Auto-Sklearn 2.0: The Next Generation

Matthias Feurer, Katharina Eggensperger, Stefan Falkner, Marius Lindauer and Frank Hutter

Abstract—Automated Machine Learning, which supports practitioners and researchers with the tedious task of manually designing machine learning pipelines, has recently achieved substantial success. In this paper we introduce new Automated Machine Learning (AutoML) techniques motivated by our winning submission to the second ChaLearn AutoML challenge, *PoSH Auto-sklearn*. For this, we extend *Auto-sklearn* with a new, simpler meta-learning technique, improve its way of handling iterative algorithms and enhance it with a successful bandit strategy for budget allocation. Furthermore, we go one step further and study the design space of AutoML itself and propose a solution towards truly hand-free AutoML. Together, these changes give rise to the next generation of our AutoML system, *Auto-sklearn (2.0)*. We verify the improvement by these additions in a large experimental study on 39 AutoML benchmark datasets and conclude the paper by comparing to *Auto-sklearn (1.0)*, reducing the regret by up to a factor of five.

Index Terms—Automated Machine Learning, Hyperparameter Optimization, Meta-Learning

◆

1 INTRODUCTION

The recent substantial progress in machine learning (ML) has led to a growing demand for hands-free ML systems that can support developers and ML novices in efficiently creating new ML applications. Since different datasets require different ML pipelines, this demand has given rise to the young area of automated machine learning (AutoML [1]). Popular AutoML systems, such as *Auto-WEKA* [2], *hyperopt-sklearn* [3] *Auto-sklearn* [4], *TPOT* [5] and *Auto-Keras* [6] perform a combined optimization across different preprocessors, classifiers, hyperparameter settings, etc., thereby reducing the effort for users substantially.

Our work is motivated by the first and second ChaLearn AutoML challenge [7], which evaluated AutoML tools in a systematic way under rigid time and memory constraints. Concretely, the AutoML tools were required to deliver predictions in up to 20 minutes which demands an efficient AutoML system that can quickly adapt to a dataset at hand. Performing well in such a setting would allow for an efficient development of new ML applications on-the-fly in daily business. We managed to win both challenges with *Auto-sklearn* [4] and *PoSH Auto-sklearn* [8], relying mostly on meta-learning and robust resource management.

While AutoML reliefs the user from making low-level design decisions (e.g. which model to use), AutoML itself opens a myriad of high-level design decisions (e.g. which model selection strategy to use [9]). Whereas our submissions to the AutoML challenges were mostly hand-desigend, in this work we go one step further by automating AutoML itself. Specifically, our contributions are:

1) We explain practical details which allow *Auto-sklearn* to handle iterative algorithms more efficiently.
2) We extend *Auto-sklearn*'s choice of model selection strategies to include ones optimized for high-throughput evaluation of ML pipelines.
3) We introduce both the practical approach as well as the theory behind building better portfolios for the meta-learning component of *Auto-sklearn*.
4) We propose a meta-learning technique based on algorithm selection to automatically choose the best AutoML-pipeline for a given dataset, further robustifying AutoML itself.
5) We conduct a large experimental evaluation, comparing against *Auto-sklearn (1.0)* and benchmarking each contribution separately.

The paper is structured as follows: First, we formally describe the AutoML problem in Section 2 and then discuss the background and basis of this work, *Auto-sklearn (1.0)* and *PoSH Auto-sklearn* in Section 3. In the following four sections, we describe in detail the improvements we propose for the next generation, *Auto-sklearn (2.0)*. For each improvement, we state the practical and theoretical motivation, mention the most closely related work (with further related work deferred to Appendix A), and discuss the methodological changes we make over *Auto-sklearn (1.0)*. We conclude each improvement section with a preview of empirical results highlighting the benefit of each change; we defer the detailed explanation of the experimental setup to Section 8.1.

In Section 8 we first perform a large-scale ablation study, assessing the contribution of each of our contributions, before comparing our new *Auto-sklearn (2.0)* against several versions of *Auto-sklearn (1.0)*. We conclude the paper with open questions, limitations and future work in Section 9.

- *M. Feurer and K. Eggensperger are with the Institute of Computer Science, University of Freiburg, Germany.*
  E-mail: {feurerm,eggenspk} @cs.uni-freiburg.de
- *S. Falkner is with the Bosch Center for Artificial Intelligence, Renningen, Germany. E-mail: Stefan.Falkner@de.bosch.com*
- *M. Lindauer is with the Institute of Information Processing, Leibniz University Hannover. E-mail: lindauer@tnt.uni-hannover.de*
- *F. Hutter is with the University of Freiburg and the Bosch Center for Artificial Intelligence. E-mail: fh@cs.uni-freiburg.de*

2 PROBLEM STATEMENT: AUTOMATED MACHINE LEARNING

Let $P(\mathcal{D})$ be a distribution of datasets from which we can sample an individual dataset's distribution $P_d = P(\mathbf{X}, \mathbf{y})$. The AutoML problem is to generate a trained pipeline $\mathcal{M}_{\boldsymbol{\lambda}} : \mathbf{x} \mapsto y$, hyper-parameterized by $\boldsymbol{\lambda} \in \boldsymbol{\Lambda}$ that automatically produces predictions for samples from the distribution P_d minimizing the generalization error:

$$GE(\mathcal{M}_{\boldsymbol{\lambda}}) = \int \mathcal{L}(\mathcal{M}_{\boldsymbol{\lambda}}(\mathbf{x}), y) P_d(\mathbf{x}, y) d\mathbf{x} dy. \quad (1)$$

Since a dataset can only be observed through a set of n independent observations $\mathcal{D}_d = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\} \sim P_d$, we can only empirically approximate the generalization error on sample data:

$$\widehat{GE}(\mathcal{M}_{\boldsymbol{\lambda}}, \mathcal{D}_d) = \frac{1}{|\mathcal{D}_d|} \sum_{(\mathbf{x}_i, y_i) \in \mathcal{D}_d} \mathcal{L}(\mathcal{M}_{\boldsymbol{\lambda}}(\mathbf{x}_i), y_i). \quad (2)$$

AutoML systems automatically search for the best $\mathcal{M}_{\boldsymbol{\lambda}^*}$:

$$\mathcal{M}_{\boldsymbol{\lambda}^*} \in \underset{\boldsymbol{\lambda} \in \boldsymbol{\Lambda}}{\arg\min} \, \widehat{GE}(\mathcal{M}_{\boldsymbol{\lambda}}, \mathcal{D}_{\text{train}}) \quad (3)$$

and estimate GE, e.g., by a k-fold cross validation:

$$\widehat{GE}_{\text{CV}}(\mathcal{M}_{\boldsymbol{\lambda}}, \mathcal{D}_{\text{train}}) = \frac{1}{k} \sum_{i=1}^{k} \widehat{GE}(\mathcal{M}_{\boldsymbol{\lambda}}^{\mathcal{D}_{\text{train}}^{(\text{train},i)}}, \mathcal{D}_{\text{train}}^{(\text{val},i)}) \quad (4)$$

where $\mathcal{M}_{\boldsymbol{\lambda}}^{\mathcal{D}_{\text{train}}^{(\text{train},i)}}$ denotes that $\mathcal{M}_{\boldsymbol{\lambda}}$ was trained on the i-th training fold $\mathcal{D}_{\text{train}}^{(\text{train},i)}$. Assuming that an AutoML system can select via $\boldsymbol{\lambda}$ both, the algorithm and its hyperparameter settings, this definition using $\widehat{GE}_{\text{CV}}$ is equivalent to the definition of the CASH problem [2], [4].

2.1 Time-bounded AutoML

In practice, users are not only interested to obtain an optimal pipeline $\mathcal{M}_{\boldsymbol{\lambda}^*}$ eventually, but have constraints on how much time and compute resources they are willing to invest. We denote the time it takes to evaluate $\widehat{GE}(\boldsymbol{\lambda}, \mathcal{D}_{\text{train}})$ as $t_{\boldsymbol{\lambda}}$ and the overall optimization budget by T. Our goal is to find

$$\mathcal{M}_{\boldsymbol{\lambda}^*} \in \underset{\boldsymbol{\lambda} \in \boldsymbol{\Lambda}}{\arg\min} \, \widehat{GE}(\boldsymbol{\lambda}, \mathcal{D}_{\text{train}}) \text{ s.t. } \left(\sum t_{\boldsymbol{\lambda}} \right) < T \quad (5)$$

where the sum is over all pipelines evaluated, explicitly honouring the optimization budget T.

2.2 Generalization of AutoML

Ultimately, a well performing and robust optimization policy $\pi : \mathcal{D} \mapsto \mathcal{M}_{\boldsymbol{\lambda}}^{\mathcal{D}}$ of an AutoML system should not only perform well on a single dataset but on the entire distribution over datasets $P(\mathcal{D})$. Therefore, the meta-problem of AutoML can be formalized as minimizing the generalization error over this distribution of datasets:

$$GE(\pi) = \int GE(\pi, \mathcal{D}) p(\mathcal{D}) d\mathcal{D}, \quad (6)$$

which in turn can again only be approximated by a finite set of meta-train datasets $\mathbf{D}_{\text{meta}}$ (each with a finite set of observations):

$$\widehat{GE}(\pi, \mathbf{D}_{\text{meta}}) = \frac{1}{|\mathbf{D}_{\text{meta}}|} \sum_{\mathcal{D}_i \in \mathbf{D}_{\text{meta}}} \widehat{GE}(\pi, \mathcal{D}_i). \quad (7)$$

Having set up the problem statement, we can use this to further formalize our goals. We will introduce a novel system for designing π from data in Section 7 and extend this to a function $\Xi : \mathcal{D} \to \pi$ which automatically suggests an AutoML optimization policy for a new dataset.

3 BACKGROUND ON *Auto-sklearn*

AutoML systems following the CASH formalism are driven by a sufficiently large and flexible pipeline configuration space and an efficient method to search this space. Additionally, to speed up this procedure, information gained on other datasets can be used to kick-start or guide the search procedure (i.e. meta-learning). Finally, one can also combine the models trained during the search phase in a post-hoc ensembling step. In the following, we give details on the background of these four components, describe how we implemented them in *Auto-sklearn (1.0)* and how we extended them for the second ChaLearn AutoML challenge.

3.1 Configuration Space

Analogously to *Auto-WEKA* [2], *AutoKeras* [6] and *Auto-PyTorch* [10], *Auto-sklearn* is also built around an existing ML library, namely *scikit-learn* [11], forming the backbone of our system. The configuration space allows ML pipelines consisting of three steps: data preprocessing, feature preprocessing and an estimator. A pipeline can consist of multiple data preprocessing steps (e.g., imputing missing data and normalizing data), one feature preprocessing step (e.g., a principal component analysis) and one estimator (e.g., gradient boosting). Our configuration space is hierarchically organized in a tree-structure and contains continuous (e.g., the learning rate), categorical (e.g., type of estimator) and conditional hyperparameters (e.g., the learning rate of a estimator). For classification, the space of possible ML pipelines currently spans across 15 classifiers, 16 feature preprocessing methods and numerous data preprocessing methods, adding up to 153 hyperparameters for the latest release.[1]

3.2 Bayesian Optimization (BO)

BO [12] is the driving force of *Auto-sklearn*. It is an optimization procedure specially designed toward sample efficiency where the evaluation time of configurations dominates the search procedure. BO is based on two mechanisms: 1) Fitting a probabilistic model mapping hyperparameters to their loss value and 2) optimizing an acquisition function to choose the next hyperparameter setting to evaluate. BO iterates these two steps and evaluates the selected setting. A common choice for the internal model are Gaussian process models [13], which perform best in low-dimensional problems with continuous hyperparameters. For *Auto-sklearn* we use random forests [14], since they have been shown to perform well for high-dimensional and structured optimization problems, like the CASH problem [2], [15].

---

1. We used software version 0.7.0, however, we note that the software version does not align with the method version.

3.3 Meta-Learning

AutoML systems often solve similar problems over and over again while starting from scratch for every task. Meta-learning techniques can exploit experience gained on previous optimization tasks and equip the optimization process with this knowledge. For *Auto-sklearn (1.0)*, we used a case-based reasoning system [16], [17], dubbed k-nearest datasets (KND). For a new dataset, this procedure warmstarts BO with the best known ML pipelines found on the k nearest datasets, where the distance between datasets is defined as the L_1 distance on meta-features describing these datasets. Meta-learning for efficient AutoML is an active area of research and we refer to a recent literature review [18].

3.4 Ensembles

While searching for the best ML pipeline, AutoML systems train numerous such ML pipelines, but would traditionally only use the single best one. An easy remedy is to combine these post-hoc into an ensemble to further improve performance and reduce overfitting [4], [19]. *Auto-sklearn* uses ensemble selection [20] to continuously output ensembles during the training process. Ensemble selection greedily adds ML pipelines to an ensemble to minimize the error and can therefore be used with any loss function without adapting the training procedure of the base models or the hyperparameter optimization.

3.5 Second AutoML Challenge

The goal of the challenge was to design an AutoML system that, without any human intervention, operates under strict time and memory limits on unknown binary classification datasets on the Codalab platform [7]. A submission had between $600 - 1200$ seconds, 2 CPUs and 16GB memory to produce predictions that were evaluated and ranked according to area under the curve and the submission with the best average rank won the competition – a predecessor of *Auto-sklearn (2.0)* which we dubbed *PoSH Auto-sklearn*, short for **Po**rtfolio and **S**uccessive **H**alving [8]. Our main changes with respect to *Auto-sklearn (1.0)* were the usage of successive halving instead of regular holdout to evaluate more machine learning pipelines, a reduced search space to complement successive halving by mostly containing iterative models, and a static portfolio instead of the KND technique to avoid the computation of meta-features. Our work picks up on the ideas of *PoSH Auto-sklearn*, describes them in more detail and provides a thorough evaluation while also presenting a novel approach to AutoML which was motivated by the manual design decisions we had to make for the competition.

