Towards Further Automation in AutoML

Matthias Feurer FEURERM@CS.UNI-FREIBURG.DE
Frank Hutter FH@CS.UNI-FREIBURG.DE
University of Freiburg

Abstract

Even though recent AutoML systems have been successful in various applications, they introduce new hyper-hyperparameters of their own, including the choice of the evaluation strategy used in the loss function, time budgets to use and the optimization strategy with its hyper-hyperparameters. We study whether it is possible to make these choices in a data-driven way for a dataset at hand. Using 437 datasets from OpenML, we demonstrate the possibility of automating these choices, that this improves over picking a fixed strategy and that for different time horizons different strategies are necessary.

Keywords: Automated Machine Learning, Meta-Learning

1. Introduction

Recent advancements in hyperparameter optimization (Snoek et al., 2012, 2014, 2015; Hutter et al., 2011; Shahriari et al., 2016) triggered the development of new AutoML systems based on them, e.g., Auto-WEKA (Thornton et al., 2013), hyperopt-sklearn (Komer et al., 2014) and *Auto-sklearn* (Feurer et al., 2015a). These systems define a suitable space of machine learning (ML) pipelines and perform a combined optimization of all of its choices (different preprocessors, classifiers, hyperparameter settings, etc). As a side-product, current AutoML systems introduce new hyper-hyperparameters regarding the choice of the evaluation strategy used in the loss function, time budgets to use and the optimization strategy. In this paper we investigate whether we can automate these choices in a data-driven way to avoid overwhelming users with too many options. As a first step, we propose a solution based on greedy submodular function minimization and algorithm selection.

Specifically, we construct several combinations of evaluation strategies and portfolios of ML pipelines, both independently and complementary to each other, and then use a selector to choose the most appropriate one given a new data set. Using 437 datasets from OpenML (Vanschoren et al., 2014), we demonstrate that it is possible to improve over picking a fixed combination of evaluation strategy and portfolio of ML pipelines (which we will refer to as *policy*). Moreover, we show that it is possible to build a portfolio of policies with the greedy algorithm which performs up to 10% better than choosing the single best policy for a time horizon from the test set.

2. Automation in Current AutoML Systems

In this section we briefly review the state of automation of the AutoML toolkit *Auto-sklearn* (Feurer et al., 2015a) under the assumption of having access to tabular data and a given target metric. Our findings also apply to other AutoML systems, such as Auto-WEKA (Thornton et al., 2013) and *TPOT* (Olson et al., 2016). We also summarize an updated version of *Auto-sklearn* for the 2nd AutoML challenge, which is the system we aim to automate further.

Auto-sklearn is a plugin-in replacement for any scikit-learn estimator (Pedregosa et al., 2011). However, users are faced with several design decisions that influence *Auto-sklearn*'s internal optimization process. More specifically, users need to specify an evaluation strategy and its arguments, a maximum budget for the evaluation of individual ML pipelines, hyper-hyperparameters for an ensembling strategy (which replaces the model selection process inside *Auto-sklearn*), and finally which classifiers and pipeline components to search over. While we have not formally studied this, for example, we expect the default holdout evaluation strategy to be a bad choice for small datasets (< 5000 data points) due to a high risk of overfitting the hyperparameters to the holdout data. For advanced users it is also possible to change hyper-hyperparameters of the internal optimization procedure SMAC (Hutter et al., 2011).

Based on our experience in the first AutoML challenge (Guyon et al., 2015, 2016), we adapted *Auto-sklearn* for the second AutoML challenge. We found that for large datasets, sophisticated algorithms were likely to run over the allocated time for individual evaluations, even when using the simple holdout strategy (rather than cross validation). Therefore, we replaced holdout with successive halving (Jamieson and Talwalkar, 2016) (see also Appendix B), but included a fallback to cross-validation for datasets with less than 1000 datapoints. Furthermore, resembling Feurer et al. (2015b,a) we constructed a portfolio of complimentary ML pipelines to warmstart the Bayesian optimization procedure which we used to find performant ML pipelines. We also made several other manual decisions, such as imposing an upper limit on the amount of datapoints and the number of features we use for training. This work is mostly inspired by the surprisingly large number of manual decisions we had to make for the updated version of *Auto-sklearn*.

