Universiteit Leiden

# Master Computer Science

Overfitting in Combined Algorithm Selection and Hyperparameter Optimization

| | |
|---|---|
| Name: | Sietse Schröder |
| Student ID: | s3374351 |
| Date: | 20/08/2024 |
| Specialization: | Bioinformatics |
| 1st supervisor: | Jan van Rijn |
| 2nd supervisor: | Mitra Baratchi |

# Abstract

Hyperparameter optimization (HPO) is crucial in designing effective machine learning algorithms that generalize well over unseen data. The fitness of a hyperparameter configuration is typically assessed through holdout or cross-validation evaluation. However, the evaluation of numerous hyperparameter configurations introduces the risk of identifying a configuration that is overly tailored to the validation procedure, resulting in poor generalization performance. This effect can be magnified by advanced HPO methods, such as Bayesian optimization, that actively incorporate evaluations in their mechanism to suggest new hyperparameter configurations, possibly leading to what is referred to as adaptive overfitting. We provide one of the first large-scale investigations of overfitting to the validation procedure in the context of the Combined Algorithm Selection and Hyperparameter optimization (CASH) problem, on 48 classification and 16 regression tasks, using random search, Bayesian optimization, and BO-THO, an algorithm we introduce that reduces overfitting to the validation procedure in HPO by 12% but comes at the expense of 0.3% generalization performance of the ultimately selected hyperparameter configuration averaged over all classification tasks. We show the significant presence of adaptive overfitting in Bayesian optimization on 41 of the 48 classification tasks, where the number of classes was not found to alleviate this effect, contrary to theoretical indications in earlier work. Finally, we show the surprising resilience of most regression tasks to this type of overfitting. Complementary, we assess the aptitude of 10-fold cross-validation as a popular benchmark evaluation procedure, showing significant adaptive overfitting leading to overinflated results when compared to evaluations on a genuinely unseen test set.

# Contents

# Chapter 1

# Introduction

Designing a supervised machine learning workflow involves selecting various components such as data preprocessors, feature selection methods, and algorithms. Each component comes with its own set of hyperparameters, leading to an immensely large space of possible workflow configurations. The problem of simultaneously optimizing the selection of components and their respective hyperparameters is known as the combined algorithm selection and hyperparameter optimization problem (CASH) [61]. In this context, the workflow can be represented using a single set of hyperparameters, with the choice of components being captured in hyperparameters itself. This set of hyperparameters describing a workflow is referred to as a hyperparameter configuration. Hyperparameter optimization (HPO) seeks to automate the selection of these configurations to optimize the workflow's generalization to the true data distribution. This automation not only reduces manual effort but also empowers non-experts to create complex machine learning workflows, making the process more accessible and reproducible [22].

Numerous HPO methodologies have been developed [5], ranging from basic approaches such as random search (RS), which involves the random selection and evaluation of hyperparameter configurations, to advanced techniques such as Bayesian optimization (BO). Bayesian optimization, employed in widely-used HPO tools such as auto-sklearn [23], is broadly regarded as a leading method for HPO [48, 61, 62].

A central component of any HPO system is the evaluation of its suggested configurations on the learning problem. Evaluating a hyperparameter configuration usually involves using data partitions, such as k-fold cross-validation or a holdout (validation) set, to estimate the model's performance on unseen data [5]. These partitions are generally predetermined to ensure fair comparison when assessing multiple configurations. However, in real-world learning problems, data scarcity often limits their size.

Evaluation on a limited validation set or cross-validation procedure can be unstable [51, 65] and may provide biased performance measures for some hyperparameter configurations compared to the performance on the true distribution. A small validation set can result in high variance among model evaluations, causing the maximum difference between the performance estimation of a configuration on the validation set and on the true distribution to increase [33]. When performance estimations using the validation procedure provide an overly optimistic measure for a configuration compared to the performance on the true distribution, we refer to this phenomenon as meta-overfitting. Note that this type of overfitting differs from the traditional overfitting in machine learning, where a model becomes overly tailored to the training data, leading to poor generalization on unseen data.

When many configurations are evaluated, a fraction can be expected to exhibit a significant amount of overfitting. This effect is visualized in Figure 1.1, displaying the difference in validation and true distribution performance of 25 000 configurations produced by random search on a binary classification problem. Although the majority of the validation estimations of configurations are reflective of the performances on the true distribution (around the $x = 0$ line), as HPO seeks to maximize validation performance and thus selects the hyperparameter configuration with the highest validation estimation, a highly overfitted configuration has a relatively high probability of ultimately being selected by the HPO mechanism. This effect has been observed empirically in model selection [8] and in Bayesian optimization for tuning the hyperparameters support vector machines [42], but overall remains a relatively under-explored and open problem in HPO [22]. Theoretically, any HPO approach utilizing multiple hyperparameter configuration evaluations is susceptible to this type of overfitting. The main metric associated with this effect is the *meta-overfitting error* (MOE) [1], which denotes the difference between the empirical performance measured on a validation set and the expected performance on the true data distribution, often approximated using a large, independent, and unobserved test set.

An additional source of overfitting is theorized to arise from not only repeatedly assessing the validation loss and selecting the configurations with the best validation performance but also actively exploiting the information gained by previous evaluations to propose solutions specifically tailored to the holdout set. The phenomenon where a fixed test set is reused while incorporating previous evaluations in the suggestion of new suggestions was first addressed by Dwork et al. [13], and is known as *adaptive data analysis*. When this effect causes the reported results to be overinflated, it is referred to as *adaptive overfitting*. There are large similarities between the reuse of a fixed test set in adaptive data analysis and the sequential use of the validation procedure in modern HPO. Many approaches, such as Bayesian optimization and evolutionary strategies [25], incorporate past evaluations to propose improved hyperparameter configurations. Understanding to what extent modern HPO methods exhibit (adaptive) overfitting and determining whether mitigation strategies identified by adaptive data

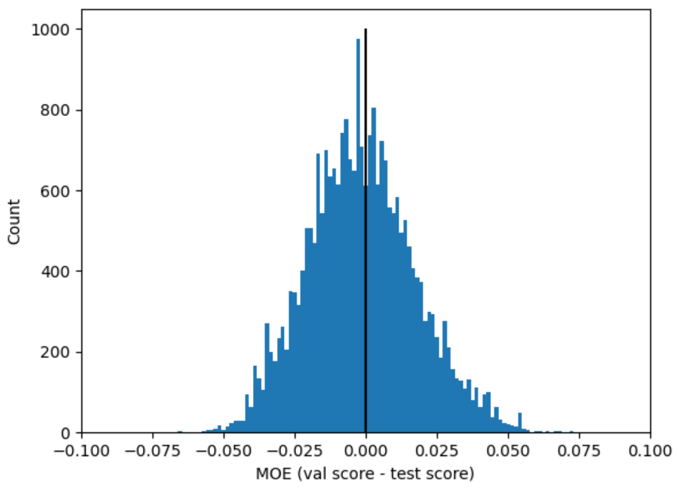


Figure 1.1: Histogram of the MOE of 25 000 evaluated configurations produced by random search, using a holdout set of 500 instances and a test set of approximately 40 000 instances on a binary classification task. Configurations on the right side are more overfitted and have on average higher validation performance, which makes them more likely to be selected by an HPO procedure.

analysis can be applied to enhance the generalizability of HPO solutions are important aspects of addressing overfitting in HPO.

This work will empirically examine overfitting resulting from HPO in a CASH problem across a variety of classification and regression tasks, answering to what extent overfitting occurs in HPO approaches for the CASH problem and whether mitigation methods such as 10-fold cross-validation or thresholdout [12] can alleviate this effect. Specifically, the following contributions are presented:

**1. Systematic analysis of overfitting in modern HPO:** We investigate overfitting in a modern HPO setting, utilizing a large search space of preprocessors, feature selectors, algorithms, and their hyperparameters. Our analysis will address inherent overfitting from selecting the best hyperparameter configuration observed so far and adaptive overfitting resulting from sampling specifically around good-performing configurations.

**2. Demonstrating the practical impact of overfitting:** We demonstrate that adaptive overfitting can be problematic in practical AutoML scenarios and benchmarking practices, particularly when data is scarce. We investigate the impact of overfitting with different validation sizes, to evaluate the extent to which overfitting can occur in real-world problems.

**3. Evaluation of mitigation methods:** We assess various mitigation methods to address overfitting in HPO, including 10-fold internal cross-validation. Furthermore, we introduce BO-THO, an algorithm based on Bayesian optimization while integrating thresholdout [12], an algorithm designed in adaptive data analysis to prevent adaptively overfitting a fixed holdout set, that to the best of our knowledge has not been studied in combination with HPO before.

Through these contributions, this thesis aims to enhance the understanding of adaptive overfitting in the context of HPO and investigate the effect of practical strategies to improve the generalizability of HPO outcomes. The remainder of this thesis will be organized as follows. Chapter 2 will formalize overfitting in the HPO context. Chapter 3 will provide a thorough overview of the fields of adaptive data analysis and (Bayesian) hyperparameter optimization. Chapter 4 will discuss related work investigating specifically the analysis and mitigation of overfitting in HPO. Chapter 5 will describe the methodology, specifically the algorithms used. Chapter 6 outlines the experiments designed, the results of which will be discussed in Chapter 7. The conclusions and directions for future work are provided in Chapter 8.

# Chapter 2

# Problem Statement

This chapter introduces the concepts and definitions used in this work. Mainly, we formalize the objective of HPO and build a framework towards studying overfitting in this context, resulting in $MOE_{avg}$ and $MOE_{sel}$, the two main metrics associated with the results presented in Chapter 7, which are presented in Equation 2.4 and Equation 2.5 respectively.

A supervised machine learning problem is approached by designing a *learning algorithm* $A$ that defines how to leverage the available data $\mathcal{D}_{train}$ to obtain a *model* that achieves maximum performance when presented with previously unobserved data from the same problem domain, generally referred to as the test set $\mathcal{D}_{test}$. Importantly, $\mathcal{D}_{test}$ should only be observed once to estimate the final generalization performance. We formalize the generalization loss of model $A$ trained on $\mathcal{D}_{train}$ and evaluated on $\mathcal{D}_{test}$ as $\mathcal{L}(A, \mathcal{D}_{train}, \mathcal{D}_{test})$, minimizing which is equivalent to maximizing the generalization performance.

## 2.1 CASH

A wide range of learning algorithms exist, and the majority of these algorithms are highly configurable with *hyperparameters* specific to them. We indicate the space of available learning algorithms as $\mathcal{A}$ and the hyperparameter space used to configure the algorithms as $\Lambda$, where a set of hyperparameters $\lambda \in \Lambda$ configures a learning algorithm $A \in \mathcal{A}$, resulting in $A_\lambda$. Simultaneously selecting an algorithm and optimizing its hyperparameters is known as the CASH problem [61].

Now, we want to carefully select and configure $A_\lambda$ to minimize $\mathcal{L}(A_\lambda, \mathcal{D}_{train}, \mathcal{D}_{test})$. However, since we are only allowed to access $\mathcal{D}_{test}$ to determine the performance of the final algorithm and hyperparameter configuration, an independent validation procedure is required to assess the performance of individual configurations in the search process. In this work, we employ two primary methods to evaluate the performance of a configuration, the holdout method and k-fold cross-validation. With the holdout method, a subset of $\mathcal{D}_{train}$ is designated as the validation set $\mathcal{D}_{val}$, which is inaccessible to the algorithm during training, providing an estimate of the generalization performance of the algorithm. Alternatively, k-fold cross-validation divides $\mathcal{D}_{train}$ into $k$ distinct subsets, where each subset serves as the validation set once, while the remaining $k-1$ subsets are used for training.

When using the holdout method as the validation procedure, the CASH problem can be formalized as follows:

$$A^*, \lambda^* \in \underset{A \in \mathcal{A}, \lambda \in \Lambda}{\arg\min}\, \mathcal{L}(A_\lambda, \mathcal{D}_{train}, \mathcal{D}_{val}) \tag{2.1}$$

Alternatively, the k-fold cross-validation procedure can be employed within the CASH formulation. In this context, $\mathcal{D}^m_{train}$ refers to data partition in fold $m$ with size $\frac{k-1}{k}$ used for training, while $\mathcal{D}^m_{val}$ denotes the validation set of that fold.

$$A^*, \lambda^* \in \underset{A \in \mathcal{A}, \lambda \in \Lambda}{\arg\min}\, \frac{1}{k} \sum_{m=1}^{k} \mathcal{L}(A_\lambda, \mathcal{D}^m_{train}, \mathcal{D}^m_{val}) \tag{2.2}$$

As elaborated on in Chapter 1, the algorithm selection can itself be captured in hyperparameters. Therefore, the CASH problem can be expressed in a single set of hyperparameters, allowing HPO methods to approach tackling a CASH problem.

## 2.2 Meta-overfitting

This work mainly investigates the disparity between the performance of a configuration measured with the internal validation procedure of the HPO mechanism and the generalization performance on the truly unseen test set $\mathcal{D}_{test}$. This difference is referred to as the *meta-overfitting error* (MOE) and is defined as:

$$MOE(A_\lambda, \mathcal{D}_{train}, \mathcal{D}_{val}, \mathcal{D}_{test}) := \mathcal{L}(A_\lambda, \mathcal{D}_{train}, \mathcal{D}_{test}) - \mathcal{L}(A_\lambda, \mathcal{D}_{train}, \mathcal{D}_{val}) \tag{2.3}$$

In the context of this work, we define an *iteration* as a sequential step of the

HPO procedure in which one configuration is proposed and evaluated. We define a *repetition* as the execution of one complete HPO procedure. In this framework, a repetition consists therefore of many sequential iterations. We measure the aggregated MOE in an HPO algorithm using two metrics, the average MOE ($MOE_{avg}$) and the selected MOE ($MOE_{sel}$). We store every configuration proposed during every iteration and repetition in $C$, where $C_{i,j}$ is the configuration proposed in iteration $i$ during repetition $j$.

Using this framework, the average MOE for one iteration $i$ is defined as the averaged MOE of all configurations proposed in that iteration over all $n$ repetitions, formalized as:

$$MOE_{avg}(C, \mathcal{D}_{train}, \mathcal{D}_{val}, \mathcal{D}_{test}, i) := \frac{1}{n}\sum_{j=1}^{n} MOE(A_{C_{i,j}}, \mathcal{D}_{train}, \mathcal{D}_{val}, \mathcal{D}_{test}) \tag{2.4}$$

Whereas the selected MOE of iteration $i$ is the best configuration observed so far at iteration $i$ within one repetition. This corresponds to the configuration that would have been selected if the repetition of HPO had been stopped at that iteration. The selected MOE is formalized as:

$$MOE_{sel}(C, \mathcal{D}_{train}, \mathcal{D}_{val}, \mathcal{D}_{test}, i) := \frac{1}{n}\sum_{j=1}^{n} MOE(A_{\lambda^*}, \mathcal{D}_{train}, \mathcal{D}_{val}, \mathcal{D}_{test}) \tag{2.5}$$

Where:

$$\lambda^* \in \operatorname*{arg\,min}_{\lambda \in C_{1\dots i,j}} \mathcal{L}(A_\lambda, \mathcal{D}_{train}, \mathcal{D}_{val}) \tag{2.6}$$

# Chapter 3

# Background

This chapter reviews the essential background for understanding overfitting in hyperparameter optimization. Firstly, it explores the field of adaptive data analysis, which, while not primarily focused on HPO, offers valuable insights into overfitting due to sequential access to validation data. This phenomenon bears similarities to overfitting in HPO and can provide a deeper understanding of the issue. Secondly, the chapter addresses the general problem of HPO and introduces the specific approach adopted in this thesis.

## 3.1 Adaptive data analysis

Assessing the performance of supervised machine learning algorithms requires estimating the generalization of the model on a previously unseen data set, such as a holdout or cross-validation procedure. The error resulting from this procedure is then considered an approximation of the performance of the model on the true underlying distribution. Performing such an assessment requires a fully independent and separate holdout set sampled from the same data distribution. For the results derived from this holdout set to be statistically valid, it can be used only once [13]. However, in practice data is often scarce, and holdout sets are often reused [11, 20, 66]. Practical examples of holdout set reuse include machine learning competitions and popular benchmarks like the MNIST dataset [37]. This reuse can lead to adaptive overfitting, where models become overly tailored to the specific characteristics of the holdout set rather than generalizing to the overall data distribution.

