance. *Test Perf* denotes the score of our submission on the private, undisclosed test set which was used to compute the *ranks*. All performances are the normalized area under the ROC curve (Guyon et al., 2015).

provided by Codalab from which we could obtain the statistics shown in Table 2. We plan to comprehensively analyze and compare our submission against Auto-sklearn in the future, on the one hand on the competition datasets[6] and on the other hand also when trained based on the same meta-datasets. (Unfortunately, we cannot do a full comparison of the current versions right now since we used all clean binary datasets we had available to train *PoSH Auto-sklearn*.)

Although all datasets in the competition were rather large, for each of them we managed to evaluate all 16 pipelines from our dataset-agnostic portfolio and to evaluate at least one pipeline on the full budget (i.e. finish one iteration of successive halving). For three of the datasets we even finished a second iteration of successive halving and for two we also reached the necessary number of function evaluations to build a model for guiding the search. Comparing columns 4 and 5, we find that there were still modest improvements after the evaluation of the 16 portfolio members with SH for 3 out of 5 datasets. Comparing columns 5 and 6, on almost all datasets we observed a large gap between validation and test performance; the largest of these occurred for dataset *Riccardo*. Our ensembles were rather small compared to the number of evaluated pipelines and consisted mainly of portfolio members. Overall, the results indicate that the portfolio provided a robust and diverse enough set of pipelines, so it was hard to find much better configurations in the remaining time.

Interestingly, even though we filtered ensemble members much more aggressively compared to our previous approaches (Feurer et al., 2015a), the number of candidate models available for building ensembles on two datasets (*Rh* and *Ri*) were still rather large. This indicates that many pipelines performed well on these datasets (at most 3% worse than the

[6] According to the competition website, the 5 private datasets will be released.

single best pipeline). We believe that this new regularization technique was very important for our method, since unregularized ensemble selection would lead to overfitting with a small number of candidate models (and due to the short time budgets small numbers of candidate models were the norm in this competition).

Although we already reduced the space of considered ML algorithms substantially compared to our previous *Auto-sklearn* (4 vs. 15 classifiers), we could have reduced this set even further since, in the end, only XGBoost models ended up in the final ensembles for the challenge.

5. Conclusion

We introduced *PoSH Auto-sklearn*, our winning entry to the second AutoML challenge. Based on our final entry to the first AutoML competition, we described the modification we introduced to cope with the shorter time budget and obtain a more principled approach. In the future, we plan to study the impact of our manual design decisions with an ablation study and develop techniques to automate these as well. Furthermore, we plan to perform a comprehensive comparison against *Auto-sklearn* as described in Section 4.

Acknowledgements

The authors acknowledge support by the state of Baden-Württemberg through bwHPC, the German Research Foundation (DFG) through grant no. INST 39/963-1 FUGG and Emmy Noether grant HU 1900/2-1, the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme under grant no. 716721. Katharina Eggensperger acknowledges funding by the State Graduate Funding Program of Baden-Württemberg.

References

R. Caruana, A. Niculescu-Mizil, G. Crew, and A. Ksikes. Ensemble selection from libraries of models. In R. Greiner, editor, *Proceedings of the 21st International Conference on Machine Learning (ICML'04)*. Omnipress, 2004.

R. Caruana, A. Munson, and A. Niculescu-Mizil. Getting the most out of ensemble selection. In *Proceedings of the 6th IEEE International Conference on Data Mining (ICDM'06)*, pages 828–833. IEEE Computer Society, 2006.

T. Chen and C. Guestrin. Xgboost: A scalable tree boosting system. In B. Krishnapuram, M. Shah, A. Smola, C. Aggarwal, D. Shen, and R. Rastogi, editors, *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 785–794. ACM Press, 2016.

S. Falkner. pynisher, 2015 – 2016. URL https://github.com/sfalkner/pynisher.

S. Falkner, A. Klein, and F. Hutter. BO-HB: Robust and efficient hyperparameter optimization at scale. In *Proceedings of the international conference on machine learning (ICML'18)*, pages 1436–1445, 2018.

M. Feurer, A. Klein, K. Eggensperger, J. T. Springenberg, M. Blum, and F. Hutter. Efficient and robust automated machine learning. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, editors, *Proceedings of the 29th International Conference on Advances in Neural Information Processing Systems (NIPS'15)*, pages 2962–2970, 2015a.

