A Similarity Based Meta-Learning Approach to Building Pipeline Portfolios for Automated Machine Learning

Wessels, Wichert M.

Award date:
2023

A Similarity Based Meta-Learning Approach to Building Pipeline Portfolios for Automated Machine Learning

Wichert, Wessels
ID: 1249096
Artificial Intelligence and Data Engineering Lab
Eindhoven University of Technology
`w.m.wessels@student.tue.nl`

August 27, 2023

Abstract

The design of a machine learning system is a comprehensive task and requires in-depth knowledge of models and understanding of the task at hand. Automated machine learning (AutoML) aims to automate this design process by objectively searching for the optimal pipeline. However, the search space of possible pipelines is infinite, and finding a well performing pipeline for every new dataset is very costly. To overcome this issue, the optimization procedure can be warm-started by using information of previous runs. The most prominent warm-starting technique to date uses pipeline portfolios for this task. This research focuses on dynamic portfolios, combined with clustering. Dynamic portfolio methodologies exploit meta-learning to provide a portfolio that is supposed to perform well on a given dataset. Previous dynamic portfolio approaches propose a portfolio by learning a relation between meta-features and the suggested portfolio. However, variations in these methods could lead to poor performing portfolios. The goal of this study is to improve upon these methods by proposing portfolios with a high intragroup performance across similar datasets. Additionally, this research aims to exploit these portfolios in an AutoML system by warm-starting the optimization procedure by first evaluating the pipelines present in the proposed portfolio. We will show that we can improve performance and converge to optimal pipelines considerably faster.

1 Introduction

The large increase of machine learning practitioners, both in industry and in academia, has led to a growing demand for systems that can support developers in creating machine learning applications. These systems exist in many forms, such as software packages, but also in low coding tools which are popular in industry, for example Rapid Miner [33] and KNIME [6]. However, these systems require developers to manually configure hyper-parameters, a task dependent on expert knowledge, which limits accessibility of optimally tuned models to only a small group of people. In addition, machine learning systems encompass not only model development challenges, but also involve pre-processing, data transformations and the composition of these into a pipeline. Automated machine learning (AutoML) aims to automate the decision process of finding the optimal composition and hyper-parameter configuration of these pipelines.

Conventional AutoML systems initiate the search for optimal pipelines anew for each dataset. AutoML systems are typically subject to a set of constraints, such as time and resource restrictions. Therefore, only a subset of possible pipelines can be evaluated within these boundaries. The challenge lies in carefully selecting pipelines for evaluation, particularly when confronted with novel datasets. Identifying high-performing pipelines for a new dataset remains uncertain and may involve prolonged search times, potentially hindering conventional optimization techniques.

These limitations highlight the necessity of enhancing AutoML search procedures. Commencing an AutoML system with a pre-established collection of high performing pipelines, known as warm-starting, holds the potential to reduce the time needed to convergence to the optimal pipeline. Such a collection, referred to as a portfolio, can significantly reduce the time required for the search. However, identifying suitable pipelines for a specific dataset is challenging due to the substantial dissimilarity between datasets. This study introduces a methodology that recommends a portfolio tailored to a given dataset, with the composition of the suggested portfolio being guided by distinct dataset attributes.

Portfolio construction can be approached by executing numerous pipelines with diverse configurations on representative datasets, subsequently selecting the top-performing pipelines using a greedy approach. Research into portfolios within the AutoML context is split into static and dynamic portfolios. Static portfolios remain invariant across datasets, while dynamic portfolios adapt their composition to the specific dataset at hand. A popular method for representing dataset characteristics involves using meta-features, which encapsulate properties and relationships among the dataset's features.

This study centers around meta-learning oriented portfolios, which are further enhanced through the incorporation of groups. In this context, the formation of groups involves partitioning a collection of datasets within a training set using a clustering methodology. The argument for adopting clustering arises from the notion that establishing a direct relationship between meta-features and the portfolio could be susceptible to inaccuracies. Such inaccuracies, if present, have the potential to yield underperforming portfolios. By initially organizing the meta-features into groups to determine the optimal portfolio based on the intragroup performance of pipelines, the objective is to mitigate the impact of this variability. Consequently, the aim is to construct more resilient portfolios that can effectively initialize AutoML systems.

The application of portfolio-based warm-starting is a rather undiscovered research field. While previous approaches focused on static portfolios or error-sensitive meta-learning methods, this research proposed a novel meta-learning approach that leverages dataset clustering to construct portfolios tailored to each partition. The objective is to balance performance variations among pipelines across individual datasets by creating portfolios that include pipelines achieving high performance across all datasets within a single partition.

The outline of this paper is as follows. Chapter 2 covers our problem statement and states the ultimate goal of this study. Chapter 3 covers the literature study, focusing on the algorithm selection problem, it's evolution to the combined selection and hyper-parameter optimization problem, and clustering algorithms in the context of our study. Following, Chapter 4 thoroughly describes our methodology, including how we the approach can be used for inference. In Chapter 5, the experimental setup and details regarding the data is discussed. Finally, Chapter 6 covers the results of these experiments, followed by a discussion in Chapter 7 and a conclusion in Chapter 8.

2 Problem Statement

As mentioned in the previous chapter, the application of pipeline portfolios to enhance the effectiveness of the AutoML search process is a rather unexplored field of research. Nevertheless, research by Feurer et al. [19][18] shows very promising results in this direction. Feurer et al.'s initial strategy involved employing solely the top-performing pipeline from the closest datasets as the basis for portfolio construction. In their subsequent approach, they adopted a fixed portfolio strategy that depended on evaluating all datasets in a training set to determine pipeline effectiveness. In contrast, our proposed approach seeks to achieve a middle ground between these two methodologies. It entails grouping similar datasets into clusters, enabling us to establish generalizations within each cluster, while concurrently tailoring distinct portfolios to suit to the diverse dataset categories.

In Chapter 1, we briefly introduced the objective of automated machine learning, which is to search for the optimal pipeline configuration. The underlying challenge addressed by AutoML is commonly known as the Combined Algorithm and Hyper-Parameter Selection (CASH) problem [42]. More formally, we are interested in finding the optimal pipeline $\mathcal{M}_\lambda^*$, hyperparameterized by $\lambda \in \Lambda$. We will base our notation for the CASH problem on the paper by Feurer et al [18], and directly use their formula for empirically approximating the generalization error.

$$\widehat{GE}(\mathcal{M}_\lambda, D) = \frac{1}{n}\sum_{i=1}^{n} \mathcal{L}(\mathcal{M}_\lambda(x_i), y_i) \tag{1}$$

In practice, the focus is on two disjoint sets, a training and a test set D_{train} and D_{test}. AutoML systems use the training set to automatically search for the best performing pipeline, while D_{test} serves to evaluate the generalization performance of the selected pipeline. As such, AutoML systems search for the optimal pipeline $\mathcal{M}_\lambda^*$ as follows:

$$\mathcal{M}_\lambda^* = \arg\min_{\lambda \in \Lambda} \widehat{GE}(\mathcal{M}_\lambda, D_{train}) \tag{2}$$

The remainder of this section is structured as follows, Section 2.1 extends Equation 2 to cover resource constraints. Next, we will generalize the AutoML problem towards an AutoML system that works well on any given dataset in some training set in Section 2.2. Finally, Section 2.3 extends the AutoML problem to include partitioning of the dataset space.

2.1 Resource Constrained AutoML

Equation 2 does not yet include any constraints, limiting its usability to theoretical use cases only. In a typical practical situation, AutoML systems operate under resource constraints such as time, computing power and RAM. We denote the time it takes to evaluate a pipeline by t_λ and the total time budget by $\mathcal{T}$. The approach in this study also incorporates the calculation of meta-features prior to the search procedure. The time for the calculation of a collection of meta-features is defined as t_ϕ. The total remaining time budget then changes to $\mathcal{T} - t_\phi$. We will denote this remaining time budget by $\mathcal{T}_\phi$. Then, we can use Equation 2 to define our constrained minimization problem.

$$\begin{aligned} \mathcal{M}_\lambda^* = \arg\min_{\lambda \in \Lambda} & \widehat{GE}(\mathcal{M}_\lambda, D), \\ & s.t. \sum t_\lambda < \mathcal{T}_\phi. \end{aligned} \tag{3}$$

In practice, the actual time available to the AutoML system is slightly smaller than $\mathcal{T}_\phi$, as some time is reserved for post-processing of the evaluated pipelines. However, this time is negligible with respect to $\mathcal{T}_\phi$, so we omit this here.

2.2 Generalization of AutoML systems

As explained earlier, AutoML systems start anew for every dataset. The approach in this study proposes a similarity-based meta learning approach using portfolios, which is used to warm-start an AutoML system. We are interested in finding an approach that works for any dataset. To this end, we define the collection of all datasets as $\mathbb{D}$, where each $D \in \mathbb{D}$ is a dataset consisting of n input-output pairs (x_i, y_i), where $1 \leq i \leq n$. Ideally, we want to have an AutoML system S that works well in general on any $D \in \mathbb{D}$. An AutoML system is defined as:

$$S : D \mapsto \mathcal{M}_\lambda, \quad D \in \mathbb{D}. \tag{4}$$

Where $\mathcal{M}_\lambda$ is the solution to Equation 3. For this research, the access to datasets is limited. In Section 5.1, we will have a more in depth discussion about the datasets used in this research. For now, we assume that the collection of datasets that is used for this research is a representative sample of all possible datasets. We define this collection as $\mathcal{D}$ with $\mathcal{D} \subset \mathbb{D}$.

To evaluate the approach taken in this study, it is important to have a separate set of datasets $\mathcal{D}_{test}$, with $\mathcal{D}_{test} \subset \mathbb{D}$, to benchmark whether the approach generalizes to unseen datasets. To this end, we can separate $\mathcal{D}$ into a training set $\mathcal{D}_{train}$ and a test set $\mathcal{D}_{test}$, where $\mathcal{D}_{train} \cap \mathcal{D}_{test} = \emptyset$.

Now, we are able to formalize the minimization of the AutoML objective to a general setting for all datasets in our training set[1]:

$$\widehat{GE}_{\mathcal{D}_{train}}(S) = \frac{1}{|\mathcal{D}_{train}|} \sum_{D_t \in \mathcal{D}_{train}} \widehat{GE}(S(D_t), D_t). \tag{5}$$

2.3 Partition-based Generalization

Equation 5 shows a generic formula for the generalization of the AutoML problem. For the scope of this study, we are interested in rewriting this equation and to make it more specific, such that it holds for a subset of the available datasets. In mathematical terms, we want to find a partition $\mathcal{K}$ of $\mathcal{D}_{train}$, where $\mathcal{K}$ contains a finite set of subsets of $\mathcal{D}_{train}$. Each element of $\mathcal{K}$ is denoted by K with $|K| > 1$, else the partition is trivial. Additionally, we have the following properties:

$$\forall i \neq j : K_i \cap K_j = \emptyset$$

$$\bigcup_{i=1}^{|\mathcal{K}|} K_i = \mathcal{D}_{train}.$$

The goal of partitioning the space of available datasets $\mathcal{D}$, is to decrease the generalization error with reference to Equation 5. We have shortly touched upon the intuition behind this in the Chapter 1. Constructing a portfolio based on only its nearest dataset(s) and its best performing pipelines seems sensitive to error. First partitioning the dataset space and determining well performing pipelines for each partition aims to average out this variation. In the context of AutoML, we want to define S for a specific K. We will denote this by $S_K : D_k \mapsto \mathcal{M}_\lambda$, $D_k \in K$. Details on S_K will be provided in the next chapter. Using our definition for S_K and S, we can write the goal of this research as:

$$\widehat{GE}(S_K(D), D) \leq \widehat{GE}(S(D), D), \quad D \in \mathcal{D}. \tag{6}$$

Important to note is that the assignment of K is not made explicit here. In chapter 4, we will have a more thorough discussion about the assignment of K from D and the construction of portfolios from K. Furthermore, the utilization of Equation 6 will enable us to further formalize the specific methodologies necessary to achieve this objective. This research will focus on empirically showing the validity of Equation 6 by evaluating all $D \in \mathcal{D}_{test}$.