Adapting analyses, modeling choices, or hypotheses to the intermediate results on the holdout set can produce overly optimistic performance or significance measures when compared to the results on the true distribution. This effect is a form of overfitting, sometimes referred to as adaptive overfitting, where models can become excessively tailored to the specific characteristics of the holdout or cross-validation procedure because of modeling choices based on holdout performance, rather than generalizing to the overall distribution [51]. The field of adaptive data analysis, initiated by Dwork et al. [13], attempts to identify and mitigate the corruption of the reliability of the holdout results in the adaptive setting. The main metric associated with this field is the *meta-overfitting error* (MOE)[1], which denotes the difference between the holdout or cross-validation error and the expected error on the true distribution.

In the non-adaptive setting, an experimenter fully designs $t$ models independently without consulting the holdout set, after which all of them are evaluated at once. Standard statistics show that the MOE of any of these models is expected to be in the order of $O(1/\sqrt{N})$, where $N$ is the size of the holdout set. Concentration bounds then show that the maximum MOE observed when evaluating $t$ models is expected to be in the order of $O(\sqrt{log(t)/N})$. However, when an experimenter is allowed to design a model at iteration $t$ after the evaluation of the first $t-1$ models on the holdout set, it has been shown that the MOE of this model can have a lower bound of $\Omega(\sqrt{t/N})$ [13], a very pessimistic view on the validity of holdout results when the number of models designed adaptively is approximating the size of the holdout set.

Several mitigations on these pessimistic bounds have been proposed by subsequent work. It has been pointed out that realistic experimenters do not incorporate all previous results efficiently and do not attempt to adversarially overfit the holdout set [66]. Furthermore, compared to binary classification, multi-class classification problems have been theorized to suffer less from adaptive overfitting with a factor linear in the number of classes [19]. Dwork et al. propose thresholdout [12], an algorithm inspired by differentiable privacy allowing an analyst to evaluate an exponential number of models on the holdout set. Thresholdout is extensively elaborated on in Chapter 5.

Despite theoretical concerns, empirical evidence for adaptive overfitting in the machine learning community is limited. Studies on benchmark s like MNIST, CIFAR-10, and ImageNet have found little evidence of significant adaptive overfitting [53, 54, 64], although it has been pointed out that the replication of datasets required for these studies can suffer from bias itself [16]. Similarly, a meta-analysis of Kaggle competitions found minimal evidence of adaptive overfitting among competitors [56].

## 3.2 Hyperparameter optimization

The selection of hyperparameters is a critical process that significantly influences the performance of a configurable algorithm [63]. Hyperparameters are generally defined by a configuration space, also referred to as a search space. This space denotes which hyperparameters are optimized and what their possible options or ranges of options are [22]. Hyperparameters can vary in type, examples being the learning rate in a neural network (continuous), the number of trees in a random forest (discrete), or the choice of activation function in a neural network (categorical).

HPO generally deploys an inner and outer evaluation loop. The inner loop evaluates suggested sets of hyperparameters to some validation procedure, fine-tuning the HPO algorithm, while the outer evaluates the final selected hyperparameters to assess generalization performance on unseen data, and is only used once [5].

A modern superset of HPO problems is CASH [61]. The CASH objective is to build the best pipeline of algorithm components, such as feature selectors, preprocessors, and learners, each with its own hyperparameters that can be optimized. In the CASH formulation, hyperparameters are often conditional on others. For instance, a hyperparameter might be inactive when a learner is selected that does not deploy that hyperparameter. Approaching learning tasks as a CASH problem has been shown to yield state-of-the-art performance and is now implemented in many popular AutoML packages, such as Auto-WEKA, auto-sklearn, AutoGluon, and TPOT [17, 21, 35, 36]. Defining a learning task as a CASH problem introduces a vast amount of possible combinations of components and hyperparameters, producing an incredibly complex search space. This complexity possibly affects the potential of overfitting in a CASH approach, the effect studied in this work.

# Chapter 4

# Related Work

Estimating the generalization performance of models has been a subject of extensive research in statistics and machine learning for decades. This chapter explores the early work on variable selection (deciding which variables to include in an analysis based on a holdout set) and model selection, as well as the recent advancements aimed at comprehending and mitigating overfitting within the context of HPO.

The realization that machine learning evaluation methods can be biased when many evaluations are performed is not novel. Freedman's paradox [24] shows that when many randomly sampled variables are available and selected based on their correlation with the target, statistically significant regression results predicting the target can be reported, while these variables would have no predictive value over the target in an unseen test set. Similarly, in 1997, it was argued that always selecting the hypothesis with the lowest CV error out of many evaluated hypotheses can lead to biased results and poor generalization performance [47].

Following this work, it was shown that the evaluation of different variable selection algorithms based on cross-validation is unreliable due to overfitting the CV procedure [55]. These effects were empirically demonstrated in an effort to determine the overfitting of manual model selection and hyperparameter tuning [8].

The evidence that evaluating many models and hyperparameters can lead to decreasing generalization performance suggests that specially HPO could be susceptible to overfitting since it algorithmically proposes and evaluates numerous configurations. However, overfitting in HPO is not regularly investigated in literature and remains largely unexplored. This section will summarize the work on overfitting in HPO, and the many uncertainties that still exist and require further research.

Nguyen et al. [48] show that BO can converge to sharp performance peaks, observed on the validation procedure, instead of stable regions, resulting in decreasing generalization, and propose an alternative acquisition function favoring regions in the search space where small variations lead to small changes in performance, over more unstable regions. Their approach shows an improved generalization of the final HPCs found when tuning an SVM on two datasets, both encapsulating around 200 observations, using both the holdout and cross-validation methods. Although their work shows the danger of overfitting in HPO and the possibility of finding viable mitigations, the empirical evaluations are too limited to be representative of a modern HPO problem.

Lévesque [42] provides a more rigorous evaluation of overfitting in BO for HPO, showing that overfitting significantly occurs when tuning an SVM using BO with a Gaussian process surrogate function. His method was evaluated on 118 datasets, using 20 repetitions per dataset, providing a basis for overfitting in BHPO. Interestingly, it was found that resampling the validation split (either the holdout split or the entire cross-validation procedure) after each evaluation increased the stability and overall performance of HPO, suggesting that overfitting was actively decreasing the generalization performance of HPO. The effect of resampling and its theoretical implications for the HPO loss surface were recently further investigated [46], finding strong evidence of the overfit of HPO algorithms, both for RS and BO implementations. Not surprisingly, overfitting occurred more strongly when using the holdout method compared to CV.

Fabris & Freitas [18] provide an analysis of the overfit of the auto-sklearn tool [23]. However, a substantial portion of this work is dedicated to understanding the general concept of overfitting in machine learning, where overly complex models are fitted to noise in the training data, producing poor generalization scores on the validation set. Nonetheless, this work presents a rare evaluation of overfitting in a CASH problem on real-world datasets, showing large differences between the validation and test performances on some datasets, although these effects were not significant across datasets, and based on one repetition of auto-sklearn for each dataset, which limits the results considering the highly stochastic nature of HPO. Additionally, auto-sklearn utilizes ensemble methods, which are known to increase stability and generalization [10].

Finally, Makarova et al. [43] introduce an intuitive termination (early stopping) criterion for BHPO that can mitigate overfitting by stopping the process when further improvements are unlikely. This method relies on estimating the simple regret bound [44] and uses cross-validation variance to refine termination decisions. Empirical results demonstrate its robustness and effectiveness in maintaining test performance while reducing overfitting.

## 4.1 Summary

Although various methods have been proposed to improve Bayesian hyperparameter optimization by mitigating significant overfitting, the underlying causes and conditions leading to this issue remain largely unexplored, particularly in the context of large-scale modern CASH approaches. To our knowledge, the only investigation into overfitting within a CASH framework is the study by Fabris & Freitas [18], which is specifically aimed at analyzing the auto-sklearn tool [23], and is constrained to 18 classification problems with a single repetition for each dataset. In contrast, this work addresses 48 classification problems and 16 regression tasks, while repeating each experiment 100 times, while isolating Bayesian optimization as an HPO method, where auto-sklearn introduces additional techniques such as ensembling.

# Chapter 5

# Methodology

In this chapter, we present the methodology underlying the experiments in this work, which are then described in Chapter 6. Specifically, we explicate the details of the three HPO algorithms utilized to approach the CASH problem on a wide range of datasets. In Section 5.1 we elaborate on random search, Section 5.2 describes Bayesian optimization applied to HPO, and Section 5.3 explains and formalizes thresholdout [12], the algorithm originating from adaptive data analysis which was introduced in Chapter 3. Subsequently, we introduce Bayesian optimization with thresholdout (BO-THO), an integration of the thresholdout algorithm within a Bayesian optimization approach for HPO. Finally, we elaborate in Section 5.5 on the different validation procedures used throughout this work.

## 5.1 Random search

Random search [3] is included as a baseline, specifically to investigate non-adaptive overfitting, since by definition random search is unaware of previously evaluated results in future configuration suggestions. Random search independently samples and evaluates a configuration each iteration.

## 5.2 Bayesian optimization

Bayesian optimization is an efficient approach for black-box optimization problems, especially when the evaluation of the objective function is computationally expensive. This technique has become increasingly popular for hyperparameter optimization (HPO) [5, 29], where training and evaluating the learner can require substantial computational resources. Bayesian optimization fits a surrogate model to the previously evaluated configurations and estimates the expected performance and uncertainty of new configurations. This approach balances the trade-off between exploring regions in the search space with high uncertainty, which have likely not been evaluated yet, and exploiting regions that are known to perform well [2]. After proposing a new configuration, the validation procedure is employed to provide an estimation of the performance of the configuration, after which the surrogate model is refitted on the updated archive of previous function evaluations. At the end of the optimization, typically the configuration with the highest performance estimation is selected, although different selection procedures have been proposed [31, 50, 57].

### 5.2.1 Surrogate model

The surrogate model approximates the true objective relationship between hyperparameter configurations and their performance estimation using the validation procedure and is iteratively refined as more evaluations of configurations are added to the Bayesian optimization archive. For each proposed configuration, the surrogate model predicts its performance, along with an uncertainty estimate regarding its prediction. Several surrogate models have been proposed, with Gaussian processes [52] and random forests [29] being popular choices.

A Gaussian process inherently models uncertainty, making it particularly suitable for many applications of Bayesian optimization. However, Gaussian processes have been shown to work well primarily with continuous input spaces [39]. Since many hyperparameters in HPO search spaces are not continuous [26]—but rather discrete (e.g., number of trees in a random forest), categorical (e.g., activation function in a neural network), or conditional (e.g., hyperparameters that are only active if the selected model deploys them)—random forests often outperform Gaussian processes in complex search spaces [15, 39]. Additionally, random forests are much cheaper to evaluate, as Gaussian processes scale cubically with the number of evaluations in the archive [30]. However, the uncertainty measure in random forests is often computed using the standard deviation of the predictions of individual trees, which is not a direct uncertainty measure and might behave sub-optimally [5].

### 5.2.2 Acquisition function

The acquisition function guides the search process in Bayesian optimization by suggesting the next point to evaluate based on the surrogate model's predictions. It quantifies the trade-off between exploration (sampling in regions of high uncertainty) and exploitation (sampling in regions of expected high performance). Many acquisition functions have been proposed, of which one of the most popular is expected improvement [32, 59], which is shown in Equation 5.1. Equation 5.1 is largely adapted from [59].

$$\mathrm{EI}(x; \{x_n, y_n\}, \lambda) = \sigma(x; \{x_n, y_n\}, \lambda) \left(\gamma(x)\Phi(\gamma(x)) + \mathcal{N}(\gamma(x); 0, 1)\right) \tag{5.1}$$

where:

- $\sigma(x; \{x_n, y_n\}, \lambda)$ is the predictive standard deviation of the Gaussian process at point $x$.

- $\gamma(x)$ is the standardized improvement, defined as:

$$\gamma(x) = \frac{f(x_{\text{best}}) - \mu(x; \{x_n, y_n\}, \lambda)}{\sigma(x; \{x_n, y_n\}, \lambda)} \tag{5.2}$$

- $\mu(x; \{x_n, y_n\}, \lambda)$ is the predictive mean of the Gaussian process at point $x$.
- $f(x_{\text{best}})$ is the best observed value so far.
- $\Phi(\cdot)$ is the cumulative distribution function of the standard normal distribution.
- $\mathcal{N}(\cdot; 0, 1)$ is the probability density function of the standard normal distribution.

### 5.2.3 Sequential model-based algorithm configuration

Sequential model-based algorithm configuration (SMAC) [29, 40] is an implementation of Bayesian optimization for HPO using a random forest surrogate model. SMAC is employed in many popular HPO tools, such as auto-sklearn [23] and AutoWEKA [35, 61].

The functionality of Bayesian optimization in this work is implemented using SMAC3 [40]. SMAC3 is a recently implemented Python package designed for various hyperparameter optimization tasks, known for its sample efficiency and robustness. In this study, SMAC3 was initialized by combining a random forest surrogate model and an expected improvement acquisition function, where the use of a random forest as the surrogate model helps in handling complex, high-dimensional configuration spaces, as are typically encountered in a CASH problem. In every iteration, the computed validation loss, either on the holdout set or using the cross-validation procedure, is used internally to update the surrogate model.

## 5.3 Thresholdout

Thresholdout [12] was designed to mitigate overfitting when reusing a holdout set adaptively, to allow the evaluation of multiple hypotheses without compromising the statistical validity of the results. In this context, a hypothesis refers to any operation that requires querying an independent set of data for evaluation and can be produced algorithmically or by a human analyst. Extrapolating to hyperparameter optimization, a hypothesis would refer to a configuration to be evaluated.

Thresholdout provides a performance estimate on the holdout set, introducing noise to ensure the subsequent, adaptively chosen hypotheses do not overfit the holdout set. When the empirical difference between the train loss and holdout loss is below some threshold, thresholdout provides the unchanged train loss to the observer. Note that this can be done infinitely without risking adaptively overfitting the holdout set since usage of the training data is unrestricted. When the solution is overfitted to the train set to an extent that exceeds the threshold, a generalization performance estimate is required to evaluate the hypothesis. In this case, thresholdout adds noise to the holdout performance, the result of which is then accessible to the observer. When thresholdout is utilized to preserve the statistical validity of the holdout results, an overfitting budget is decremented when the threshold is exceeded, refusing further holdout evaluations when exhausted. In the context of HPO however, we are interested in reducing or preventing adaptive overfitting and improving generalization rather than ensuring our results are statistically valid, which is why the budget was omitted from thresholdout in this work.

The noise within the thresholdout algorithm is sampled from the Laplacian distribution (a symmetrical exponential distribution) in the full version of the author's work [12], motivated by its successful use within the field of differentiable privacy [14]. However, in a short version of the work where thresholdout is presented, the authors propose sampling from the Gaussian distribution [11]. In this work, the Laplacian distribution is used for the noise injection within thresholdout.

Algorithm 1 contains the pseudo-code for a thresholdout in the general context. The validation set $\mathcal{D}_{val}$ represents any holdout set that is adaptively reused. Note that this differs from our requirements for a test set, which can only be accessed once. Algorithm 1 is configurable with the threshold $T$ and noise rate $\rho$, introducing a trade-off between the amount of information released about the holdout set and the resistance against adaptive overfitting, where an unsuitably low noise rate allows adaptive overfitting, while a very high noise rate would give too little information about the generalization of the hypothesis. The initialization of the noise rate and threshold in this work is further elaborated on in Section 5.4.

**Algorithm 1:** Thresholdout

```
Input: Training set $\mathcal{D}_{train}$, holdout set $\mathcal{D}_{val}$, threshold $T$, noise rate $\rho$,
       loss $\mathcal{L}$
1  Sample $\gamma \sim \text{Laplace}(2 \cdot \rho)$;
2  $\hat{T} \leftarrow T + \gamma$;
3  foreach hypothesis $\phi$ do
4      Sample $\eta \sim \text{Laplace}(4 \cdot \rho)$;
5      if $|\mathcal{L}_{\mathcal{D}_{val}}(\phi) - \mathcal{L}_{\mathcal{D}_{train}}(\phi)| > \hat{T} + \eta$ then
6          Sample $\xi \sim \text{Laplace}(\rho)$;
7          Sample $\gamma \sim \text{Laplace}(2 \cdot \rho)$;
8          $\hat{T} \leftarrow T + \gamma$;
9          Output $\mathcal{L}_{\mathcal{D}_{val}}(\phi) + \xi$;
10     else
11         Output $\mathcal{L}_{\mathcal{D}_{train}}(\phi)$;
```

## 5.4 Bayesian optimization with thresholdout

To investigate the potentially mitigating effect of methods proposed in adaptive data analysis on adaptive overfitting, thresholdout, as introduced by Dwork et al. [12] and described in Section 5.3 is applied to the internal Bayesian optimization procedure, henceforth referred to as BO-THO.