4 IMPROVEMENT 0: PRACTICAL CONSIDERATIONS

The performance of an AutoML system not only relies on efficient hyperparameter optimization and model selection strategies, but also on practical considerations. Here, we will describe design decisions we applied for all further experiments since they in general improve performance.
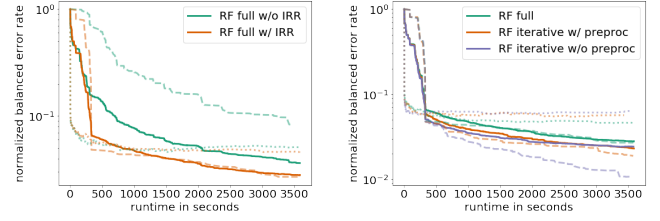

Fig. 1. Balanced error rate over time. We report aggregated results across 39 datasets and 10 repetitions for different settings of our AutoML system using holdout as the model selection strategy. The solid line aggregates across all datasets and the dotted (dashed) line aggregates across the 5 smallest (largest) datasets. Left: Comparing with intermittent results retrieval (IRR) and without. Right: Comparing different search spaces.

4.1 Early Stopping and Retrieving Intermittent Results

Estimating the generalization error of a pipeline $\mathcal{M}_{\boldsymbol{\lambda}}$ practically requires to restrict the CPU-time per evaluation to prevent that one single, very long algorithm run stalls the optimization procedure [2], [4]. If an algorithm run exceeds the assigned time limit, it is terminated and the worst possible generalization error is assigned. If the time limit is set too low, a majority of the algorithms do not return a result and thus provide very scarce information for the optimization procedure. A too high time limit, however, might as well not return any meaningful results since all time may be spent on long-running, under-performing pipelines. To mitigate this risk, for algorithms that can be trained iteratively (e.g., gradient boosting and linear models trained with stochastic gradient descent) we implemented two measures. Firstly, we allow a pipeline to stop training based on a heuristic at any time, i.e. early stopping, which prevents overfitting. Secondly, we make use of intermittent results retrieval, e.g., saving the results at checkpoints spaced at geometrically increasing iteration numbers, thereby ensuring that every evaluation returns a performance and thus yields information for the optimizer. With this, our AutoML tool can robustly tackle large datasets without the necessity to finetune the number of iterations dependent on the time limit.

4.2 Search Space

The first search space of *Auto-sklearn* consisted of $\mid \boldsymbol{\Lambda} \mid = 110$ hyperparameters, whereas the latest release has grown to even 153 hyperparameters. Current hyperparameter optimization algorithms can cope with such spaces, given enough time, but, in this work, we consider a heavily time-bounded setting. Therefore, we reduced our space to 42 hyperparameters only including iterative models to benefit from the early stopping and intermittent results retrieval.

4.3 Preview of Experimental Results

4.3.1 Do we need intermittent results retrieval?

Ideally, we want all iterative algorithms to converge on a dataset, i.e. allow them to run for as many iterations as required until the early-stopping mechanism terminates training. However, on large datasets this might be infeasible, so one would need to carefully set the number of iterations such that the algorithm terminates within the given

time-limit or tune the number of iterations as part of the configuration space. The left plot in Figure 1 shows the substantial improvement of intermittent results retrieval. While the impact on small datasets is negligible (dotted line), on large datasets this is crucial (dashed line).

4.3.2 Do we need the full search space?

The larger the search space, the greater the flexibility to adapt a method to a new dataset. However, a strict time-limit might prohibit a thorough search in a large search space. Therefore, we studied two pruned versions of our search space: 1) reducing the classifier space to only contain models that can be fitted iteratively and 2) further reduce the preprocessing space to only contain necessary preprocessing steps, such as imputation of missing values and one-hot-encoding of categorical values.

Focusing on a subset of classifiers that always return a result reduces the chance of wasteful timeouts which motivates the first point. This subset mostly contains tree-based methods, which often inherently contain forms of feature selection which lead us to the second point above. In Figure 1, we compare our AutoML system using different configuration spaces. Again, we see that the impact of this decision is most evident on large datasets. We provide the reduced search space in Appendix C.

5 IMPROVEMENT 1: MODEL SELECTION STRATEGY

A key component of any efficient AutoML system is its model selection strategy, which addresses the two following problems: 1) how to approximate the generalization error of a single ML pipeline and 2) how many resources to allocate for each pipeline evaluation. In this section, we discuss different combinations of these to increase the flexibility of *Auto-sklearn (2.0)* for different use cases.

5.1 Assessing the Performance of a Model

Given a training set $\mathcal{D}_{\text{train}}$, the goal is to best approximate the generalization error to 1) provide a precise signal for the optimization procedure and 2) based on this to select $\mathcal{M}_{\boldsymbol{\lambda}}$ in the end. We usually compute the validation loss, which is obtained by splitting the training data into into two smaller, disjoint sets $\mathcal{D}_{\text{train}}^{train}$ and $\mathcal{D}_{\text{train}}^{valid}$, by following the common *train-validation-test* protocol [21], [22]. The two most common ways to assess the performance of a model are holdout and the K-fold cross-validation [9], [22], [23], [24], [25], [26]. We expect the holdout strategy to be a better choice for large datasets where the holdout set is representative of the test set, and where it is computationally wasteful to apply cross-validation. Consequently, we expect cross-validation to yield the best results for small datasets, where its computational overhead does not play a role, and where only the use of all available samples can result in a reliable estimate of the generalization error.[2]

2. Different model selection strategies could be ignored from an optimization point of view, where the goal is to optimize performance given a loss function, as is often done in the research fields of meta-learning and hyperparameter optimization. However, for AutoML systems this is highly relevant as we are not interested in the optimization performance (of some subpart) of these systems, but the final generalization performance when applied to new data.

5.2 Allocating Resources to Choose the Best Model

Considering that the available resources are limited, it is important to trade off the time spent assessing the performance of each ML pipeline versus the number of pipelines to evaluate. Currently, *Auto-sklearn (1.0)* implements a conceptually simple approach and evaluates each pipeline under the same resource limitations and on the same budget (e.g., number of iterations using iterative algorithms). The recent bandit strategy successive halving (SH) [27], [28] employs the concept of assigning higher *budgets* to more promising pipelines when evaluating them; the budgets can, e.g., be the number of iterations in gradient boosting, the number of epochs in neural networks or the number of data points. Given a minimal and maximal budget per ML pipeline, SH starts by training a fixed number of ML pipelines for the smallest budget. Then, it iteratively selects $\frac{1}{\eta}$ of the pipelines with lowest generalization error, multiplies their budget by η, and re-evaluates. This process is continued until only a single ML pipeline is left or the maximal budget is spent.

While SH itself uses random search to propose new pipelines $\mathcal{M}_{\boldsymbol{\lambda}}$, we follow recent work combining SH with BO [29]. BO iteratively suggests new ML pipelines $\mathcal{M}_{\boldsymbol{\lambda}}$, which we evaluate on the lowest budget until a fixed number of pipelines has been evaluated. Then, we run SH as described above. We build the model for BO on the highest available budget where we have observed the performance of $\frac{|\Lambda|}{2}$ pipelines.

SH potentially provides large speedups, but it could also too aggressively cut away good configurations that need a higher budget to perform best. Thus, we expect SH to work best for large datasets, for which there is not enough time to train many ML pipelines for the full budget, but for which training a ML pipeline on a small budget already yields a good indication of the generalization error.

5.3 Preview of Experimental Results

Choosing the correct evaluation strategy not only depends on the characteristics of the dataset at hand, but also on the given time-limit. While there exist general recommendations, we observed in practice that this is a crucial design decision that drastically impacts performance. To highlight this effect, in Figure 2 we show exemplary results comparing holdout, 3CV, 5CV, 10CV with and without SH on different optimization budgets and datasets. We give details on the SH hyperparameters in Appendix C.

The top row shows results obtained using the same optimization budget of 10 minutes on two different datasets. While holdout without SH is best on dataset *robert* (top left) the same strategy performs worst on dataset *fabert* (top right). Also, on *robert*, SH performs slightly worse in contrast to *fabert*, where SH performs better on average. The bottom rows shows how the given time-limit impacts the performance. Using a quite restrictive optimization budget of 10 minutes (bottom left), SH with holdout, which aggressively cuts ML pipelines on lower budgets, performs best on average. With a higher optimization budget (bottom right), the overall results improve, but holdout is also no longer the best option and 3CV performs best.
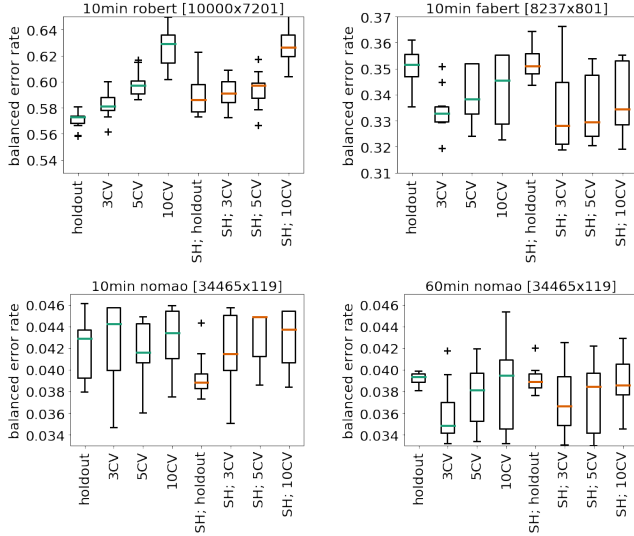
Fig. 2. Final performance for BO using different model selection strategies averaged across 10 repetitions. Top row: Results for an optimization budget of 10 minutes on two different datasets. Bottom row: Results for an optimization budget of 10 and 60 minutes on the same dataset.

6 IMPROVEMENT 2: PORTFOLIO BUILDING

Finding the optimal solution to the optimization problem from Eq. (5) requires to search a large space of possible solutions as efficiently as possible. BO is built to work under exactly these conditions, however, it starts from scratch for every new problem. A better solution would be to warmstart BO with ML pipelines that are expected to work well, such as KND described in Section 3.3. However, we found this solution to introduce new problems: First, it is time consuming since it requires to compute meta-features describing a new dataset, where good meta-features are often quite expensive to compute. Second, it adds complexity to the system as the computation of the meta-features must also be done with a time and memory limit. Third, a lot of meta-features are not defined with respect to categorical data and missing values, making them hard to apply for most datasets. Fourth, it is not immediately clear which meta-features work best for which problem. Fifth, in the KND approach mentioned in Section 3.3, there is no mechanism to guarantee that we do not execute redundant ML pipelines. Therefore, here we propose a meta-feature-free approach which does not warmstart with a set of configurations specific to a new dataset, but which uses a *portfolio* – a set of complementary configurations that covers as many diverse datasets as possible and minimizes the risk of failure when facing a new task.

Portfolios were introduced for hard combinatorial optimization problems, where the runtime between different algorithms varies drastically and allocating time shares to multiple algorithms instead of allocating all available time to a single one reduces the average cost for solving a problem [30], [31]. Algorithm portfolios were introduced to ML with the goal of reducing the required time to perform model selection compared to running all ML pipelines under consideration [32], [33], [34]. Portfolios of ML pipelines can be superior to BO for hyperparameter optimization [35] or BO with a model that takes past performance data into

```
1: Input: Set of candidate ML pipelines C,
   D_meta = {D_1, ..., D_K}, maximal portfolio size P, model
   selection strategy S
2: P = ∅
3: while |P| < P do
4:    c* ∈ argmin_{c∈C} GE_S(P ∪ {c}, D_meta)
5:    P = P ∪ c*, C = C \ {c*}
6: end while
7: return Portfolio P
   Algorithm 1: Greedy Portfolio Building
```

account [36]. They can also be applied when there is simply no time to perform full hyperparameter optimization [8] which is our main motivation.

6.1 Approach

To improve the efficiency of *Auto-sklearn (2.0)* in its early phase and to obtain results if there is no time for thorough hyperparameter optimization, we build a portfolio consisting of high-performing and complementary ML pipelines to perform well on as many datasets as possible. All pipelines in the portfolio are simply evaluated one after the other instead of an initial design or pipelines proposed by a global optimization algorithm.

We outline the proposed process in Algorithm 1, which is motivated by the *Hydra* algorithm [37], [38]. First, we initialize our portfolio P to the empty set (Line 2). Then, we repeat the following procedure until |P| reaches a predefined limit: from a set of candidate ML pipelines C, we greedily add a candidate $c^* \in C$ to P that reduces the estimated generalization error over all meta-train datasets most (Line 4), and then remove the c^* from C (Line 5).

We define the estimated generalization error of P across all meta-train datasets $\mathbf{D}_{meta} = D_1, ..., D_K$ as

$$\widehat{GE}_S(P, \mathbf{D}_{meta}) =$$
$$\sum_{k=1}^{K} \widehat{GE}_S(S(\mathcal{L}, P, D'_k \subset D_k), D_k \setminus D'_k), \quad (8)$$

which is the estimated generalization error of selecting the ML pipeline $\mathcal{M}_{\lambda} \in P$ according to the model selection strategy S, where S is a function which trains different $\mathcal{M}_{\lambda}$, compares them with respect to their estimated generalization error and returns the best one as described in the previous section, see Appendix B for further details.