3 Related Work

This chapter starts with an introduction of the model selection problem, followed by its evolution to the Combined Algorithm and Hyper-Parameter Selection (CASH) problem. Subsequently, we discuss different approaches to the CASH problem, including methods within AutoML to effectively cope with this problem. Finally, the last section discusses popular clustering algorithms that are interesting in the scope of this study.

[1]Note that the formal setting on $\mathbb{D}$ is omitted.

3.1 Combined Selection and Hyper-Parameter Optimization

3.1.1 Model Selection

Selecting the best machine learning algorithm for a specific task and dataset is a problem that has been studied for several decades. We define the model selection problem as follows.

Let $\mathcal{H}$ be a set of candidate models, where each $h_\theta \in \mathcal{H}$ is a function from a set of possible input data $\mathcal{X}$ to a set of possible output values $\mathcal{Y}$. We can thus view $h_\theta \in \mathcal{H}$ as a model, parameterized by a set of hyper-parameters θ, where $\theta \in \Theta$. Let D be a dataset consisting of n input-output pairs $(x_i, y_i) \in \mathcal{X} \times \mathcal{Y}$, where $1 \leq i \leq n$. The goal of model selection is to select a model $h^*_\theta \in \mathcal{H}$ that best fits the data D in the sense that the model achieves the lowest generalization error on the dataset D.

Early approaches tackle the model selection problem by predicting beforehand what the performance of a machine learning algorithm will be [5]. One way is to focus on the runtime of models, which is especially relevant for larger datasets. In order to evaluate multiple models on larger datasets in a reasonable time, information about the runtime of the model is of crucial importance. Doan et al. [13] present such a method, where meta learning is used to fit a spline-based regression model to predict the runtime of machine learning algorithms. Similarly, Reif et al. [38] use a simple regression model to fit a model for predicting the runtime of a series of classifiers. Their work also includes a parameter optimization search procedure, where the optimal configuration of hyper-parameters is given per classifier with a low runtime as objective. Additionally, Reif et al. [39] propose a meta-learning method to optimize the parameters of a given classifier by providing starting points for an evolutionary algorithm. Finding good configurations of hyper-parameters is also an active research field, and will be elaborated on later in this chapter.

3.1.2 Model Selection with Ensembles

The papers mentioned until now focus solely on the effectiveness of a single machine learning model. An alternative approach to the model selection problem is to construct ensembles of models to increase the efficacy of the predictive model. Caruana et al. [9] successfully implemented an ensemble constructed on a large library of models with different configurations using forward step-wise selection. Later work focuses on improving the selection procedure by pruning the number of available models for the ensemble selection, as well as scaling the model predictions on a canonical scale [8].

3.1.3 Pipeline Selection

Although the selection of algorithms is widely researched, these approaches merely focus on the model itself. It remains to pre-process and/or transform the data accordingly for the selected model. The approaches discussed so far only focus on the model and leave the remaining steps to the developer. Selecting the right combination of pre-processors and data transformers without automation is a tedious task and requires in-depth knowledge of the problem at hand and experience from the developer. Automated machine learning aims to include all these steps in the selection procedure with the end goal of optimizing the performance of the entire pipeline. As such, the problem shifts from model selection to the Combined Algorithm and Hyper-Parameter Selection(CASH) problem [42]. With the introduction of pre-processors and data transformers, the amount of possible combinations, including all the configurations of hyper-parameters, increases tremendously. As such, the need arises for tailored techniques that can effectively navigate this large search space.

Escalante et al. [15] attempt to solve the CASH problem by finding the optimal composition of pre-processors, feature selectors and classification models using particle swarm optimization [27]. Bürger et al. [7] provide a similar approach, but use evolutionary optimization to find optimal pipelines. Later work by Escalante et al. [14] adds ensembles to their previously discussed paper. While searching for the optimal pipeline, well performing compositions are stored and used to construct intermediate pipeline ensembles. Sun et al. [41] also propose a framework for finding the optimal pipeline by using particle swarm optimization. Rather than outputting a static pipeline, their approach outputs a DAG with multiple well performing operators that are found to be capable to solve the classification problem.

3.1.4 AutoML Frameworks

With the increasing popularity of the AutoML research community in the past decade, a range of practical AutoML frameworks were created and most frameworks are made available to the public to use. The first AutoML framework is Auto-WEKA [42]. Their approach uses bayesian optimization to select a suitable pipeline for the dataset at hand. The framework selects pipeline components from the WEKA library [25]. Olson et al. proposed TPOT [34], where pipelines are constructed using genetic programming (GP) trees, with each component of the tree being a GP primitive. For their pipeline optimization, 100 trees are generated, where the best performing 20 trees are selected and copied 5 times. Following, the trees mutate and this procedure is repeated until termination of the system. Later work on TPOT includes the use of successive halving to increase the feasibility of the tool for larger datasets [35]. Another tool is developed by Ledell et al. [28], and is named H2O AutoML. Their work is based on the H2O framework, which is a scalable machine learning framework designed to handle large datasets. H2O AutoML uses a combination of random search and stacked ensembles, as opposed to bayesian optimization and evolutionary algorithms, which are generally a more popular choice for the search procedure within the AutoML research community.

Feurer et al. [19] propose the Auto-Sklearn framework. Their approach uses pipeline components from the sklearn library and uses bayesian optimization to find the optimal pipeline. Additionally, they use a meta-learning approach to warm-start the optimization. In the next section, where portfolios are discussed, a more in depth description of this approach will be given. Moreover, Auto-Sklearn 2.0 will be discussed [18], which is related to the approach proposed in this research.

The final AutoML framework that is discussed in this section is GAMA, developed by Gijsbers et al. [22]. The overarching goal of this framework is to make it as accessible as possible for AutoML researchers to change certain parts of the system. GAMA distinguishes itself from the previously discussed frameworks by its modularity and extensibility. More specifically, the framework consists of components (e.g. search algorithm module), where each component can easily be replaced. GAMA is most related to auto-sklearn and TPOT, whereas GAMA uses pipeline components from the sklearn library while searching for the optimal pipeline. Additionally, GAMA uses genetic programming, which, as discussed earlier, is also used in the TPOT framework.

AutoML systems typically operate under certain constraints, such as time and memory. The previously discussed tools offer the possibility to select well-performing pipelines, but start searching from scratch for every new dataset. Especially for larger datasets, or harder datasets, the AutoML system may fail to select a well-performing pipeline within the boundaries of these constraints. Consequently, this highlights the necessity to enhance the search procedure's efficiency [24], as discussed in the subsequent section.

3.2 Automating the Selection of Starting Pipelines

In this section, we will first shortly discuss the origin of portfolios, followed by a discussion of static and dynamic defaults. Finally, research in the field of static and dynamic portfolios in the context of AutoML is discussed.

A portfolio is a set of high-performing and complementary machine learning pipelines that perform well on a set of datasets. Here, the set of datasets should be sufficiently large and representative. More specifically, the set of datasets should be a representative set of all datasets such that we can assume the portfolio works well on *any* dataset. In Chapter 4, a more formal notion of portfolios will be provided.

Portfolio approaches have been used before in the context of search problems. An early approach by Gomes et al. [23] uses portfolios as a solution to hard combinatorial problems. They mentioned that the runtime of different algorithms can vary drastically, and using all computational resources on one algorithm is inefficient. As an alternative, they use algorithm portfolios to allocate resources to different algorithms.

3.2.1 Defaults

In software packages, there is typically a default configuration of hyper-parameters when using a machine learning model. The default configuration of hyper-parameters is tuned by the developers of these packages such that the model is expected to perform well on most datasets. Several approaches to determine one or more alternative default configurations have been reported in literature. Mantovani et al. [31] proposed a method to find a set of default configurations for SVMs. However, to extend this method to AutoML, a more sophisticated set of defaults is required, as we are interested in finding the best possible configuration of pre-processors, data transformers and models, as opposed to only the configuration of model parameters. In AutoML systems, the set of possible components is usually almost the entire library of pre-processors, data transformers and machine learning models in a package like sklearn [36] or WEKA [25].

An approach to find a more general set of default configurations is proposed by Pfisterer et al. [37]. Rather than using the given defaults from a software package, they perform a parallel search on a small set of well performing default hyper-parameter configurations, which are learned by considering a larger set of configurations on a collection of datasets. As opposed to the previously discussed method by Mantovani et al., this approach considers a wide range of machine learning models.

The paper by Pfisterer et al. uses a static approach, whereas the hyper-parameter configurations are pre-learned and the collection of hyper-parameter is the same for every dataset. Contrary to this static configuration, Gijsbers et al. [21] propose a dynamic approach to the problem of finding optimal hyper-parameters. In their research, they learned symbolic configurations as formulas of dataset properties, or meta-features. As such, for every new dataset, the meta-features map to the corresponding optimal hyper-parameter configuration according to their approach.

3.2.2 Portfolios

In Section 3.1, Auto-Sklearn was shortly introduced. Both the first approach of Auto-Sklearn and Auto-Sklearn 2.0, are important in the context of this study, as the approaches closely relate to the methodology discussed in our research. The first version of Auto-Sklearn is proposed by Feurer et al. [19]. In this study, they introduce the Auto-Sklearn framework, including a meta-learning approach to warm-start the optimization algorithm. Starting from a collection of datasets, the same set of pipelines is evaluated on every dataset and the best performing pipeline is stored on a per dataset basis. Additionally, meta-features are calculated for each of these datasets. This results in a meta-feature space where we have a collection of vectors for each dataset with its corresponding meta-features. Then, for a new dataset, the meta-features are calculated and mapped to the same vector space. Finally, the closest 25 datasets according to a particular distance metric are determined and the best performing pipeline per dataset is extracted and stored in a portfolio. This portfolio is then used to warm-start the optimization algorithm of the system.

The next version of the system is also introduced by Feurer et al. [18]. Opposed to the previous version of Auto-Sklearn, this version uses a static approach to construct the portfolio that is used for warm-starting the optimization algorithm. From a collection of datasets, they ran the first implementation of Auto-Sklearn on all of these datasets and added the best performing pipeline per dataset to a set of candidate pipelines. Next, the set of candidate pipelines is evaluated on all datasets. The result is a performance matrix, where every row is a dataset and every column is a pipeline from the set of candidate pipelines. Then, a portfolio is determined using a greedy approach, where the best performing P pipelines across all datasets are saved. Here, P is the portfolio size. The most notable difference between the first and the second version of Auto-Sklearn is that the first approach does take the dataset characteristics into account, and the second one follows a static approach while constructing the portfolio. Most of their motivation for using a static approach comes from the additional time complexity that computing meta-features adds to the optimization objective.