3. Related Work

Our proposed methodology draws inspiration from several previous works and we reference those where appropriate. The idea of optimizing the target function of hyperparameter optimization in ML has not yet been tackled. To the best of our knowledge, only Guyon et al. (2015) acknowledge the lack of such an approach; more specifically, they mention that "*there is no attempt to treat the problem* [of full model selection] *as the optimization of a regularized functional J2 with respect to both (1) modeling choices and (2) data split*." We aim to solve a more general problem, which is not limited to avoiding overfitting. For example, our approach also supports determining a reasonable subsampling strategy (e.g., depending on the given time budgets, there may not be enough time to perform hyperparameter optimization using a holdout strategy).

Several existing works compare different evaluation strategies (Kohavi, 1995; Petrak, 2000; Yadav and Shukla, 2016). Our goal is not to compare different strategies per se, but to automate the choice between them based on meta-data.

Most similar in spirit is Lindauer et al.'s (2015) AutoFolio, which automatically configures an extensive algorithm selection framework for hard combinatorial problems. Our proposed approach applies similar ideas to the domain of AutoML, but algorithm selection is only one part of it.

4. Methodology

As highlighted in Section 2, current AutoML systems still have several adjustable knobs (hyper-hyperparameters) which affect the global optimization strategy used to find ML pipelines (a pipeline consisting of data cleaning such as missing value imputation and feature normalization as well a classifier, see for example Feurer et al. (2015a). Extending Feurer et al. (2015a), we aim to create an AutoML system by optimizing:

$$\mathbf{V}^*, \omega^* \in \operatorname*{argmin}_{\mathbf{V}\in\mathcal{V}, \omega\in\mathbf{\Omega}} \sum_{D\in\mathcal{D}} \pi(\mathbf{V}, \omega, D, \mathbf{b}), \tag{1}$$

where $\mathbf{V}$ is an evaluation strategy, such as holdout or cross-validation, π is a global optimization algorithm such as SMAC (Hutter et al., 2011) with hyper-hyperparameter searchspace $\mathbf{\Omega}$, which is instantiated with a hyper-hyperparameter vector ω. The remaining variables need to be provided by the user – $\mathcal{D}$ is a set of meta training datasets which we use to optimize the AutoML-system and $\mathbf{b}$ is a budget the AutoML system must abide (such as total runtime).

4.1. Optimizing Average-Case Performance for a Set of Datasets

We now describe a way to construct an AutoML system by optimizing Equation 1; we note that this is similar to our submission to the second AutoML challenge discussed in Section 2. It constructs a portfolio of ML pipelines and chooses the evaluation strategy for these pipelines.

In our setting, a portfolio $\mathcal{P}$ is an ordered set of untrained ML pipelines, which is selected offline. The portfolio aims to provide a diverse set of ML pipelines to be applied to new datasets. Together with an evaluation strategy $\mathbf{V}$ (see Appendix B for the evaluation strategies we consider), it provides a policy $\pi(\mathcal{P}, \mathbf{V})$ of how to tackle a new dataset. Our base strategy for portfolio building is greedy submodular function minimization (Krause and Golovin, 2014), which has already been successfully applied to combinatorial optimization (Xu et al., 2010, 2011; Seipp et al., 2015).

Given a set of ML pipelines $\mathcal{S}$, a set of meta training datasets $\mathcal{D}$, the performances of all ML pipelines in $\mathcal{S}$ on all training datasets, and the empty portfolio $\mathcal{P}$, for each $s \in \mathcal{S}$ we evaluate the performance of $\mathcal{P}' = \mathcal{P} \cup \{s\}$. We continue with $\mathcal{P}'$ which minimizes $\mathcal{L}(\mathcal{P}') = \sum_{D\in\mathcal{D}} \mathcal{L}(\operatorname{argmin}_{s\in\mathcal{P}'} \mathcal{L}(s, D_{test}), D_{test})$, remove the selected s from $\mathcal{S}$, and repeat until $|\mathcal{P}|$ reaches a pre-defined limit. In the naive setting we only look at the test error, but to reduce the risk of overfitting, we use the following alternative strategy: For each extended portfolio $\mathcal{P}' = \mathcal{P} \cup \{s\}$ and each dataset $D \in \mathcal{D}$ we select the pipeline s with the lowest validation error on D. The loss of $\mathcal{P}'$ is calculated by applying the pipelines selected on the validation set to the test set: $\mathcal{L}(\mathcal{P}') = \sum_{D\in\mathcal{D}} \mathcal{L}(\operatorname{argmin}_{s\in\mathcal{P}'} \mathcal{L}(s, D_{val}), D_{test})$.