M. Feurer, T. Springenberg, and F. Hutter. Initializing Bayesian hyperparameter optimization via meta-learning. In B. Bonet and S. Koenig, editors, *Proceedings of the Twenty-nineth National Conference on Artificial Intelligence (AAAI'15)*, pages 1128–1135. AAAI Press, 2015b.

P. Geurts, D. Ernst, and L. Wehenkel. Extremely randomized trees. *Machine Learning*, 63 (1):3–42, Apr 2006.

I. Guyon, K. Bennett, G. Cawley, H. J. Escalante, S. Escalera, Tin Kam Ho, N. Macià, B. Ray, M. Saeed, A. Statnikov, and E. Viegas. Design of the 2015 ChaLearn AutoML challenge. In *International Joint Conference on Neural Networks (IJCNN'2015)*, pages 1–8, 2015.

I. Guyon, I. Chaabane, H. J. Escalante, S. Escalera, D. Jajetic, J. R. Lloyd, N. Macià, B. Ray, L. Romaszko, M. Sebag, A. Statnikov, S. Treguer, and E. Viegas. A brief review of the ChaLearn AutoML challenge: Any-time any-dataset learning without human intervention. In Frank Hutter, Lars Kotthoff, and Joaquin Vanschoren, editors, *Proceedings of the Workshop on Automatic Machine Learning*, volume 64 of *Proceedings of Machine Learning Research*, pages 21–30. PMLR, 2016.

I. Guyon, L. Sun-Hosoya, M. Boullé, H. Escalante, S. Escalera, Z. Liu, D. Jajetic, B. Ray, M. Saeed, M. Sebag, A. Statnikov, W. Tu, and E. Viegas. Analysis of the automl challenge series 2015-2018. In Frank Hutter et al., editor, *AutoML*, Springer Series on Challenges in Machine Learning (to appear). Springer, 2018.

F. Hutter, H. Hoos, and K. Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In C. Coello, editor, *Proceedings of the Fifth International Conference on Learning and Intelligent Optimization (LION'11)*, volume 6683 of *Lecture Notes in Computer Science*, pages 507–523. Springer-Verlag, 2011.

K. Jamieson and A. Talwalkar. Non-stochastic best arm identification and hyperparameter optimization. In *Proceedings of the Seventeenth International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2016.

A. Krause and D. Golovin. Submodular function maximization. In L. Bordeaux, Y. Hamadi, and P. Kohli, editors, *Tractability: Practical Approaches to Hard Problems*, pages 71–104. Cambridge University Press, 2014.

L. Li, K. Jamieson, G. DeSalvo, A. Rostamizadeh, and A. Talwalkar. Hyperband: Bandit-based configuration evaluation for hyperparameter optimization. In *Proceedings of the International Conference on Learning Representations (ICLR'17)*, 2017. Published online: iclr.cc.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

C. Thornton, F. Hutter, H. Hoos, and K. Leyton-Brown. Auto-WEKA: combined selection and hyperparameter optimization of classification algorithms. In I. Dhillon, Y. Koren, R. Ghani, T. Senator, P. Bradley, R. Parekh, J. He, R. Grossman, and R. Uthurusamy, editors, *The 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'13)*, pages 847–855. ACM Press, 2013.

J. Vanschoren, J. van Rijn, B. Bischl, and L. Torgo. OpenML: Networked science in machine learning. *SIGKDD Explorations*, 15(2):49–60, 2014.

R. Vinayak and R. Gilad-Bachrach. DART: Dropouts meet Multiple Additive Regression Trees. In Guy Lebanon and S. V. N. Vishwanathan, editors, *Proceedings of the Eighteenth International Conference on Artificial Intelligence and Statistics (AISTATS'15)*, pages 489–497, 2015.

Appendix A. Details

A.1. Configuration Space

Name	Values	Default
Splitting criterion	{gini, entropy}	gini
Min #samples per split	$[2, 20]$	2
Min #samples per leaf	$[1, 20]$	1
Bootstrap	{True, False}	True
Max % of features	$10^{[0,1]}$	0.5

Table 3: Configuration space of the random forest.

Name	Values	Default	Log
C	$[2^{-5}, 2^{15}]$	1	✓
gamma	$[2^{-15}, 2^{3}]$	0.1	✓
shrinking	True, False	True	

Table 4: Configuration space of the support vector machine.