3.3 Clustering

In the previous section, the use case of clustering meta-features was discussed. To the best of our knowledge, the specific use case of applying clustering on meta-features has not been reported in literature. Nevertheless, we can consider the task of clustering meta-features as a regular clustering task. As clustering is a widely studied field, and many approaches to the clustering problem have been reported in literature, we will only discuss the most prominent clustering directions in research and discuss their use case in the context of this study.

K-means is a popular clustering algorithm, and is part of the centroid-based clustering sub-field. [30]. Improvements to K-means include better initial centroid placement [4] and kernelizing the objective function to allow for identification of non-linearly separable data [12]. Another clustering research area is density-based clustering. Popular algorithms in this area are DBSCAN [16] and OPTICS [3]. A favorable property of density-based clustering in the scope of our study is that outliers or noisy data points do not belong to a cluster. As such, separate portfolios can be learned for this outlier group. Furthermore, distribution-based clustering [11] involves defining clusters as statistical distributions. Nonetheless, when dealing with higher dimensional data, the task of estimating distribution parameters becomes increasingly challenging, leading to limitations in the effectiveness of these approaches. Lastly, hierarchical clustering employs iterative bottom-up or top-down strategies to gradually form or disband clusters [40][10]. However, hierarchical clustering techniques often yield suboptimal performance when faced with datasets containing outliers, as these data points are frequently treated as distinct clusters.

Using the right clustering algorithm for the task at hand cannot be determined beforehand. As such, the best clustering algorithm has to be decided by experimenting [17] and determining the required clustering algorithm properties for the problem at hand. We will further elaborate on the requirements in the next chapter.

In summary, model selection and the CASH problem have been extensively studied. Various methods address default configurations using static and meta learning techniques. The CASH problem has driven the development of AutoML tools for selecting optimal pipeline configurations. Efficiently exploring this space is challenging. Leading techniques employ portfolios to enhance search optimization and achieve state-of-the-art outcomes. Notably, there's a gap in literature regarding a clustering approach within meta learning and portfolios. This research aims to augment portfolio research within the AutoML domain by introducing an approach that leverages an efficient meta-learning method, resilient to resource constraints.

4 Portfolios with Similarity-based Meta-Learning

Chapter 2 formalized the high level objective of this research (Equation 6). In this section, we will introduce the methodology of this research. We start with an overview of the different steps of the approach in Section 4.1. Following, we will discuss the different components of the approach in more depth, including the interconnection between them.

4.1 Compositional Approach to Portfolios

The methodology in this research can be described as a sequential process. The process is depicted in Figure 1. Starting from a set of datasets, we calculate a set of meta-features. For every dataset, we store the meta-features in a vector ϕ. In practise, two different datasets will never have the same meta-feature vector. As such, this results in a collection of $|\mathcal{D}_{train}|$ vectors, which we will refer to as $\mathcal{D}_{meta}$. The next step is to partition $\mathcal{D}_{meta}$ into $\mathcal{K}_{meta}$ according to a specified similarity measure, using certain clustering algorithms. We will further formalize this similarity measure and the selected clustering algorithms in this chapter. Finally, we want to construct a set of candidate pipelines $\mathcal{C}$ and evaluate its performance on each dataset in $\mathcal{D}_{train}$. Note that the space of datasets is partitioned in $\mathcal{K}_{train}$ according to the similarity of meta-features. After evaluating $\mathcal{C}$ on $\mathcal{D}_{train}$, we can directly infer the performance of $\mathcal{C}$ on K, with $K \in \mathcal{K}_{train}$. As the per partition performance is known, we can construct our portfolio $\mathcal{P}$, by simply selecting the top performing $|\mathcal{P}|$ pipelines that perform best on $K \in \mathcal{K}_{train}$ according to a pre-defined evaluation metric. $|\mathcal{P}|$ is the size of the portfolio, which can be interpreted as a hyper-parameter and thus configured and optimized.

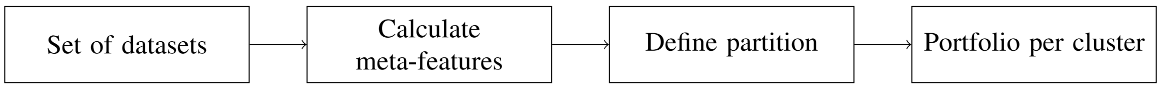

Figure 1: Portfolio Construction Process

4.2 Similarity-based Partitioning

We have touched upon the calculation of meta-features earlier. Here, we introduce a more formal notion of this calculation. To this end, we first define the goal of calculating meta-features. The goal of calculating meta-features is to construct a vector ϕ, where ϕ is a representation of a dataset D. We can write this calculation as a mapping $\mathcal{F} : D \mapsto \phi$.

In Section 2.3, it was shortly mentioned that we want to warm-start an AutoML system S for a specific partition K, such that Equation 6 holds. The first step in this process, is to find a finite partition of $\mathcal{D}_{train}$. This study partitions $\mathcal{D}_{train}$ using meta-features. We start by calculating the meta-features on all datasets in $\mathcal{D}_{train}$. This procedure is shown in Algorithm 1.

Algorithm 1 Construction of Meta Dataset

```
Input: $\mathcal{D}_{train}$, collection of datasets
Output: $\mathcal{D}_{meta}$, meta dataset
Initialize $\mathcal{D}_{meta}$
for $D \in \mathcal{D}_{train}$ do
    $\phi \leftarrow \mathcal{F}(D)$
    add $\phi$ to $\mathcal{D}_{meta}$
return $\mathcal{D}_{meta}$
```

Every element of $\mathcal{D}_{meta}$ is a representation of a dataset $D \in \mathcal{D}_{train}$. As meta-features represent a dataset by its characteristics, similar datasets are represented in a similar manner. Consequently, we are interested in partitioning $\mathcal{D}_{train}$ using this similarity.

In Chapter 3.3, we explored popular clustering algorithms. For this study, we opted for centroid-based and density-based clustering. We'll start by explaining centroid-based clustering in our context, followed by density-based clustering.

In centroid-based clustering, centroids are used to represent clusters. A centroid is typically defined as a representative point within a cluster that minimizes the distance to other data points within the same cluster. For this study, we use K-means as centroid-based clustering algorithm. It partitions a dataset into K clusters by iteratively assigning data points to the nearest centroid and updating the centroid positions based on the newly assigned points [30]. For K-means to work well and converge to the optimal solution, it requires the data to be convex and isotropic, which might not hold for all datasets in $\mathcal{D}_{meta}$. To address these limitations, we normalize $\mathcal{D}_{meta}$ to improve its uniformity and employ the kernelized K-means objective [12] to allow for the formation of non-convex clusters. Then, we define our partition as $\mathcal{K}$, where $|\mathcal{K}|$ is a parameter reflecting the number of centroids.

The alternative to centroid-based clustering that we will consider in this study is density-based clustering. Instead of relying on explicit centroid computations or distance measurements, density-based clustering methods focus on the local density of data points. The key idea is to identify regions of high data concentration, referred to as dense regions, and separate them from regions with lower density, which we consider noise or outliers. This is favourable for our study, as we can learn a separate portfolio for the low density group.

In this study, we will use OPTICS [3] as our density-based clustering algorithm. OPTICS extends the concept of density-based clustering by introducing the notion of *reachability distance*. It constructs an ordering of data points based on their reachability distances, allowing for the detection of clusters at multiple scales.

Using OPTICS, we can define our partition as $\mathcal{K}$, where $|\mathcal{K}|$ is determined based on the amount of dense regions contained in $\mathcal{D}_{meta}$. When using OPTICS, we have $K_1 \in \mathcal{K}$ containing the noise or outliers of $\mathcal{D}_{meta}$. Here,

$$\forall i \in \{2, .., |\mathcal{K}_{meta}|\} : K_1 \cap K_i = \emptyset$$

Note that for the sake of this study, we restrict $|\mathcal{K}_{meta}|$ to $|\mathcal{K}_{meta}| > 1$, otherwise the partition and Equation 6 are trivial.

Independent of the chosen clustering algorithm, we can write the mapping of ϕ to an element of $\mathcal{K}_{meta}$ as $\mathcal{Q}$, where $\mathcal{Q}$ is defined as $\mathcal{Q} : \phi \mapsto K$.

4.3 Portfolio Construction

After partitioning $\mathcal{D}_{meta}$ into $\mathcal{K}_{meta}$, we need one additional step to be able to construct our portfolios. A portfolio is a set of pipelines, where each pipeline is denoted by $\mathcal{M}_{\lambda}$. Portfolios are constructed by considering a set of candidate pipelines $\mathcal{C}$, and selecting the best performing n pipelines evaluated on a set of datasets. Self-evidently, best performing pipelines is directly related to a pre-defined evaluation metric. In this study, we will use a greedy approach, where the selection of the top n best performing pipelines is determined based on the evaluation metric. Note that complementarity of the portfolios is not enforced by the greedy selection, but depends on the diversity of the pipelines contained in $\mathcal{C}$. $\mathcal{C}$ is constructed by running GAMA without warm-starting on a sample of datasets from the training set(repeated for every task). Then, the top five highest performing pipelines per dataset are selected and added to $\mathcal{C}$. Algorithm 2 shows the construction of a portfolio in pseudo code. For convenience of notation, we will define an arbitrary[2] scorer function for the performance of a pipeline c on a dataset d as:

$$score : D \times C \mapsto \mathbb{R}$$

Algorithm 2 Portfolio Building

```
Input: Set of candidate pipelines $\mathcal{C}$, a set of datasets $D$, portfolio size $p$. $|\mathcal{C}| \geq p$
Output: Portfolio of size $p$
Initialize $\mathcal{P} = \emptyset$
while $|\mathcal{P}| < p$ do
    $cBest = \arg\min_{c \in \mathcal{C}}(\frac{1}{|D|} \sum_{d \in D} score(d, c))$
    $\mathcal{P} = \mathcal{P} \cup \{cBest\}$
    $\mathcal{C} = \mathcal{C} \setminus \{cBest\}$
end while
return $\mathcal{P}$
```

This greedy approach is only dependent on a *performance matrix*. We will define a performance matrix as a matrix with dimensions $|\mathcal{D}_{train}|$x$|\mathcal{C}|$, where each element of the performance matrix is the outcome of our $score$ function. The domain of the performance matrix depends on the prediction task.

As mentioned earlier, this study focuses on three prediction tasks, binary classification, multi-class classification and regression. For binary classification we use the area under the Receiver Operating Characteristic curve (ROC AUC) [32]. The main reason for choosing the ROC AUC, is that the metric is robust against class imbalance. As we evaluate models on a wide range of datasets, we need a versatile, robust evaluation metric that is eligible and representative on different types of datasets. The domain of the ROC AUC is $[0, 1]$, with 1 being a perfect classifier.

Multi-class classification problems also require a versatile evaluation metric, as we have both the problem of class imbalance, as well as needing an evaluation metric suited for different amount of classes. As such, we choose to use the Log Loss. This metric internally works based on probability estimates, where a probability is calculated for every class. The domain of the Log Loss is $[0, \infty)$, where lower is better (0 is a perfect classifier).

Finally, we have regression as a prediction task. As we have datasets with a continuous predictor on a scale specific to that dataset, we require an evaluation metric that accommodates the scale of continuous predictors and avoids overfitting to outliers. As such, the Root Mean Squared Error (RMSE) is chosen as an evaluation metric due to its ability to generalize well and provide a suitable scale for the continuous predictor. The domain of the RMSE is $[0, \infty)$, with lower values better performance (0 is a perfect regressor).