BO-THO works similarly to the general Bayesian optimization for HPO but internally keeps two archives storing configurations and their evaluations, $\mathcal{H}_{true}$ and $\mathcal{H}_{tho}$. $\mathcal{H}_{true}$ stores the actual evaluations of configurations on the validation procedure, and is used to select the final configuration after all iterations have been completed. In contrast, $\mathcal{H}_{tho}$ is used to fit the surrogate function and is updated each iteration with the reported evaluation by thresholdout. Using this algorithmic design, in theory, BO-THO selects the configuration with the smallest validation loss similar to regular Bayesian optimization, but is prevented from sampling actively around highly overfitted solutions by thresholdout.

BO-THO is formalized in Algorithm 2. In lines 1 & 2 the Laplacian noise used within thresholdout is sampled. Lines 3 & 4 initialize the two archives explained earlier in this chapter. Line 5 defines the number of iterations performed in BO-THO, which is an experimental setting. Then, in line 6 the surrogate model of Bayesian optimization is fitted on all previous configurations and their evaluations produced by the thresholdout mechanism, stored in $\mathcal{H}_{tho_{t-1}}$. In this formulation, $\mathcal{H}_{tho_{t-1}}$ is the set of every configuration and its performance estimation from thresholdout up until iteration $t$. The predictive mean $\mu$ and predictive standard deviation $\sigma$ from the surrogate model of the Bayesian optimization procedure, the acquisition function described in Section 5.2.2 is built in line 7. In line 8, the configuration with the highest expected improvement is selected and proposed, which is used in line 9 to initialize a learning algorithm. Line 10 uses the available training data to train this algorithm. Lines 11 & 12 sample Laplacian noise for the internal use of thresholdout and compute whether the difference in train and validation loss exceeds the threshold. If this is the case, lines 13-16 are aimed at sampling the required noise and adding the configuration and its noisy evaluation to the archive. If the difference did not exceed the threshold, in line 18 the loss computed on the training set is added to the archive. In any case, line 19 adds the empirical validation loss to the true archive, which is not accessible to the BO-THO algorithm before all iterations are completed. Finally, the algorithm outputs the configuration in the true archive with the highest validation performance.

BO-THO is configurable with the noise rate hyperparameter, which controls for the amount of information provided about the validation set when overfitting is found. In line with the experiments performed by Dwork et al. [12], the noise rate was divided by the square root of the validation size, since a larger validation set is theoretically less prone to overfitting and therefore allows for less noisy evaluations. The threshold was set to double the noise rate, also divided by the square root of the validation set size. We attempted several noise rates between 0.00625 and 1.0, where the range between 0.25 and 0.5 visually seemed to balance information and overfitting best. For all future experiments, we selected a noise rate of 0.25.

**Algorithm 2:** BO-THO

```
Input: Search space $\Lambda$, Training set $\mathcal{D}_{train}$, holdout set $\mathcal{D}_{val}$, threshold
       $T$, noise rate $\rho$, loss $\mathcal{L}$, iterations $N$
1  Sample $\gamma \sim$ Laplace($2 \cdot \rho$);
2  $\hat{T} \leftarrow T + \gamma$;
3  $\mathcal{H}_{true_0} \leftarrow \emptyset$ // Initialize true archive ;
4  $\mathcal{H}_{tho_0} \leftarrow \emptyset$ // Initialize thresholdout archive ;
5  for $t = 0$ to $N$ do
6      Fit surrogate model $\mu(x)$, $\sigma(x)$ on $\mathcal{H}_{tho_{t-1}}$;
7      Define acquisition function $a(x)$ from $\mu(x)$ and $\sigma(x)$;
8      Propose new configuration with $\lambda_t \leftarrow \arg\max_{\lambda \in \Lambda} a(\lambda)$;
9      Initialize new algorithm $A_{\lambda_t}$ with configuration $\lambda_t$;
10     Fit model $A_{\lambda_t}$ on $\mathcal{D}_{train}$;
11     Sample $\eta \sim$ Laplace($4 \cdot \rho$);
12     if $|\mathcal{L}(A_{\lambda_t}, \mathcal{D}_{train}, \mathcal{D}_{val}) - \mathcal{L}(A_{\lambda_t}, \mathcal{D}_{train}, \mathcal{D}_{train})| > \hat{T} + \eta$ then
13         Sample $\xi \sim$ Laplace($\rho$);
14         Sample $\gamma \sim$ Laplace($2 \cdot \rho$);
15         $\hat{T} \leftarrow T + \gamma$;
16         $\mathcal{H}_{tho_t} \leftarrow \mathcal{H}_{tho_{t-1}} \cup (\lambda_t, \mathcal{L}(A_{\lambda_t}, \mathcal{D}_{train}, \mathcal{D}_{val}) + \xi)$;
17     else
18         $\mathcal{H}_{tho_t} \leftarrow \mathcal{H}_{tho_{t-1}} \cup (\lambda_t, \mathcal{L}(A_{\lambda_t}, \mathcal{D}_{train}, \mathcal{D}_{train}))$;
19     $\mathcal{H}_{true_t} \leftarrow \mathcal{H}_{true_{t-1}} \cup (\lambda_t, \mathcal{L}(A_{\lambda_t}, \mathcal{D}_{train}, \mathcal{D}_{val}))$;
Output: $\lambda^* \in \arg\min_{\lambda \in \mathcal{H}_{true}} \mathcal{L}(A_\lambda, \mathcal{D}_{train}, \mathcal{D}_{val})$
```

## 5.5 Resampling and evaluation

As mentioned in earlier chapters, the evaluation of a machine learning algorithm aims to provide a reliable approximation of the generalization performance of that algorithm on unseen data. Although many methods have been proposed, the two most popular remain the holdout and cross-validation methods [4]. The holdout method sets aside a portion of data for validation purposes before the model selection procedure, which is used as validation data in every evaluation during the selection process. Alternatively, the k-fold cross-validation procedure repeats the training and validation procedure k times independently, where each of these uses a fixed, different fold with size $1/k$ of the available data as validation. The final evaluation of that pipeline is then averaged over the $k$ validation folds. While the holdout method is simple and cheap to evaluate, it is often unstable and provides unreliable estimations of generalization performance, as mentioned in Chapter 4. The holdout method mitigates this effect partially by leveraging all data as validation data throughout the procedure, at the cost of training the pipeline $k$ times instead of once for the holdout method. This can be prohibitive in many practical applications of HPO, especially when $k$ is large.

Our experimental procedure leverages both the holdout and cross-validation methods. Unless explicitly mentioned otherwise, in the case of the holdout method, 1/3 of the available data is used for validation and 2/3 for training purposes, mirroring the default setting of auto-sklearn [21]. When cross-validation is applied, we use 10 fixed folds, in line with the standard evaluation procedure for popular AutoML benchmarks [27, 28].

The reported metrics in this work are accuracy for classification tasks, and root mean squared error (RMSE) for regression tasks.

# Chapter 6

# Experimental Setup

In this chapter, we present the range of real-world tasks and experiments used to assess the effects of overfitting in a modern CASH context. To broaden the impact of our conclusions, binary classification, multi-class classification, and regression were considered extensively. Since the results of HPO can differ massively across separate repetitions of the algorithms, the results of every experiment are averaged over 100 repetitions. The delivered code implementing the experiments can be examined in this repository.

## 6.1 Datasets

A representative range of datasets associated with popular benchmarks in HPO was selected for this work. In total, for both regression and classification tasks, 64 datasets were included, resulting in 29 binary classification tasks, 19 multi-class classification tasks, and 16 regression tasks.

### 6.1.1 Classification

For the classification datasets, the OpenML-CC18 Curated Classification benchmark [6] was utilized, including every dataset with at most 50 features. High-dimensional datasets were thus excluded due to limitations in computational resources. This resulted in 29 datasets for binary classification and 19 datasets for multi-class classification. The division of the number of classes in the resulting 48 datasets is visualized in Figure 6.1. A complete overview of the included datasets in this work and some of their characteristics can be examined in Appendix B.

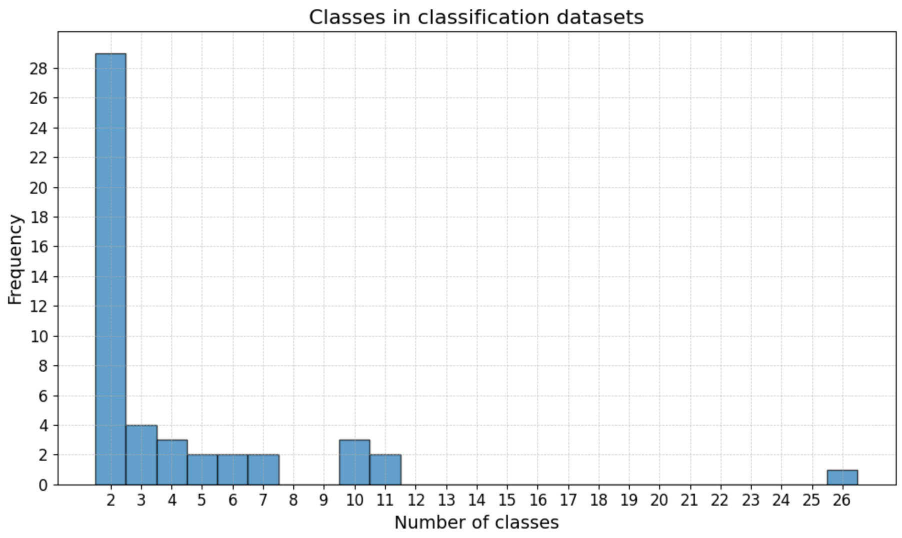


Figure 6.1: Number of classes in classification tasks used for experiments.

### 6.1.2 Regression

The regression tasks from the AutoML benchmark were included [27]. Similarly to the selection of the classification tasks, datasets with more than 50 features were excluded, resulting in 16 datasets. Appendix B.2 provides a complete overview of these datasets. Note that two datasets were omitted due to compatibility issues.

## 6.2 Search space

The search spaces utilized throughout the experiments described in this section follow the CASH formulation, as explained in Section 3.2. Two search spaces were defined, for regression and classification, and were designed to resemble each other as closely as possible, to allow for fair comparison. Both search spaces include the same preprocessing components with roughly the same associated hyperparameters. If appropriate, for each classifier the corresponding regressor was included in the search space. For example, a decision tree classifier and a decision tree regressor are included in respectively the classification and regression search spaces.

Both search spaces encase six top-level categorical hyperparameters corresponding to machine learning pipeline components, which are displayed in Table 6.1 in order of appearance in the pipeline. Only the base model and imputer are always included. For the other components, one of the categories indicates the absence of this preprocessor. Accumulating the total pipeline possibilities, not regarding hyperparameters of individual components, results in 4 896 possible classification pipelines and 4 320 regression pipelines, highlighting the complexity of the CASH problem. For multiclass classification, two base classifiers (Lasso and ElasticNet) were excluded due to compatibility issues with this type of problem, resulting in 4 320 possible pipelines.

As expected, each possible component of the pipeline, such as a specific base classifier, is controlled by one or several hyperparameters as well. An extensive overview of hyperparameters and their associated ranges used in this work is given in Appendix A. To increase compatibility across pipelines and a wide range of datasets, several combinations of hyperparameter choices were forbidden, such as the absence of a scaler in combination with a multi-layered perceptron as the base model, which often raises an error.

All pipeline components are implemented using the Python scikit-learn library [49]. The implementation of the configuration space relies heavily on the Python ConfigSpace library [41]. For an overview of all software packages and versions used, the reader is referred to Appendix C.

| **Hyperparameter** | **# choices** |
|---|---|
| encoder | 2 |
| missing values imputer | 3 |
| feature selector | 4 |
| scaler | 8 |
| dimensionality reducer | 3 |
| base classifier | 17 |
| base regressor | 15 |

Table 6.1: High-level functional pipeline components and the number of possible values.

## 6.3 Experiments

The empirical contribution of this work addresses three lines of experimentation. A large-scale experiment (Section 6.3.1) using the holdout method as the validation procedure is performed over 64 datasets, including classification and regression problems. Furthermore, different validation set sizes are compared to broaden the applicability of the results over datasets of varying sizes (Section 6.3.2). Finally, 10-fold cross-validation is investigated as a mitigation method of overfitting (Section 6.3.3). The results of these experiments are respectively discussed in Section 7.1, Section 7.2, and Section 7.3 respectively.

### 6.3.1 Large-scale holdout

The large-scale holdout experiment is conducted on a comprehensive set of 64 datasets, encompassing both classification and regression tasks. This experiment aims to investigate the following high-level research question:

*To what extent does overfitting occur in HPO approaches for the CASH problem when utilizing the holdout method for validation?*

Additionally, the study addresses the following subquestions:

- *Does Bayesian optimization exhibit adaptive overfitting?*
- *Can thresholdout mitigate adaptive overfitting when integrated within Bayesian optimization?*
- *Does overfitting occur less in multiclass classification and regression tasks compared to binary classification?*

Although the modeling decision on how to divide the available data into train, validation, and test sets is not trivial and often to be determined by the modeler, we believe a 2/1 ratio between $\mathcal{D}_{train}$ and $\mathcal{D}_{val}$ is representative for modern machine learning tasks, which is also the default setting in auto-sklearn [21]. Furthermore, our experimental setup requires a test set of sufficient size. Therefore, in the large-scale holdout experiment, we utilize the following sampling procedure for dividing the available data in ($\mathcal{D}_{train}$, $\mathcal{D}_{val}$, and $\mathcal{D}_{test}$):

- Datasets larger than 3 000 instances are split in (1 000, 500, remainder).
- Datasets smaller than 3 000 instances are split using (40%, 20%, 40%).

Arguably, the selection of effectively 1 500 instances per dataset can be considered relatively small. However, the median dataset size of the OpenML-CC18 [6] is approximately 1 800, indicating our experimental setup is a realistic representation of real-world tasks.

Note that due to computational limitations, the maximum number of test instances was limited at 100 000. In the case of datasets with more than 101 500 instances, the remainder was discarded. Every repetition of every experiment randomly shuffled the entire dataset before sampling these splits, to ensure validity of the averaged results.

### 6.3.2 Holdout with varying validation sizes

To investigate the effect of varying validation set sizes on adaptive overfitting, an experiment was conducted using four datasets for binary classification from the OpenML-CC18 benchmark [6]. These datasets were chosen for their large instance counts, allowing for validation sets up to 10 000 instances, while still providing a sufficiently large test set to reliably approximate the performance on the true distribution. This experiment aims to answer the following research question:

*How does the size of the validation set influence the extent of overfitting in HPO approaches for the CASH problem, and does this effect differ between random search, Bayesian optimization, and BO-THO?*

Answering this question provides insight regarding overfitting in HPO approaches to CASH for datasets of varying sizes. In this experiment, every repetition involved re-sampling the test set to ensure 11 000 instances were available for training and validation. The training set size was fixed at 1 000 instances, while validation set sizes were varied, with 100, 200, 500, 1 000, 2 000, 5 000, and 10 000 instances being used. Any remaining instances after sampling were discarded for that run. Each repetition consisted of seven runs for random search, Bayesian optimization, and BO-THO algorithms, using a fixed test set and resampled training and validation sets of the specified sizes.

### 6.3.3 10CV

Due to the tenfold increase in the number of times a configured pipeline is fitted and evaluated, the 10CV experiments were restricted to a selection of 4 datasets, which can be examined in Appendix B.3. This experimental setup aims to answer the following research question:

*Does 10-fold cross-validation mitigate overfitting in HPO approaches for the CASH problem compared to the holdout method?*

Answering this question will also explicitly show insight into the suitability of 10CV as a benchmark evaluation metric, showing the expected amount of overfitting for a configuration produced by HPO and evaluated using 10CV.

The data splitting practices are replicated from the holdout experiment, elaborated on in Section 6.3.1, using either 60% of available data for datasets smaller than 3 000 instances, and 1 500 instances otherwise. Every repetition of an experiment, the available instances for training and validation were resampled.

The validation loss of the 10CV procedure is averaged over the validation splits of the 10 folds, and equivalently to the holdout experiment described in Section 6.3.1 can be exploited by the CASH algorithms to suggest new configurations. However, assessing the generalization performance of a configuration on the unseen test set in the 10CV procedure requires some thought, since essentially we obtain 10 trained pipelines. Several methods exist, such as using the fitted pipeline of one of the folds, averaging the predictions of all 10 fitted models on the test set, or retraining the same configuration on all available data. In the experiments in this work, the fitted model in each fold predicts both the validation and test set, and the final test performance is estimated by averaging the predictions of the 10 fitted pipelines, identical to the computation of the cross-validation score. Selecting the fitted pipeline from one fold might provide a biased estimation, retraining the configuration was computationally undesirable, and averaging the predictions of the fitted cross-validation pipelines is a form of ensembling, arguably benefiting from the often positive effects of ensembling on performance, leading to unfair comparison with the validation estimates, which are not the result of ensembling several models.