As the losses for datasets can live on different scales, we employ the simple regret scaled between zero and one for each dataset. We compute the statistics for zero-one scaling by looking at all evaluation strategies we consider.

Because we need to choose a maximal time budget for invoking each configuration, we also propose a strategy to learn these budgets with the greedy algorithm: When calculating the gain of adding a configuration to the portfolio, we pick the time limit resulting in the lowest loss (in case of ties, we pick the shorter time limit).

4.2. Constructing a Portfolio of specialized AutoML Systems

Our approach for constructing an AutoML system presented in the previous subsection only optimizes the hyper-hyperparameters $\mathbf{\Omega}$ by constructing a portfolio of ML pipelines, but has no measure to optimize the evaluation strategy. Since the evaluation strategy and its parametrization is one of the most important choices and highly dataset-dependent, we actually want to construct multiple complementary AutoML systems by optimizing Equation 1 for different subsets of $\mathcal{D}$ and decide on a per-dataset basis which one to use.

We propose to optimize the evaluation strategy by enumerating all different choices; for each evaluation strategy $\mathbf{V}$, we greedily build a portfolio $\mathcal{P}$ and compute a score for $\mathbf{V}$ by simulating the validation performance that $\mathcal{P}$ would achieve using $\mathbf{V}$. After having chosen one $(\mathbf{V}, \mathcal{P})$ combination, we continue to use the same approach, but use the marginal improvement over the existing set of policies as our loss function (both for adding new ML pipelines to a portfolio and selecting a policy). Marginal improvement is the improvement of the policy under construction over the current set of policies; this can be computed using an oracle selector[1] to always choose the correct policy for each training dataset. While this procedure could be repeated until the training loss is zero, we stop the construction after the third policy. We will refer to this set of policies as **complementary policies**. As a baseline, we construct a set of policies by constructing one portfolio for each evaluation strategy using the greedy algorithm described in Section 4.1 (we will refer to these as **independent policies**).

Given a set of AutoML policies, we can use dataset meta-features to select, on a per-dataset basis, which policy to use. For this, we loosely follow the selector design of Hydra (Xu et al., 2010, 2011): for each pair of AutoML policies, we fit a random forest to predict whether playing policy π_A outperforms playing policy π_B given dataset meta-features.

We collect training targets by simply applying the policies to the training datasets, measuring their real-valued performance values and using their difference as the label for a cost-sensitive pairwise binary classification. At prediction time (i.e., when tackling a new dataset), we evaluate all pairwise models, for each policy π add up the probabilities for π to outperform each other policies, and select the policy with the highest score.

5. Experiments

5.1. Setup

We briefly describe our experimental setup and refer to Appendix C for a detailed description. For this paper we restrict ourselves to constructing ML pipelines based on XGBoost (Chen and Guestrin, 2016); specifically, we optimize 13 hyperparameters of XGBoost and 9 hyperparameters of an additional data preprocessing pipeline, which we give in Table 2 in Appendix A. We restrict our study to binary classification.

1. We refer to oracle as a policy selector which selects the policy with the lowest test loss.

Time limit (s):	60		600		3600		No limit	
	mean	std	mean	std	mean	std	mean	std
CV	0.2052	0.0021	0.1178	0.0012	***0.0799***	***0.0020***	***0.0797***	***0.0021***
Holdout	***0.0874***	***0.0009***	***0.0841***	***0.0019***	0.0829	0.0020	0.0829	0.0020
SH	0.0916	0.0008	0.0865	0.0028	0.0860	0.0017	0.0859	0.0018
Independent+Selector	0.1017	0.0046	0.0951	0.0041	0.0852	0.0022	0.0852	0.0022
Independent+Oracle	0.0640	0.0013	0.0593	0.0010	0.0598	0.0013	0.0597	0.0012
Complementary+Selector	**0.0799**	**0.0012**	**0.0728**	**0.0020**	**0.0710**	**0.0023**	**0.0712**	**0.0020**
Complementary+Oracle	0.0593	0.0009	0.0549	0.0009	0.0542	0.0011	0.0548	0.0018