In contrast to *Hydra*, we first run BO on each meta-dataset and use the best found solution as a candidate. Then, we evaluate each of these candidates on each meta-train dataset in $\mathbf{D}_{meta}$ to obtain a *performance matrix* which we use as a lookup table to construct the portfolio. To build a portfolio across datasets, we need to take into account that the generalization errors for different datasets live on different scales [39]. Thus, before taking averages, we transform them to the simple regret scaled between zero and one for each dataset [36], [40]. We compute the statistics for zero-one scaling by taking the results of all model selection strategies into account (i.e., we use the lowest observed test loss and the largest observed test loss for each meta-train dataset).

For each meta-train dataset D_k, as mentioned before, we split the training set $D_{k,train}$ into two smaller disjoint

sets $\mathcal{D}^{train}_{k,train}$ and $\mathcal{D}^{valid}_{k,train}$. We usually train models using $\mathcal{D}^{train}_{k,train}$, use $\mathcal{D}^{valid}_{k,train}$ to choose a ML pipeline $\mathcal{M}_{\boldsymbol{\lambda}}$ from the portfolio by means of the model selection strategy, and judge the portfolio quality by the generalization loss of $\mathcal{M}_{\boldsymbol{\lambda}}$ on $\mathcal{D}_{k,test}$. However, if we instead select the ML pipeline on the test set $\mathcal{D}_{k,test}$, we obtain a submodular algorithm which we detail in Section 6.2. Therefore, we follow this approach in practice, but we emphasize that this only affects the offline phase; for a new dataset, our algorithm of course does not access the test set.

6.2 Theoretical Properties of the Greedy Algorithm

Besides the already mentioned practical advantages of the proposed greedy algorithm, the worst-case performance of the portfolio is even bounded.

Proposition 1. Minimizing the test loss of a portfolio $\mathcal{P}$ on a set of datasets $\mathcal{D}_1, \ldots, \mathcal{D}_K$, when choosing a ML pipeline from $\mathcal{P}$ for $\mathcal{D}_k$ using holdout or cross-validation based on its performance on $\mathcal{D}_{k,test}$, is equivalent to the *sensor placement problem* for minimizing detection time [41].

We detail this equivalence in Appendix B. Thereby, we can apply existing results for the sensor placement problem to our problem and can conclude that the greedy portfolio building algorithm choosing on $\mathcal{D}_{k,test}$ proposed in Section 6.1 is submodular and monotone. Using the test set of the meta-train datasets $\mathbf{D}_{\mathrm{meta}}$ to construct a portfolio is perfectly fine as long as we do not use the meta-test datasets $\mathbf{D}_{\mathrm{test}}$.

This finding has several important implications. First, we can directly apply the proof from Krause et al. [41] that the so-called penalty function (maximum estimated generalization error minus the observed estimated generalization error) is submodular and monotone to our problem setup. Since linear combinations of submodular functions are also submodular [42], the penalty function for all meta-train datasets is also submodular. Second, we know that the problem of finding an optimal portfolio $\mathcal{P}$ is NP-hard [41], [43]. Third, the reduction of regret achieved by the greedy algorithm is at least $(1-\frac{1}{e})GE(\mathcal{P}^*, \mathbf{D}_{\mathrm{meta}})$, meaning that we reduce our regret to at most 37% of what the best possible portfolio $\mathcal{P}^*$ would achieve [42], [43]. A generalization of this result given by Krause and Golovin [42] also allows to reduce the regret to 1% of what the best possible portfolio of size k would achieve by extending the portfolio to size $5k$. This means that we can find a close-to-optimal portfolio on the meta-train datasets $\mathbf{D}_{\mathrm{meta}}$. Under the assumption that we apply the portfolio to datasets from the same distribution of datasets, we have a strong set of default ML pipelines. Fourth, we can apply other strategies for the sensor set placement in our problem setting, such as mixed integer programming strategies; however, these do not scale to portfolio sizes of a dozen ML pipelines [41]. The same proof and consequences apply if we select a ML pipeline based on an intermediate step in a learning curve or use cross-validation instead of holdout. We describe the properties of the greedy algorithm when using SH, and when choosing an algorithm on the validation set in Appendix B.

6.3 Preview of Experimental Results

We introduced the portfolio-based warmstarting to avoid computing meta-features for a new dataset. However, the

TABLE 1
Averaged normalized balanced error rate. We report the aggregated performance across 10 repetitions and 39 datasets of our AutoML system using only Bayesian optimization (*BO*), or BO warmstarted with k-nearest-datasets (*KND*) or a greedy portfolio (*Port*). We boldface the best mean value (per model selection strategy and optimization budget, and underline results that are not statistically different according to a Wilcoxon-signed-rank Test ($\alpha = 0.05$).

| | 10 minutes | | | 60 minutes | | |
	BO	KND	Port	BO	KND	Port
Holdout	4.31	**3.40**	3.48	2.95	2.84	**2.76**
SH, Holdout	4.01	3.51	**3.43**	2.91	2.74	**2.66**
3CV	6.82	5.87	**5.78**	5.39	**5.17**	5.20
SH, 3CV	6.50	6.00	**5.76**	5.43	5.21	**4.97**
5CV	9.73	8.66	9.12	7.83	**7.46**	7.62
SH, 5CV	9.58	8.43	8.93	7.85	7.43	**7.41**
10CV	17.37	15.82	**15.70**	16.15	**15.07**	17.23
SH, 10CV	16.79	15.72	**15.65**	15.74	**14.98**	15.25

portfolios work inherently differently. While KND aimed at using only well performing configurations, a portfolio is built such that there is at least one configuration that works well, which also provides a different form of initial design for BO. Here, we study the performance of the learned portfolio and compare it against *Auto-sklearn (1.0)*'s default meta-learning strategy using 25 configurations. Additionally, we also study how pure BO would perform. We give results in Table 1. For the new AutoML-hyperparameter $|\mathcal{P}|$ we chose 32 to allow two full iterations of SH with our hyperparameter setting of SH. Unsurprisingly, warmstarting improves the performance on all datasets, often by a large margin. Although the KND approach mostly does not perform statistically worse, the portfolio approach achieves a better average performance while being conceptually simpler and theoretically motivated.

7 IMPROVEMENT 3: AUTOMATED POLICY SELECTION

The goal of AutoML is to yield state-of-the-art performance without requiring the user to make low-level decisions, e.g., which model and hyperparameter configurations to apply. However, some high-level design decisions remain and thus AutoML systems suffer from a similar problem as they are trying to solve. We consider the case, where an AutoML system can be run with different optimization policies $\pi \in \Pi$ (e.g., model selection strategies) and study how to further automate AutoML using algorithm selection. In practice, we extend the formulation introduced in Eq. 7 to not contain a fixed policy π, but to contain a selector $\Xi : \mathcal{D} \to \pi$:

$$\widehat{GE}(\Xi, \mathbf{D}_{\mathrm{meta}}) = \frac{1}{|\mathbf{D}_{\mathrm{meta}}|} \sum_{\mathcal{D}_i \in \mathbf{D}_{\mathrm{meta}}} \widehat{GE}(\Xi(\mathcal{D}_i), \mathcal{D}_i). \quad (9)$$

In the remainder of this section, we describe how to construct such a selector.

7.1 Design Decisions in AutoML

Optimization strategies in AutoML itself are often heavily hyper-parameterized. In our case, we deem the model selection strategy S (see Section 5) as the most important design decision of an AutoML system. This decision depends on
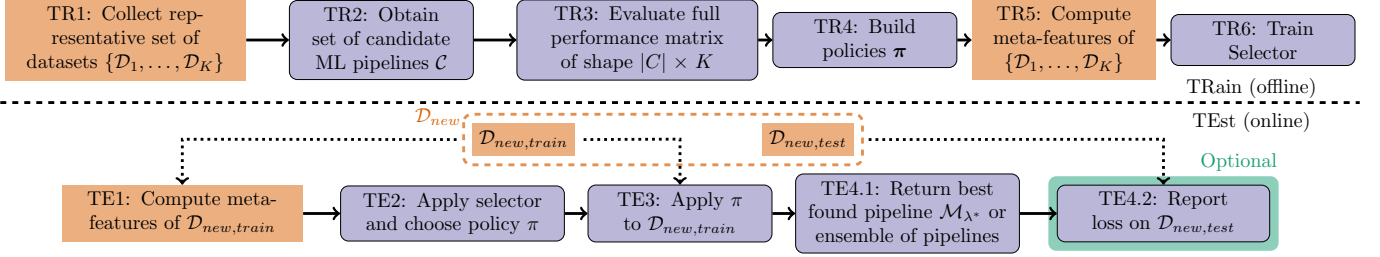
Fig. 3. Schematic overview of the proposed *Auto-sklearn (2.0)* system with the training phase above and the test phase below the dashed line. Rounded boxes refer to computational steps while rectangular boxes output data.

both the given dataset and the available resources. As there is also an interaction between the model selection strategy and the optimal portfolio $\mathcal{P}$, we consider here that the optimization policy π is parameterized by a combination of model selection strategy and a portfolio optimized for this strategy: $S \times \mathcal{P}$.

7.2 Automated Algorithm Selection of AutoML-Policies

We introduce a new layer on top of AutoML systems that automatically selects a policy π for a new dataset. We show an overview of this system in Figure 3 which consists of a training (TR1–TR6) and a testing stage (TE1–TE4). In brief, in training steps TR1–TR3, we obtain a performance matrix of size $|C| \times K$, where C is a set of candidate ML pipelines, and K is the number of representative *meta-train* datasets. This matrix is used to build policies in training step TR4, e.g., including portfolios greedily built, see Section 6. In steps TR5 and TR6, we compute meta-features and use them to train a selector which will be used in the online test phase.

For a new dataset $\mathcal{D}_{new}$, we first compute meta-features describing $\mathcal{D}_{new}$ (TE1) and use the selector from step TR6 to automatically select an appropriate policy for $\mathcal{D}_{new}$ based on the meta-features (TE2). This will relieve users from making this decision on their own. Given a policy, we then apply the AutoML system using this policy to $\mathcal{D}_{new}$ (TE3). Finally, we return the best found pipeline $\mathcal{M}_{\boldsymbol{\lambda}*}$ based on the training set of $\mathcal{D}_{new}$ (TE4.1). Optionally, we can then compute the loss of $\mathcal{M}_{\boldsymbol{\lambda}*}$ on the test set of $\mathcal{D}_{new}$ (TE4.2); we emphasize that this would be the only time we ever access the test set of $\mathcal{D}_{new}$.

7.2.1 Meta-Features

To train our selector and to select a policy, we use meta-features [18], [44] describing all meta-train datasets (TR4) and new datasets (TE1). To avoid the problems discussed in Section 6 we only use very simple and robust meta-features, which can be reliably computed in linear time for every dataset: 1) the number of data points, 2) the number of features and 3) the number of classes. In our experiments we will show that even with only these trivial and cheap meta-features we can substantially improve over a static policy.

7.2.2 Constructing the Selector

To construct the meta selection model (TR6), we follow the selector design of HydraMIP [45]: for each pair of AutoML policies, we fit a random forest to predict whether policy

π_A outperforms policy π_B given the current dataset's meta-features. Since the misclassification loss depends on the difference of the losses of the two policies (i.e. the regret when choosing the wrong policy), we weight each meta-observation by their loss difference. To make errors comparable across different datasets [39], we scale the individual error values for each dataset across all policies to be between zero and one wrt the minimal and maximal observed loss. At test time (TE2), we query all pairwise models for the given meta-features, and use voting to choose a policy π. We will refer to this strategy as *Selector*.

To obtain an estimate of the generalization error of a policy π on a dataset we run the proposed AutoML system. In order to not overestimate the performance of π on a dataset k, dataset k must not be part of the meta-data for constructing the portfolio. To overcome this issue, we perform an inner 5-fold cross-validation and build each π on four fifths of the datasets and evaluate it on the final fifth of training datasets. As we have access to the performance matrix we introduced in the previous section, constructing these additional portfolios for cross-validation comes at little cost.

To improve the performance of the selection system, we applied random search to optimize the selector's hyperparameters (its random forest's hyperparameters) [46] to minimize the error of the selector computed on out-of-bag samples [47]. Hyperparameters are shared between all pairwise models to avoid factorial growth of the number of hyperparameters with the number of new model selection strategies.

7.2.3 Backup strategy

Since our selector may not extrapolate well to datasets outside of the meta-datasets, we use a fallback measure to avoid failures due to the fact that random forests struggle to extrapolate well. Such failures can be harmful if a new dataset is much larger than any dataset in the meta-dataset and the selector proposes to use a policy that would time out without any solution. More specifically, if there is no dataset in the meta-train datasets that has higher or equal values for each meta-feature (i.e. dominates the dataset meta-features), our system falls back to use *holdout* with SH.

7.3 Preview of Experimental Results

To study the performance of the selector, we compare three different selector strategies: 1) a random policy for each dataset and each repetition (*Random*), 2) the policy that is

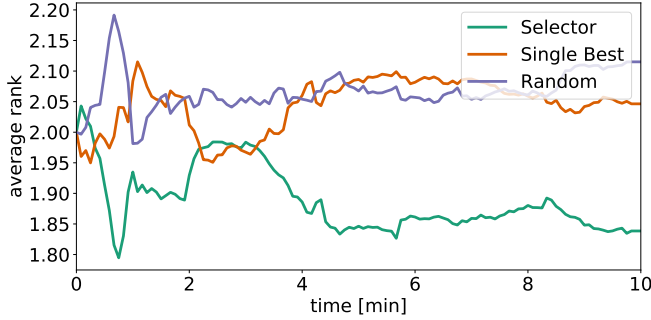| | $\varnothing$ regret | | STD | |
	10min	60min	10min	60min
Selector	**<u>3.09</u>**	**<u>2.66</u>**	0.17	0.61
Single Best	5.76	<u>4.84</u>	0.07	0.83
Oracle	2.01	1.46	0.05	0.07
Random	9.39	8.89	1.82	1.65

Fig. 4. Average rank over time. We report the averaged rank over time
across 39 datasets and 10 repetitions of our AutoML system using three
different ways to select an AutoML policy.

best on average wrt balanced error rate for each repetition
with 5-fold cross-validation on meta-train (*Single Best*), 3)
our trained selector (*Selector*) and 4) the optimal policy
(*Oracle*), which marks the lowest possible error that can
theoretically be achieved by a policy.[3]

In Table 2 we report quantitative results for short (10
minutes) and long (60 minutes) optimization budgets. As
expected, the random strategy performs worst and also
yields the highest variance across repetitions. Choosing the
policy that is best on average performs substantially better,
but still worse than using a selector.