In the previous section, we have shown how to partition $\mathcal{D}_{meta}$ into $\mathcal{K}_{meta}$. After constructing a performance matrix of $|\mathcal{D}_{train}|$x$|\mathcal{C}|$ for a given task, we can apply the same partitioning of $\mathcal{D}_{train}$ into $\mathcal{K}_{train}$ to partition the performance matrix. To achieve this, the rows of $\mathcal{D}_{meta}$ are indexed by a dataset ID, which can be used to directly map a vector with meta-features to its corresponding dataset ID in the performance matrix. In Table 1 and Table 2, an example is shown, where the datasets with ID 2 and 5 are partitioned together based on their meta-features. Consequently, this partition is directly transferable to the performance matrix. As such, we can select only the items in the performance matrix that have the same ID as the items in the partition. Repeating this process for all elements in $\mathcal{K}_{train}$ results in $|\mathcal{K}_{train}|$ performance matrices. Then, for $K \in \mathcal{K}_{train}$, we can greedily construct portfolios and retrieve a per partition portfolio as result. We will further refer to these per partition portfolios as $\mathcal{P}_K$. Note that if we consider two portfolios $\mathcal{P}_{K_1}$ and $\mathcal{P}_{K_2}$, $K_1, K_2 \in \mathcal{K}_{train}$, the two are not enforced to contain only different pipelines. More specifically, this means that $\mathcal{P}_{K_1} \cap \mathcal{P}_{K_2} \neq \emptyset$ is allowed.

4.4 Inference

This chapter has focused on explaining the methodology of the construction and training of portfolios using meta learning. For every $D \in \mathcal{D}_{train}$, the meta-features are calculated and $\mathcal{C}$ is evaluated. Due to the usage of a dataset ID,

[2]We have three tasks with different scorers, and thus different domains. This will be elaborated on later in this section.

Dataset ID	**Size**	**...**	**No. of Features**
2	300	...	10
5	320	...	11
...	...	...	...
100	40000	...	100

Table 1: Dataset Information

Dataset ID	**Pipeline 1**	**...**	**Pipeline $\lvert\mathcal{C}\rvert$**
2	0.74	...	0.93
5	0.92	...	0.78
...	...	...	...
100	0.94	...	0.97

Table 2: Pipeline Performance

we can use this ID as a primary key and directly create a relation between $\mathcal{D}_{meta}$ and the performance matrix. Then, we can greedily construct portfolios by selecting the top n best performing pipelines per partition.

This compositional approach can be viewed as a standard machine learning training procedure, where the portfolios have been constructed from a training set of datasets. Next, we will explain how this approach can be used on a novel dataset D_{new}, where $D_{new} \notin \mathcal{D}_{train}$. We assumed that $\mathcal{D}$ is a representative sample of $\mathbb{D}$, as the dataset repository used in this study provides both a great variety and a vast amount of datasets [43]. As a result, we would expect that our approach still works on D_{new}.

To test the approach proposed in this study, the following phases have to be executed. First, we calculate the meta-features $\phi \leftarrow \mathcal{F}(D)$. Then, using the function Q, we can determine cluster K via the assignment of $K \leftarrow \mathcal{Q}(\phi)$. Finally, we can return portfolio $\mathcal{P}_K$. This portfolio can then be used to warm-start an AutoML system, which is the main purpose of portfolios in this research. The process is depicted in Figure 2.

$$\begin{tikzcd}
\boxed{\begin{array}{c}\text{Calculate}\\ \phi \leftarrow \mathcal{F}(D)\end{array}} \arrow[r] & \boxed{\begin{array}{c}\text{Decide partition}\\ K \leftarrow \mathcal{Q}(\phi)\end{array}} \arrow[r] & \boxed{\begin{array}{c}\text{Assign Portfolio}\\ \mathcal{P}_K \leftarrow K\end{array}} \arrow[r] & \boxed{\text{Warm-start } S_K(D)}
\end{tikzcd}$$

Figure 2: Portfolio Inference and Warm-Starting

In this research, we provide our portfolios as starting candidates to an asynchronous evolutionary algorithm. Without warm-starting, the algorithm initializes from random pipelines. We expect that warm-starting with a portfolio containing pipelines specific to a partition K results in both faster convergence and better performance.

Note that this is not the only use case of these portfolios, as the portfolios can also be solely evaluated as potential high performing candidates for a machine learning task at hand. In this case, we select the best performing pipeline by slightly modifying Equation 2. Rather than searching for the optimal $\lambda \in \Lambda$, we are instead interested in finding the optimal pipeline in a portfolio $\mathcal{P}$:

$$\mathcal{M}_\lambda^{\mathcal{P}} = \operatorname*{argmin}_{\mathcal{M}_\lambda \in \mathcal{P}} \widehat{GE}(\mathcal{M}_\lambda, D) \tag{7}$$

5 Experimental Design

The previous chapter focused on a thorough background discussion of the methodology of this study. The focal point was more towards the theoretical boundaries and capabilities of the approach. In practise, there are a lot of choices regarding the implementation that need to be carefully chosen and studied. The first choice is to decide on the application scope of the supervised machine learning problems. This study will limit its scope to three supervised learning categories: binary classification, multi-class classification and regression algorithms. As mentioned earlier, the implementation is built as an extension of the GAMA AutoML system, which is an AutoML tool built in Python with a focus on modularity and extendability. GAMA currently supports two search algorithms, asynchronous successive halving [29] and asynchronous evolutionary optimization. We will focus solely on the evolutionary algorithm in our experiments.

All of the experiments are executed on a high performance cluster. In this cluster, we have access to 32 CPU cores and 32 GB RAM. Furthermore, all of the pipelines are evaluated with a 15 minute timeout, after which we forcefully terminate the evaluation procedure. Note that we choose 15 minutes for the experiments outside of the scope of AutoML, and for the construction of the performance matrix. When running experiments for AutoML, we set the timeout time to 6 minutes, which is exactly $1/10$ of 60 minutes, being the maximum time we set for the AutoML experiments.

Ideally, we would want to set a timeout of up to 60 minutes for pipeline evaluation outside of the AutoML scope, but since we had to evaluate a large amount of pipelines, it was necessary to reduce this time to remain within the budget constraints of this research. However, if we study the impact of timing out pipelines after 15 minutes, we find that this timeout has little impact, if not found to be beneficial for our portfolios. Nearly all of the pipelines that often timed out, were not found to have a higher performance, meaning that we have no use of these pipelines outside of the context of AutoML. Additionally, within AutoML systems, we ideally want to construct a stable portfolio. This means that on average, the pipelines in the portfolio should perform well and finish without any kind of error (including timeout).

This chapter starts with a summary of the selected datasets in this study (Section 5.1). Next, Section 5.2 discusses the meta-features that are chosen. Finally, Section 5.3 provides a detailed overview of the experiments that were conducted.

5.1 Dataset Selection

This section clarifies the attempt of selecting datasets to construct $\mathcal{D}$ from $\mathbb{D}$. The method's applicability spans across three domains, necessitating the inclusion of datasets that contain binary classification targets, multi-class classification targets, and regression targets.

A substantial collection of datasets is available in the OpenML repository [43], which is an open source platform for sharing datasets, algorithms and experiments. A set of logical rules was applied on simple dataset characteristics to avoid including datasets that are either too small, or too large. The reason for excluding these is to avoid technical issues like excessive memory usage or simply taking too long for large datasets. Similarly, for small datasets we ran into issues with cross validation, not being able to represent every class in every fold. The selection rules are shown in Table 3 and Table 4.

Name of Feature	**Logical Expression**
Number of instances	> 250 AND < 500.000
Number of features	> 2 AND < 2.500
Number of classes	> 1
Size[3]	< 10.000.000

Table 3: Classification Selection Rules

Name of Feature	**Logical Expression**
Number of instances	> 20 AND < 500.000
Number of features	> 2 AND < 2.500
Number of classes	< 1
Size[4]	> 2000 AND < 10.000.000

Table 4: Regression Selection Rules

Before continuing our discussion on the dataset selection process, it is important to state the data that compromises $\mathcal{D}_{test}$. For this, the datasets from the AutoML benchmark are used [20]. These datasets are also available in the OpenML repository. In the available datasets, there are several datasets that are exact copies, or derivatives of the AutoML benchmark datasets. To prevent information leakage to the test set, we removed these datasets from our training set.

Additionally, there are also datasets in our training set that are exact copies, later versions or derivatives of each other. To prevent creating clusters of datasets that strongly reflect these similarities, we remove these from our set of datasets as

[4]The size is a dummy variable calculated by multiplying the number of instances by the number of features.

well. Some other datasets have been removed for dataset specific reasons, for example because they contain unstructured data like tweets. We will not fully cover these specifics in full detail in this section. For all the details, we refer to the codebase, which can be found in our repository[5]. For the test sets, we provide a detailed overview in Appendix B.

5.2 meta-feature Selection

The methodology of this study is designed for three supervised learning problems. Each of these learning fields have its own specifics in terms of dataset properties. We want to capture these properties as accurately as possible for each of the three learning fields. The meta-features include statistical-, information theoretical-, and landmarking features. For this, we selected a subset of meta-features based on the groups as defined in PyMFE [2].

We want to distinguish the way meta-features are calculated for the different learning fields. Most of the statistical meta-features include aggregations of column statistics, for example the maximum or minimum of a numerical column. Information theoretical meta-features capture between column relations. However, information theoretical meta-features also capture relations between columns and the target feature. This raises the issue that for classification and regression, we have different types of targets (discrete and continuous), meaning that we require different meta-features for these targets.

As PyMFE currently does not support the following features:

- Parallel computing of meta-features
- Land-marking features for regression
- Regression target features and features capturing the relation between features and the target

We could not directly use PyMFE and had to pivot to an own implementation. An overview of all the different meta-features per task is provided in Table 11 in the Appendix.

Finally, some meta-features are only defined on either categorical features or numerical features. As discussed in Chapter 4, the approach of this study is compositional, meaning that earlier steps influence the performance of downstream steps. The partitioning part requires our meta-feature vector to contain only numerical values. As such, missing values need to be imputed. The imputation strategy for each meta-feature has to be carefully chosen, as the imputed value directly influences the partition the dataset at hand is assigned to. For this, we first distinguished the columns that contain outliers and those that do not, where we used an interquartile range with a 1.5 whisker to filter outliers per column. Following, we decided whether or not these columns were flagged as outlier columns. Next, the columns with outliers are imputed with the median value of the respective column, and the columns without outliers with the mean. We choose the median for outlier columns, as some outlier columns contained values only skewed towards very small numbers or large numbers. The median is less sensitive to these particular cases.

5.3 Experimental Setup

In all of our experiments, we will refer to the results of the methodology presented in this paper as SBPort. For the execution of our methodology, we use the workflow presented in Figure 2, where the last step is only executed for the AutoML experiment.

5.3.1 Combined Cluster and Portfolio Selection

Clustering the dataset space based on meta-features is a comprehensive task, where we cannot decide on the exact configuration of the clustering algorithm beforehand. Therefore, the first step towards finding the optimal similarity-based portfolio strategy is executing a grid search on the configuration of the clustering algorithms, as well as varying the size of the portfolio.