### 6.3.4 Repetitions

Every experiment on one dataset is repeated 100 times using different random seeds. Every repetition consists of 250 iterations of random search, Bayesian optimization, and BO-THO. At each iteration, the pipeline is trained and evaluated on both the validation and unseen test set. The validation scores, either calculated using holdout or 10-fold CV, are in the case of Bayesian optimization and BO-THO used internally to propose new configurations. The test scores are stored at each iteration but remain unobserved for the algorithms.

Every dataset therefore includes training and evaluating a machine pipeline 75,000 times. Including the 10CV experiments described in Section 6.3.3 and experiments with altering validation sizes, roughly 10 million pipelines were trained and evaluated in our experiments. On 20 Intel(R) Xeon(R) CPU @ 2.20GHz with 2 cores, the full experiments ran in roughly 13 days.

## 6.4 Statistical analysis

Throughout the discussion of the results produced by the experiments described in this chapter, several statistical tests will be conducted to support the analyses. This section will briefly introduce and motivate the use of these tests.

As motivated in earlier chapters, adaptive overfitting occurs when repeatedly using a validation procedure causes results to be overinflated compared to the performance measured on the true distribution. In the context of HPO, this entails suggesting progressively more overfitted configurations when more evaluations have been completed since more information about the validation procedure is incorporated in the suggestion of new configurations. This indicates the existence of a relationship between the number of iterations and the average amount of the configuration is overfitted in that iteration. We statistically investigate this effect using Spearman's rank-order correlation [60], which is a non-parametric test and therefore can assess the relationship between variables that are not normally distributed. Since the variable containing the iterations is uniformly distributed, a non-parametric test is most appropriate. A higher coefficient indicates a stronger relationship between the iteration in which a configuration is proposed and its MOE. The p-value from Spearman's correlation test is consulted to determine the significance of the effect, where the null hypothesis states no significant relation between the variables exists.

To compare the selected MOE of the three algorithms after 250 iterations, we use a test of equal variances (ANOVA) [45]. For this test to be appropriate, all groups have to be normally distributed and homogeneity of the variances of the groups is assumed. We test the normality of each of the groups individually using Shapiro's test [58], where the null hypothesis is that the sample is normally distributed. We test the homogeneity of variances using Levene's test [38], where the null hypothesis is that the variances of the groups are equal.

# Chapter 7

# Results

In this chapter, we will thoroughly elaborate on the results of the methods and experiments introduced in Chapter 5 and Chapter 6 respectively, attempting to answer the research questions stated in Section 6.3, with the first three sections in this chapter corresponding to the three experimental subsections in Section 6.3. Throughout this chapter, overfitting will be addressed in two primary ways, as formally defined in Chapter 2:

1. Average MOE: the average MOE of all configurations suggested in iteration $i$, across all repetitions. Formalized in Equation 2.4.
2. Selected MOE: the MOE of the best configuration observed so far at iteration $i$ measured on the validation procedure, averaged over all repetitions. Formalized in Equation 2.5.

We build this chapter by extensively reporting and visualizing the results using the *adult* dataset, which consists of 15 features, 48 842 instances, and a binary classification task to predict if the salary of a person is over 50 000 dollars a year [34]. We then extrapolate these results to all datasets relevant to the analysis and report the full results in tabular format. The remainder of this chapter is organized as follows. Section 7.1 presents the large-scale holdout experiments, including classification and regression tasks, and is therefore the majority of this chapter. Section 7.2 and Section 7.3 present the experiments with varying validation sizes and 10-fold cross-validation respectively. Finally, the limitations of this work are discussed in Section 7.4.

## 7.1 Large-scale holdout results

In this section, we investigate to what extent overfitting occurs in HPO approaches for the CASH problem when utilizing the holdout method for validation. Since binary classification, multiclass classification, and regression tasks are considered in combination with random search, Bayesian optimization, and

BO-THO, this section will encompass the majority of this chapter.

### 7.1.1 Results on the *Adult* dataset

Figure 7.1 shows the holdout and test evaluations of all configurations produced by random search, Bayesian optimization, and BO-THO against the $y = x$ line, providing a comparison with the situation in which no overfitting is observed. Unsurprisingly, the results show that all algorithms are susceptible to overfitting, as indicated by the deviation from the $y = x$ line for the configurations with the highest validation evaluations, which are isolated and magnified in Figure 7.2. Logically, for random search, this effect is not driven by the iteration in which the configuration was suggested, in contrast to Bayesian optimization, which seems to be sampling more around overfitted regions in later iterations.

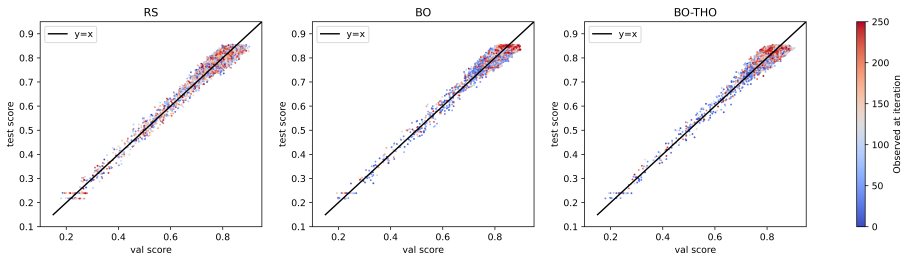


Figure 7.1: Test and validation accuracy scores of all configurations produced by the different HPO algorithms on the *adult* dataset. The color indicates in what iteration they were suggested.

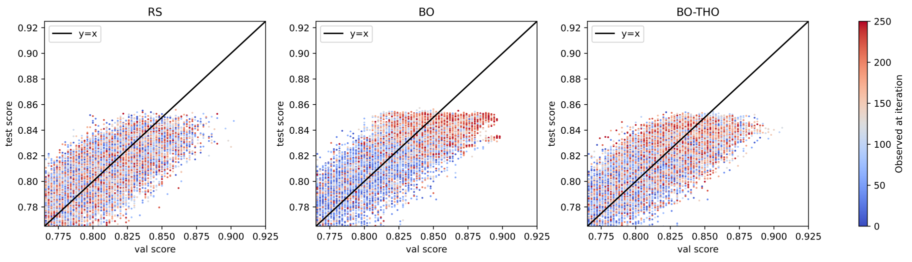
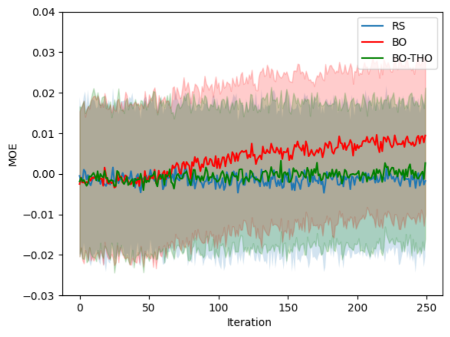


Figure 7.2: Test and validation accuracy scores of highest performing configurations produced by the different HPO algorithms on the *adult* dataset. The color indicates in what iteration they were suggested.

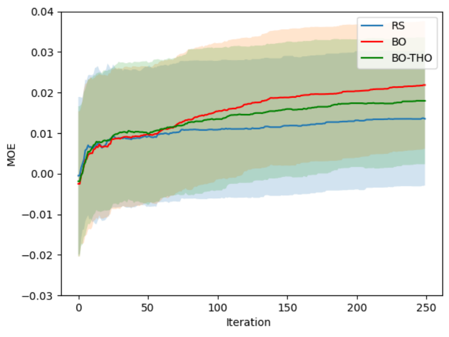
To investigate this effect on the *adult* dataset further, we consider the results in Figure 7.3a which visually shows a linear relationship between the iteration in which a configuration is proposed by Bayesian optimization and its average

MOE. Assessing Spearman's correlation coefficient [60], as motivated in Section 6.4, resulted in a coefficient of 0.179 ($p = 0.000$), providing strong evidence against the null hypothesis, showing that a relation exists between the iteration in which a configuration is proposed and its MOE. This indicates that Bayesian optimization is adaptively overfitting a validation set with 500 instances from the *adult* dataset. As should be expected, no significant relationship between iteration and MOE was found for random search. In the case of BO-THO, a significant relationship was found ($p = 0.000$) with a coefficient of 0.029, showing much more resilience against adaptive overfitting when compared to Bayesian optimization. This answers two of our research questions, showing that Bayesian optimization exhibits adaptive overfitting and that adopting thresholdout within the Bayesian optimization mechanism partially mitigates this effect.

Projecting these results to the practical HPO space, the selected MOE was considered, which is shown for random search, Bayesian optimization, and BO-THO in Figure 7.3b. These results show that for the binary classification task in the *adult* dataset, the selected configuration found by random search, Bayesian optimization, and BO-THO after 250 iterations has a MOE of 0.014, 0.022, and 0.018 respectively.



(a) Average MOE



(b) Selected MOE

Figure 7.3: The average MOE (a) and selected MOE (b) observed at each iteration by RS, BO, and BO-THO on the *adult* dataset, averaged over 100 repetitions.

In order to statistically compare the selected configurations by random search and Bayesian optimization after 250 iterations, we conducted a series of tests to confirm the assumptions required for an ANOVA test [45], as motivated in Section 6.4. We confirmed the normality of all three groups using Shapiro's test [58], resulting in random search, Bayesian optimization, and BO-THO reporting significance levels of $p = 0.733$, $p = 0.453$, and $p = 0.216$ respectively, not rejecting the null hypothesis that the sample is normally distributed for any of the algorithms. Furthermore, Levene's test ($p = 0.642$) does not reject the null hypothesis that the variances of the groups are significantly different, validating the use of an ANOVA test for comparison. The ANOVA results revealed a statistically significant difference in the MOE of the selected configurations of random search, Bayesian optimization, and BO-THO after 250 iterations on the *adult* dataset ($p = 0.001$), indicating that the results presented in Figure 7.3 are significantly different between the algorithms.

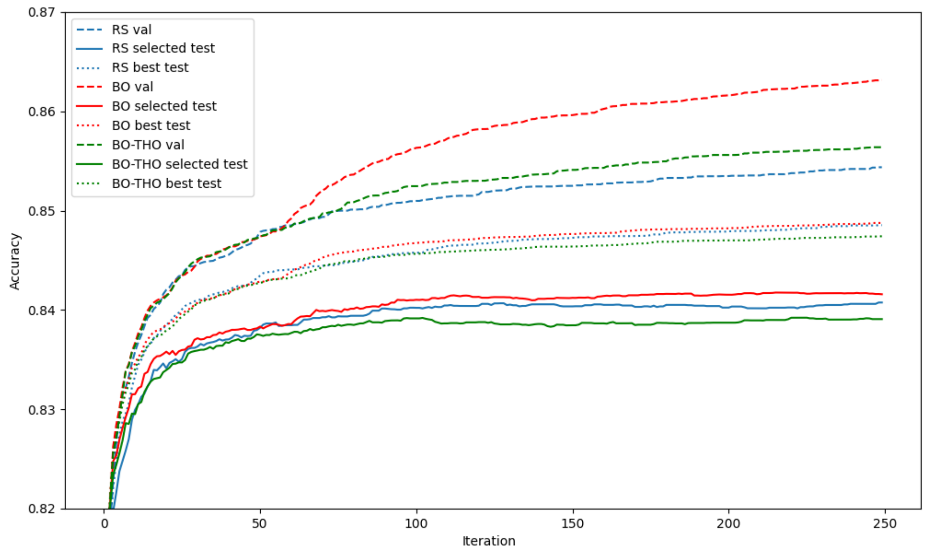


Figure 7.4: Average development of observed validation and test performances of RS, BO, and BO-THO for 250 iterations on the *adult* dataset.

Additionally, we examine the average progression of three metrics across all repetitions: the best validation performance observed (val), the test performance of the corresponding configuration (selected test), and the best test performance observed (best test). The latter represents the potential performance if the best configuration on the test set is selected. Figure 7.4 illustrates this progression, averaged over 100 repetitions, consistent with the previously reported results. Notably, the substantial gap between the best-observed generalization performance and the selected generalization performance highlights the need for more sophisticated evaluation procedures than the naive holdout procedure. Important to mention is the size of the test set (47 342 instances), which unusually large size provides stable and reliable performance estimations, making it unlikely a configuration performing very well on the test set was sampled by chance. Interestingly, the performance of the selected configurations after 250 iterations deviates relatively little between the algorithms, although Bayesian optimization outperforms random search and BO-THO on this dataset. Curiously, the increase on the test set seems hardly present after 100 iterations for all algorithms, while the performance of the configuration scoring best on the validation procedure keeps increasing, especially for Bayesian optimization, indicating that much of the optimization after a certain number of iterations does not yield actual improvement, but rather is only overfitting the validation procedure more.

### 7.1.2 All binary classification results

In this section, we summarize the results of all binary classification datasets. Table 7.1 shows the results using the holdout method described in 6.3.1. The values are sorted based on the coefficient of Spearman's correlation test on the average MOE per iteration for Bayesian optimization. The p-value and significance of each correlation are shown. The significance was determined using the conservative Bonferroni correction [7] to correct for testing multiple hypotheses, setting the p-value threshold to 1.9e-38, resulting in 25 significant linear relationships out of 29 datasets, indicating that adaptive overfitting by Bayesian optimization is widely spread across binary classification tasks. Furthermore, we can observe that overfitting can be expected more for tasks with lower empirical performance of the best classifier, which is not unexpected, considering less room for improvement exists in case the best classifier shows high accuracy on the task.

| ID | Val size | RS score | RS MOE | BO score | BO MOE | BO-THO score | BO-THO MOE | Correlation | p-value | Significant |
|---|---|---|---|---|---|---|---|---|---|---|
| 23381 | 100 | 0.585 | 0.095 | **0.593** | 0.098 | 0.586 | 0.094 | 0.241 | 0.0 | Yes |
| 23517 | 500 | 0.505 | 0.058 | **0.506** | 0.062 | 0.505 | 0.057 | 0.232 | 8.29e-302 | Yes |
| 1510 | 114 | 0.956 | 0.030 | 0.958 | 0.030 | **0.959** | 0.027 | 0.215 | 1.38e-259 | Yes |
| 1480 | 116 | 0.692 | 0.079 | **0.697** | 0.085 | 0.693 | 0.083 | 0.216 | 5.03e-261 | Yes |
| 31 | 200 | 0.729 | 0.046 | **0.731** | 0.055 | 0.727 | 0.050 | 0.206 | 3.24e-238 | Yes |
| 1494 | 211 | **0.848** | 0.030 | 0.848 | 0.042 | 0.842 | 0.041 | 0.198 | 2.55e-219 | Yes |
| 6332 | 108 | 0.725 | 0.066 | **0.729** | 0.083 | 0.722 | 0.078 | 0.193 | 2.41e-209 | Yes |
| 37 | 153 | 0.752 | 0.047 | **0.755** | 0.048 | 0.748 | 0.053 | 0.192 | 1.25e-206 | Yes |
| 29 | 138 | 0.858 | 0.037 | **0.860** | 0.045 | 0.858 | 0.038 | 0.181 | 1.44e-183 | Yes |
| 40994 | 108 | 0.932 | 0.033 | **0.933** | 0.038 | 0.930 | 0.035 | 0.181 | 1.33e-183 | Yes |
| 1464 | 149 | **0.772** | 0.038 | 0.772 | 0.043 | 0.770 | 0.039 | 0.180 | 4.69e-181 | Yes |
| 1590 | 500 | 0.841 | 0.014 | **0.842** | 0.022 | 0.839 | 0.018 | 0.179 | 1.75e-178 | Yes |
| 1461 | 500 | 0.892 | 0.015 | **0.894** | 0.018 | 0.891 | 0.016 | 0.179 | 1.37e-181 | Yes |
| 1049 | 291 | 0.897 | 0.025 | **0.898** | 0.028 | 0.897 | 0.026 | 0.178 | 8.85e-177 | Yes |
| 1063 | 104 | 0.822 | 0.053 | **0.823** | 0.057 | **0.823** | 0.052 | 0.177 | 1.64e-174 | Yes |
| 38 | 500 | 0.979 | 0.004 | **0.979** | 0.008 | 0.978 | 0.007 | 0.177 | 1.64e-174 | Yes |
| 15 | 140 | **0.961** | 0.019 | 0.959 | 0.022 | 0.959 | 0.020 | 0.174 | 3.44e-169 | Yes |
| 1053 | 500 | 0.808 | 0.017 | **0.809** | 0.020 | 0.807 | 0.019 | 0.157 | 8.11e-138 | Yes |
| 1067 | 422 | 0.853 | 0.019 | **0.854** | 0.023 | 0.852 | 0.022 | 0.145 | 2.62e-117 | Yes |
| 1489 | 500 | 0.846 | 0.012 | **0.849** | 0.024 | 0.848 | 0.019 | 0.143 | 4.69e-115 | Yes |
| 4534 | 500 | 0.937 | 0.010 | **0.937** | 0.016 | 0.935 | 0.013 | 0.139 | 8.26e-109 | Yes |
| 1068 | 222 | 0.929 | 0.015 | 0.929 | 0.019 | **0.930** | 0.015 | 0.134 | 8.49e-100 | Yes |
| 1050 | 312 | **0.892** | 0.020 | 0.892 | 0.023 | 0.892 | 0.020 | 0.124 | 4.52e-86 | Yes |
| 151 | 500 | 0.792 | 0.012 | **0.797** | 0.022 | 0.791 | 0.018 | 0.119 | 4.65e-79 | Yes |
| 40701 | 500 | 0.932 | 0.006 | **0.937** | 0.012 | 0.932 | 0.010 | 0.104 | 1.50e-60 | Yes |
| 3 | 500 | 0.982 | 0.003 | **0.984** | 0.007 | 0.983 | 0.005 | 0.080 | 1.88e-36 | No |
| 40983 | 500 | 0.980 | 0.005 | **0.980** | 0.007 | 0.979 | 0.006 | 0.076 | 1.34e-33 | No |
| 1462 | 274 | 0.997 | 0.003 | 0.997 | 0.003 | **0.997** | 0.003 | 0.054 | 8.14e-18 | No |
| 50 | 191 | 0.981 | 0.004 | **0.983** | 0.006 | 0.981 | 0.009 | 0.019 | 0.003 | No |

Table 7.1: Results after 250 iterations on all binary classification tasks, averaged over 100 repetitions using the holdout method. The score for each method is the accuracy of the final selected configuration.