When turning to the ranking shown in Figure 4, we ob-
serve that the random policy is competitive with the single
best policy (in contrast to the results shown in Table 2).
This is due to the fact that some policies, especially cross-
validation with a high number of folds, fail to produce re-
sults on a few datasets and thus get the worst possible error,
but work best on the majority of other, smaller datasets.
The random strategy can select these policies and therefore
achieves a similar rank as the single best policy. In contrast,
our proposed selection approach does not suffer from this
issue and outperforms both baseline methods after the first
few minutes.

---

3. Also the oracle performance is not necessarily zero, because even
evaluating the best policy on a dataset can exhibit overfitting compared
to the single best model we use to normalize data.

8 EVALUATION

To thoroughly assess the impact of our proposed improve-
ments, we now study the performance of *Auto-sklearn
(2.0)* and compare it to ablated variants of itself and *Auto-sklearn
(1.0)*. We first describe the experimental setup in Section 8.1,
conduct a large-scale ablation study in Section 8.2 and then
compare *Auto-sklearn (2.0)* against different versions of *Auto-
sklearn (1.0)* in Section 8.3.

8.1 Experimental Setup

So far, AutoML systems were designed without any opti-
mization budget or with a single, fixed optimization bud-
get T in mind (see Equation 5).[4] Our system takes the
optimization budget into account when constructing the
portfolio. When choosing an AutoML system using meta-
learning, we select a strategy and a portfolio based on both
the optimization budget and the dataset meta-features. We
will study two optimization budgets: a short, 10 minute
optimization budget and a long, 60 minute optimization
budget as in the original *Auto-sklearn* paper. To have a sin-
gle metric for binary classification, multiclass classification
and unbalanced datasets, we report the balanced error rate
($1 -$ balanced accuracy), following the 1st AutoML chal-
lenge [7]. As different datasets can live on different scales,
we apply a linear transformation to obtain comparable val-
ues. Concretely, we obtain the minimal and maximal error
obtained by executing *Auto-sklearn (2.0)* without portfolios
and ensembles, but with all available model selection strate-
gies per dataset, and rescale by subtracting the minimal
error and dividing by the difference between the maximal
and minimal error [40]. With this transformation, we obtain
a normalized error which can be interpreted as the regret of
our method.

As discussed in Section 4, we also limit the time and
memory for each machine learning pipeline. For the time
limit we allow for at most $1/10$ of the optimization budget,
while for the memory we allow the pipeline 4GB before
forcefully terminating the execution.

8.1.1 Datasets

We require two disjoint sets of datasets for our setup:
(i) $\mathbf{D}_{\text{meta}}$, on which we build portfolios and our selector
and (ii) $\mathbf{D}_{\text{test}}$, on which we evaluate our method. The
distribution of both sets ideally spans a wide variety of
problem domains and dataset characteristics. For $\mathbf{D}_{\text{test}}$, we
rely on 39 datasets selected for the AutoML benchmark
proposed in [49], which consists of datasets for comparing
classifiers [50] and datasets from the AutoML challenges [7].

We collected the meta-train datasets $\mathbf{D}_{\text{meta}}$ based on
OpenML [51] using the OpenML-Python API [52]. To obtain
a representative set, we considered all datasets on OpenML
with more than 500 and less than $1\,000\,000$ samples with
at least two attributes. Next, we dropped all datasets that
are sparse, contain time attributes or string type attributes
as the $\mathbf{D}_{\text{test}}$ does not contain any such datasets. Then, we
automatically dropped synthetic datasets and subsampled

---

4. The OBOE AutoML system [48] is a potential exception that takes
the optimization budget into consideration, but the experiments in [48]
were only conducted for a single optimization budget, not demonstrat-
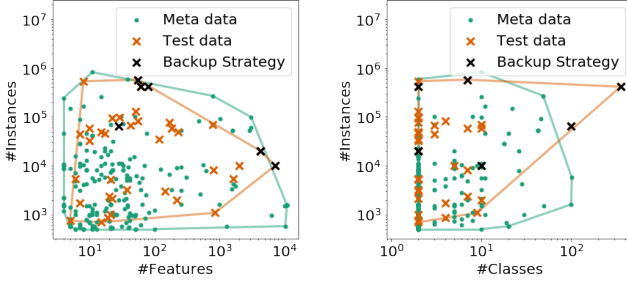ing that the system adapts itself to multiple optimization budgets.

Fig. 5. Distribution of meta and test datasets. We visualize each dataset w.r.t. its metafeatures and highlight the datasets that lie outside our meta distribution; for these, we apply a backup strategy.

clusters of highly similar datasets. Finally, we manually checked for overlap with $\mathbf{D}_{test}$ and ended up with a total of 209 training datasets and used them to design our method.

We show the distribution of the datasets in Figure 5. Green points refer to $\mathbf{D}_{meta}$ and orange crosses to $\mathbf{D}_{test}$. We can see that $\mathbf{D}_{meta}$ spans the underlying distribution of $\mathbf{D}_{test}$ quite well, but that there are several datasets which are outside of the $\mathbf{D}_{meta}$ distribution, which are marked with a black cross and for which our AutoML system selected a backup strategy (see Section 7.2.3). We give the full list of datasets for $\mathbf{D}_{meta}$ and $\mathbf{D}_{test}$ in Appendix D.

For all datasets we use a single holdout test set of 33.33% which is defined by the corresponding OpenML task. The remaining 66.66% are the training data of our AutoML systems, which handle further splits for model selection themselves based on the chosen model selection strategy.

8.1.2 Meta-data Generation

For each optimization budget we created four performance matrices, see Section 6. Each matrix refers to one way of assessing the generalization error of a model: holdout, 3-fold CV, 5-fold CV or 10-fold CV. To obtain each matrix, we did the following. For each dataset $\mathcal{D}$ in $\mathbf{D}_{meta}$, we used combined algorithm selection and hyperparameter optimization to find a customized ML pipeline. In practice, we ran SMAC [15], [53] three times for the prescribed optimization budget and picked the best resulting ML pipeline on the test split of $\mathcal{D}$. Then, we ran the cross-product of all ML pipelines and datasets to obtain the performance matrix.

8.1.3 Other Experimental Details

We always report results averaged across 10 repetitions to account for randomness and report the mean and standard deviation over these repetitions. To check whether performance differences are significant, where possible, we ran the Wilcoxon signed rank test as a statistical hypothesis test with $\alpha = 0.05$ [54]. In addition, we plot the average rank as follows. For each dataset, we draw one run per method (out of 10 repetitions) and rank these draws according to performance, using the average rank in case of ties. We repeat this sampling 200 times to obtain the average rank on a dataset, before averaging these into the total average.

We conducted all previous results without ensemble selection to focus on the individual improvements. From now on, all results include ensemble selection (and we construct ensembles of size 50 with replacement).

TABLE 3
Final performance (averaged normalized balanced error rate) for the full system and when not considering all model selection strategies.

		10 Min		60 Min	
		∅	std	∅	std
All	selector	2.27	0.16	1.88	0.12
	random	6.04	1.93	5.49	1.85
	oracle	1.15	0.07	0.92	0.05
Only Holdout	selector	2.61	0.12	2.22	0.18
	random	2.67	0.12	2.22	0.13
	oracle	2.20	0.09	1.83	0.13
Only CV	selector	4.76	0.12	4.36	0.06
	random	7.08	0.76	6.35	0.88
	oracle	3.91	0.03	3.64	0.07
Full budget	selector	2.26	0.13	1.85	0.13
	random	6.17	1.50	5.59	1.51
	oracle	1.52	0.05	1.12	0.07
Only SH	selector	2.26	0.15	1.80	0.09
	random	5.31	2.01	4.70	1.92
	oracle	1.39	0.09	1.11	0.07

All experiments were conducted on a compute cluster with machines equipped with 2 *Intel Xeon Gold 6242 CPUs* with 2.8GHz (32 cores) and 128 GB RAM, running Ubuntu 18.04.3. We provide scripts for reproducing all our experimental results at https://github.com/mfeurer/ASKL2.0_experiments and we provide an implementation within Auto-sklearn at https://automl.github.io/auto-sklearn/master/.

8.2 Ablation Study

Now, we study the contribution of each of our improvements in an ablation study. We iteratively disable one component and compare the performance to the full system. These components are (1) using only a subset of the model selection strategies, (2) warmstarting BO with a portfolio and (3) using a selector to choose a model selection strategy.

8.2.1 Do we need different model selection strategies?

We now examine whether we need the different model selection strategies discussed in Section 5. For this, we build selectors on different subsets of the available model selection strategies: *Only Holdout* consists of holdout with and without SH; *Only CV* comprises 3-fold CV, 5-fold CV and 10-fold CV, all of them with and without SH; *Full budget* contains both holdout and cross-validation and assigns each pipeline evaluation the same budget; while *Only SH* uses successive halving to assign budgets.

In Table 3, the performance of the oracle selector shows how good a set of model selection strategies could be if we could build a perfect selector. It turns out that both *Only Holdout* and *Only CV* have a much worse oracle performance than *All*, with the oracle performance of *Only CV* being even worse than the performance of the learned selector for *All*. Looking at the two budget allocation strategies, it turns out that using either of them alone (*Full budget* or *Only SH*) would be slightly preferable in terms of performance with a selector. However, the oracle performance of both is worse than that of *All* which shows that there is some complementarity in them which cannot yet be exploited by the selector.

TABLE 4
Final performance (averaged normalized balanced error rate) after 60 minutes for the full system on the left-hand side (*Auto-sklearn (2.0)*, including portfolios, BO and ensembles) and without portfolios on the right-hand side (*Auto-sklearn (2.0)*, only including BO and ensembles). We boldface the best mean value (per optimization budget) and underline results that are not statistically different according to a Wilcoxon-signed-rank Test ($\alpha = 0.05$).

| 60min | Portfolio | | No portfolio | |
	Selector	Single best	Selector	Single best
mean	**<u>1.88</u>**	<u>4.67</u>	2.21	5.08
std	0.12	0.24	0.17	0.34

While these results question the usefulness of choosing from all model selection strategies, we believe this points to the research question whether we can learn on the meta-train datasets which model selection strategies to include in the set of strategies to choose from. Also, with an ever-growing availability of meta-train datasets and continued research on robust selectors, we expect this flexibility to eventually yield improved performance.

8.2.2 Do we need portfolios?

Now we study the impact of the portfolio. For this study, we completely remove the portfolio from our AutoML system, meaning that we only run BO and construct ensembles – both for creating the data we train our selector on and for reporting performance. We compare this reduced system against full *Auto-sklearn (2.0)* in Table 4.

Comparing the performance of the AutoML system with and without portfolios (column 1 and 3), there is a clear drop in performance showing the benefit of using portfolios in our system. To demonstrate that the selector indeed helps and we do not only measure the impact of warmstarting with a portfolio, we also show the performance of the single best selector (columns 2 and 4), which is always worse than our learned selector.

8.2.3 Do we need selection at all?

Next, we examine how much performance we gain by having a selector to decide between different AutoML strategies based on meta-features and how to construct this selector. We compare the performance of the full system using a learned selector to using (1) a single, static learned strategy (single best) and (2) the selector without a fallback mechanism for out-of-distribution datasets. As a baseline, we provide results for a random selector and the oracle selector; we give all results in Table 5. All results show the performance of using a portfolio and then running BO and building ensembles for the remaining time.

An important metric in the field of algorithm selection is how much of the gap between the single best and the oracle performance one can close. We see that indeed the selector described in Section 7 is able to close most of this gap, demonstrating that there is value in using three simple meta-features to decide on the model selection strategy.

To study how much resources we need to spend on generating training data for our selector, we consider three approaches: (P) only using the portfolio performance, (P+BO) actually running the portfolio and BO for 10 and 60 minutes,

TABLE 5
Final performance (averaged normalized balanced error rate) for 10 and 60 minutes. We report the theoretical best results (*oracle*) and results for a *random* selector as baselines. The second part of the table shows the performance for the *single best* selector and the learned dynamic *selector* when trained on different data obtained on $\mathbf{D}_{\text{meta}}$ (P = Portfolio, BO = Bayesian Optimization, E = Ensemble) as well as the learned *selector without the fallback*. We boldface the best mean value (per optimization budget) and underline results that are not statistically different according to a Wilcoxon-signed-rank Test ($\alpha = 0.05$).

	10 Min			60 Min		
oracle		1.15			0.92	
random		6.64			6.06	
trained on	P	P+BO	P+BO+E	P	P+BO	P+BO+E
single best	2.65	5.00	<u>5.53</u>	2.69	<u>4.27</u>	<u>4.67</u>
selector	<u>2.39</u>	<u>2.31</u>	**2.27**	<u>1.90</u>	<u>1.91</u>	**1.88**
selector w/o fallback	<u>2.35</u>	<u>4.32</u>	3.45	<u>2.09</u>	<u>4.08</u>	<u>4.44</u>

respectively, and (P+BO+E) additionally also constructing ensembles, which yield the most correct meta-data. Running BO on all 209 datasets (P+BO) is by far more expensive than the table lookups (P); building an ensemble (P+BO+E) adds only several seconds to minutes on top compared to (P+BO).