First, we define α as the number of clusters in the Kernel K-means algorithm. For OPTICS, we will only use one configuration of hyper-parameters, as the tuning of the hyper-parameters of OPTICS are not as effective as changing α, and leads to almost identical clusters. Therefore, we simply choose one hyper-parameter configuration to be included in the grid. For convenience of notation, we denote this hyper-parameter configuration as $\mathcal{R}$.

[5] github.com/Wman1001/MasterThesis

We choose $\alpha \in \{3, 5, 8\}$ to test general larger clusters, as well as smaller clusters specific to a small partition. Similarly, for portfolio $\mathcal{P}$, we choose the sizes $|\mathcal{P}| \in \{4, 8, 16\}$ for our grid search. Then, we can define our grid as: $G = \{(a, b)|\ a \in \alpha \cup \{\mathcal{R}\} \text{ and } b \in |\mathcal{P}|\}$

As mentioned earlier, the datasets contained in the AutoML Benchmark [20] is used as a test suite for validating our approach. As we are first conducting a grid search, we split the datasets from the benchmark using a $33.33\%, 66.66\%$ validation test split, where we evaluate our approach on the validation set to decide on the best configuration of hyper-parameters. We will refer to these two sets as $\mathcal{D}_{val}$ and $\mathcal{D}_{test}$.

Our methodology serves two purposes. We can evaluate the portfolios outside of the context of AutoML, meaning that we use Equation 7 to determine the performance of the portfolio. However, in the case of using portfolios in the context of AutoML, we have different requirements than simply selecting the best performing pipeline out of the portfolio. More specifically, we are interested in the stability of the pipelines that are evaluated. Unstable portfolios lead to pipelines being terminated by the system due to for example excessive memory usage or running out of time. Therefore, for warm-starting the search procedure, we are interested in constructing a metric that does not only take into account the performance of the best performing pipeline. First, we define failCount as a counter of the amount of pipelines that were forcefully terminated because of an arbitrary error while evaluating a portfolio $\mathcal{P}$. Then, we can define our metric as:

$$\frac{\frac{1}{|\mathcal{P}|-\text{failCount}} \sum_{\mathcal{M}_\lambda \in \mathcal{P}} \widehat{GE}(\mathcal{M}_\lambda, D) + \frac{\text{failCount}}{|\mathcal{P}|}}{2}, \quad D \in \mathcal{D}_{val}.^{6} \tag{8}$$

Where we take the average of the pipelines that finished evaluating, add this to the ratio of pipelines that were forcefully terminated during execution with respect to the size of the portfolio, divided by two. This provides a trade-off between average performance and the stability of the portfolio.

From the results of our grid search, we select two portfolios for every learning task, the first one using Equation 8 for AutoML warm-starting, and the second portfolio using the metric defined by Equation 7 for detached portfolio evaluation. We will refer to these portfolios as stable portfolio and max portfolio, respectively.

5.3.2 Cluster Selection Evaluation

Directly evaluating the clusters is hard, as we do not have a metric to evaluate the performance of a cluster without testing the approach of this study as a whole. Nonetheless, we can evaluate the performance of the cluster by comparing the performance of the predicted cluster, to all other clusters. This provides an insight whether the predicted cluster, determined by the meta-features, also provides well performing pipelines for the dataset at hand. We will use the max policy to determine the performance of the portfolios per cluster.

5.3.3 Baselines

Static Portfolio

Prior portfolio research in the context of AutoML focused on static portfolios [18]. As such, we are interested in testing whether portfolios constructed using the approach in this study are beneficial when compared to static portfolios. For this purpose, we construct two static portfolios per task, one with the size of the portfolio as selected by the max policy and the other with the size of the portfolio as selected by the stable policy.

Alternatively, our max and stable portfolios are evaluated against the Auto-Sklearn 2 portfolios by Feurer et al. [18]. As their portfolios have a size of 32, we will select the first n pipelines in their portfolio, where n again is the size of the portfolio of the task at hand. They constructed portfolios only for classification problems, where the portfolio is constructed for both binary and multi-class classification problems. As such, we will compare both our binary classification portfolios and multi-class classification portfolios to the same Auto-Sklearn 2 portfolios.

One Nearest dataset

[6] In practise, we employ a two-step scaling approach for $\widehat{GE}(\mathcal{M}_\lambda, D)$: initially applying outlier scaling, and subsequently normalizing to constrain the values within the range of 0 to 1.

Our methodology builds upon the intuition that creating portfolios based on the most similar dataset in the training set seems sensitive to error. This experiment aims to validate this claim by implementing the one nearest dataset algorithm based on meta learning, and comparing the results to the max and stable portfolio. The one nearest dataset algorithm is shown in Algorithm 3

Algorithm 3 One Nearest dataset Portfolio Construction

```
Input: Set of candidate pipelines $\mathcal{C}$, set of training datasets $\mathcal{D}_{train}$, a new dataset $D_{new}$, portfolio size $p$
Output: Portfolio of size $p$
Initialize $\mathcal{P} = \emptyset$
$closest\_set = \arg\min_{D_m \in \mathcal{D}_{train}}(\|\mathcal{F}(D_{new}) - \mathcal{F}(D_m)\|)$
while $|\mathcal{P}| < p$ do
    $cBest = \arg\min_{c \in \mathcal{C}}(score(closest\_set, c))$
    $\mathcal{P} = \mathcal{P} \cup cBest$
    $\mathcal{C} = \mathcal{C} \setminus cBest$
end while
return $\mathcal{P}$
```

Here, the size of p is equal to the size of the portfolio we are comparing the one nearest dataset approach to. Additionally, we use the $score$ function that we defined prior to Algorithm 2 and the function $\mathcal{F}$ defined in Chapter 4 for the assignment of meta-features.

5.3.4 Portfolios in AutoML

Finally, we want to evaluate our portfolios in the context of AutoML. As mentioned earlier, we will use GAMA as AutoML system, where the focus for this research is on the evolutionary optimization algorithm. The algorithm defaults to random starting pipelines. In this experiment, we will use our stable portfolios to provide starting candidates that are expected to work well on the dataset at hand. To evaluate the difference in performance, we will run GAMA without warm-starting on $\mathcal{D}_{test}$ for 10 minutes and 60 minutes. Following, we will execute the same experiment, but with warm-starting. The deviations between these setups are assessed both by the pipeline evaluation over time, as well as the final performance after 10 and 60 minutes. For post-processing, we will use the single best pipeline.

6 Results

6.1 Results Combined Cluster and Portfolio Selection

For this experiment, we evaluated our different configurations on $\mathcal{D}_{val}$, which is a 33.33% holdout from the AutoML benchmark, where we selected only datasets using the same set of rules as listed in Table 3 and Table 4. First of all, we have evaluated OPTICS on $\mathcal{D}_{val}$. However, not all of the clusters were of the desired sizes and quality for all three tasks. This limitation was especially highlighted for the multi-class classification task, which resulted in rather poor performing configurations for OPTICS.

As described earlier, OPTICS uses the notion of reachability distance to construct clusters. For prediction, we extracted the per cluster maximum reachability distance and set it as a threshold for new data points. For every new data point, we calculated the distance to near data points to decide whether the data point falls within the boundaries of this threshold for one the clusters. If this was the case, we assigned the new data point to the same cluster as the nearest data point in the training set.

Table 5 shows the results of the grid search on binary classification datasets in our validation set for the optimal two portfolios, using the max policy and stable policy for evaluation. To determine the most suitable configuration using the max policy, we aggregated the results over the validation datasets and calculated the cumulative frequency of occurrences wherein the configuration was identified as best performing. The determination of the most suitable configuration for the stable policy involves the computation of the mean performance across all datasets contained within the validation set. The specific values for each dataset are derived utilizing the formulation outlined in Chapter 5. Important to note is that we selected the best performing portfolio for the stable policy only based on the results of the configurations with $p = 16$, as most of the smaller portfolios contain similar pipelines. This leads to little gain from

	$MaxPolicy$		$StablePolicy$	
DataSet Name	$\alpha = 8, p = 16$	$BestMax$	$\alpha = 5, p = 16$	$BestStable$
bank-marketing	0.779	0.779	0.942	0.997
Bioresponse	0.882	0.882	0.873	0.873
Click_prediction	0.698	0.705	0.997	0.999
PhisingWebsites	0.996	0.996	0.999	0.999
ozone-level-8hr	0.873	0.883	0.996	0.998
ada	0.917	0.918	0.990	0.999
pc4	0.949	0.949	0.996	0.999
MiniBoone	0.985	0.985	0.812	0.812
churn	0.920	0.922	0.998	0.999
gina	0.988	0.989	0.874	0.874

Table 5: Grid Search Results on Binary Classification tasks using a 33.33% holdout of the AutoML Benchmark. The values represent the maximum ROC AUC calculated across the portfolio using 10-fold cross-validation (1 is best). The best portfolio per dataset across the entire grid is shown for ease of reference.

	$MaxPolicy$		$StablePolicy$	
DataSet Name	$\alpha = 8, p = 16$	$BestMax$	$\alpha = 8, p = 16$	$BestStable$
cmc	0.924	0.902	0.246	0.046
okcupid-stem	0.570	0.569	0.099	0.001
eucalyptus	0.864	0.864	0.320	0.088
car	0.207	0.207	0.261	0.052
theorem-proving	1.241	1.240	0.115	0.026
helena	2.908	2.896	0.446	0.375
micro-mass	0.396	0.396	0.034	0.004

Table 6: Grid Search Results on Multiclass Classification tasks using a 33.33% holdout of the AutoML Benchmark. The values represent the minimum Log Loss calculated across the portfolio using 10-fold cross-validation (0 is best). The best portfolio per dataset across the entire grid is shown for ease of reference.

warm-starting. The chosen portfolios for binary classification have parameters $\alpha = 8$, $p = 16$ and $\alpha = 5$, $p = 16$ using the max and stable policy, respectively. The results for the entire grid can be found in Table 15 for the max policy and Table 16 for the stable policy. The pipelines contained in each portfolio can be found in our repository[7].

For multi-class classification, the results are shown in Table 6. Here, we find slightly more spread between the different configurations. For multi-class classification, we choose portfolios with parameters $\alpha = 8$, $p = 16$ for both the optimal portfolio using the max policy and the stable policy (using the same aggregation strategy as described above). Again, the results for the other datasets can be found in Table 17 and Table 18 for the max and stable policy, respectively.

Lastly, we ran the defined grid of portfolio configurations on the validation set of the regression AutoML benchmark. The results can be found in Table 7. Here, we indeed see that the results per dataset are on completely different scales, which highlights the necessity to scale accordingly when aggregating the results for the stable policy. The selected portfolios for regression have parameters $\alpha = 3$, $p = 16$ and $\mathcal{R}$, $p = 16$ using the max and stable policy, respectively. The results of the entire grid can be found in Table 19 and Table 20 for the max and stable policy, respectively.

6.2 Cluster Selection Results

As mentioned earlier, we could not directly assess the quality of the clusters. The results of this experiment attempt to provide an insight in the performance of the predicted cluster, when compared to all of the other clusters. Figure 3 shows the results for this experiment. We utilized portfolios created through the max policy to assess how well the performance of the anticipated cluster matches up against the portfolios associated with other clusters that were not anticipated. The results reflect the performance compared to the best portfolio across all labels for a single dataset. This allows for ease of comparison as the results across datasets are placed on a similar scale.