### 7.1.3 Multiclass classification

This section reports the results of the holdout experiment on multiclass classification tasks. Specifically, an attempt is made to identify to what extent we can expect adaptive overfitting on multiclass classification and regression tasks compared to binary classification tasks. This comparison has been a region of interest and was theoretically approached in adaptive data analysis [19], and was phrased as a research question in Section 6.3.1, but to our knowledge has not been investigated empirically. As described in Section 6.4, we consider the coefficient of Spearman's correlation between the iteration in which a configuration is proposed and its MOE to be an indication of the amount of adaptive overfitting, allowing for comparison between tasks, where comparison based on accuracy or even RMSE for regression tasks would not be appropriate.

Since we consider 19 multiclass tasks, the Bonferroni correction shows a significance threshold of 1.9e-25. This indicates a significant linear correlation between the number of iterations and the average MOE of Bayesian optimization in 16 datasets, indicating that Bayesian optimization is in fact adaptively overfitting to the multiclass task as well. Furthermore, we do not find a linear connection between the number of classes and Spearman's coefficient, or the MOE of the selected configuration by Bayesian optimization. Therefore, although we are limited in our comparison, we do not find evidence that less adaptive overfitting occurs in classification tasks when the number of classes increases. Considering random search and BO-THO, we observe similar results as in binary classification, both being outperformed by Bayesian optimization, but less prone to overfitting.

| **ID** | **Classes** | **Val size** | **RS score** | **RS MOE** | **BO score** | **BO MOE** | **BO-THO score** | **BO-THO MOE** | **Coefficient** | **p-value** | **Significant** |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 469 | 6 | 159 | 0.194 | 0.074 | **0.196** | 0.083 | 0.190 | 0.080 | 0.280 | 0.000 | Yes |
| 23 | 3 | 294 | 0.531 | 0.044 | **0.535** | 0.053 | 0.533 | 0.045 | 0.212 | 1.90e-253 | Yes |
| 188 | 5 | 147 | 0.622 | 0.046 | 0.621 | 0.072 | **0.625** | 0.049 | 0.198 | 1.46e-219 | Yes |
| 22 | 10 | 400 | 0.807 | 0.021 | **0.811** | 0.030 | 0.807 | 0.023 | 0.185 | 1.75e-191 | Yes |
| 54 | 4 | 169 | 0.781 | 0.032 | **0.786** | 0.043 | 0.780 | 0.036 | 0.158 | 3.32e-140 | Yes |
| 40984 | 7 | 462 | 0.915 | 0.011 | **0.918** | 0.021 | 0.918 | 0.014 | 0.135 | 1.87e-101 | Yes |
| 40499 | 11 | 500 | 0.991 | 0.004 | **0.992** | 0.006 | 0.991 | 0.006 | 0.128 | 1.40e-91 | Yes |
| 182 | 6 | 500 | 0.878 | 0.011 | **0.882** | 0.021 | 0.879 | 0.016 | 0.112 | 4.83e-71 | Yes |
| 4538 | 5 | 500 | 0.807 | 0.018 | **0.812** | 0.026 | 0.807 | 0.021 | 0.111 | 2.11e-70 | Yes |
| 18 | 10 | 400 | 0.731 | 0.015 | **0.731** | 0.022 | 0.724 | 0.022 | 0.096 | 1.25e-47 | Yes |
| 307 | 11 | 198 | 0.871 | 0.014 | **0.889** | 0.033 | 0.876 | 0.026 | 0.093 | 5.11e-49 | Yes |
| 40668 | 3 | 500 | 0.747 | 0.007 | **0.749** | 0.024 | 0.743 | 0.015 | 0.089 | 1.17e-45 | Yes |
| 40982 | 7 | 388 | 0.740 | 0.014 | **0.750** | 0.031 | 0.741 | 0.024 | 0.088 | 1.57e-44 | Yes |
| 11 | 3 | 125 | 0.932 | 0.014 | **0.939** | 0.024 | 0.931 | 0.017 | 0.082 | 2.14e-31 | Yes |
| 41027 | 3 | 500 | 0.785 | 0.007 | **0.786** | 0.022 | 0.785 | 0.016 | 0.081 | 3.11e-37 | Yes |
| 32 | 10 | 500 | 0.969 | 0.006 | **0.974** | 0.010 | 0.971 | 0.009 | 0.079 | 7.36e-36 | Yes |
| 6 | 26 | 500 | 0.795 | 0.005 | **0.813** | 0.019 | 0.810 | 0.016 | 0.051 | 4.49e-16 | No |
| 40975 | 4 | 345 | 0.959 | 0.003 | **0.962** | 0.013 | 0.959 | 0.010 | 0.043 | 6.39e-012 | No |
| 1497 | 4 | 500 | 0.986 | 0.003 | 0.988 | 0.006 | **0.988** | 0.005 | 0.027 | 2.15e-05 | No |

Table 7.2: Results after 250 iterations on all multiclass classification tasks, averaged over 100 repetitions using the holdout method.

## 7.1.4 Regression

The results of the experiments on the regression tasks are displayed in Table 7.3. The scoring metric used in the regression tasks, RMSE, differs between datasets and is therefore hard to interpret. Similarly as the analysis for other tasks, we apply a Bonferroni correction to the significance threshold for the Spearman's correlation test, resulting in a threshold of 1.5e-22 for 16 regression tasks. Surprisingly, we find only 3 datasets display a significant linear relationship between the number of iterations and the MOE of the average configuration of Bayesian optimization, providing evidence for the conclusion that Bayesian optimization does not adaptively overfit most regression tasks. This is in sharp contrast with the classification tasks examined, where a significant relationship was found, even with a highly conservative correction on the significance.

| **ID** | **Val size** | **RS score** | **RS MOE** | **BO score** | **BO MOE** | **BO-THO score** | **BO-THO MOE** | **Coefficient** | **p-value** | **Significant** |
|---|---|---|---|---|---|---|---|---|---|---|
| 201 | 500 | -4.182 | 0.294 | -4.001 | 0.708 | **-3.934** | 0.620 | 0.145 | 7.8e-11 | No |
| 41540 | 500 | -2951.925 | 49.887 | -2923.218 | 90.469 | **-2921.574** | 80.786 | 0.088 | 1.11e-33 | Yes |
| 287 | 500 | -0.543 | 0.009 | **-0.539** | 0.014 | -0.544 | 0.012 | 0.087 | 7.62e-45 | Yes |
| 42726 | 500 | -1.538 | 0.048 | **-1.526** | 0.063 | -1.527 | 0.056 | 0.063 | 2.57e-20 | No |
| 531 | 101 | **-2.654** | 0.120 | -2.673 | 0.235 | -2.685 | 0.227 | 0.057 | 1.53e-24 | Yes |
| 42727 | 500 | -0.118 | 0.002 | **-0.118** | 0.002 | -0.118 | 0.002 | 0.048 | 1.64e-14 | No |
| 541 | 231 | -7.616 | 0.548 | **-7.263** | 0.814 | -7.289 | 0.861 | 0.044 | 3.62e-11 | No |
| 546 | 115 | -0.610 | 0.040 | **-0.609** | 0.043 | -0.611 | 0.038 | 0.044 | 2.28e-08 | No |
| 507 | 500 | -0.083 | 0.001 | **-0.082** | 0.002 | -0.084 | 0.001 | 0.044 | 6.74e-16 | No |
| 42729 | 500 | -0.792 | 0.015 | **-0.776** | 0.0332 | -0.785 | 0.023 | 0.040 | 9.00e-15 | No |
| 574 | 500 | -19100.138 | -144.101 | -18615.466 | 132.063 | **-18540.787** | 162.249 | 0.033 | 8.859e-08 | No |
| 42225 | 500 | -489.276 | 9.172 | **-455.198** | 18.517 | -463.538 | 18.456 | 0.033 | 2.15e-06 | No |
| 550 | 435 | -0.142 | 0.001 | -0.142 | 0.002 | **-0.141** | 0.001 | 0.021 | 7.45e-05 | No |
| 41021 | 246 | **-16.898** | 0.169 | -16.934 | 0.316 | -16.916 | 0.278 | 0.017 | 0.001 | No |
| 42728 | 500 | -11.378 | 0.150 | **-11.371** | 0.174 | -11.373 | 0.174 | 0.004 | 0.507 | No |
| 42688 | 500 | -0.720 | 0.030 | -0.702 | 0.037 | **-0.697** | 0.035 | -0.137 | 3.22e-102 | No |

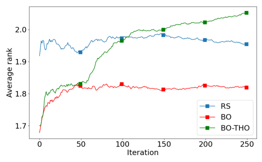
Table 7.3: Results after 250 iterations on all regression tasks, averaged over 100 repetitions using the holdout method.

### 7.1.5 Summary of different tasks

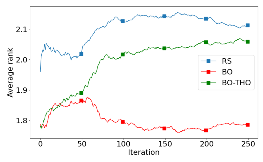
Concluding the holdout experiments on three different tasks, binary classification, multiclass classification, and regression, our results do not provide evidence to distinguish between classification tasks. Therefore, we can not conclude that multiclass classification datasets are less sensitive to adaptive overfitting. However, we found that the relationship between the number of iterations and the average MOE of suggested configurations for regression tasks does not present itself, suggesting that regression tasks are largely immune to adaptive overfitting.

### 7.1.6 Evaluation of RS, BO, and BO-THO

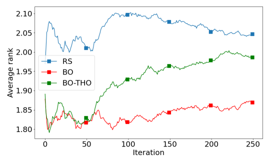
Finally, we assess the relative performance of random search, Bayesian optimization, and BO-THO used throughout this thesis in a competitive setting. To compare performance across tasks, we use average ranking [9], a common measure of comparison between tasks with different metrics and complexities. Average ranking provides a rank for each task for every algorithm and averages these ranks across datasets. Figure 7.5 shows the average ranking across all binary classification tasks (Figure 7.5a), all multiclass classification tasks (Figure 7.5b), and all regression tasks (Figure 7.5c). Additionally, Figure 7.6 provides the ranking overall of 64 tasks.



(a) Average ranking on all binary tasks.



(b) Average ranking on all multiclass tasks.



(c) Average ranking on all regression tasks.

Figure 7.5: Average rankings of random search, Bayesian optimization, and BO-THO using accuracy for classification and RMSE for regression, computed on all binary classification tasks (a), all multiclass classification tasks (b), and all regression tasks (c).

Investigating the results in Figure 7.5 and Figure 7.6, we observe the superiority of Bayesian optimization, which is consistent with results from the literature. BO-THO is outperformed by Bayesian optimization in all three tasks, and is in terms of performance equal to random search, specifically with more iterations. These results show that thresholdout, although preventing adaptive overfitting effectively, does not increase the performance of Bayesian optimization. However, the simplicity of thresholdout, and its ability to limit adaptive overfitting while sampling from higher-performing regions compared to random search, perhaps allows the existence of a more sophisticated algorithm sampling from highly generalized areas, a direction that future research could explore.

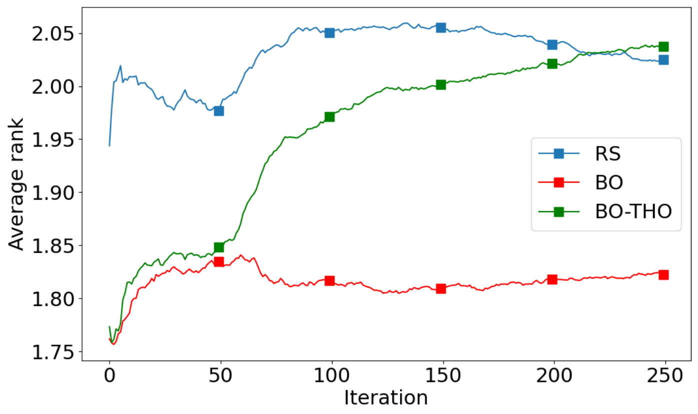


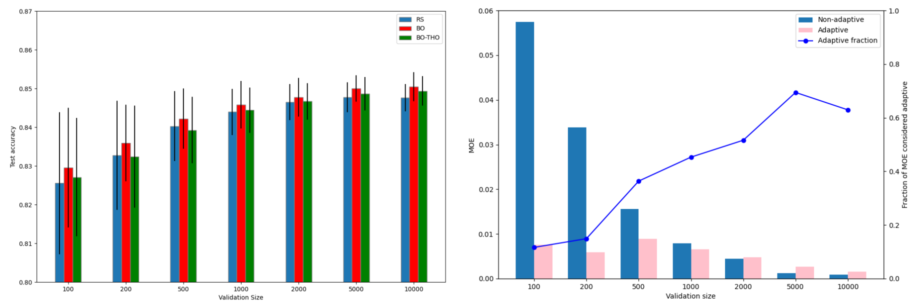
Figure 7.6: Average ranking of random search, Bayesian optimization, and BO-THO across tasks.

## 7.2 Varying validation sizes

Although data scarcity often limits the available data for the validation procedure, investigating validation sets of varying sizes increases the transferability of the results in this work to datasets of different sizes. In Chapter 3, we mentioned the expected square root relationship between the validation set size and the expected maximum (non-adaptive) MOE of any configuration. This section discusses the empirical results of altering the validation set size while keeping the train and test set sizes constant. As described in detail in Chapter 5, this experiment was conducted using 4 datasets for binary classification.

We investigate the performance of all three algorithms in the general HPO application, plotting the test scores of the selected models after 250 iterations, as shown in Figure 7.7a. These results demonstrate the benefits of rigorous configuration evaluation. We found that better-performing configurations on the fixed test set were selected as the validation size increased. This indicates that a larger validation size allows for a more accurate assessment of model performance, leading to improved configuration selection.

An interesting observation can be made regarding the MOE with varying validation set sizes. As shown in Figure 7.7b, we compare the selected MOE after 250 iterations for random search and the additional MOE introduced by Bayesian optimization. The selected MOE by random search decreases clearly as the validation size increases, in line with the expectations in the field. However, the additional MOE introduced by Bayesian optimization decreases at a slower rate, making up for a larger fraction of the total MOE as the validation set size increases, as indicated by the blue line in Figure 7.7b. This arguably suggests that adaptive overfitting is less mitigated by increasing validation set sizes, which might be an intriguing angle for future research.



(a) Test performance of selected configurations after 250 iterations for different validation set sizes, for the RS, BO, and BO-THO algorithms.

(b) MOE of selected configurations after 250 iterations. Line denotes the fraction of total overfit that is contributed to adaptive overfitting.