For both optimization budgets using P+BO+E yields the best results using the selector closely followed by P+BO, see Table 5. The cheapest method, P, yields the worst results showing that it is worth to invest resources into computing good meta-data. Looking at the single best, surprisingly, performance gets worse when using seemingly better meta-data. This is due to a few selection strategies failing on large datasets: When only looking at portfolios, the robust hold-out strategy is selected as the single best model selection strategy. However, when also considering BO, there is a greater risk to overfit and thus a crossvalidation variant performs best on average on the meta-datasets; unfortunately this variant fails on some test datasets due to violating resource limitations. For such cases our fallback mechanism is quite important.

Finally, we also take a closer look at the impact of the fallback mechanism to verify that our improvements are not solely due to this component. We observe that the performance drops for five out of six of the selectors when we not include this fallback mechanism, but that the selector still outperforms the single best. The rather stark performance degradation compared to the regular selector can mostly be explained by a few, huge dataset. Based on these observations we suggest research into an adaptive fallback strategy which can change the model selection strategy during the execution of the AutoML system so that a selector can be used on out-of-distribution datasets. We conclude that using a selector is very beneficial, and using a fallback strategy to cope with out-of-distribution datasets can substantially improve performance.

8.3 Auto-sklearn (1.0) vs. Auto-sklearn (2.0)

In this section, we demonstrate the superior performance of *Auto-sklearn (2.0)* against the previous version, *Auto-sklearn (1.0)*. In Table 6 and Figure 6 we compare the performance of *Auto-sklearn (2.0)* to six different setups of *Auto-sklearn (1.0)*, including the full and the reduced search space, using only BO without KND, and using random search.
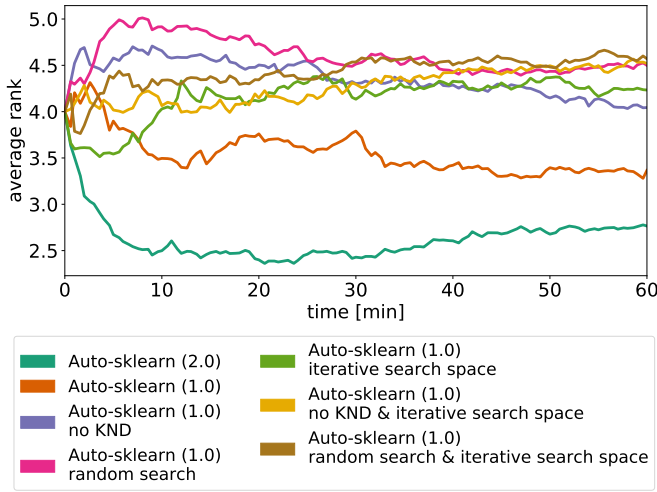
Fig. 6. Performance over time. We report the normalized BER and the rank over time averaged across 10 repetitions and 39 datasets comparing our system to our previous AutoML systems.

TABLE 6
Final performance of *Auto-sklearn (2.0)* and *Auto-sklearn (1.0)*. We report the normalized balanced error rate averaged across 10 repetitions on 39 datasets. We compare our final system (1) to the previous version downloaded as is using BO and KND on the full search space (2) (the reduced search space (5)), using only BO and no KND on the full search space (3) (the reduced search space (6)) and using only random search on the full space (4) (the reduced search space (7)). We boldface the best mean value (per optimization budget) and underline results that are not statistically different according to a Wilcoxon-signed-rank Test ($\alpha = 0.05$)

	10MIN		60MIN	
	∅	std	∅	std
(1) *Auto-sklearn (2.0)*	**2.27**	0.16	**1.88**	0.12
(2) *Auto-sklearn (1.0)*	11.76	0.09	<u>8.59</u>	0.13
(3) *Auto-sklearn (1.0)*, no KND	7.68	0.72	3.31	0.34
(4) *Auto-sklearn (1.0)*, RS	7.56	1.77	3.79	0.86
(5) *Auto-sklearn (1.0)*, only iterative	11.82	0.10	7.29	0.14
(6) *Auto-sklearn (1.0)*, no KND, only iterative	7.89	0.71	4.04	1.00
(7) *Auto-sklearn (1.0)*, RS, only iterative	8.06	0.93	3.81	0.65

Looking at the first two rows in Table 6, we see that *Auto-sklearn (2.0)* achieves the lowest error for both optimization budgets, being significantly better for the 10 minute setting. Most notably, *Auto-sklearn (2.0)* reduces the relative error by 80% (10m) and 78%, respectively, which means a reduction by a factor of five. The large difference between *Auto-sklearn (1.0)* and *Auto-sklearn (2.0)* is mainly a result of *Auto-sklearn (2.0)* being very robust by avoiding ML pipelines that cannot be trained within the given time limit while the KND approach in *Auto-sklearn (1.0)* does not avoid this failure mode. Thus, using only BO (3 and 6) or random search (4 and 7) results in better performance. It turns out that these results are skewed by three large datasets (task IDs 189873, 189874, 75193) on which the KND initialization of *Auto-sklearn (1.0)* only suggests ML pipelines that time out or hit the memory limit and thus exhaust the optimization budget for the full search space. Not taking these three datasets into account, *Auto-sklearn (1.0)* using BO and meta-learning improves over the versions without meta-learning. Our new AutoML system does not suffer from this problem as it a)

selects SH to avoid spending too much time on unpromising ML pipelines and b) can return predictions and results even if a ML pipeline was not evaluated for the full budget or converged early.

Figure 6 provides another view on the results, presenting average ranks (where failures obtain less weight compared to the averaged performance). Thus, under this view, *Auto-sklearn (1.0)* using KND and BO achieves a much better rank than the methods without meta-learning. We emphasize that despite the quite different relative performances of the *Auto-sklearn (1.0)* variants under this different evaluation setup, *Auto-sklearn (2.0)* still clearly yields the best results.

9 DISCUSSION AND CONCLUSION

Auto-sklearn (2.0) constitutes the next generation of our AutoML system *Auto-sklearn*, aiming to provide a truly hands-free system which, given a new task and resource limitations, automatically chooses the best setup. We proposed three improvements for faster and more efficient AutoML: (i) we show the necessity of different model selection strategies to work well on various datasets, (ii) to get strong results quickly we propose to use portfolios, which can be built offline and thus reduce startup costs and (iii) to close the design space we opened up for AutoML we propose to automatically select the best configuration of our system.

We conducted a large-scale study based on 209 meta-datasets and 39 datasets for testing and obtained substantially improved performance compared to *Auto-sklearn (1.0)*, reducing the regret by up to a factor of five and achieving a lower loss after 10 minutes than *Auto-sklearn (1.0)* after 60 minutes. Our ablation study showed that using a selector to choose the model selection strategy has the greatest impact on performance and allows *Auto-sklearn (2.0)* to run robustly on new, unseen datasets. In future work, we would like to compare against other AutoML systems, and the AutoML benchmark from Gijsbers et al. [49], from which we already obtained the datasets, would be a natural candidate.[5]

Our system also introduces some shortcomings since it is constructed towards a single optimization budget, a single metric and a single search space. Although all of these, along with the meta-training datasets, could be provided by a user to automatically build a customized version of *Auto-sklearn (2.0)*, it would be interesting to see whether we can learn how to transfer a specific AutoML system to different optimization budgets and metrics. Also, there still remain several hand-picked hyperparameters on the level of the AutoML system, which we would like to automate in future work, too, for example by automatically learning the portfolio size, learning more hyper-hyperparameters of the different model selection strategies (for example of SH) and learning which parts of the search space to use. Finally, building the training data is currently quite expensive. Even though this has to be done only once, it will be interesting to see whether we can take shortcuts here, for example by using a joint ranking model [55].

5. However, running the AutoML benchmark from [49] in a fair way is not trivial, since it only offers a parallel setting. We focused on other research questions first and have not yet engineered our system to exploit parallel resources. Nevertheless, *Auto-sklearn (1.0)* already performed well on that benchmark and we therefore expect the improvements of *Auto-sklearn (2.0)* to transfer to this setting as well.

ACKNOWLEDGMENTS

REFERENCES

[1] F. Hutter, L. Kotthoff, and J. Vanschoren, Eds., *Automatic Machine Learning: Methods, Systems, Challenges*, ser. SSCML. Springer, 2019.

[2] C. Thornton, F. Hutter, H. Hoos, and K. Leyton-Brown, "Auto-WEKA: combined selection and hyperparameter optimization of classification algorithms," in *Proc. of KDD'13*, 2013, pp. 847–855.

[3] B. Komer, J. Bergstra, and C. Eliasmith, "Hyperopt-sklearn: Automatic hyperparameter configuration for scikit-learn," in *ICML Workshop on AutoML*, 2014.

[4] M. Feurer, A. Klein, K. Eggensperger, J. Springenberg, M. Blum, and F. Hutter, "Efficient and robust automated machine learning," in *Proc. of NeurIPS'15*, 2015, pp. 2962–2970.

[5] R. Olson, N. Bartley, R. Urbanowicz, and J. Moore, "Evaluation of a Tree-based Pipeline Optimization Tool for Automating Data Science," in *Proc. of GECCO'16*, 2016, pp. 485–492.

[6] H. Jin, Q. Song, and X. Hu, "Auto-Keras: An efficient neural architecture search system," in *Proc. of KDD'19*, 2019, pp. 1946–1956.

[7] 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 *Automatic Machine Learning: Methods, Systems, Challenges*, ser. SSCML, F. Hutter, L. Kotthoff, and J. Vanschoren, Eds. Springer, 2019, ch. 10, pp. 177–219.

[8] M. Feurer, K. Eggensperger, S. Falkner, M. Lindauer, and F. Hutter, "Practical automated machine learning for the automl challenge 2018," in *ICML 2018 AutoML Workshop*, 2018.

[9] I. Guyon, A. Saffari, G. Dror, and G. Cawley, "Model selection: Beyond the Bayesian/Frequentist divide," *JMLR*, vol. 11, pp. 61–87, 2010.

[10] H. Mendoza, A. Klein, M. Feurer, J. Springenberg, M. Urban, M. Burkart, M. Dippel, M. Lindauer, and F. Hutter, "Towards automatically-tuned deep neural networks," in *Automatic Machine Learning: Methods, Systems, Challenges*, ser. SSCML, F. Hutter, L. Kotthoff, and J. Vanschoren, Eds. Springer, 2019, pp. 135–149.

[11] 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," *JMLR*, vol. 12, pp. 2825–2830, 2011.

[12] 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*, vol. 104, no. 1, pp. 148–175, 2016.

[13] C. Rasmussen and C. Williams, *Gaussian Processes for Machine Learning*. The MIT Press, 2006.

[14] L. Breimann, "Random forests," *MLJ*, vol. 45, pp. 5–32, 2001.

[15] F. Hutter, H. Hoos, and K. Leyton-Brown, "Sequential model-based optimization for general algorithm configuration," in *Proc. of LION'11*, 2011, pp. 507–523.

[16] M. Reif, F. Shafait, and A. Dengel, "Meta-learning for evolutionary parameter optimization of classifiers," *Machine Learning*, vol. 87, pp. 357–380, 2012.

[17] M. Feurer, J. Springenberg, and F. Hutter, "Initializing Bayesian hyperparameter optimization via meta-learning," in *Proc. of AAAI'15*, 2015, pp. 1128–1135.

[18] J. Vanschoren, "Meta-learning," in *Automatic Machine Learning: Methods, Systems, Challenges*, ser. SSCML, F. Hutter, L. Kotthoff, and J. Vanschoren, Eds. Springer, 2019, pp. 35–61.

[19] H. Escalante, M. Montes, and E. Sucar, "Particle Swarm Model Selection," *JMLR*, vol. 10, pp. 405–440, 2009.

[20] R. Caruana, A. Niculescu-Mizil, G. Crew, and A. Ksikes, "Ensemble selection from libraries of models," in *Proc. of ICML'04*, 2004.

[21] C. M. Bishop, *Neural Networks for Pattern Recognition*. Oxford University Press, Inc., 1995.

[22] S. Raschka, "Model evaluation, model selection, and algorithm selection in machine learning," *arXiv:1811.12808 [stat.ML]*, 2018.

[23] R. J. Henery, "Methods for comparison," in *Machine Learning, Neural and Statistical Classification*. Ellis Horwood, 1994, ch. 7, pp. 107 – 124.

[24] R. Kohavi and G. John, "Automatic Parameter Selection by Minimizing Estimated Error," in *Proc. of ICML'95*, 1995, pp. 304–312.

[25] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning*. Springer-Verlag, 2001.

[26] B. Bischl, O. Mersmann, H. Trautmann, and C. Weihs, "Resampling methods for meta-model validation with recommendations for evolutionary computation," *Evolutionary Computation*, vol. 20, no. 2, pp. 249–275, 2012.

[27] K. Jamieson and A. Talwalkar, "Non-stochastic best arm identification and hyperparameter optimization," in *Proc. of AISTATS'16*, 2016.

[28] Z. Karnin, T. Koren, and O. Somekh, "Almost optimal exploration in multi-armed bandits," in *Proc. of ICML'13*, 2013, pp. 1238–1246.

[29] S. Falkner, A. Klein, and F. Hutter, "BOHB: Robust and Efficient Hyperparameter Optimization at Scale," in *Proc. of ICML'18*, 2018, pp. 1437–1446.