Table 1: Results of executing the different policies for different time horizons. For each time horizon we report the average normalized regret and standard deviation. Oracle models are printed with a grey background. We print the best strategy using multiple policies in bold, and the best strategy using a single policy in bold italic. Abbreviations are CV: cross-validation, SH: successive halving.

We use a total of 437 binary datasets from OpenML.org (Vanschoren et al., 2014). For each dataset we optimized the hyperparameters of our pipeline with SMAC (Hutter et al., 2011), with both a holdout strategy with a 33%-validation set, and a 5-fold cross-validation, which resulted in 871 different ML pipelines. As target metric we used balanced error rate as defined by Guyon et al. (2015). We limited each call to the ML pipeline to 10 minutes and restricted the available memory to 6000MB. For each ML pipeline (and learning curve step) we stored the time taken so far, the validation and the test loss.

We then ran the cross-product of all 871 ML pipelines and 437 datasets with both holdout and 5-fold cross-validation to obtain four performance matrices.[2] Having access to these, we were able to run all experiments as table lookups. To obtain learning curve data for the holdout strategy, we saved the predictions for XGBoost following a geometric schedule (2, 4, 8, 16, . . . 512 iterations). For the successive halving strategy (see Appendix B) we had to set values for the following hyper-hyperparameters: minimum budget, maximum budget and discard ratio η. We used 16, 256, and 2.

All experiments were conducted with 10-fold cross-validation. We used 90% of the datasets as training data for our AutoML algorithms and tested them on the remaining 10% of the datasets. We repeated this procedure ten times to account for randomness due to the shuffling and report both the mean and standard deviation over these repetitions. Furthermore, we ran our procedure with four alternative (simulated) total time budgets: 60s, 600s, 3600s seconds, and no limit.

5.2. Results

We present the results of our experiments in Table 1. The first three rows (CV, Holdout and SH) are results for using a single policy for all datasets and the last four rows (Independent and Complementary) are results for using a selector or oracle to choose between different policies on a per-dataset basis. Strategies using an oracle are printed with a grey background. We print the best strategy using multiple policies in bold, and the best strategy using a single policy in bold italic.

2. For both resampling strategies we measure the validation and test loss.

We report the normalized average regret of evaluating a portfolio of size 16 until the specified total time budget is reached. The regret of each dataset is scaled between zero and one. As a proxy for the lowest achievable loss for a dataset, we use the lowest observed loss (of all evaluation strategies). Reaching a regret of zero is only possible when selecting the correct evaluation strategy and the ML pipeline which has the lowest test loss; this is most likely not possible in practice because we choose the final ML pipeline to apply to the test set based on the validation set.

Best single policy. (Top three rows.) The best single policy is different for different time horizons. For 60s and 600s, it is on average best to use the holdout strategy, while for 3600s and no limit cross-validation results in a lower loss. Cross-validation does not perform well for small time horizons because there is not enough time to run it on the larger datasets. Successive Halving can make use of the iterative nature of XGBoost and therefore perform well on large datasets, but it is not the best evaluation strategy for small datasets and therefore only has a mediocre performance on average across datasets.

Multiple policies with a selector. (Rows four and six.) When using a selector, we have a clear winner: Complementary policies. They clearly outperform selecting a policy from independently constructed policies.

Using multiple policies. Using multiple policies with a selector can outperform a single policy if the policies to choose from are constructed correctly. Simply fitting a selector to independently constructed policies does not outperform the single best independent policy for any of the time budgets. (We note, though, that the performance with a selector is not much worse than the single best policy, so this can still be useful since it frees us from choosing the best policy.) The complementary policies perform much better than the independent ones, yielding the best performance for all four time budgets; we can therefore conclude that using multiple policies is clearly better than using only a single policy for all datasets.