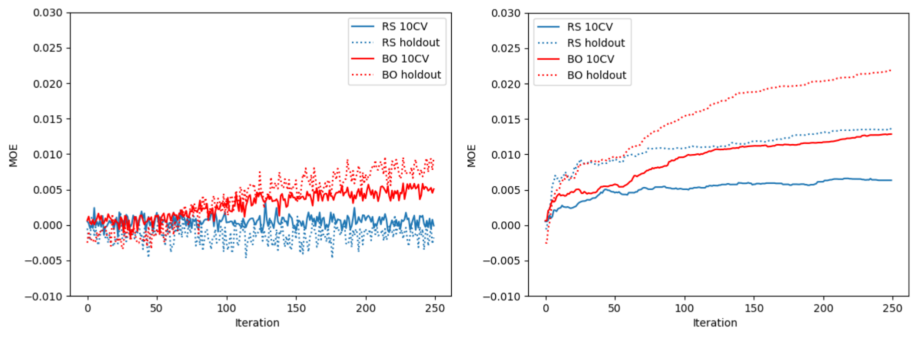
Figure 7.7: Performance and causes of overfitting when considering validation sets of different sizes.

## 7.3 10CV

The instability and potential of overfitting of the holdout method are widely recognized, as discussed in earlier chapters. Cross-validation is the de facto method to provide more stable and reliable model evaluation. In this section, 10CV will be investigated in the context of overfitting in HPO, assessing the mitigation effect of 10CV on overfitting compared to the holdout method, and the impact of using 10CV as an inner evaluation method on the overall performance of random search, Bayesian optimization, and BO-THO. This section will additionally discuss the widespread use of 10CV as benchmark practice, and the expected overfitting of complex pipelines evaluated using this procedure.

The results of the comparison between the overfitting effects of the holdout method and the 10CV method are displayed in Figure 7.8. In the average case (Figure 7.8a, Bayesian optimization evaluated with 10CV appears to sample from less overfitted configurations, although the difference is relatively small. The Spearman's correlation coefficients for the *adult* dataset are 0.142 and 0.179 for Bayesian optimization with 10CV and Bayesian optimization with holdout, respectively, with both tests reporting significant results (both $p = 0.000$). This indicates a statistically significant but minor reduction in overfitting when using 10CV compared to the holdout method. These results suggest that leveraging 10CV instead of the holdout method can partially mitigate adaptive overfitting. However, the effects are small and may not justify the tenfold increase in computational costs in many practical scenarios.

Figure 7.8b shows the MOE of the selected configuration at each iteration and is therefore of more practical interest. 10CV has a stabilizing effect, reducing the average MOE of the best configuration found using the holdout evaluation. Especially in the absence of adapting the procedure to previous evaluations, the MOE resulting from random search suggests a relatively strong foundation for 10CV for robust model evaluation. However, although the average MOE of Bayesian optimization is reduced, it is perhaps not as resilient as often thought. Our conclusions are limited to the binary classification task on a dataset with 1 500 instances and 15 features, but considering the median instance count of approximately 1 800 in the OpenML-CC18 dataset, these tasks are arguably representative. Following these results, benchmarking practices on similar binary classification tasks for Bayesian optimization with a similar number of iterations can expect to be overfitted between 0.010 and 0.015, which is in many practical scenarios a large margin. Concluding the research question stated in Section 6.3.3, 10CV mitigates overfitting, but should still be treated carefully as benchmark practice.



(a) Average MOE (b) Selected MOE

Figure 7.8: Average MOE (a) selected MOE (b) for algorithms evaluated with holdout and 10CV.

Table 7.4 shows the 10CV and holdout results on the 4 datasets selected in Chapter 5, indicating that 10CV is successful in reducing overfitting in all 4 datasets between 40% and 45% and improving generalization performance of the final selected configuration in 3 of the 4 datasets.

| ID | Holdout | | | | | | 10CV | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | RS score | RS MOE | BO score | BO MOE | Coefficient | p-value | RS score | RS MOE | BO score | BO MOE | Coefficient | p-value |
| 151 | 0.792 | 0.012 | 0.797 | 0.022 | 0.118 | 4.65e-79 | 0.803 | 0.004 | 0.805 | 0.012 | 0.062 | 8.30e-33 |
| 1461 | 0.892 | 0.015 | 0.894 | 0.018 | 0.179 | 1.37e-181 | 0.878 | 0.006 | 0.878 | 0.010 | 0.134 | 6.11e-113 |
| 1590 | 0.841 | 0.014 | 0.842 | 0.022 | 0.179 | 1.75e-178 | 0.847 | 0.006 | 0.848 | 0.013 | 0.142 | 4.17e-112 |
| 23517 | 0.505 | 0.058 | 0.506 | 0.062 | 0.232 | 8.29e-302 | 0.507 | 0.033 | 0.508 | 0.037 | 0.327 | 0.000 |

Table 7.4: All 10CV results and corresponding holdout results.

## 7.4 Limitations

Despite these contributions, the work has certain limitations. Due to computational constraints, our cross-validation and validation size experiments were limited to four datasets, where the 10CV experiments took roughly 9 days per dataset on one machine with two cores to compute for all repetitions. Also, the fixed nature of the sizes of the training and validation sets in the majority of the experiments limits the portion of the problem space covered. The same argument can be made for excluding additional scoring metrics besides accuracy and RMSE. Finally, the simplicity of the thresholdout algorithm, which was the earliest published approach to mitigating adaptive overfitting on a holdout set, prompts the question of whether more sophisticated algorithms can actively cause Bayesian optimization to sample from high-performing and generalized regions, for example by adapting the noise rate based on the specific problem.

# Chapter 8

# Conclusion & Future Work

Throughout this thesis, we have delved into the phenomenon of overfitting in hyperparameter optimization. Two types of overfitting were examined; overfitting from evaluating many configurations, therefore selecting configurations that evaluate highly on the validation procedure by chance, and adaptive overfitting, where information from evaluated configurations is actively incorporated into the mechanism for proposing new configurations. Our approach involved comparing random search, Bayesian optimization, and BO-THO, which was introduced in this work. Concluding this work, our analyses provided the following contributions:

**Systematic analysis of overfitting in modern HPO**
To the best of our knowledge, this thesis presents one of the first large-scale assessments of overfitting in a CASH scenario, a method central to many popular HPO packages. Our analysis spanned a wide array of real-world datasets, including binary classification, multiclass classification, and regression tasks, providing comprehensive insights that have not been extensively explored in prior research. We introduce a simple metric to measure and compare adaptive overfitting across tasks, demonstrating that adaptive overfitting significantly affects CASH on the majority of classification tasks. Notably, it was found that regression tasks are minimally susceptible to overfitting, both by chance and adaptively. Surprisingly, no evidence was found for the theoretical hypothesis from adaptive data analysis [19] that multiclass classification problems are less susceptible to adaptive overfitting compared to binary classification tasks.

**Demonstrating the practical impact of overfitting**
We further demonstrated that (adaptive) overfitting can pose challenges in practical HPO scenarios, especially when data is scarce. Our experiments revealed that even widely accepted benchmark practices, such as 10CV, can be prone to adaptive overfitting. Notably, 10CV exhibited significant adaptive overfitting across all four datasets examined, which raises concerns about its suitability as an evaluation practice in HPO benchmarking. Furthermore, we observed

increasing the validation size successfully limits practical overfitting from evaluating many configurations, suggesting large datasets are less susceptible to this effect. However, adaptive overfitting in Bayesian optimization was shown to be much more resilient to this effect, providing an interesting direction for future research.

**Evaluation of mitigation methods**
Finally, we designed and evaluated mitigation methods for overfitting in HPO. We found 10CV to be successful in reducing overfitting when compared to the holdout method with 40% to 45% while increasing generalization performance in 3 out of 4 examined datasets, at the cost of a 10-fold increase in computational resources required. Furthermore, we introduced BO-THO, an integration of thresholdout [12] in Bayesian optimization for HPO, which successfully limits adaptive overfitting, but overall achieves less generalization performance compared to the regular implementation of Bayesian optimization.

**Future work**
In this work, we observe relatively limited generalization improvement for sophisticated methods compared to random search, while a lot of performance gains are reported on the validation procedure, specifically for small to medium datasets. Additionally, we observe that significantly higher-performing configurations on the test set exist, but are not selected. The most interesting direction for future research regarding this is the distinction between making progress in HPO by scoring even higher in less computational time on the validation procedure, often leading to increases in generalization as well, and altering the search process to allow optimizing for generalization performance rather than overfitting the validation set. The success observed in reshuffling the validation procedure every evaluation [42, 46] indicates the potential value of the latter, while little research has been directed at this. Multiple directions could be promising, including further investigating techniques developed in adaptive data analysis. Furthermore, the existence of more generalized configurations shows that post-selection procedures potentially could increase generalization performance, specifically while incorporating the knowledge that modern HPO methods exhibit adaptive overfitting.

A different direction aimed at directing the search process to more generalized solutions could be meta-learning, where previous results on many datasets can provide value in subsequent HPO approaches, identifying the risk of overfitting and adjusting the search procedure accordingly, for example by designing an acquisition function incorporating some measure of overfitting risk in the proposal of new configurations.

Finally, a thorough continuation of assessing overfitting in modern HPO is required, especially regarding the advancements in ensemble methods. Curiosity arises regarding whether the widespread success of ensemble methods partially comes from its mitigating effect on (adaptive) overfitting.

# Bibliography

[1] Sanjeev Arora and Yi Zhang. Rip van Winkle's razor: A simple estimate of overfit to test data. *Computing Research Repository*, abs/2102.13189, 2021.

[2] George De Ath, Richard M. Everson, Alma As-Aad Rahat, and Jonathan E. Fieldsend. Greed is good: Exploration and exploitation trade-offs in Bayesian optimisation. *ACM Transactions on Evolutionary Learning and Optimization*, 1(1):1:1–1:22, 2021.

[3] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13:281–305, 2012.

[4] Srikanth Bethu, B. Sankara Babu, K. Madhavi, and P. Gopala Krishna. Algorithm selection and model evaluation in application design using machine learning. In *Machine Learning for Networking - Second IFIP TC 6 International Conference, MLN 2019, Paris, France, December 3-5, 2019, Revised Selected Papers*, volume 12081 of *Lecture Notes in Computer Science*, pages 175–195. Springer, 2019.

[5] Bernd Bischl, Martin Binder, Michel Lang, Tobias Pielok, Jakob Richter, Stefan Coors, Janek Thomas, Theresa Ullmann, Marc Becker, Anne-Laure Boulesteix, Difan Deng, and Marius Lindauer. Hyperparameter optimization: Foundations, algorithms, best practices, and open challenges. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 13(2), 2023.

[6] Bernd Bischl, Giuseppe Casalicchio, Matthias Feurer, Pieter Gijsbers, Frank Hutter, Michel Lang, Rafael Gomes Mantovani, Jan N. van Rijn, and Joaquin Vanschoren. OpenML benchmarking suites. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks 1, NeurIPS Datasets and Benchmarks 2021, December 2021, virtual*, 2021.

[7] Carlo Bonferroni. Teoria statistica delle classi e calcolo delle probabilita. *Pubblicazioni del R istituto superiore di scienze economiche e commericiali di firenze*, 8:3–62, 1936.

[8] Gavin C. Cawley and Nicola L. C. Talbot. On over-fitting in model selection and subsequent selection bias in performance evaluation. *Journal of Machine Learning Research*, 11:2079–2107, 2010.

[9] Janez Demsar. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research*, 7:1–30, 2006.

[10] Thomas G. Dietterich. Ensemble methods in machine learning. In *Multiple Classifier Systems, First International Workshop, MCS 2000, Cagliari, Italy, June 21-23, 2000, Proceedings*, volume 1857 of *Lecture Notes in Computer Science*, pages 1–15. Springer, 2000.

[11] Cynthia Dwork, Vitaly Feldman, Moritz Hardt, Toniann Pitassi, Omer Reingold, and Aaron Roth. Generalization in adaptive data analysis and holdout reuse. In *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, NeurIPS 2015, December 7-12, 2015, Montreal, Quebec, Canada*, pages 2350–2358, 2015.

[12] Cynthia Dwork, Vitaly Feldman, Moritz Hardt, Toniann Pitassi, Omer Reingold, and Aaron Roth. The reusable holdout: Preserving validity in adaptive data analysis. *Science*, 349(6248):636–638, 2015.

[13] Cynthia Dwork, Vitaly Feldman, Moritz Hardt, Toniann Pitassi, Omer Reingold, and Aaron Leon Roth. Preserving statistical validity in adaptive data analysis. In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*, pages 117–126. ACM, 2015.

[14] Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam D. Smith. Calibrating noise to sensitivity in private data analysis. In *Theory of Cryptography, Third Theory of Cryptography Conference, TCC 2006, New York, NY, USA, March 4-7, 2006, Proceedings*, volume 3876 of *Lecture Notes in Computer Science*, pages 265–284. Springer, 2006.

[15] Katharina Eggensperger, Matthias Feurer, Frank Hutter, James Bergstra, Jasper Snoek, Holger H. Hoos, and Kevin Leyton-Brown. Towards an empirical foundation for assessing Bayesian optimization of hyperparameters. In *NIPS workshop on Bayesian Optimization in Theory and Practice*, volume 10, 2013.

[16] Logan Engstrom, Andrew Ilyas, Shibani Santurkar, Dimitris Tsipras, Jacob Steinhardt, and Aleksander Madry. Identifying statistical bias in dataset replication. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, volume 119 of *Proceedings of Machine Learning Research*, pages 2922–2932. PMLR, 2020.

[17] Nick Erickson, Jonas Mueller, Alexander Shirkov, Hang Zhang, Pedro Larroy, Mu Li, and Alexander J. Smola. Autogluon-tabular: Robust and accurate AutoML for structured data. *Computing Research Repository*, abs/2003.06505, 2020.

[18] Fabio Fabris and Alex Alves Freitas. Analysing the overfit of the auto-sklearn automated machine learning tool. In *Machine Learning, Optimization, and Data Science - 5th International Conference, LOD 2019, Siena, Italy, September 10-13, 2019, Proceedings*, volume 11943 of *Lecture Notes in Computer Science*, pages 508–520. Springer, 2019.

[19] Vitaly Feldman, Roy Frostig, and Moritz Hardt. The advantages of multiple classes for reducing overfitting from test set reuse. In *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pages 1892–1900. PMLR, 2019.

[20] Jean Feng, Gene Pennello, Nicholas Petrick, Berkman Sahiner, Romain Pirracchio, and Alexej Gossmann. Sequential algorithmic modification with test data reuse. In *Uncertainty in Artificial Intelligence, Proceedings of the Thirty-Eighth Conference on Uncertainty in Artificial Intelligence, UAI 2022, 1-5 August 2022, Eindhoven, The Netherlands*, volume 180 of *Proceedings of Machine Learning Research*, pages 674–684. PMLR, 2022.

[21] Matthias Feurer, Katharina Eggensperger, Stefan Falkner, Marius Lindauer, and Frank Hutter. Auto-sklearn 2.0: Hands-free AutoML via meta-learning. *Journal of Machine Learning Research*, 23:261:1–261:61, 2022.

[22] Matthias Feurer and Frank Hutter. Hyperparameter optimization. In *Automated Machine Learning - Methods, Systems, Challenges*, The Springer Series on Challenges in Machine Learning, pages 3–33. Springer, 2019.

[23] Matthias Feurer, Aaron Klein, Katharina Eggensperger, Jost Tobias Springenberg, Manuel Blum, and Frank Hutter. Efficient and robust automated machine learning. In *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, pages 2962–2970, 2015.

[24] David A. Freedman. A note on screening regression equations. *The American Statistician*, 37(2):152–155, 1983.

[25] Frauke Friedrichs and Christian Igel. Evolutionary tuning of multiple SVM parameters. *Neurocomputing*, 64:107–117, 2005.

[26] Eduardo C. Garrido-Merchán and Daniel Hernández-Lobato. Dealing with categorical and integer-valued variables in Bayesian optimization with Gaussian processes. *Neurocomputing*, 380:20–35, 2020.

[27] Pieter Gijsbers, Marcos L. P. Bueno, Stefan Coors, Erin LeDell, Sébastien Poirier, Janek Thomas, Bernd Bischl, and Joaquin Vanschoren. AMLB: An AutoML benchmark. *Computing Research Repository*, abs/2207.12560, 2022.

[28] Pieter Gijsbers, Erin LeDell, Janek Thomas, Sébastien Poirier, Bernd Bischl, and Joaquin Vanschoren. An open source AutoML benchmark. *Computing Research Repository*, abs/1907.00909, 2019.