[7]https://github.com/Wman1001/MasterThesis/tree/main/portfolios

	$MaxPolicy$		$StablePolicy$	
DataSet Name	$\alpha = 3, p = 16$	$BestMax$	$\mathcal{R}, p = 16$	$BestStable$
pol	31.845	7.376	0.425	0.157
MIP-2016-regression	28883.452	28883.452	0.154	0.042
house_sales	270815.395	248885.402	0.202	0.008
tecator	2.635	2.635	0.094	0.089
black_friday	4645.942	4645.942	0.140	0.032
OnlineNewsPopularity	10953.811	10953.811	0.060	0.021
SAT11-HAND-runtime-regression	2136.192	2010.112	0.059	0.024
space_ga	1818.843	1818.843	0.201	0.003

Table 7: Grid Search Results on Regression tasks using a 33.33% holdout of the AutoML Benchmark. The values repesent the minimum RMSE calculated across the portfolio using 10-fold cross-validation (0 is best). The best portfolio per dataset across the entire grid is shown for ease of reference.

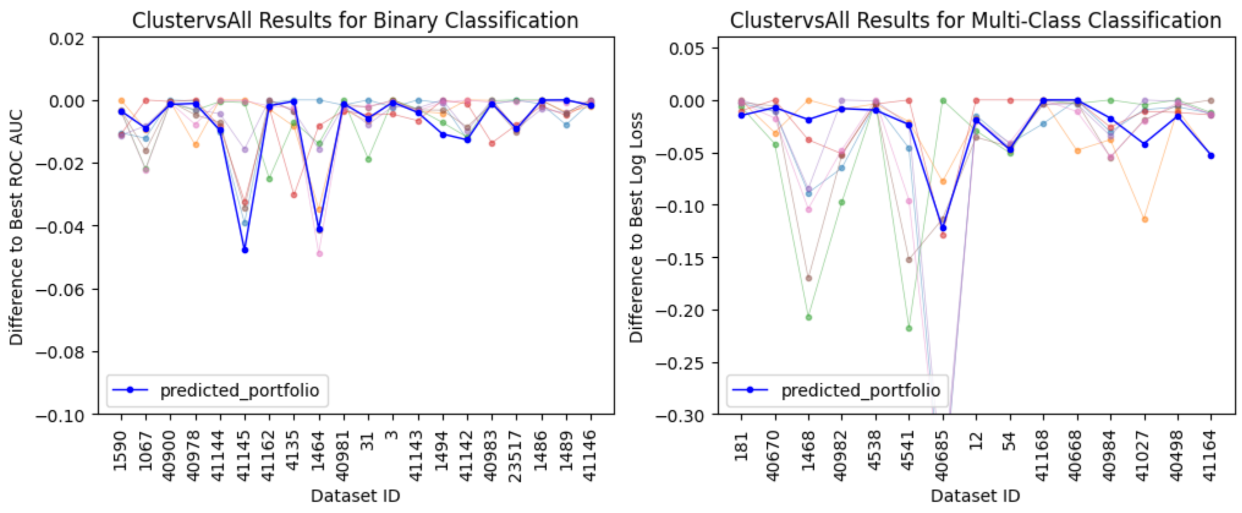

Figure 3: Cluster vs All results. The portfolios for all labels are evaluated on $\mathcal{D}_{test}$ and compared to the predicted label. The portfolios used are the portfolios as selected by the max policy. Depicted data points reflect the best performing pipeline in the portfolio for every label. Predicted label is highlighted for ease of comparison.

For binary classification, we find that in most cases, the portfolio attached to our predicted cluster provides either the best performance at least a satisfactory level of performance. Although in 18/20 datasets we see this behavior, we can observe that for dataset 41145 and dataset 1464, the chosen portfolio does not result in a near optimal performance due to a wrong assignment to a partition.

For multi-class classification, we applied a log transformation to the data to get a visualization taking into account various scales upon which the datasets exist. We can observe a bit more variation when compared to binary classification, but in most cases, except from dataset 40685, close to optimal results can be observed.

For regression, a representative visualization was not possible, as we had multiple missing values and some results were too far apart from each other to provide a proper reflection of the performance of the different clusters. Instead, we report the results per cluster in Table 21 in the appendix.

6.3 Results of Portfolio Comparison

In order to assess the performance of our portfolios, we compared them against three baseline portfolios: a portfolio constructed using the one nearest dataset algorithm (OND), a static portfolio that we build using our specific datasets and candidate pipelines, and the static portfolio approach presented by Feurer et al. [18]. First, we show the results of our approach(SBPort), the OND algorithm and our own static portfolio on the binary classification test set in Table 8.

	Max Policy			Stable Policy		
Dataset Name	**OND**	**Static**	**SBPort**	**OND**	**Static**	**SBPort**
adult	0.917	0.916	**0.926**	0.948	**0.956**	0.952
kc1	0.777	0.775	**0.777**	**0.882**	0.880	0.877
Satellite	0.993	0.994	**0.994**	0.978	0.994	**0.995**
Internet-Advertisements	0.969	0.968	**0.973**	**0.913**	0.606	0.854
madeline	0.873	0.950	**0.951**	**0.924**	0.659	0.878
philippine	0.870	**0.887**	0.872	**0.923**	0.564	0.558
kick	0.663	0.675	**0.688**	0.669	0.581	**0.739**
Amazon_employee_access	0.861	0.863	**0.867**	0.895	**0.929**	0.926
blood-transfusion-service-center	**0.948**	0.738	0.717	**0.829**	0.822	0.817
Australian	0.933	0.932	**0.936**	0.961	0.964	**0.964**
credit-g	**0.799**	0.798	0.796	0.864	**0.896**	0.890
kr-vs-kp	0.998	**0.999**	0.998	0.994	0.999	**0.999**
jasmine	**0.886**	0.884	0.885	0.872	0.938	**0.938**
qsar-biodeg	0.925	**0.927**	0.921	0.960	**0.960**	0.957
christine	**0.825**	0.820	0.817	**0.902**	0.532	0.782
wilt	0.994	**0.994**	0.993	0.962	0.989	**0.994**
numerai28.6	**0.530**	0.521	0.521	0.666	0.571	**0.697**
nomao	0.988	0.985	**0.988**	0.837	0.617	**0.992**
phoneme	0.956	0.968	**0.972**	0.969	0.979	**0.979**
sylvine	0.988	0.990	**0.990**	0.993	0.993	**0.993**
Total	5/20	4/20	11/20	6/20	4/20	10/20

Table 8: Portfolio results for Binary Classification task on the test split of the AutoML benchmark. We provide both results using the max policy and the stable policy to evaluate the algorithms. The used metric is the ROC AUC.

We find that our portfolio constructed by using the max policy gives us the model with the best performance in 11 out of 20 datasets. Additionally, the portfolios constructed using the stable policy show the best overall performance in 10/20 cases. The static portfolio tends to be quite unstable in numerous cases. For the SBPort portfolio, we only find an instable portfolio for the philippine dataset. A possible cause could be a wrong assignment to a partition, leading to a portfolio not well suited for the corresponding dataset.

For multi-class classification, the results are shown in Table 9. While observing the results for the stable policy, we see that SBPort again shows the most stable portfolios overall. The results aggregated over the datasets are especially more stable for SBPort when compared to OND. For OND, we see some very well performing portfolios for the shuttle and jungle_chess datasets, but the opposite can be observed for most of the other datasets. Additionally, The static portfolios tend to slightly more stability in the case of multi-class classification, when compared to the binary classification results.

For the comparison to the Auto-Sklearn portfolios, we ran their entire portfolio and retrieved the results of the first 16 pipelines that did not crash. We have chosen to do this, because there were some technical problems while running portfolios out of the context of AutoML. As we wanted to have a fair comparison between SBPort and Auto-Sklearn, including only the successfully evaluated pipelines, this was found to be the most suitable option. Also, for the multi-class classification datasets, we had to remove some datasets, as we could not evaluate any of the Auto-Sklearn pipelines on these datasets due to internal errors in the software for detached portfolio evaluation. Because of these issues, we could not evaluate the results of this experiment using the stable policy, as we did not have an accurate representation of the amount of failed pipelines. The results of the max policy for both binary and multi-class classification is shown in Figure 4. We can see that Auto-Sklearn finds the best performing pipeline in most cases compared to SBPort, for both binary and multi-class classification.

We show a strip plot to give an insight in the spread of the SBPort portfolios and Auto-Sklearn portfolios in Figure 5 for binary classification and Figure 6 for multi-class classification. Here, it's evident that the SBPort portfolios demonstrate greater consistency in providing well performing pipelines when compared to Auto-Sklearn.

Finally, we report the results of the regression experiment for OND, our own static portfolio and SBPort, using both the max policy and the stable policy to evaluate the performance. The results on the test set are shown in Table 10. Here, we see that SBPort performs best on every dataset in our test set when compared to OND and our own static portfolio

	Max Policy			Stable Policy		
Dataset Name	**OND**	**Static**	**SBPort**	**OND**	**Static**	**SBPort**
yeast	1.227	**1.107**	1.108	0.414	0.389	**0.282**
dna	0.111	0.113	**0.109**	0.467	0.437	**0.404**
cnae-9	0.254	0.258	**0.219**	0.409	0.434	**0.300**
steel-plates-fault	1.012	0.969	**0.943**	0.361	0.393	**0.182**
GesturePhaseSegmentation	1.304	**1.277**	1.287	0.348	0.355	**0.299**
Diabetes130US	0.901	**0.900**	0.943	0.384	**0.342**	0.459
shuttle	0.112	0.001	**0.001**	**0.210**	0.355	0.247
mfeat-factors	0.105	0.108	**0.104**	0.431	**0.333**	0.356
vehicle	0.452	**0.429**	0.446	0.338	0.271	**0.246**
jannis	0.752	0.710	**0.692**	0.466	**0.454**	0.595
connect-4	**0.763**	0.780	0.791	**0.421**	0.426	0.507
segment	0.162	0.168	**0.161**	**0.314**	0.421	0.336
jungle_chess	**0.713**	0.728	0.790	0.451	**0.347**	0.409
wine-quality	1.396	**1.108**	1.127	0.386	0.361	**0.320**
fabert	0.878	**0.864**	0.917	0.406	**0.401**	0.486
Total	2/15	6/15	7/15	3/15	5/15	7/15

Table 9: Portfolio results for the multi-class classification task on the test split of the AutoML benchmark. We provide both results using the max policy and the stable policy to evaluate the algorithms. The used metric is the Log Loss.

	Max Policy			Stable Policy		
Dataset Name	**OND**	**Static**	**SBPort**	**OND**	**Static**	**SBPort**
wine quality	0.891	1.459	**0.718**	**0.090**	0.382	0.143
yprop_4_1	0.041	0.070	**0.029**	0.514	0.468	**0.212**
elevators	0.010	0.009	**0.003**	0.214	**0.141**	0.312
boston	9.86	194.39	**4.093**	0.189	0.465	**0.037**
Moneyball	91.833	89.751	**24.793**	0.115	**0.083**	0.43
Brazilian_houses	29446.95	29377.88	**9668.785**	0.338	0.373	**0.153**
abalone	3.086	3.088	**2.149**	0.273	0.329	**0.161**
us_crime	1.535	0.766	**0.136**	0.442	0.499	**0.370**
topo_2_1	0.045	0.047	**0.029**	0.423	0.381	**0.256**
diamonds	3672.09	3544.031	**752.754**	0.304	**0.265**	0.279
sensory	1.337	3.9	**0.877**	0.405	0.496	**0.168**
socmob	37.337	37.337	**15.073**	**0.234**	0.280	0.284
Mercedes_Benz_Greener_Manufacturing	21.646	26.976	**8.79**	0.5	0.407	**0.213**
colleges	0.277	0.312	**0.149**	0.312	0.365	**0.195**
house_prices_nominal	82786.677	89895.681	**29165.251**	**0.153**	0.319	0.302
quake	4.482	4.483	**0.198**	0.464	0.484	**0.069**
house_16H	65730.604	2323033.876	**31688.139**	0.252	0.467	**0.037**
Average	0/17	0/17	**17/17**	3/17	3/17	**11/17**

Table 10: Portfolio results for the regression task on the test split of the AutoML benchmark. We provide both results using the max policy and the stable policy to evaluate the algorithms. The used metric is the RMSE.