[30] B. Huberman, R. Lukose, and T. Hogg, "An economic approach to hard computational problems," *Science*, vol. 275, pp. 51–54, 1997.

[31] C. Gomes and B. Selman, "Algorithm portfolios," *AIJ*, vol. 126, no. 1-2, pp. 43–62, 2001.

[32] P. Brazdil and C. Soares, "A comparison of ranking methods for classification algorithm selection," in *Proc. of ECML'00*, 2000, pp. 63–74.

[33] C. Soares and P. Brazdil, "Zoomed ranking: Selection of classification algorithms based on relevant performance information," in *Proc. of PKDD'00*, 2000, pp. 126–135.

[34] P. Brazdil, C. Soares, and R. Pereira, "Reducing rankings of classifiers by eliminating redundant classifiers," in *Proc. of EPAI'01*, 2001, pp. 14–21.

[35] M. Wistuba, N. Schilling, and L. Schmidt-Thieme, "Sequential Model-Free Hyperparameter Tuning," in *Proc. of ICDM '15*, 2015, pp. 1033–1038.

[36] ——, "Learning hyperparameter optimization initializations," in *Proc. of DSAA'15*, 2015, pp. 1–10.

[37] L. Xu, H. Hoos, and K. Leyton-Brown, "Hydra: Automatically configuring algorithms for portfolio-based selection," in *Proc. of AAAI'10*, 2010, pp. 210–216.

[38] M. Lindauer, H. Hoos, K. Leyton-Brown, and T. Schaub, "Automatic construction of parallel portfolios via algorithm configuration," *AIJ*, vol. 244, pp. 272–290, 2017.

[39] R. Bardenet, M. Brendel, B. Kégl, and M. Sebag, "Collaborative hyperparameter tuning," in *Proc. of ICML'13*, 2013, pp. 199–207.

[40] M. Wistuba, N. Schilling, and L. Schmidt-Thieme, "Scalable Gaussian process-based transfer surrogates for hyperparameter optimization," *Machine Learning*, vol. 107, no. 1, pp. 43–78, 2018.

[41] A. Krause, J. Leskovec, C. Guestrin, J. VanBriesen, and C. Faloutsos, "Efficient sensor placement optimization for securing large water distribution networks," *JWRPM*, vol. 134, pp. 516–526, 2008.

[42] A. Krause and D. Golovin, "Submodular function maximization," in *Tractability: Practical Approaches to Hard Problems*, L. Bordeaux, Y. Hamadi, and P. Kohli, Eds. Cambridge University Press, 2014, pp. 71–104.

[43] G. Nemhauser, L. Wolsey, and M. Fisher, "An analysis of approximations for maximizing submodular set functions," *Mathematical Programming*, vol. 14, no. 1, pp. 265–294, 1978.

[44] P. Brazdil, C. Giraud-Carrier, C. Soares, and R. Vilalta, *Metalearning: Applications to Data Mining*, 1st ed. Springer Publishing Company, Incorporated, 2008.

[45] 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.

[46] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *JMLR*, vol. 13, pp. 281–305, 2012.

[47] P. Probst, M. N. Wright, and A.-L. Boulesteix, "Hyperparameters and tuning strategies for random forest," *WIREs Data Mining and Knowledge Discovery*, vol. 9, no. 3, p. e1301, 2019.

[48] C. Yang, Y. Akimoto, D. W. Kim, and M. Udell, "OBOE: Collaborative filtering for AutoML initialization," in *Proc. of KDD'19*, 2019.

[49] P. Gijsbers, E. LeDell, S. Poirier, J. Thomas, B. Bischl, and J. Vanschoren, "An open source automl benchmark," in *ICML 2019 AutoML Workshop*, 2019.

[50] B. Bischl, G. Casalicchio, M. Feurer, F. Hutter, M. Lang, R. Mantovani, J. van Rijn, and J. Vanschoren, "Openml benchmarking suites," *arXiv*, vol. 1708.0373v2, pp. 1–6, Sep. 2019.

[51] J. Vanschoren, J. van Rijn, B. Bischl, and L. Torgo, "OpenML: Networked science in machine learning," *SIGKDD*, vol. 15, no. 2, pp. 49–60, 2014.

[52] M. Feurer, J. van Rijn, A. Kadra, P. Gijsbers, N. Mallik, S. Ravi, A. Müller, J. Vanschoren, and F. Hutter, "Openml-python: an extensible python api for openml," *arXiv:1911.02490 [cs.LG]*, 2019.

[53] M. Lindauer, K. Eggensperger, M. Feurer, S. Falkner, A. Biedenkapp, and F. Hutter, "Smac v3: Algorithm configuration in python," https://github.com/automl/SMAC3, 2017.

[54] J. Demšar, "Statistical comparisons of classifiers over multiple data sets," *JMLR*, vol. 7, pp. 1–30, 2006.

[55] A. Tornede, M. Wever, and E. Hüllermeier, "Extreme algorithm selection with dyadic feature representation," *arXiv:2001.10741 [cs.LG]*, 2020.

[56] I. Guyon, K. Bennett, G. Cawley, H. J. Escalante, S. Escalera, T. K. Ho, N. Macià, B. Ray, M. Saeed, A. Statnikov, and E. Viegas, "Design of the 2015 chalearn automl challenge," in *Proc. of IJCNN'15*. IEEE, 2015, pp. 1–8.

[57] A. Kalousis and M. Hilario, "Representational Issues in Meta-Learning," in *Proceedings of the 20th International Conference on Machine Learning*, 2003, pp. 313–320.

[58] R. 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*, ser. IJCAI'95, 1995, pp. 1137–1143.

[59] F. Mohr, M. Wever, and E. Hüllermeier, "ML-Plan: Automated machine learning via hierarchical planning," *Machine Learning*, vol. 107, no. 8-10, pp. 1495–1515, 2018.

[60] O. Maron and A. Moore, "The racing algorithm: Model selection for lazy learners," *Artificial Intelligence Review*, vol. 11, no. 1-5, pp. 193–225, 1997.

[61] A. Zheng and M. Bilenko, "Lazy Paired Hyper-Parameter Tuning," in *Proceedings of the 23rd International Joint Conference on Artificial Intelligence*, F. Rossi, Ed., 2013, pp. 1924–1931.

[62] T. Krueger, D. Panknin, and M. Braun, "Fast cross-validation via sequential testing," *JMLR*, 2015.

[63] A. Anderson, S. Dubois, A. Cuesta-infante, and K. Veeramachaneni, "Sample, Estimate, Tune: Scaling Bayesian Auto-Tuning of Data Science Pipelines," in *2017 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*, 2017, pp. 361–372.

[64] L. Li, K. Jamieson, G. DeSalvo, A. Rostamizadeh, and A. Talwalkar, "Hyperband: A novel bandit-based approach to hyperparameter optimization," *JMLR*, vol. 18, no. 185, pp. 1–52, 2018.

[65] I. Tsamardinos, E. Greasidou, and G. Borboudakis, "Bootstrapping the out-of-sample predictions for efficient and accurate cross-validation," *Machine Learning*, vol. 107, no. 12, pp. 1895–1922, 2018.

[66] K. Smith-Miles, "Cross-disciplinary perspectives on meta-learning for algorithm selection," *ACM*, vol. 41, no. 1, 2008.

[67] L. Kotthoff, "Algorithm selection for combinatorial search problems: A survey," *AI Magazine*, vol. 35, no. 3, pp. 48–60, 2014.

[68] P. Kerschke, H. Hoos, F. Neumann, and H. Trautmann, "Automated algorithm selection: Survey and perspectives," *Evolutionary Computation*, vol. 27, no. 1, pp. 3–45, 2019.

[69] F. Pfisterer, J. van Rijn, P. Probst, A. Müller, and B. Bischl, "Learning multiple defaults for machine learning algorithms," *arXiv:1811.09409 [stat.ML]* , 2018.

[70] J. Seipp, S. Sievers, M. Helmert, and F. Hutter, "Automatic configuration of sequential planning portfolios," in *Proc. of AAAI'15*, 2015.

[71] R. Leite, P. Brazdil, and J. Vanschoren, "Selecting classification algorithms with active testing," in *Proc. of MLDM*, 2013, pp. 117–131.

[72] N. Fusi, R. Sheth, and M. Elibol, "Probabilistic matrix factorization for automated machine learning," in *Advances in Neural Information Processing Systems 31*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, Eds. Curran Associates, Inc., 2018, pp. 3348–3357.

[73] A. Klein, S. Falkner, J. Springenberg, and F. Hutter, "Learning curve prediction with Bayesian neural networks," in *Proc. of ICLR'17*, 2017.

[74] M. Lindauer, H. Hoos, F. Hutter, and T. Schaub, "Autofolio: An automatically configured algorithm selector," *JAIR*, vol. 53, pp. 745–778, 2015.

[75] F. Hutter, H. Hoos, K. Leyton-Brown, and T. Stützle, "ParamILS: An automatic algorithm configuration framework," *JAIR*, vol. 36, pp. 267–306, 2009.

[76] R. Caruana, A. Munson, and A. Niculescu-Mizil, "Getting the most out of ensemble selection," in *Proc. of ICDM'06*, 2006, pp. 828–833.

[77] A. Niculescu-Mizil, C. Perlich, G. Swirszcz, V. Sindhwani, Y. Liu, P. Melville, D. Wang, J. Xiao, J. Hu, M. Singh, W. Shang, and Y. Zhu, "Winning the kdd cup orange challenge with ensemble selection," in *Proceedings of KDD-Cup 2009 Competition*, G. Dror, M. Boullé, I. Guyon, V. Lemaire, and D. Vogel, Eds., vol. 7, 2009, pp. 23–34.

[78] R. Olson and J. Moore, *TPOT: A Tree-Based Pipeline Optimization Tool for Automating Machine Learning*, ser. SSCML. Springer, 2019, pp. 151–160.

[79] M. Wistuba, N. Schilling, and L. Schmidt-Thieme, "Automatic Frankensteining: Creating Complex Ensembles Autonomously," in *Proceedings of the 2017 SIAM International Conference on Data Mining*, 2017, pp. 741–749.

[80] B. Chen, H. Wu, W. Mo, I. Chattopadhyay, and H. Lipson, "Autostacker: A Compositional Evolutionary Learning System," in *Proc. of GECCO'18*, 2018, pp. 402–409.

[81] Y. Zhang, M. Bahadori, H. Su, and J. Sun, "FLASH: Fast Bayesian Optimization for Data Analytic Pipelines," in *Proc. of KDD'16*, 2016, pp. 2065–2074.

[82] H. Rakotoarison, M. Schoenauer, and M. Sebag, "Automated machine learning with Monte-Carlo tree search," in *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*. International Joint Conferences on Artificial Intelligence Organization, 2019, pp. 3296–3303.

[83] A. Alaa and M. van der Schaar, "AutoPrognosis: Automated Clinical Prognostic Modeling via Bayesian Optimization with Structured Kernel Learning," in *Proc. of ICML'18*, 2018, pp. 139–148.

[84] L. Parmentier, O. Nicol, L. Jourdan, and M. Kessaci, "Tpot-sh: A faster optimization algorithm to solve the automl problem on large datasets," in *2019 IEEE 31st International Conference on Tools with Artificial Intelligence (ICTAI)*, 2019, pp. 471–478.

[85] H. Mendoza, A. Klein, M. Feurer, J. Springenberg, and F. Hutter, "Towards automatically-tuned neural networks," in *ICML 2016 AutoML Workshop*, 2016.

[86] J. Krarup and P. Pruzan, "The simple plant location problem: Survey and synthesis," *European Journal of Operations Research*, vol. 12, pp. 36–81, 1983.

[87] S. van der Walt, S. C. Colbert, and G. Varoquaux, "The numpy array: A structure for efficient numerical computation," *Computing in Science Engineering*, vol. 13, no. 2, pp. 22–30, 2011.

[88] P. Virtanen, R. Gommers, T. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. van der Walt, M. Brett, J. Wilson, K. Jarrod Millman, N. Mayorov, A. Nelson, E. Jones, R. Kern, E. Larson, C. Carey, İ. Polat, Y. Feng, E. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. Harris, A. Archibald, A. Ribeiro, F. Pedregosa, P. van Mulbregt, and S. . . Contributors, "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python," *Nature Methods*, vol. 17, pp. 261–272, 2020.

[89] Wes McKinney, "Data Structures for Statistical Computing in Python," in *Proceedings of the 9th Python in Science Conference*, Stéfan van der Walt and Jarrod Millman, Eds., 2010, pp. 56 – 61.

[90] J. Hunter, "Matplotlib: A 2d graphics environment," *Computing in Science & Engineering*, vol. 9, no. 3, pp. 90–95, 2007.

[91] *Proc. of KDD'19*, 2019.

[92] *Proc. of ICML'18*, 2018.

[93] *Proc. of ICML'13*, 2013.

[94] *Proc. of AAAI'15*, 2015.

APPENDIX A
RELATED WORK

In this section we provide additional related work on the improvements we presented in the main paper. First, we will provide further literature on model selection strategies.

Second, we give more details on existing work on portfolios of ML pipelines. Third, we give pointers to literature overviews of algorithm selection before discussing other existing AutoML systems in detail.

A.1 Related work on model selection strategies

Automatically choosing a model selection strategy to assess the performance of an ML pipeline for hyperparameter optimization has not previously been tackled, and only Guyon et al. [56] acknowledge the lack of such an approach. The influence of the model selection strategy on the validation and test performance is well known [57] and researchers have studied their impact [58]. Following up on this, the OpenML platform [51] stores which validation strategies were used for an experiment, but so far no work has operationalized this information. Recently, Mohr et al. [59] noted that the choice of the model selection strategy has an effect on the final test performance of an AutoML system but only made a general recommendation, too.