[29] Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *Learning and Intelligent Optimization - 5th International Conference, LION 5, Rome, Italy, January 17-21, 2011. Selected Papers*, volume 6683 of *Lecture Notes in Computer Science*, pages 507–523. Springer, 2011.

[30] Frank Hutter, Lars Kotthoff, and Joaquin Vanschoren, editors. *Automated Machine Learning - Methods, Systems, Challenges.* The Springer Series on Challenges in Machine Learning. Springer, 2019.

[31] Hamed Jalali, Inneke Van Nieuwenhuyse, and Victor Picheny. Comparison of kriging-based algorithms for simulation optimization with heterogeneous noise. *European Journal of Operational Research*, 261(1):279–301, 2017.

[32] Donald R. Jones, Matthias Schonlau, and William J. Welch. Efficient global optimization of expensive black-box functions. *Journal of Global Optimization*, 13(4):455–492, 1998.

[33] Ron Kohavi. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence, IJCAI 95, Montréal Québec, Canada, August 20-25 1995, 2 Volumes*, pages 1137–1145. Morgan Kaufmann, 1995.

[34] Ron Kohavi. Scaling up the accuracy of naive-bayes classifiers: A decision-tree hybrid. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD-96), Portland, Oregon, USA*, pages 202–207. AAAI Press, 1996.

[35] Lars Kotthoff, Chris Thornton, Holger H. Hoos, Frank Hutter, and Kevin Leyton-Brown. Auto-weka 2.0: Automatic model selection and hyperparameter optimization in WEKA. *Journal of Machine Learning Research*, 18:25:1–25:5, 2017.

[36] Trang T. Le, Weixuan Fu, and Jason H. Moore. Scaling tree-based automated machine learning to biomedical big data with a feature set selector. *Bioinformatics*, 36(1):250–256, 2020.

[37] Yann LeCun and Corinna Cortes. The MNIST database of handwritten digits, 2005.

[38] Howard Levene. Robust tests for equality of variances. *Contributions to probability and statistics*, pages 278–292, 1960.

[39] Julien-Charles Lévesque, Audrey Durand, Christian Gagné, and Robert Sabourin. Bayesian optimization for conditional hyperparameter spaces. In *2017 International Joint Conference on Neural Networks, IJCNN 2017, Anchorage, AK, USA, May 14-19, 2017*, pages 286–293. IEEE, 2017.

[40] Marius Lindauer, Katharina Eggensperger, Matthias Feurer, André Biedenkapp, Difan Deng, Carolin Benjamins, Tim Ruhkopf, René Sass, and Frank Hutter. SMAC3: A versatile Bayesian optimization package for hyperparameter optimization. *Journal of Machine Learning Research*, 23:54:1–54:9, 2022.

[41] Marius Lindauer, Katharina Eggensperger, Matthias Feurer, André Biedenkapp, Joshua Marben, Philipp Müller, and Frank Hutter. BOAH: A tool suite for multi-fidelity Bayesian optimization & analysis of hyperparameters. *Computing Research Repository*, abs/1908.06756, 2019.

[42] Julien-Charles Lévesque. *Bayesian hyperparameter optimization: Overfitting, ensembles and conditional spaces.* PhD thesis, Université Laval, 2018.

[43] Anastasia Makarova, Huibin Shen, Valerio Perrone, Aaron Klein, Jean Baptiste Faddoul, Andreas Krause, Matthias W. Seeger, and Cédric Archambeau. Overfitting in Bayesian optimization: An empirical study and early-stopping solution. *Computing Research Repository*, abs/2104.08166, 2021.

[44] Mark McLeod, Stephen J. Roberts, and Michael A. Osborne. Optimization, fast and slow: Optimally switching between local and Bayesian optimization. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pages 3440–3449. PMLR, 2018.

[45] Douglas C Montgomery. *Design and analysis of experiments.* John Wiley & sons, 2017.

[46] Thomas Nagler, Lennart Schneider, Bernd Bischl, and Matthias Feurer. Reshuffling resampling splits can improve generalization of hyperparameter optimization. *Computing Research Repository*, abs/2405.15393, 2024.

[47] Andrew Y. Ng. Preventing ”overfitting” of cross-validation data. In Douglas H. Fisher, editor, *Proceedings of the Fourteenth International Conference on Machine Learning (ICML 1997), Nashville, Tennessee, USA, July 8-12, 1997*, pages 245–253. Morgan Kaufmann, 1997.

[48] Thanh Dai Nguyen, Sunil Gupta, Santu Rana, and Svetha Venkatesh. Stable Bayesian optimization. In *Advances in Knowledge Discovery and Data Mining - 21st Pacific-Asia Conference, PAKDD 2017, Jeju, South Korea,*

*May 23-26, 2017, Proceedings, Part II*, volume 10235 of *Lecture Notes in Computer Science*, pages 578–591, 2017.

[49] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake VanderPlas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Edouard Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[50] Victor Picheny, Tobias Wagner, and David Ginsbourger. A benchmark of kriging-based infill criteria for noisy optimization. *Structural and multidisciplinary optimization*, 48:607–626, 2013.

[51] R. Bharat Rao and Glenn Fung. On the dangers of cross-validation. An experimental evaluation. In *Proceedings of the SIAM International Conference on Data Mining, SDM 2008, April 24-26, 2008, Atlanta, Georgia, USA*, pages 588–596. SIAM, 2008.

[52] Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian processes for machine learning.* Adaptive computation and machine learning. MIT Press, 2006.

[53] Benjamin Recht, Rebecca Roelofs, Ludwig Schmidt, and Vaishaal Shankar. Do CIFAR-10 classifiers generalize to CIFAR-10? *Computing Research Repository*, abs/1806.00451, 2018.

[54] Benjamin Recht, Rebecca Roelofs, Ludwig Schmidt, and Vaishaal Shankar. Do ImageNet classifiers generalize to ImageNet? In *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pages 5389–5400. PMLR, 2019.

[55] Juha Reunanen. Overfitting in making comparisons between variable selection methods. *Journal of Machine Learning Research*, 3:1371–1382, 2003.

[56] Rebecca Roelofs, Vaishaal Shankar, Benjamin Recht, Sara Fridovich-Keil, Moritz Hardt, John Miller, and Ludwig Schmidt. A meta-analysis of overfitting in machine learning. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 9175–9185, 2019.

[57] Warren R. Scott, Peter I. Frazier, and Warren B. Powell. The correlated knowledge gradient for simulation optimization of continuous parameters using Gaussian process regression. *SIAM J. Optim.*, 21(3):996–1026, 2011.

[58] Samuel S. Shapiro and Martin B. Wilk. An analysis of variance test for normality (complete samples). *Biometrika*, 52(3-4):591–611, 1965.

[59] Jasper Snoek, Hugo Larochelle, and Ryan P. Adams. Practical Bayesian optimization of machine learning algorithms. In *Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held December 3-6, 2012, Lake Tahoe, Nevada, United States*, pages 2960–2968, 2012.

[60] Charles Spearman. The proof and measurement of association between two things. *The American Journal of Psychology*, 100(3/4):441–471, 1987.

[61] Chris Thornton, Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. Auto-weka: Combined selection and hyperparameter optimization of classification algorithms. In *The 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD 2013, Chicago, IL, USA, August 11-14, 2013*, pages 847–855. ACM, 2013.

[62] Ryan Turner, David Eriksson, Michael McCourt, Juha Kiili, Eero Laaksonen, Zhen Xu, and Isabelle Guyon. Bayesian optimization is superior to random search for machine learning hyperparameter tuning: Analysis of the black-box optimization challenge 2020. In *NeurIPS 2020 Competition and Demonstration Track, 6-12 December 2020, Virtual Event / Vancouver, BC, Canada*, volume 133 of *Proceedings of Machine Learning Research*, pages 3–26. PMLR, 2020.

[63] Jan N. van Rijn and Frank Hutter. Hyperparameter importance across datasets. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD 2018, London, UK, August 19-23, 2018*, pages 2367–2376. ACM, 2018.

[64] Chhavi Yadav and Léon Bottou. Cold case: The lost MNIST digits. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 13443–13452, 2019.

[65] Xiang Zhou, Yixin Nie, Hao Tan, and Mohit Bansal. The curse of performance instability in analysis datasets: Consequences, source, and suggestions. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing, EMNLP 2020, Online, November 16-20, 2020*, pages 8215–8228. Association for Computational Linguistics, 2020.

[66] Tijana Zrnic and Moritz Hardt. Natural analysts in adaptive data analysis. In *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pages 7703–7711. PMLR, 2019.

# Appendix A

# Search spaces

This appendix describes the full search space utilized throughout the experiments. Every classifier, regressor, or pipeline component was implemented using the scikit-learn library [49]. This section is divided in the list of utilized classifiers A.1, the list of utilized regressors A.2, the hyperparameters used to configure the classifiers A.3, the hyperparameters used to configure the regressors A.4, the encoders used A.5, the imputers and their hyperparameters A.6, the feature selectors and their hyperparameters A.7, the scalers and their hyperparameters A.8, and finally the dimensionality reducers and their hyperparameters A.9.

## A.1 List of classifiers

The included classifiers for the search spaces in the experiments described in this thesis are shown in Table A.1.

## A.2 List of regressors

All regressors are implemented using the scikit-learn implementation [49]. Table A.2 shows all included regressors.

| **Classifier** | **Multiclass** |
|---|---|
| RandomForestClassifier | Yes |
| AdaBoostClassifier | Yes |
| DecisionTreeClassifier | Yes |
| GradientBoostingClassifier | Yes |
| BaggingClassifier | Yes |
| ExtraTreeClassifier | Yes |
| ExtraTreesClassifier | Yes |
| LogisticRegression | Yes |
| SGDClassifier | Yes |
| PassiveAggressiveClassifier | Yes |
| RidgeClassifier | Yes |
| Lasso | No |
| ElasticNet | No |
| GaussianNB | Yes |
| MLPClassifier | Yes |
| KNeighborsClassifier | Yes |
| LinearDiscriminantAnalysis | Yes |

Table A.1: All classifiers used for binary classification tasks. The Multiclass column indicates if classifiers were included in search space for multiclass classification tasks.

| **Regressor** |
|---|
| RandomForestRegressor |
| AdaBoostRegressor |
| DecisionTreeRegressor |
| GradientBoostingRegressor |
| BaggingRegressor |
| ExtraTreeRegressor |
| ExtraTreesRegressor |
| SGDRegressor |
| PassiveAggressiveRegressor |
| Ridge |
| Lasso |
| ElasticNet |
| MLPRegressor |
| KNeighborsRegressor |
| ARDRegression |

Table A.2: List of Regressors included in search space for regression tasks.

## A.3 Classifier hyperparameters

| **Hyperparameter** | **Active if model selected** | **Type** | **Valid Range/Choices** | **Sampled from log scale** |
|---|---|---|---|---|
| n_estimators | RandomForestClassifier | discrete | 10-200 | No |
| max_depth | RandomForestClassifier | discrete | 1-25 | No |
| rf_min_samples_split | RandomForestClassifier | discrete | 2-20 | No |
| rf_min_samples_leaf | RandomForestClassifier | discrete | 1-20 | No |
| rf_criterion | RandomForestClassifier | categorical | gini, entropy | No |
| rf_bootstrap | RandomForestClassifier | categorical | True, False | No |
| lr_C | LogisticRegression | continuous | 0.0001-50 | Yes |
| lr_solver | LogisticRegression | categorical | newton-cg, liblinear, lbfgs, saga, sag, newton-cholesky | No |
| mlp_hidden_layer_size | MLPClassifier | discrete | 10-1000 | No |
| mlp_activation | MLPClassifier | categorical | identity, logistic, tanh, relu | No |
| mlp_solver | MLPClassifier | categorical | lbfgs, sgd, adam | No |
| mlp_lr_init | MLPClassifier | continuous | 0.0001-1.0 | Yes |
| mlp_max_iter | MLPClassifier | discrete | 10-1000 | No |
| mlp_early_stop | MLPClassifier | categorical | True, False | No |
| mlp_alpha | MLPClassifier | continuous | 0.00001-1.0 | Yes |
| gb_loss | GradientBoostingClassifier | categorical | log_loss, exponential | No |
| gb_n_estimators | GradientBoostingClassifier | discrete | 10-200 | No |
| gb_learning_rate | GradientBoostingClassifier | continuous | 0.001-1.0 | Yes |
| gb_max_depth | GradientBoostingClassifier | discrete | 1-15 | No |
| gb_subsample | GradientBoostingClassifier | continuous | 0.05-1.0 | No |
| gb_min_samples_split | GradientBoostingClassifier | discrete | 2-20 | No |
| gb_min_samples_leaf | GradientBoostingClassifier | discrete | 1-20 | No |
| ab_n_estimators | AdaBoostClassifier | discrete | 10-250 | No |
| ab_learning_rate | AdaBoostClassifier | continuous | 0.001-2.0 | Yes |
| dt_max_depth | DecisionTreeClassifier | discrete | 1-25 | No |
| dt_min_samples_split | DecisionTreeClassifier | discrete | 2-20 | No |
| dt_min_samples_leaf | DecisionTreeClassifier | discrete | 1-20 | No |
| bagging_n_estimators | BaggingClassifier | discrete | 10-50 | No |
| bagging_max_samples | BaggingClassifier | continuous | 0.01-1.0 | No |
| bagging_max_features | BaggingClassifier | continuous | 0.01-1.0 | No |
| bagging_bootstrap | BaggingClassifier | categorical | True, False | No |
| et_max_depth | ExtraTreeClassifier | discrete | 1-150 | No |
| et_criterion | ExtraTreeClassifier | categorical | gini, entropy | No |
| et_splitter | ExtraTreeClassifier | categorical | random, best | No |
| et_min_samples_split | ExtraTreeClassifier | discrete | 2-20 | No |
| et_min_samples_leaf | ExtraTreeClassifier | discrete | 1-20 | No |
| pac_C | PassiveAggressiveClassifier | continuous | 0.01-10.0 | Yes |
| pac_max_iter | PassiveAggressiveClassifier | discrete | 1-3000 | Yes |
| pac_tol | PassiveAggressiveClassifier | continuous | 0.0001-0.1 | Yes |
| ridge_alpha | RidgeClassifier | continuous | 0.01-10.0 | Yes |
| ridge_solver | RidgeClassifier | categorical | auto, svd, cholesky, lsqr, sag, saga | No |
| lasso_alpha | Lasso | continuous | 0.0001-1.0 | Yes |
| en_alpha | ElasticNet | continuous | 0.0001-1.0 | Yes |
| en_l1_ratio | ElasticNet | continuous | 0.0-1.0 | No |
| knc_n_neighbors | KNeighborsClassifier | discrete | 1-100 | No |
| knc_weights | KNeighborsClassifier | categorical | uniform, distance | No |
| knc_algorithm | KNeighborsClassifier | categorical | ball_tree, kd_tree, auto | No |
| knc_leaf_size | KNeighborsClassifier | discrete | 10-50 | No |
| knc_p | KNeighborsClassifier | discrete | 1-3 | No |
| gnb_var_smoothing | GaussianNB | continuous | 1e-12-1e-6 | Yes |
| lda_solver | LinearDiscriminantAnalysis | categorical | svd, lsqr, eigen | No |
| lda_shrinkage | LinearDiscriminantAnalysis | continuous | 0.0-1.0 | No |
| sgd_loss | SGDClassifier | categorical | huber, squared_epsilon_insensitive, squared_error, squared_hinge, perceptron, hinge, log_loss, modified_huber, epsilon_insensitive | No |
| sgd_penalty | SGDClassifier | categorical | l2, l1, elasticnet | No |
| sgd_alpha | SGDClassifier | continuous | 1e-6-1e-1 | Yes |
| sgd_learning_rate | SGDClassifier | categorical | constant, optimal, invscaling, adaptive | No |
| sgd_l1_ratio | SGDClassifier | continuous | 0.0-1.0 | No |
| sgd_power_t | SGDClassifier | continuous | 0.0-50 | No |
| sgd_eta0 | SGDClassifier | continuous | 1e-7-1e-2 | Yes |
| ets_n_estimators | ExtraTreesClassifier | discrete | 10-200 | No |
| ets_criterion | ExtraTreesClassifier | categorical | gini, entropy | No |
| ets_max_features | ExtraTreesClassifier | categorical | sqrt, log2 | No |
| ets_min_samples_split | ExtraTreesClassifier | discrete | 2-20 | No |
| ets_min_samples_leaf | ExtraTreesClassifier | discrete | 1-20 | No |
| ets_min_weight_fraction_leaf | ExtraTreesClassifier | continuous | 0.0-0.5 | No |
| ets_max_leaf_nodes | ExtraTreesClassifier | discrete | 10-1000 | No |
| ets_min_impurity_decrease | ExtraTreesClassifier | continuous | 0.0-0.5 | No |
| ets_bootstrap | ExtraTreesClassifier | categorical | True, False | No |

Table A.3: All classifier hyperparameters, their type, valid ranges or choices, and by what classifier choice they are active.