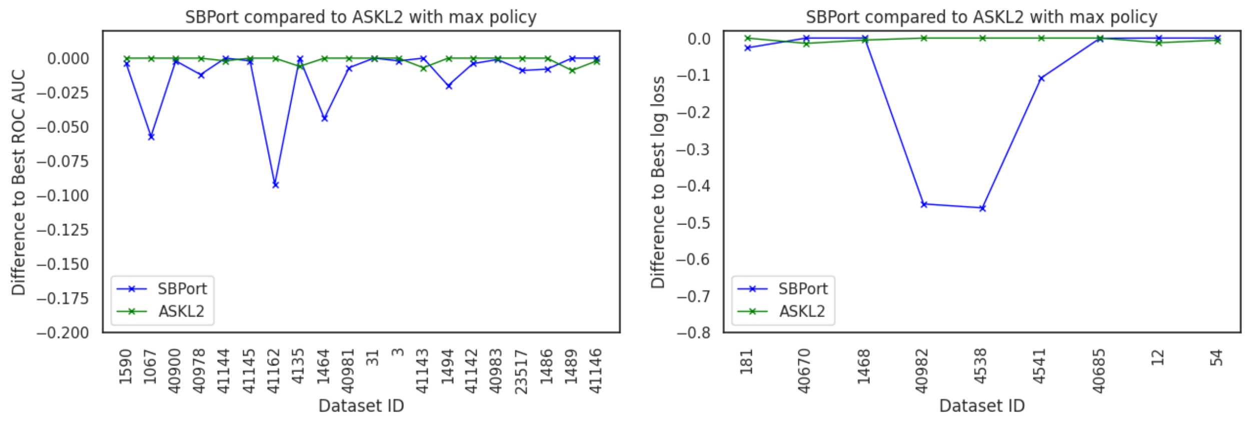

Figure 4: Performance of SBPort and ASKL2 using the max policy. Here, the reported values are the difference to the best value of SBPort and ASKL2 to provide a comparable metric for results that are on a different scale.

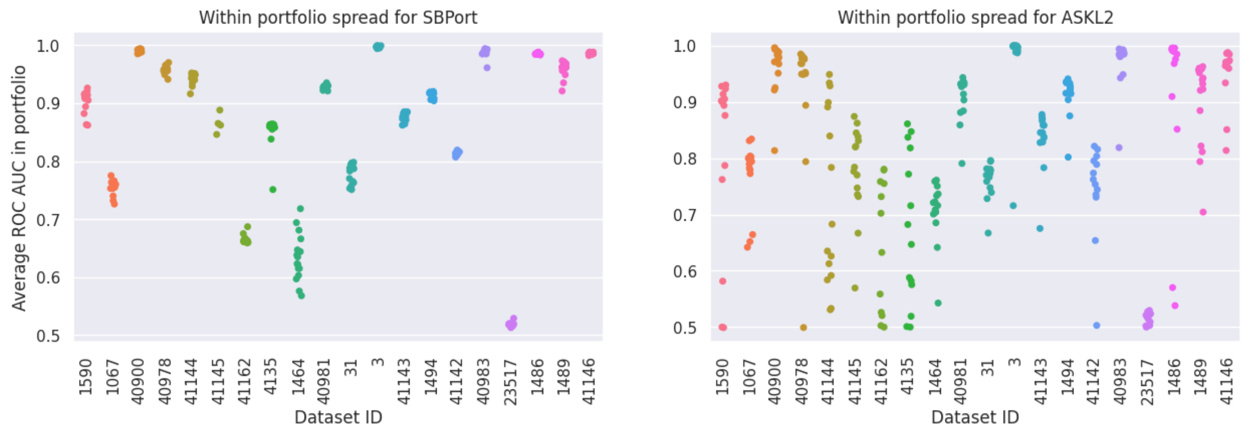

Figure 5: Strip plot of the individual pipeline performance of SBPort compared to Auto-Sklearn for binary classification datasets.

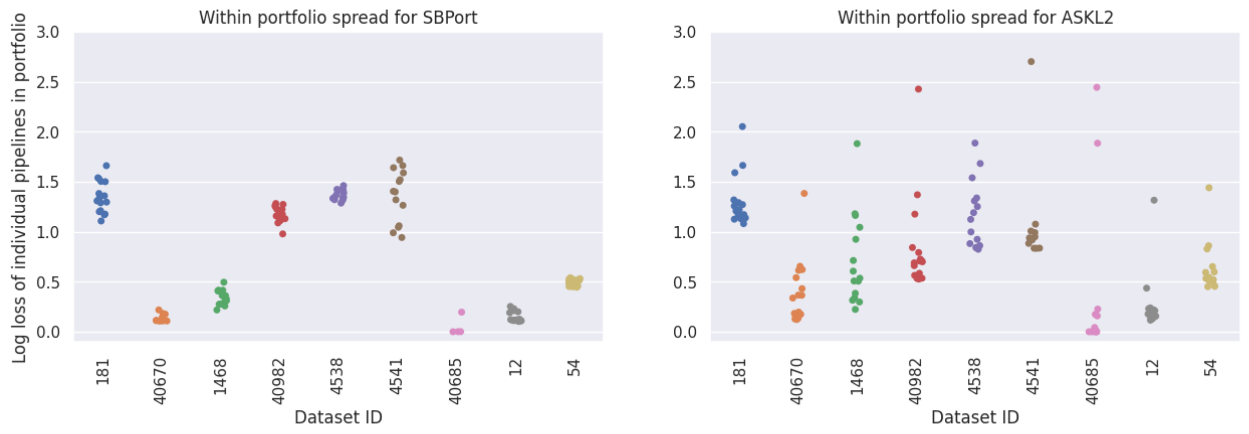

Figure 6: Strip plot of the individual pipeline performance of SBPort compared to Auto-Sklearn for multi-class classification datasets.

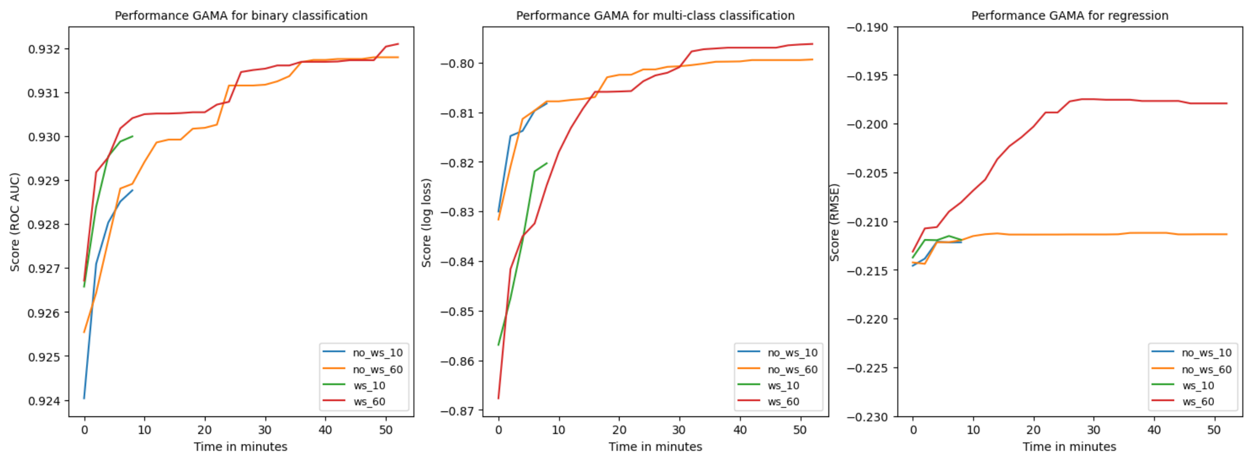

Figure 7: Results of warm-starting GAMA with portfolios. The figures represent aggregated results for binary classification, multi-class classification and regression datasets, from left to right. We evaluate its performance using the same metrics as for the other experiments.

when evaluated with the max policy. Additionally, the results using the stable policy show that the pipelines in our portfolio rarely crash and perform well on average.

6.4 Results Portfolios in AutoML

As a last experiment, we assessed the effectiveness of our portfolios within the context of warm-starting GAMA. To achieve this, we subjected each dataset to four separate runs of GAMA, allocating time limits of 10 and 60 minutes for each run, while alternating between employing warm-starting and not. The combined outcomes across all datasets are illustrated in Figure 7.

In the domain of binary classification, we find that warm-starting proves advantageous. The warm-starting approach consistently outperforms its counterpart without warm-starting across the entire timespan. While observing multi-class classification results, warm-starting appears not to lead to better pipelines in the first 20 minutes. Nonetheless, as time unfolds, the difference becomes more noticeable, yielding the best overall performance after 60 minutes.

For regression, we evidently see that warm-starting is beneficial, leading to a clear increase of performance compared to GAMA without warm-starting. The full results can be found in Table 22, Table 23 and Table 24 in the appendix for binary classification, multi-class classification and regression respectively.

7 Discussion

7.1 Discussion of Experimental Results

In consideration of both predictive power and stability, high performing portfolios were elected through the utilization of a grid search to determine the appropriate hyper-parameters for the number of pipelines in the portfolio and the cluster configuration. To achieve this, we used a validation split of the AutoML benchmark. The results on the test split demonstrate the advantage of using SBPort over our constructed static portfolios and portfolios constructed using the OND algorithm. Especially for binary classification and regression prediction tasks, we clearly found improvements.

While Auto-Sklearn portfolios exhibit superior performance in predicting the optimal pipeline, they demonstrate less stability compared to SBPort. A possible reason for this is that the pipeline configuration space of Auto-Sklearn is notably different compared to our configuration space. The disparity in stability also highlights the drawback of using static portfolios, whereas for every dataset, several pipelines may not be well suited to the dataset at hand and demonstrate poor performance.

In the cluster quality experiment, we found that the assignment of a partition from the meta-features of a dataset is not always optimal. We find that in some cases, another cluster would have yielded a more effective portfolio, which may be the result of a suboptimal selection of meta-features. Finally, the pipelines contained in the portfolios of SBPort were used to warm-start GAMA. We found several improvements compared to GAMA without warm-starting. On all three tasks, we found clear benefits of using warm-starting. For the majority of the datasets, GAMA with warm-starting given a time budget of 10 minutes reaches similar performance compared to GAMA without warm-starting with a time budget of 60 minutes for most datasets. However, wrong assignments to a partition lead to poor warm-starting, which in turn also results in worse performance than GAMA without warm-starting.