Complementary policies. (Rows five and seven) To answer the question whether we can build complementary policies we need to remove the selector as a potential source of error. By comparing the oracle performance of the independent policies with the oracle performance of complementary policies, we find that the complementary policies always result in a lower regret. Furthermore, the correct selection appears to be easier for the complementary policies than for independently constructed policies, as using a selector on complementary policies results in an improvement over the single best policy, while using a selector on independently constructed policies hurts performance.

6. Conclusion

We presented a novel methodology for automatically constructing AutoML strategies that are optimized to perform well for a given set of representative datasets, a specific budget, and with a specific set of base level classifiers. Our method also supports selecting between specialized AutoML strategies on a per-dataset basis using simple dataset metafeatures. Through extensive experiments on 437 datasets we have shown that our solution not only matches, but outperforms the single best AutoML system from our set.

Although our approach shows promising results we are keen on studying the effect of jointly learning all portfolios and the selector to see if we can reduce the rather high average regret of the oracle selector, and also whether we can improve the selector to be closer to the oracle selector performance. With an improved optimization and selection strategy we also expect that it will be straightforward to optimize additional evaluation strategy hyper-hyperparameters, such as the number of cross-validation folds and the choice among more ML models. Finally, as the method highly depends on having a large amount of representative meta training datasets, we want to study the effect of the training set size and whether we can learn the optimal number of complementary policies for different meta training datasets $\mathcal{D}$.

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.

References

J. Bergstra and Y. Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13:281–305, 2012.

B. Bonet and S. Koenig, editors. *Proceedings of the Twenty-nineth National Conference on Artificial Intelligence (AAAI'15)*, 2015. AAAI Press.

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.

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 Bonet and Koenig (2015), pages 1128–1135.

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 *2015 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, July 2015. doi: 10.1109/IJCNN.2015.7280767.

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, New York, New York, USA, 24 Jun 2016. PMLR.

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.

Ron Kohavi. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence - Volume 2*, IJCAI'95, pages 1137–1143, 1995.

B. Komer, J. Bergstra, and C. Eliasmith. Hyperopt-sklearn: Automatic hyperparameter configuration for scikit-learn. In F. Hutter, R. Caruana, R. Bardenet, M. Bilenko I. Guyon, B. Kégl, , and H. Larochelle, editors, *ICML workshop on Automated Machine Learning (AutoML workshop 2014)*, 2014.

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.

M. Lindauer, H. Hoos, F. Hutter, and T. Schaub. Autofolio: An automatically configured algorithm selector. *Journal of Artificial Intelligence Research*, 53:745–778, August 2015.

Randal S. Olson, Ryan J. Urbanowicz, Peter C. Andrews, Nicole A. Lavender, La Creis Kidd, and Jason H. Moore. *Applications of Evolutionary Computation: 19th European Conference, EvoApplications 2016, Porto, Portugal, March 30 – April 1, 2016, Proceedings, Part I*, chapter Automating Biomedical Data Science Through Tree-Based Pipeline Optimization, pages 123–137. Springer International Publishing, 2016.

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.

Johann Petrak. Fast Subsampling Performance Estimates for Classification Algorithm Selection. In *Proceedings of the ECML-00 Workshop on Meta-Learning: Building Automatic Advice Strategies for Model Selection and Method Combination*, pages 3–14, 2000.

J. Seipp, S. Sievers, M. Helmert, and F. Hutter. Automatic configuration of sequential planning portfolios. In Bonet and Koenig (2015).

B. Shahriari, K. Swersky, Z. Wang, R. Adams, and N. de Freitas. Taking the human out of the loop: A review of Bayesian optimization. *Proceedings of the IEEE*, 104(1):148–175, 2016.

J. Snoek, H. Larochelle, and R. P. Adams. Practical Bayesian optimization of machine learning algorithms. In P. Bartlett, F. Pereira, C. Burges, L. Bottou, and K. Weinberger, editors, *Proceedings of the 26th International Conference on Advances in Neural Information Processing Systems (NIPS'12)*, pages 2960–2968, 2012.