Our method, in general, is not specific to holdout, cross-validation or successive halving and could generalize to any other method assessing the performance of a model [9], [22], [26] or allocating resources to the evaluation of a model [15], [60], [61], [62], [63], [64], [65]. While these are important areas of research, we focus here on the most commonly used methods and leave studying these extensions for future work.

A.2 Related work on Portfolios

Using portfolios has a long history [30], [31] for leveraging the complementary strengths of algorithms (or hyperparameter settings) and had applications in different sub-fields of AI [66], [67], [68].

Algorithm portfolios were introduced to machine learning by the name of *algorithm ranking* with the goal of reducing the required time to perform model selection compared to running all algorithms under consideration [32], [33], ignoring redundant ones [34]. ML portfolios can be superior to hyperparameter optimization with Bayesian optimization [35], Bayesian optimization with a model which takes past performance data into account [36] or can be applied when there is simply no time to perform full hyperparameter optimization [8]. Furthermore, such a portfolio-based model-free optimization is both easier to implement than regular Bayesian optimization and meta-feature based solutions, and the portfolio can be shared easily across researchers and practitioners without the necessity of sharing meta-data [35], [36], [69] or additional hyperparameter optimization software.

The efficient creation of algorithm portfolios is an active area of research with the Greedy Algorithm being a popular choice [8], [35], [37], [45], [70] due to its simplicity. Wistuba et al. [35] first proposed the use of the Greedy Algorithm for pipelines of machine learning portfolios, minimizing the average rank on meta-datasets for a single machine learning algorithm. Later, they extended their work to update the members of a portfolio in a round-robin fashion, this time using the average normalized misclassification error as a loss function and relying on a Gaussian process model [36]. The loss function of the first method can suffer

from irrelevant alternatives, while the second method does not guarantee that well-performing algorithms are executed early on, which could be harmful under time constraints. In work parallel to our submission to the second AutoML challenge, Pfisterer et al. [69] also suggested using a set of default values to simplify hyperparameter optimization. They argued that constructing an optimal portfolio of hyperparameter settings is a generalization of the *Maximum coverage problem* and propose two solutions based on *Mixed Integer Programming* and the *Greedy Algorithm* which we also use as the base of our algorithm. The main difference of our work is that we demonstrate the usefulness of portfolios for high-dimensional configuration spaces of AutoML systems under strict time limits and that we give concrete worst-case performance guarantees.

Extending these portfolio strategies which are learned offline, there are online portfolios which can select from a fixed set of machine learning pipelines, taking previous evaluations into account [35], [35], [48], [71], [72]. However, such methods cannot be directly combined with all resampling strategies as they require the definition of a special model for extrapolating learning curves [29], [73] and also introduce additional complexity into AutoML systems.

There exists other work on building portfolios without prior discretization (which we do for our work and was done for all work mentioned above), which directly optimizes the hyperparameters of ML pipelines to add next to the portfolio [37], [45], [70] and to also build parallel portfolios [38].

A.3 Related Work on Algorithm Selection

Treating the choice of model selection strategy as an algorithm selection problem allows us to apply methods from the field of algorithm selection [66], [67], [68] and we can in future work reuse existing techniques besides pairwise classification [45]. An especially promising candidate is AutoFolio [74], an AutoAI system which automatically constructs a selector for a given algorithm selection problem using algorithm configuration [75].

A.4 Related Work on AutoML systems

AutoML systems have recently gained traction in the research community and there exist a multitude of approaches with many of them being either available as supplementary material or open source software.

To the best of our knowledge, the first AutoML system which tunes both hyperparameters and chooses algorithms was an ensemble method [20]. The system randomly produces 2000 classifiers from a wide range of ML algorithms and constructs a post-hoc ensemble. It was later robustified [76] and employed in a winning KDD challenge [77].

The first AutoML system to jointly optimize the whole pipeline is *Particle Swarm Model Selection*. Later systems started employing model-based global optimization algorithms, such as Auto-WEKA [2] and Hyperopt-sklearn [3]. We extended this approach using meta-learning and including ensembles in *Auto-sklearn* [4].

Relieving the limitation of a fixed search space, the *tree-based pipeline optimization tool* (TPOT [78]) uses a pipeline

grammar and grammatical evolution to construct ML pipelines of arbitrary length.

Instead of a single layer of ML algorithms followed by an ensembling mechanism, Wistuba et al. [79] proposed two-layer stacking, applying AutoML to the outputs of an AutoML system. Auto-Stacker went one step further, directly optimizing for a two-layer AutoML system [80].

Another strain of work on AutoML systems aims at more efficient optimization. FLASH [81] proposed a pruning mechanism to reduce the pipeline space to search through, MOSAIC [82] uses Monte-Carlo Tree search to efficiently search the tree-structured space of ML pipelines and ML-PLAN uses hierarchical task-networks and a randomized depth-first search [59]. Auto-Prognosis [83] splits the optimization problem of the full pipeline into smaller optimization problems which can then be tackled by Gaussian process-based BO. TPOT-SH [84], inspired by our submission to the second AutoML challenge, uses successive halving to speed up TPOT on large datasets.

Finally, while the AutoML tools discussed so far focus on "traditional" machine learning, there is also work on creating AutoML systems that can leverage recent advancements in deep learning. Auto-Net extended the Auto-WEKA approach to deep neural networks [85] and Auto-Keras employs Neural Architecture Search to find well-performing neural networks [6].

Of course, there are also many techniques related to AutoML which are not used in one of the AutoML systems discussed in this section and we refer to Hutter et al. [1] for an overview of the field of Automated Machine Learning and Brazdil et al. [44] for an overview on meta-learning research which pre-dates the work on AutoML.

APPENDIX B
DETAILS ON GREEDY PORTFOLIO CONSTRUCTION
B.1 Holdout as a Model Selection Strategy

In the main paper we have only defined $S : \mathcal{L} \times \mathcal{P} \times \mathcal{D} \to \mathcal{M}_{\boldsymbol{\lambda}}^{\mathcal{P}}$, but not how it practically works. For holdout, S is defined as:

$$\mathcal{M}_{\boldsymbol{\lambda}}^{\mathcal{D}^{train}} \in \operatorname*{argmin}_{\mathcal{M}_{\boldsymbol{\lambda}}^{\mathcal{D}^{train}} \in \mathcal{P}} \widehat{GE}_{holdout}(\mathcal{M}_{\boldsymbol{\lambda}}^{\mathcal{D}^{train}}, \mathcal{D}^{valid}) \quad (10)$$

, while for cross-validation we can plug in the definition for $\widehat{GE}_{CV}$ from Section 2 of the main paper. Successive Halving is a bit more involved an cannot be written in a single equation, but would require pseudo-code.

B.2 Theoretical properties of the greedy algorithm
B.2.1 Definitions

Definition 1. (Discrete derivative, from Krause & Golovin [42]) For a set function $f : 2^{\mathcal{V}} \to \mathbb{R}, \mathcal{S} \subseteq \mathcal{V}$ and $e \in \mathcal{V}$ let $\Delta_f(e|\mathcal{S}) = f(\mathcal{S} \cup \{e\}) - f(\mathcal{S})$ be the *discrete derivative* of f at S with respect to e.

Definition 2. (Submodularity, from Krause & Golovin [42]): A function $f : 2^{\mathcal{V}} \to \mathbb{R}$ is *submodular* if for every $\mathcal{A} \subseteq \mathcal{B} \subseteq \mathcal{V}$ and $e \in \mathcal{V} \setminus \mathcal{B}$ it holds that $\Delta_f(e|\mathcal{A}) \geq \Delta_f(e|\mathcal{B})$.

Definition 3. (Monotonicity, from Krause & Golovin [42]): A function $f : 2^{\mathcal{V}} \to \mathbb{R}$ is *monotone* if for every $\mathcal{A} \subseteq \mathcal{B} \subseteq V, f(\mathcal{A}) \leq f(\mathcal{B})$.

B.2.2 Choosing on the test set

In this section we give a proof of Proposition 1 from the main paper:

Proposition 2. Minimizing the test loss of a portfolio $\mathcal{P}$ on a set of datasets $\mathcal{D}_1, \ldots, \mathcal{D}_K$, when choosing a ML pipeline from $\mathcal{P}$ for $\mathcal{D}_k$ based on performance on $\mathcal{D}_{k,test}$, is equivalent to the *sensor placement problem* for minimizing detection time [41].

Following Krause et al. [41], sensor set placement aims at maximizing a so-called *penalty reduction* $R(\mathcal{A}) = \sum_{i \in \mathcal{I}} P(i) R(\mathcal{A}, i)$, where $\mathcal{I}$ are intrusion scenarios following a probability distribution P with i being a specific intrusion. $\mathcal{A} \subset \mathcal{C}$ is a *sensor placement*, a subset of all possible locations $\mathcal{C}$ where sensors are actually placed. Penalty reduction R is defined as the reduction of the penalty when choosing $\mathcal{A}$ compared to the maximum penalty possible on scenario i: $R(\mathcal{A}, i) = \text{penalty}_i(\infty) - \text{penalty}_i(T(\mathcal{A}, i))$. In the simplest case where action is taken upon intrusion detection, the penalty is equal to the detection time ($\text{penalty}_i(t) = t$). The detection time of a sensor placement $T(\mathcal{A}, i)$ is simply defined as the minimum of the detection times of its individual members: $\min_{s \in \mathcal{A}} T(s, i)$.

In our setting, we need to do the following replacements to find that the problems are equivalent:

1) Intrusion scenarios $\mathcal{I}$: datasets $\{\mathcal{D}^{(1)}, \ldots, \mathcal{D}_k\}$,
2) Possible sensor locations $\mathcal{C}$: set of candidate ML pipelines of our algorithm $\mathcal{C}$, Detection time $T(s \in \mathcal{A}, i)$ on intrusion scenario i: test performance $\mathcal{L}(\mathcal{M}_C, \mathcal{D}_{k,test})$ on dataset $\mathcal{D}_k$,
3) Detection time of a sensor placement $T(\mathcal{A}, i)$: test loss of applying portfolio $\mathcal{P}$ on dataset $\mathcal{D}$: $\min_{p \in \mathcal{P}} \mathcal{L}(p, \mathcal{D}_{k,test})$
4) Penalty function $\text{penalty}_i(t)$: loss function $\mathcal{L}$, in our case, the penalty is equal to the loss.
5) Penalty reduction for an intrusion scenario $R(\mathcal{A}, i)$: the penalty reduction for successfully applying a portfolio $\mathcal{P}$ to dataset k: $R(\mathcal{P}, k) = \text{penalty}_k(\infty) - \min_{p \in \mathcal{P}} \mathcal{L}(p, \mathcal{D}_{k,test})$.

∎

B.2.3 Choosing on the validation set

We demonstrate that choosing an ML pipeline from the portfolio via holdout (i.e. a validation set) and reporting its test performance is neither submodular nor monotone by a simple example. To simplify notation we argue in terms of performance instead of penalty reduction, which is equivalent.

Let $\mathcal{B} = \{(5, 5), (7, 7), (10, 10)\}$ and $\mathcal{A} = \{(5, 5), (7, 7)\}$, where each tuple represents the validation and test performance. For $e = (8, 6)$ we obtain the discrete derivatives $\Delta_f(e|\mathcal{A}) = -1$ and $\Delta_f(e|B) = 0$ which violates Definition 2. The fact that the discrete derivative is negative violates Definition 3 because $f(\mathcal{A}) > f(\mathcal{A} \cup \{e\})$.

B.2.4 Successive Halving

As in the previous subsection, we use a simple example to demonstrate that selecting an algorithm via the successive halving model selection strategy is neither submodular nor

monotone. To simplify notation we argue in terms of performance instead of penalty reduction, which is equivalent.

Let $\mathcal{B} = \{((5,5),(8,8)),((5,5),(6,6)),((4,4),(5,5))\}$ and $\mathcal{A} = \{((5,5),(7,7))\}$, where each tuple is a learning curve of validation-, test performance tuples. For $e = ((6,5),(6,5))$, we eliminate entries 2 and 3 from $\mathcal{B}$ in the first iteration of successive halving (while we advance entries 1 and 4), and we eliminate entry 1 from $\mathcal{A}$. After the second stage, the performances are $f(\mathcal{B}) = 8$ and $f(\mathcal{A}) = 5$, and the discrete derivatives $\Delta_f(e|\mathcal{A}) = -1$ and $\Delta_f(e|B) = 0$ which violates Definition 2. The fact that the discrete derivative is negative violates Definition 3 because $f(\mathcal{A}) > f(\mathcal{A} \cup \{e\})$.

B.2.5 Further equalities

In addition, our problem can also be phrased as a *facility location* problem [86] and statements about the facility location problem can be applied to our problem setup as well.

APPENDIX C
IMPLEMENTATION DETAILS

C.1 Software

We implemented the AutoML systems and experiments in the Python3 programming language, using *numpy* [87], *scipy* [88], *scikit-learn* [11], *pandas* [89], and *matplotlib* [90].

C.2 Configuration Space

We give the configuration space we use in *Auto-sklearn (2.0)* in Table 7.

C.3 Successive Halving hyperparameters

We used the same hyperparameters for all experiments. First, we set to $eta = 4$. Next, we had to choose the minimal and maximal budgets assigned to each algorithm. For the tree-based methods we chose to go from 32 to 512, while for the linear models (SGD and passive aggressive) we chose 64 as the minimal budget and 1024 as the maximal budget. Further tuning these hyperparameters would be an interesting, but an expensive way forward.