Name	Values	Default	Log
Loss	{hinge, log, modified huber, squared hinge, perceptron}	log	
Penalty	{l1, l2, elastic net}	l2	
α	$[1, 20]$	1	
l1 ratio	$[1e{-}9, 1e{-}1]$	0.15	✓
tolerance	$[1e{-}5, 1e{-}1]$	$1e{-}4$	✓
epsilon	$[1e{-}5, 1e{-}1]$	$1e{-}4$	✓
learning rate schedule	{optimal, invscaling, constant}	invscaling	
eta_0	$[1e{-}7, 1e{-}1]$	$1e{-}2$	✓
power_t	$[1e{-}5, 1]$	0.5	
average	False, True	False	

Table 5: Configuration space of the linear classifier trained by stochastic gradient descent. l1 ratio is only active for the elastic net penalty, epsilon only for the modified huber loss function, and power_t only for the invscaling learning rate schedule.

Name	Values	Default	Log
Max depth	[1, 10]	3	
Learning rate	[0.01, 1]	0.1	✓
Booster	GBTree, DART	GBTree	
Subsample	[0.01, 1.0]	1.0	
Min child weight	$[1e^{-10}, 20]$	1	
Sample type	uniform, weighted	uniform	
Normalization type	tree, forest	tree	
Dropout rate	$[1e-10, 1-1e-10]$	0.5	

Table 6: Configuration space of Extreme Gradient Boosting (Chen and Guestrin, 2016). Hyperparameters in the lower half are only active if the *Dropout Additive Regression Trees (DART)*-Booster (Vinayak and Gilad-Bachrach, 2015) is chosen.

Name	Values	Default	Log
Imputation strategy	{mean, median, most frequent}	mean	
One Hot Encoding (OHE)	{On, Off}	On	
OHE use minimum fraction	{On, Off}	On	
OHE mimimum fraction	[0.0001, 0.5]	0.01	✓
Rescaling strategy	{Standardize, None, MinMax, Normalize, Quantile Transformer, Percentile MinMax}	Standardize	
# Quantiles	[10, 2000]	1000	
Quantile distribution	{uniform, normal}	uniform	
Lower percentile	[0.001, 0.3]	0.25	
Upper percentile	[0.7, 0.999]	0.75	

Table 7: Configuration space of the preprocessing steps.

A.2. Budgets for Successive Halving:

Budgets:

- SVM: percentage of data
- Random Forest: 128 · percentage
- SGD: 512 · percentage (+ early stopping)
- XGBoost: 512 · percentage (no early stopping)

To make sure that no single configuration can make our submission fail by running too long, or running until the end and not giving other configurations time, we allocated a time budget to each run which was proportional to the total number of iterations or dataset subset budget of successive halving. To practically enforce the limits we used the Pynisher (Falkner, 2015 – 2016), which we also used in *Auto-sklearn* (Feurer et al., 2015a). First, we computed the total budget required for successive halving in relation to the max budget. Given an individual budget, we calculated what percentage of the total budget we could allocate. The final time budget allocated is this percentage of the total time budget for a dataset. To keep our code simple we did not reload configurations from disk but re-trained them from scratch. This is at most twice as slow.

A.3. Portfolio

We used a portfolio of size 16 which we obtained by greedily adding machine learning pipelines that performed well across our datasets. Specifically, our portfolio consisted of seven XGBoost, two Random Forest, one linear regression and five SVM configurations plus one manually selected linear regression configuration as a backup solution.

A.4. Additional Measures for Robustness

We implemented the following fallbacks and preprocessing steps in *PoSH Auto-sklearn* which were not relevant for the competition in the end:

- We first saved a dummy predictions consisting of predictions of the majority class to disk. In case something would crashes afterwards, constantly predicting the majority class gives a score of at least 0.0, while not producing a single predictions is punished with a score of -1.0.
- If the dataset had less than 1000 data points, we reverted to cross-validation instead of successive halving.
- For datasets with more than 45.000 data points, we capped the number of training points at 30.000 to retain decent computational complexity. 45.000 was used as a threshold as we used $\frac{2}{3}$ of the data for training only (30.000 data points).
- We prepared a backup solution using Extremely Randomized Trees (Geurts et al., 2006) for the case that no configuration from our portfolio completed in the first iteration (smallest budget) of successive halving.

A.5. Development Process

We developed our submission using Python3 on top of *Auto-sklearn* using a custom version of XGBoost (Chen and Guestrin, 2016) which enabled iterative training. We automated several steps of the development process to efficiently evaluate design decisions. First, we had a *bash*-script that automatically collects all dependencies and creates a submission ready to be uploaded. Second, we used the provided Docker image to reconstruct challenge conditions and test our submission for robustness and bugs.