## A.4 Regressor hyperparameters

| Hyperparameter | Active if model selected | Type | Valid Range/Choices | Sampled from log scale |
|---|---|---|---|---|
| ridge_alpha | Ridge | continuous | 0.01-10.0 | Yes |
| ridge_solver | Ridge | categorical | auto, svd, cholesky, lsqr, sag, saga | No |
| lasso_alpha | Lasso | continuous | 0.0001-1.0 | Yes |
| en_alpha | ElasticNet | continuous | 0.0001-1.0 | Yes |
| en_l1_ratio | ElasticNet | continuous | 0.0-1.0 | No |
| sgd_loss | SGDRegressor | categorical | huber, squared_epsilon_insensitive, squared_error, epsilon_insensitive | No |
| sgd_penalty | SGDRegressor | categorical | l2, l1, elasticnet | No |
| sgd_alpha | SGDRegressor | continuous | 1e-6-1e-1 | Yes |
| sgd_learning_rate | SGDRegressor | categorical | constant, optimal, invscaling, adaptive | No |
| sgd_l1_ratio | SGDRegressor | continuous | 0.0-1.0 | No |
| sgd_power_t | SGDRegressor | continuous | 0.0-50 | No |
| sgd_eta0 | SGDRegressor | continuous | 1e-7-1e-2 | Yes |
| ard_alpha_1 | ARDRegression | continuous | 1e-6-1e-3 | Yes |
| ard_alpha_2 | ARDRegression | continuous | 1e-6-1e-3 | Yes |
| ard_lambda_1 | ARDRegression | continuous | 1e-6-1e-3 | Yes |
| ard_lambda_2 | ARDRegression | continuous | 1e-6-1e-3 | Yes |
| ard_threshold_lambda | ARDRegression | continuous | 1e4-1e5 | No |
| pac_C | PassiveAggressiveRegressor | continuous | 0.01-10.0 | Yes |
| pac_max_iter | PassiveAggressiveRegressor | discrete | 1-3000 | Yes |
| pac_tolerance | PassiveAggressiveRegressor | continuous | 1e-4-1e-1 | Yes |
| dt_max_depth | DecisionTreeRegressor | discrete | 1-25 | No |
| dt_min_samples_split | DecisionTreeRegressor | discrete | 2-20 | No |
| dt_min_samples_leaf | DecisionTreeRegressor | discrete | 1-20 | No |
| rf_n_estimators | RandomForestRegressor | discrete | 10-200 | No |
| rf_max_depth | RandomForestRegressor | discrete | 1-25 | No |
| rf_min_samples_split | RandomForestRegressor | discrete | 2-20 | No |
| rf_min_samples_leaf | RandomForestRegressor | discrete | 1-20 | No |
| rf_criterion | RandomForestRegressor | categorical | friedman_mse, absolute_error, poisson, squared_error | No |
| rf_bootstrap | RandomForestRegressor | categorical | True, False | No |
| ada_n_estimators | AdaBoostRegressor | discrete | 10-250 | No |
| ada_learning_rate | AdaBoostRegressor | continuous | 0.001-2.0 | Yes |
| gb_loss | GradientBoostingRegressor | categorical | squared_error, absolute_error, huber, quantile | No |
| gbr_n_estimators | GradientBoostingRegressor | discrete | 10-200 | No |
| gbr_learning_rate | GradientBoostingRegressor | continuous | 0.001-1.0 | Yes |
| gbr_max_depth | GradientBoostingRegressor | discrete | 1-15 | No |
| gb_subsample | GradientBoostingRegressor | continuous | 0.05-1.0 | No |
| gbr_min_samples_split | GradientBoostingRegressor | discrete | 2-20 | No |
| gbr_min_samples_leaf | GradientBoostingRegressor | discrete | 1-20 | No |
| bagging_n_estimators | BaggingRegressor | discrete | 10-50 | No |
| bagging_max_samples | BaggingRegressor | continuous | 0.01-1.0 | No |
| bagging_max_features | BaggingRegressor | continuous | 0.01-1.0 | No |
| bagging_bootstrap | BaggingRegressor | categorical | True, False | No |
| etr_max_depth | ExtraTreeRegressor | discrete | 1-150 | No |
| etr_criterion | ExtraTreeRegressor | categorical | squared_error, poisson, absolute_error, friedman_mse | No |
| etr_min_samples_split | ExtraTreeRegressor | discrete | 2-20 | No |
| etr_min_samples_leaf | ExtraTreeRegressor | discrete | 1-20 | No |
| etr_splitter | ExtraTreeRegressor | categorical | best, random | No |
| knr_n_neighbors | KNeighborsRegressor | discrete | 1-100 | No |
| knr_weights | KNeighborsRegressor | categorical | uniform, distance | No |
| knr_algorithm | KNeighborsRegressor | categorical | auto, ball_tree, kd_tree | No |
| knr_leaf_size | KNeighborsRegressor | discrete | 10-50 | No |
| knr_p | KNeighborsRegressor | discrete | 1-3 | No |
| mlp_hidden_layer_size | MLPRegressor | discrete | 10-1000 | No |
| mlp_activation | MLPRegressor | categorical | identity, logistic, tanh, relu | No |
| mlp_solver | MLPRegressor | categorical | lbfgs, sgd, adam | No |
| mlp_lr_init | MLPRegressor | continuous | 0.0001-1.0 | Yes |
| mlp_max_iter | MLPRegressor | discrete | 10-1000 | No |
| mlp_early_stop | MLPRegressor | categorical | True, False | No |
| mlp_alpha | MLPRegressor | continuous | 1e-5-1.0 | Yes |
| etrs_n_estimators | ExtraTreesRegressor | discrete | 10-200 | No |
| etrs_criterion | ExtraTreesRegressor | categorical | squared_error, absolute_error, poisson | No |
| etrs_max_features | ExtraTreesRegressor | categorical | sqrt, log2 | No |
| etrs_min_samples_split | ExtraTreesRegressor | discrete | 2-20 | No |
| etrs_min_samples_leaf | ExtraTreesRegressor | discrete | 1-20 | No |
| etrs_min_weight_fraction_leaf | ExtraTreesRegressor | continuous | 0.0-0.5 | No |
| etrs_max_leaf_nodes | ExtraTreesRegressor | discrete | 10-1000 | No |
| etrs_min_impurity_decrease | ExtraTreesRegressor | continuous | 0.0-0.5 | No |
| etrs_bootstrap | ExtraTreesRegressor | categorical | True, False | No |

Table A.4: All regressor hyperparameters, their type, valid ranges or choices, and by what classifier choice they are active.

## A.5 Encoder choices & hyperparameters

The included encoders were the OrdinalEncoder and OneHotEncoder. Both encoders are not configurable, and therefore do not have associated hyperparameters.

## A.6 Imputer choices & hyperparameters

SimpleImputer, IterativeImputer, and KNNImputer were incorporated in the search space. Not selecting any imputer is not permitted, due to the inability of many base models to handle missing values. In case of categorical imputation for missing values in categorical variables, two strategies were deployed, described by the cat_imputer hyperparameter. All imputer hyperparameters can be referenced to in Table A.5.

| Hyperparameter | Active if imputer selected | Type | Valid range/choices | Sampled from log space |
|---|---|---|---|---|
| `simple_strategy` | SimpleImputer | categorical | {'mean', 'median', 'constant'} | No |
| `iterative_max_iter` | IterativeImputer | discrete | 10 to 100 | No |
| `iterative_imputation_order` | IterativeImputer | categorical | {'ascending', 'descending', 'roman', 'arabic'} | No |
| `knn_n_neighbors` | KNNImputer | discrete | 1 to 10 | No |
| `knn_weights` | KNNImputer | categorical | {'uniform', 'distance'} | No |
| `cat_imputer` | all | categorical | {'constant', 'most_frequent'} | No |

Table A.5: Imputer hyperparameters and their details.

## A.7 Feature selector choices & hyperparameters

The allowed choices for feature selectors are set to VarianceThreshold, SelectKBest, and SelectPercentile. Optionally, no feature selector can be used, not changing the selection of features to be used in the pipeline. Table A.6 shows the hyperparameters associated with the feature selectors.

| Hyperparameter | Active if selector selected | Type | Valid range/choices | Sampled from log space |
|---|---|---|---|---|
| `variance_threshold` | VarianceThreshold | continuous | 0.0 to 0.05 | No |
| `k_best` | SelectKBest | discrete | 3 to 50 | No |
| `score_func` | SelectKBest | categorical | {'f', 'mutual_info'} | No |
| `percentile` | SelectPercentile | discrete | 10 to 100 | No |
| `score_func_per` | SelectPercentile | categorical | {'f', 'mutual_info'} | No |

Table A.6: Feature Selector hyperparameters and their details.

## A.8 Scaler choices & hyperparameters

The available scalers, many of which are limitly configurable with hyperparameters, are StandardScaler, MinMaxScaler, MaxAbsScaler, RobustScaler, Normalizer, QuantileTransformer, and PowerTransformer. Additionaly, not selecting any scaler is an option. Note that for some classifiers and regressors, the presence of a scaler is required to be compatible with many datasets. In these cases, forbidden clauses were utilized to prevent configurations with incompatible combinations to be sampled. Table A.7 shows the hyperparameters associated with the scalers.

| Hyperparameter | Active if scaler selected | Type | Valid range/choices | Sampled from log space |
|---|---|---|---|---|
| `quantile_transformer_n_quantiles` | QuantileTransformer | discrete | 10 to 1000 | No |
| `quantile_transformer_output_distribution` | QuantileTransformer | categorical | {`'uniform', 'normal'`} | No |

Table A.7: Scaler hyperparameters and their details

## A.9 Dimensionality reduction choices & hyperparameters

Two dimensionality reducers were included in the search space, PCA and FastICA. Not selecting a dimensionality algorithm was an additional choice. Table A.8 shows the hyperparameters associated with the dimensionality reducers.

| Hyperparameter | Active if reducer selected | Type | Valid range/choices | Sampled from log space |
|---|---|---|---|---|
| `num_components` | PCA, FastICA | discrete | 5 to 50 | No |
| `pca_whiten` | PCA | categorical | {`True, False`} | No |
| `fastica_algorithm` | FastICA | categorical | {`'parallel', 'deflation'`} | No |
| `fastica_max_iter` | FastICA | discrete | 50 to 100 | No |
| `fastica_fun` | FastICA | categorical | {`'logcosh', 'exp', 'cube'`} | No |

Table A.8: Dimensionality reducer hyperparameters and their details

# Appendix B

# Datasets

This appendix contains additional information about the datasets used in this thesis.

## B.1 Classification datasets

As elaborated on in Chapter 5, 49 classification datasets were selected, 48 of which from the OpenML-CC18 benchmark. These datasets and their specifications are listed in Table B.1.

| **ID** | **Name** | **Instances** | **Features** | **Classes** |
|---|---|---|---|---|
| 3 | kr-vs-kp | 3196 | 37 | 2 |
| 6 | letter | 20000 | 17 | 26 |
| 11 | balance-scale | 625 | 5 | 3 |
| 15 | breast-w | 699 | 10 | 2 |
| 18 | mfeat-morphological | 2000 | 7 | 10 |
| 22 | mfeat-zernike | 2000 | 48 | 10 |
| 23 | cmc | 1473 | 10 | 3 |
| 29 | credit-approval | 690 | 16 | 2 |
| 31 | credit-g | 1000 | 21 | 2 |
| 32 | pendigits | 10992 | 17 | 10 |
| 37 | diabetes | 768 | 9 | 2 |
| 50 | tic-tac-toe | 958 | 10 | 2 |
| 54 | vehicle | 846 | 19 | 4 |
| 151 | electricity | 45312 | 9 | 2 |
| 182 | satimage | 6430 | 37 | 6 |
| 188 | eucalyptus | 736 | 20 | 5 |
| 38 | sick | 3772 | 30 | 2 |
| 307 | vowel | 990 | 13 | 11 |
| 469 | analcatdata_dmft | 797 | 5 | 6 |
| 1049 | pc4 | 1458 | 38 | 2 |
| 1050 | pc3 | 1563 | 38 | 2 |
| 1053 | jm1 | 10885 | 22 | 2 |
| 1063 | kc2 | 522 | 22 | 2 |
| 1067 | kc1 | 2109 | 22 | 2 |
| 1068 | pc1 | 1109 | 22 | 2 |
| 1590 | adult | 48842 | 15 | 2 |
| 1510 | wdbc | 569 | 31 | 2 |
| 1489 | phoneme | 5404 | 6 | 2 |
| 1494 | qsar-biodeg | 1055 | 42 | 2 |
| 1497 | wall-robot-navigation | 5456 | 25 | 4 |
| 1480 | ilpd | 583 | 11 | 2 |
| 1462 | banknote-authentication | 1372 | 5 | 2 |
| 1464 | blood-transfusion-service-center | 748 | 5 | 2 |
| 4534 | PhishingWebsites | 11055 | 31 | 2 |
| 6332 | cylinder-bands | 540 | 40 | 2 |
| 1461 | bank-marketing | 45211 | 17 | 2 |
| 4538 | GesturePhaseSegmentationProcessed | 9873 | 33 | 5 |
| 23381 | dresses-sales | 500 | 13 | 2 |
| 40499 | texture | 5500 | 41 | 11 |
| 40668 | connect-4 | 67557 | 43 | 3 |
| 40982 | steel-plates-fault | 1941 | 28 | 7 |
| 40994 | climate-model-simulation-crashes | 540 | 21 | 2 |
| 40983 | wilt | 4839 | 6 | 2 |
| 40975 | car | 1728 | 7 | 4 |
| 40984 | segment | 2310 | 20 | 7 |
| 41027 | jungle_chess_2pcs_raw_endgame_complete | 44819 | 7 | 3 |
| 23517 | numerai28.6 | 96320 | 22 | 2 |
| 40701 | churn | 5000 | 21 | 2 |

Table B.1: Classification datasets from OpenML used in experiments. Column ID corresponds to the OpenML dataset ID.

## B.2 Regression datasets

For the experiments in this thesis, 16 regression datasets were selected from the AutoML regression benchmark [27]. These datasets are listed in Table A.2.

| **Dataset ID** | **Name** | **Instances** | **Features** |
|---|---|---|---|
| 41021 | Moneyball | 1232 | 15 |
| 42225 | diamonds | 53940 | 10 |
| 42728 | Airlines_DepDelay_10M | 10000000 | 10 |
| 550 | quake | 2178 | 4 |
| 546 | sensory | 576 | 12 |
| 541 | socmob | 1156 | 6 |
| 507 | space_ga | 3107 | 7 |
| 287 | wine_quality | 6497 | 12 |
| 41540 | black_friday | 166821 | 10 |
| 42688 | Brazilian_houses | 10692 | 13 |
| 42727 | colleges | 7063 | 48 |
| 42729 | nyc-taxi-green-dec-2016 | 581835 | 19 |
| 42726 | abalone | 4177 | 9 |
| 201 | pol | 15000 | 49 |
| 531 | boston | 506 | 14 |
| 574 | house_16H | 22784 | 17 |

Table B.2: Specifications of OpenML regression datasets used in experiments.

## B.3 Datasets for 10CV and altering validation sizes

| **ID** | **Name** | **Instances** | **Features** | **Classes** |
|---|---|---|---|---|
| 151 | electricity | 45312 | 9 | 2 |
| 1590 | adult | 48842 | 15 | 2 |
| 1461 | bank-marketing | 45211 | 17 | 2 |
| 23517 | numerai28.6 | 96320 | 22 | 2 |

Table B.3: Binary classification datasets used to investigate 10CV and altering the validation size. All binary datasets with more than 20,000 sample were selected from OpenML-CC18.

# Appendix C

# Software packages and versions

The following software packages and versions were used in the implementation of this work.

| **Package** | **Version** |
|---|---|
| Python | 3.10.12 |
| ConfigSpace | 0.7.1 |
| dask | 2024.4.1 |
| dask-expr | 1.0.11 |
| dask-jobqueue | 0.8.5 |
| numpy | 1.26.4 |
| openml | 0.14.2 |
| pandas | 2.2.2 |
| pyarrow | 15.0.2 |
| scikit-learn | 1.4.2 |
| scipy | 1.13.0 |
| smac | 2.0.2 |

Table C.1: Software packages and versions used in the implementation of the experiments in this thesis.