APPENDIX D
DATASETS

We give the name, OpenML dataset ID, OpenML task ID and the size of all datasets we used in Table 8.

TABLE 7

Configuration space for *Auto-sklearn (2.0)* using only iterative models and only preprocessing to transform data into a format that can be usefully employed by the different classification algorithms. The final column (log) states whether we actually search $log_{10}(\lambda)$.

Name	Domain	Default	Log
Classifier	(Extra Trees, Gradient Boosting, Passive Aggressive, Random Forest, Linear Model (SGD)	Random Forest	-
Extra Trees: Bootstrap	(True, False)	False	-
Extra Trees: Criterion	(Gini, Entropy)	Gini	-
Extra Trees: Max Features	$[0.0, 1.0]$	0.5	No
Extra Trees: Min Samples Leaf	$[1, 20]$	1	No
Extra Trees: Min Samples Split	$[2, 20]$	2	No
Gradient Boosting: Early Stopping	(Off, Train, Valid)	Off	-
Gradient Boosting: L_2 Regularization	$[1e-10, 1.0]$	1e-10	Yes
Gradient Boosting: Learning Rate	$[0.01, 1.0]$	0.1	Yes
Gradient Boosting: Max Leaf Nodes	$[3, 2047]$	31	Yes
Gradient Boosting: Min Samples Leaf	$[1, 200]$	20	Yes
Gradient Boosting: #Iter No Change	$[1, 20]$	10	No
Gradient Boosting: Validation Fraction	$[0.01, 0.4]$	0.1	No
Passive Aggressive: C	$[1e-05, 10.0]$	1	Yes
Passive Aggressive: Average	(False, True)	False	-
Passive Aggressive: Loss	(Hinge, Squared Hinge)	Hinge	-
Passive Aggressive: Tolerance	$[1e-05, 0.1]$	0.0001	Yes
Random Forest: Bootstrap	(True, False)	True	-
Random Forest: Criterion	(Gini, Entropy)	Gini	-
Random Forest: Max Features	$[0.0, 1.0]$	0.5	No
Random Forest: Min Samples Leaf	$[1, 20]$	1	No
Random Forest: Min Samples Split	$[2, 20]$	2	No
Sgd: α	$[1e-07, 0.1]$	0.0001	Yes
Sgd: Average	(False, True)	False	-
Sgd: ϵ	$[1e-05, 0.1]$	0.0001	Yes
Sgd: η_0	$[1e-07, 0.1]$	0.01	Yes
Sgd: l_1 Ratio	$[1e-09, 1.0]$	0.15	Yes
Sgd: Learning Rate	(Optimal, Invscaling, Constant)	Invscaling	-
Sgd: Loss	(Hinge, Log, Modified Huber, Squared Hinge, Perceptron)	Log	-
Sgd: Penalty	$(L_1, L_2, \text{Elastic Net})$	l2	-
Sgd: Power t	$[1e-05, 1.0]$	0.5	No
Sgd: Tolerance	$[1e-05, 0.1]$	0.0001	Yes
Balancing	(None, Weighting)	None	-
Categorical Encoding: Choice	(None, One Hot Encoding)	One Hot Encoding	-
Category Coalescence: Choice	(Minority Coalescer, No Coalescense)	Minority Coalescer	-
Minority Coalescer: Minimum percentage samples	$[0.0001, 0.5]$	0.01	Yes
Imputation (numerical only)	(Mean, Median, Most Frequent)	Mean	-
Rescaling (numerical only)	(Min/Max, None, Normalize, Quantile, Standardize, Robust)	Standardize	-
Quantile Transformer: N Quantiles	$[10, 2000]$	1000	No
Quantile Transformer: Output Distribution	(Uniform, Normal)	Uniform	-
Robust Scaler: Q Max	$[0.7, 0.999]$	0.75	No
Robust Scaler: Q Min	$[0.001, 0.3]$	0.25	No

TABLE 8
Characteristics of the 209 datasets in $\mathbf{D}_{meta}$ (first part) and the 39 datasets in $\mathbf{D}_{test}$ (second part) sorted by number of features. We report for each dataset the task id and dataset id as used on OpenML.org, the number of observations, the number of features and the number of classes.

name	tid	#obs	#feat	#cls
OVA_O ...	75126	1545	10937	2
OVA_C ...	75125	1545	10937	2
OVA_P ...	75121	1545	10937	2
OVA_E ...	75120	1545	10937	2
OVA_K ...	75116	1545	10937	2
OVA_L ...	75115	1545	10937	2
OVA_B ...	75114	1545	10937	2
UMIST ...	189859	575	10305	20
amazo ...	189878	1500	10001	50
eatin ...	189786	945	6374	7
CIFAR ...	167204	60000	3073	10
SVHN	189857	99289	3073	10
GTSRB ...	190156	51839	2917	43
Biore ...	75156	3751	1777	2
hiva_ ...	166996	4229	1618	2
GTSRB ...	190157	51839	1569	43
GTSRB ...	190158	51839	1569	43
Inter ...	168791	3279	1559	2
micro ...	146597	571	1301	20
Devna ...	167203	92000	1025	46
GAMET ...	167085	1600	1001	2
Kuzus ...	190154	270912	785	49
mnist ...	75098	70000	785	10
Kuzus ...	190159	70000	785	10
isole ...	75169	7797	618	26
har	126030	10299	562	6
madel ...	146594	2600	501	2
KDD98 ...	211723	82318	478	2
phili ...	189864	5832	309	2
madel ...	189863	3140	260	2
USPS	189858	9298	257	10
semei ...	75236	1593	257	10
GTSRB ...	190155	51839	257	43
India ...	211720	9144	221	8
dna	167202	3186	181	3
musk	75108	6598	170	2
Speed ...	146679	8378	123	2
hill- ...	146592	1212	101	2
fri_c ...	166866	500	101	2
MiceP ...	167205	1080	82	8
meta_ ...	2356	45164	75	11
ozone ...	75225	2534	73	2
analc ...	146576	841	71	4
kdd_i ...	166970	10108	69	2
optdi ...	258	5620	65	10
one-h ...	75154	1600	65	100
synth ...	146574	600	62	6
splic ...	275	3190	61	3
spamb ...	273	4601	58	2
first ...	75221	6118	52	6
fri_c ...	75180	1000	51	2
fri_c ...	166944	500	51	2
fri_c ...	166951	500	51	2
Diabe ...	189828	101766	50	3
oil_s ...	3049	937	50	2
pol	75139	15000	49	2
tokyo ...	167100	959	45	2
qsar- ...	75232	1055	42	2
textu ...	126031	5500	41	11
autoU ...	189899	750	41	8
ailer ...	75146	13750	41	2
wavef ...	288	5000	41	3
cylin ...	146600	540	40	2
water ...	166953	527	39	2
annea ...	232	898	39	5
mc1	75133	9466	39	2
pc4	75092	1458	38	2
pc3	75129	1563	38	2
porto ...	211722	595212	38	2
pc2	75100	5589	37	2
satim ...	2120	6430	37	6
Satel ...	189844	5100	37	2
soybe ...	271	683	36	19
cardi ...	75217	2126	36	10
cjs	146601	2796	35	6
colle ...	75212	1302	35	2
puma3 ...	75153	8192	33	2
Gestu ...	75109	9873	33	5
kick	189870	72983	33	2
bank3 ...	75179	8192	33	2
wdbc	146596	569	31	2
Phish ...	75215	11055	31	2
fars	189840	100968	30	8

name	tid	# obs	# feat	# class
hypot ...	3044	3772	30	4
steel ...	168785	1941	28	7
eye_m ...	189779	10936	28	3
fri_c ...	75136	1000	26	2
fri_c ...	75199	1000	26	2
wall- ...	75235	5456	25	4
led24 ...	189841	3200	25	10
colli ...	189845	1000	24	30
rl	189869	31406	23	2
mushr ...	254	8124	23	2
meta	166875	528	22	2
jm1	75093	10885	22	2
pc1	75159	1109	22	2
kc2	146583	522	22	2
cpu_a ...	75233	8192	22	2
autoU ...	75089	1000	21	2
GAMET ...	167086	1600	21	2
GAMET ...	167087	1600	21	2
bosto ...	166905	506	21	2
GAMET ...	167088	1600	21	2
GAMET ...	167089	1600	21	2
churn ...	167097	5000	21	2
clima ...	167106	540	21	2
micro ...	189875	20000	21	5
GAMET ...	167090	1600	21	2
Traff ...	211724	70340	21	3
ringn ...	75234	7400	21	2
twono ...	75187	7400	21	2
eucal ...	2125	736	20	5
eleva ...	75184	16599	19	2
pbcse ...	166897	1945	19	2
baseb ...	2123	1340	18	3
house ...	75174	22784	17	2
colle ...	75196	1161	17	2
BachC ...	189829	5665	17	102
pendi ...	262	10992	17	10
lette ...	236	20000	17	26
spoke ...	75178	263256	15	10
eeg-e ...	75219	14980	15	2
wind	75185	6574	15	2
Japan ...	126021	9961	15	9
compa ...	211721	5278	14	2
vowel ...	3047	990	13	11
cpu_s ...	75147	8192	13	2
autoU ...	189900	700	13	3
autoU ...	75118	1100	13	5
dress ...	146602	500	13	2
senso ...	166906	576	12	2
wine- ...	189836	4898	12	7
wine- ...	189843	1599	12	6
Magic ...	75112	19020	12	2
mv	75195	40768	11	2
parit ...	167101	1124	11	2
mofn- ...	167094	1324	11	2
fri_c ...	75149	1000	11	2
poker ...	340	829201	11	10
fri_c ...	166950	500	11	2
page- ...	260	5473	11	5
ilpd	146593	583	11	2
2dpla ...	75142	40768	11	2
fried ...	75161	40768	11	2
rmfts ...	166859	508	11	2
stock	166915	950	10	2
tic-t ...	279	958	10	2
breas ...	245	699	10	2
xd6	167096	973	10	2
cmc	253	1473	10	3
profb ...	146578	672	10	2
diabe ...	267	768	9	2
abalo ...	2121	4177	9	28
bank8 ...	75141	8192	9	2
elect ...	336	45312	9	2
kdd_e ...	166913	782	9	2
house ...	75176	20640	9	2
nurse ...	256	12960	9	5
kin8n ...	75166	8192	9	2
yeast ...	2119	1484	9	10
puma8 ...	75171	8192	9	2
analc ...	75143	4052	8	2
ldpa	75134	164860	8	11
pm10	166872	500	8	2
no2	166932	500	8	2
LED-d ...	146603	500	8	10

name	tid	#obs	#feat	#cls
artif ...	126028	10218	8	10
monks ...	3055	554	7	2
space ...	75148	3107	7	2
kr-vs ...	75223	28056	7	18
monks ...	3054	601	7	2
Run_o ...	167103	88588	7	2
delta ...	75173	9517	7	2
strik ...	166882	625	7	2
mammo ...	3048	11183	7	2
monks ...	3053	556	7	2
kropt ...	2122	28056	7	18
delta ...	75163	7129	6	2
wilt	167105	4839	6	2
fri_c ...	75131	1000	6	2
mozil ...	126024	15545	6	2
polle ...	75192	3848	6	2
socmo ...	75213	1156	6	2
irish ...	146575	500	6	2
fri_c ...	166931	500	6	2
arsen ...	166957	559	5	2
arsen ...	166956	559	5	2
walki ...	75250	149332	5	22
analc ...	146577	797	5	6
bankn ...	146586	1372	5	2
arsen ...	166959	559	5	2
visua ...	75210	8641	5	2
balan ...	241	625	5	3
arsen ...	166958	559	5	2
volca ...	189902	10130	4	5
skin- ...	75237	245057	4	2
tamil ...	189846	45781	4	20
quake ...	75157	2178	4	2
volca ...	189893	8654	4	5
volca ...	189890	8753	4	5
volca ...	189887	9989	4	5
volca ...	189884	10668	4	5
volca ...	189883	10176	4	5
volca ...	189882	1515	4	5
volca ...	189881	1521	4	5
volca ...	189880	1623	4	5
Titan ...	167099	2201	4	2
volca ...	189894	1183	4	5

name	tid	#obs	#feat	#cls
rober ...	168794	10000	7201	10
ricca ...	168797	20000	4297	2
guill ...	168796	20000	4297	2
dilbe ...	189871	10000	2001	5
chris ...	189861	5418	1637	2
cnae- ...	167185	1080	857	9
faber ...	189872	8237	801	7
Fashi ...	189908	70000	785	10
KDDCu ...	75105	50000	231	2
mfeat ...	167152	2000	217	10
volke ...	168793	58310	181	10
APSFa ...	189860	76000	171	2
jasmi ...	189862	2984	145	2
nomao ...	126026	34465	119	2
alber ...	189866	425240	79	2
dioni ...	189873	416188	61	355
janni ...	168792	83733	55	4
cover ...	75193	581012	55	7
MiniB ...	168798	130064	51	2
conne ...	167201	67557	43	3
kr-vs ...	167149	3196	37	2
higgs ...	167200	98050	29	2
helen ...	189874	65196	28	100
kc1	167181	2109	22	2
numer ...	167083	96320	22	2
credi ...	167161	1000	21	2
sylvi ...	189865	5124	21	2
segme ...	189906	2310	20	7
vehic ...	167168	846	19	4
bank- ...	126029	45211	17	2
Austr ...	167104	690	15	2
adult ...	126025	48842	15	2
Amazo ...	75097	32769	10	2
shutt ...	168795	58000	10	7
airli ...	75127	539383	8	2
car	189905	1728	7	4
jungl ...	189909	44819	7	3
phone ...	167190	5404	6	2
blood ...	167184	748	5	2