J. Snoek, K. Swersky, R. Zemel, and R. Adams. Input warping for Bayesian optimization of non-stationary functions. In E. Xing and T. Jebara, editors, *Proceedings of the 31th International Conference on Machine Learning, (ICML'14)*, pages 1674–1682. Omnipress, 2014.

J. Snoek, O. Rippel, K. Swersky, R. Kiros, N. Satish, N. Sundaram, M. Patwary, Prabhat, and R. Adams. Scalable Bayesian optimization using deep neural networks. In F. Bach and D. Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning (ICML'15)*, volume 37, pages 2171–2180. Omnipress, 2015.

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*, volume 38 of *Proceedings of Machine Learning Research*, pages 489–497, San Diego, California, USA, 09–12 May 2015. PMLR.

L. Xu, H. Hoos, and K. Leyton-Brown. Hydra: Automatically configuring algorithms for portfolio-based selection. In M. Fox and D. Poole, editors, *Proceedings of the Twenty-fourth National Conference on Artificial Intelligence (AAAI'10)*, pages 210–216. AAAI Press, 2010.

L. Xu, F. Hutter, H. Hoos, and K. Leyton-Brown. Hydra-MIP: Automated algorithm configuration and selection for mixed integer programming. In *Proc. of RCRA workshop at IJCAI*, 2011.

S. Yadav and S. Shukla. Analysis of k-fold cross-validation over hold-out validation on colossal datasets for quality classification. In *2016 IEEE 6th International Conference on Advanced Computing (IACC)*, pages 78–83, Feb 2016.

Appendix A. Configuration space

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	
Column subsample by tree	[0.1, 1.0]	1	
Column subsample by level	[0.1, 1.0]	1	
α	$[1e-10, 1e-1]$	$1e-10$	✓
λ	$[1e-10, 1e-1]$	$1e-10$	✓
Balancing	Off, On	Off	
Sample type	{uniform, weighted}	uniform	
Normalization type	tree, forest	tree	
Dropout rate	$[1e-10, 1-1e-10]$	0.5	
Imputation strategy	{mean, median, most frequent}	mean	
One Hot Encoding	{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 2: Configuration Space of Extreme Gradient Boosting (Chen and Guestrin, 2016). Sample type, Normalization type and dropout rate are only active if the *Dropout Additive Regression Trees (DART)*-Booster (Vinayak and Gilad-Bachrach, 2015) is chosen.

Appendix B. Evaluation strategies

Holdout. This is the default evaluation strategy in *Auto-sklearn*. Configurations are executed one after the other, and the configuration with the lowest loss on the validation set is chosen. For iterative algorithms, we keep track of the learning curve and use the lowest validation loss to select a configuration (and a time step) in the end. If we enforce a time limit, we take the learning curve up to that limit into account.

Cross-validation (CV). We expect this to perform best for small datasets, because it results in a more robust estimate of the validation loss and thus reduces overfitting. However, it comes at the cost of increasing the time to evaluate individual ML pipelines by roughly k times. Similar to holdout, cross-validation executes one configuration after the other and uses the validation loss to choose the best configuration.

Successive Halving (SH). This bandit strategy by Jamieson and Talwalkar (2016) makes use of partial evaluations of a configuration. Given a minimal and maximal budget per configuration, successive halving starts by running a fixed number of configurations on the smallest budget. Then, it iteratively selects the best half of configurations with best validation loss, doubles their budget, and re-evaluates. This process is continued until only one configuration is left or the maximal budget is reached.[3] We expect successive halving to work best for large datasets, for which there is not enough time to train multiple configurations for the full budget, but for which running configurations on a small budget already gives good indications of the validation performance.

In our implementation, successive halving uses the same holdout set as the holdout strategy to compute the loss of an ML pipeline. The difference to the regular holdout strategy is that all ML pipelines are compared only on a very small budget, while the holdout strategy compares the ML pipelines on the full budget. For increasing budgets, successive halving only compares a decreasing number of ML pipelines to each other, thereby saving a lot of time. We note that a the budget of successive halving could also be changed to be the number of cross-validation folds, which would result in a strategy speeding up CV.

Different holdout strategies could be ignored from an optimization point of view as often done in the research of meta-learning and algorithm selection. For AutoML systems this is highly relevant as we are not interested in the optimization performance (of some subpart) of the system, but the generalization performance when applied to a dataset for a certain amount of time.

Appendix C. Experimental details

For this paper we restrict ourselves to XGBoost (Chen and Guestrin, 2016), a versatile and performent implementation of gradient boosting. Additionally, we use a preprocessing pipeline consisting of 1) One-Hot-Encoding (OHE) of categorical features, 2) imputation of missing values, 3) rescaling of features and 4) removal of constant features. While step 1) and 2) could be handled by XGBoost, we follow the pipeline implementation of *Auto-sklearn* which allows us to easily add further models later on. We optimize 13 hyperparameters for

3. In practice, the fraction of configurations to continue and by how much to increase the budget is governed by a hyper-hyperparameter called the *discard ratio* η.

XGBoost and 9 hyperparameters for our preprocessing pipeline which we give in Table 2 in Appendix A. As gradient boosting requires to build a tree for each class in iteration if there are more than two classes, we restrict ourselves to binary classification.

We use a total of 437 binary datasets from OpenML.org (Vanschoren et al., 2014) with at least 150 instances each. While our pipeline could handle sparse data, we found only 3 datasets fulfilling our criteria and therefore dropped them to not run into any downstream issues. The meta-features we used as inputs are: number of missing values, majority class size, dimensionality, majority class percentage, number of binary features, number of classes, number of features, number of instances, number of instances with missing values, number of numeric features, number of symbolic features, percentage of binary features, percentage of instances with missing values, percentage of missing values, percentage of numeric features and the percentage of symbolic features and are available on OpenML (Vanschoren et al., 2014). We give rudimentary details on the datasets in Table 3.

Meta-Feature	Min	Max
# Features	2	61359
# Instances	151	1496391
Majority class percentage	53.06%	99.84 %

Table 3: Dataset statistics.

For each dataset we optimized the hyperparameters of our pipeline with SMAC (Hutter et al., 2011), once with a holdout strategy with a 33%-validation set, and once with 5-fold cross-validation, which resulted in 871 different configurations for our ML pipeline. As target metric we used balanced accuracy as defined by Guyon et al. (2015).

To obtain learning curve data for the holdout strategy, we saved the predictions for XGBoost with a geometric schedule, i.e. after 2, 4, 8, 16, . . . 512 iterations. Moreover, we used 10% of the training data for XGBoost's early stopping. We limited each call to the machine learning pipeline to 10 minutes and restricted the available memory to 6000MB. For each configuration (and learning curve step) we stored the time taken so far, the validation and the test loss. The test loss for the holdout strategy can simply be obtained by applying the trained model to the test set. For cross-validation we apply all models to the test set and use the average predictions (of the probabilities) to compute the loss.

After gathering 871 configurations, we ran the cross-product of all 871 configurations and 437 datasets with both holdout and 5-fold cross-validation to obtain four performance matrices (two evaluation strategies, each with the validation and test loss). Having access to these, we were able to run all experiments as table lookups. For the successive halving strategy we had to set values for the hyper-hyperparameters minimum budget, maximum budget and discard ratio η, without any special intention we set these to $16, 256$ and 2.

Initial optimization and computation of the performance matrices was done on a compute cluster using CentOS 7.4.1708. Each node is equipped with two Intel(R) Xeon(R) CPU E5-2630 v4 @ 2.20GHz (20 cores) and 120 GB of RAM.

All experiments were conducted in a 10-fold cross-validation fashion. We used 90% of the datasets as training data for our AutoML algorithms and tested them on the remaining 10% of the datasets. We repeated this procedure ten times to account for randomness due to the shuffling and report both the mean and standard deviation over these repetitions. As we are interested in the performance for different time horizons, we ran our procedure with (a simulated) total time budget of 60 (1m), 600 (10m) and 3600 (1h) seconds and without any total time budget.

To maximize the performance of our policy selector, we also tune its hyperparameters (minimal number of samples to create a new split, minimal number of samples required for a leaf, number of features considered at each split). For of each pairwise classifier inside the selector we run 30 iterations of random search (Bergstra and Bengio, 2012) on the cost-weighted out-of-bag score, using a total of 512 trees inside the random forest.