7.2 Limitations and Future Research

Several limitations were identified during this research. First of all, we used OPTICS as a clustering algorithm to create portfolios for high density clusters. However, especially for multi-class classification, no satisfactory configuration could be found. The usage of other density-based algorithms, or other types of clustering as a whole, may work better for this task. Nevertheless, as we conducted a grid search, which included Kernel K-means, we could simply select solely Kernel K-means configurations to proceed with the experiments. For future research, it remains interesting to investigate the usage of a clustering algorithm that creates a separate cluster for outliers or datasets that are not in the span of the training set of datasets. Even though Kernel K-means has proven its usability in this study, we predicted poor performing portfolios for a few datasets, highlighting the necessity to investigate other algorithms to partition the set of datasets. Alternatively, the problem of predicting poor portfolios could also stem from the choice of meta-features. A recommendation for future research is learn meta-features through the application of deep learning [26]. By adopting this methodology, it becomes possible to represent individual datasets as embeddings. Much like the approach employed in this paper, these embeddings can be clustered, thereby offering the potential for enhanced portfolio quality.

Another limitation of our research is that we selected our candidate pipelines based on 50 AutoML runs for 60 minutes, and selected the top five best performing pipelines. This procedure was repeated for every task. However, as the top five best performing pipelines on a dataset are sometimes very similar, this led to clusters having a portfolio with a collection of near identical pipelines, especially for smaller portfolios. We particularly observed this for our binary classification portfolios. A simple solution to this would be to run the AutoML system on all datasets in our training set, and pick the single best performing pipeline for this respective dataset. However, in the timeline of this research, this was not feasible.

Finally, the performance of a pipeline in our performance matrix is solely based on the predictive power of the pipeline. However, especially for constructing portfolios in the context of AutoML, it would be interesting to extend the performance to take into account the time it takes to evaluate a pipeline. For example, say we have a pipeline $\mathcal{M}_{\lambda 1}$ and another pipeline $\mathcal{M}_{\lambda 2}$, which we evaluate on a dataset D. After successfully evaluating both pipelines, we find that $\widehat{GE}(\mathcal{M}_{\lambda 1}, D)$ is slightly better than $\widehat{GE}(\mathcal{M}_{\lambda 2}, D)$. However, $t_{\lambda 2} << t_{\lambda 1}$. While inspecting the performance matrix, we indeed found several cases where the difference in performance is negligible, but the evaluation time is up to 100 times faster. In this case, $\mathcal{M}_{\lambda 2}$ would be a better pipeline to append to our stable portfolio. As such, an interesting direction for future research would be to have a metric composed of a predictive power component and a time component. Abdulrahman et al. [1] present a multi-objective measure with both these components for algorithm selection. This measure could be adopted and used as a portfolio construction approach. This might benefit AutoML warm-starting even more, especially for large datasets.

8 Conclusion

In this research, a novel meta-learning approach to building pipeline portfolios was developed. We partitioned the dataset space and stored the best performing pipelines within this partition in a portfolio. We broadly tested the performance of SBPort compared to alternative portfolios created by a static approach and by using the one nearest dataset algorithm. Moreover, we compared SBPort to the portfolios of Auto-Sklearn 2.0, which is the current state-of-the-art for warm-starting of AutoML systems. Here, it was found that SBPort did not find similar performing pipelines using the max policy. Nevertheless, SBPort has clearly proven to be a competitive alternative solution while observing the stability of the portfolio.

Prior portfolio research has only evaluated its performance on binary and multi-class classification problems. Our research is the first to also include the regression task in our methodology. As such, no direct comparison could be made to other similar techniques. Nonetheless, competitive performance was observed compared to our static portfolios and the OND portfolios.

Besides detached portfolio evaluation, SBPort has proven its usability within the context of warm-starting GAMA. Across all three tasks, the utilization of warm-starting was found to be beneficial compared to GAMA without warm-starting across a majority of the datasets in the AutoML benchmark.

Compared to static portfolios, we can conclude that in terms of predictive performance, our approach provides a marginal advantage. However, the stability of the portfolios by SBPort has proven to be better compared to our own static portfolios, but also the static portfolios by Feurer et al. [18]. Nevertheless, our approach is directly dependent on the predicted cluster. Wrong cluster assignments directly lead to poor performing behaviour, both in terms of predictive performance and stability. A potential solution to mitigate this behaviour in the context of AutoML would be to use successive halving to provide less resources to poor performing pipelines.

References

[1] S. M. Abdulrahman, P. Brazdil, J. N. van Rijn, and J. Vanschoren. Speeding up algorithm selection using average ranking and active testing by introducing runtime. *Machine learning*, 107:79–108, 2018.

[2] E. Alcobaça, F. Siqueira, A. Rivolli, L. P. F. Garcia, J. T. Oliva, and A. C. P. L. F. de Carvalho. Mfe: Towards reproducible meta-feature extraction. *Journal of Machine Learning Research*, 21(111):1–5, 2020.

[3] M. Ankerst, M. M. Breunig, H.-P. Kriegel, and J. Sander. Optics: Ordering points to identify the clustering structure. *ACM Sigmod record*, 28(2):49–60, 1999.

[4] D. Arthur and S. Vassilvitskii. k-means++: The advantages of careful seeding. Technical report, Stanford, 2006.

[5] H. Bensusan and A. Kalousis. Estimating the predictive accuracy of a classifier. In *European Conference on Machine Learning*, pages 25–36. Springer, 2001.

[6] M. R. Berthold, N. Cebron, F. Dill, T. R. Gabriel, T. Kötter, T. Meinl, P. Ohl, C. Sieb, K. Thiel, and B. Wiswedel. KNIME: The Konstanz Information Miner. In *Studies in Classification, Data Analysis, and Knowledge Organization (GfKL 2007)*. Springer, 2007.

[7] F. Bürger and J. Pauli. A holistic classification optimization framework with feature selection, preprocessing, manifold learning and classifiers. In *Pattern Recognition: Applications and Methods: 4th International Conference, ICPRAM 2015, Lisbon, Portugal, January 10-12, 2015, Revised Selected Papers 4*, pages 52–68. Springer, 2015.

[8] R. Caruana, A. Munson, and A. Niculescu-Mizil. Getting the most out of ensemble selection. In *Sixth International Conference on Data Mining (ICDM'06)*, pages 828–833. IEEE, 2006.

[9] R. Caruana, A. Niculescu-Mizil, G. Crew, and A. Ksikes. Ensemble selection from libraries of models. In *Proceedings of the twenty-first international conference on Machine learning*, page 18, 2004.

[10] D. Defays. An efficient algorithm for a complete link method. *The computer journal*, 20(4):364–366, 1977.

[11] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society: Series B (Methodological)*, 39(1):1–22, 1977.

[12] I. S. Dhillon, Y. Guan, and B. Kulis. Kernel k-means: Spectral clustering and normalized cuts. In *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '04, page 551–556, New York, NY, USA, 2004. Association for Computing Machinery.

[13] T. Doan and J. Kalita. Predicting run time of classification algorithms using meta-learning. *International Journal of Machine Learning and Cybernetics*, 8(6):1929–1943, 2017.

[14] H. J. Escalante, M. Montes, and E. Sucar. Ensemble particle swarm model selection. In *The 2010 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2010.

[15] H. J. Escalante, M. Montes, and L. E. Sucar. Particle swarm model selection. *Journal of Machine Learning Research*, 10(2), 2009.

[16] M. Ester, H.-P. Kriegel, J. Sander, X. Xu, et al. A density-based algorithm for discovering clusters in large spatial databases with noise. In *kdd*, volume 96, pages 226–231, 1996.

[17] V. Estivill-Castro. Why so many clustering algorithms: a position paper. *ACM SIGKDD explorations newsletter*, 4(1):65–75, 2002.

[18] M. Feurer, K. Eggensperger, S. Falkner, M. Lindauer, and F. Hutter. Auto-sklearn 2.0: Hands-free automl via meta-learning. *arXiv preprint arXiv:2007.04074*, 2020.

[19] M. Feurer, A. Klein, K. Eggensperger, J. Springenberg, M. Blum, and F. Hutter. Efficient and robust automated machine learning. *Advances in neural information processing systems*, 28, 2015.

[20] P. Gijsbers, M. L. Bueno, S. Coors, E. LeDell, S. Poirier, J. Thomas, B. Bischl, and J. Vanschoren. Amlb: an automl benchmark. *arXiv preprint arXiv:2207.12560*, 2022.

[21] P. Gijsbers, F. Pfisterer, J. N. van Rijn, B. Bischl, and J. Vanschoren. Meta-learning for symbolic hyperparameter defaults. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pages 151–152, 2021.

[22] P. Gijsbers and J. Vanschoren. Gama: A general automated machine learning assistant. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 560–564. Springer, 2021.

[23] C. P. Gomes and B. Selman. Algorithm portfolios. *Artificial Intelligence*, 126(1-2):43–62, 2001.

[24] I. Guyon, K. Bennett, G. Cawley, H. J. Escalante, S. Escalera, T. K. Ho, N. Macià, B. Ray, M. Saeed, A. Statnikov, et al. Design of the 2015 chalearn automl challenge. In *2015 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2015.

[25] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. The weka data mining software: an update. *ACM SIGKDD explorations newsletter*, 11(1):10–18, 2009.

[26] H. S. Jomaa, L. Schmidt-Thieme, and J. Grabocka. Dataset2vec: Learning dataset meta-features. *Data Mining and Knowledge Discovery*, 35:964–985, 2021.

[27] J. Kennedy and R. Eberhart. Particle swarm optimization. In *Proceedings of ICNN'95 - International Conference on Neural Networks*, volume 4, pages 1942–1948 vol.4, 1995.

[28] E. LeDell and S. Poirier. H2o automl: Scalable automatic machine learning. In *Proceedings of the AutoML Workshop at ICML*, volume 2020, 2020.

[29] L. Li, K. Jamieson, A. Rostamizadeh, E. Gonina, M. Hardt, B. Recht, and A. Talwalkar. A system for massively parallel hyperparameter tuning, 2020.

[30] J. MacQueen. Classification and analysis of multivariate observations. In *5th Berkeley Symp. Math. Statist. Probability*, pages 281–297, 1967.

[31] R. G. Mantovani, A. L. D. Rossi, E. Alcobaça, J. C. Gertrudes, S. B. Junior, and A. C. P. d. L. F. de Carvalho. Rethinking default values: a low cost and efficient strategy to define hyperparameters. *arXiv preprint arXiv:2008.00025*, 2020.

[32] D. K. McClish. Analyzing a portion of the roc curve. *Medical decision making*, 9(3):190–195, 1989.

[33] I. Mierswa, M. Wurst, R. Klinkenberg, M. Scholz, and T. Euler. Yale: Rapid prototyping for complex data mining tasks. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 935–940. ACM, 2006.

[34] R. S. Olson and J. H. Moore. Tpot: A tree-based pipeline optimization tool for automating machine learning. In F. Hutter, L. Kotthoff, and J. Vanschoren, editors, *Proceedings of the Workshop on Automatic Machine Learning*, volume 64 of *Proceedings of Machine Learning Research*, pages 66–74, New York, New York, USA, 24 Jun 2016. PMLR.

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

[36] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, et al. Scikit-learn: Machine learning in python. *the Journal of machine Learning research*, 12:2825–2830, 2011.

[37] F. Pfisterer, J. N. van Rijn, P. Probst, A. C. Müller, and B. Bischl. Learning multiple defaults for machine learning algorithms. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pages 241–242, 2021.

[38] M. Reif, F. Shafait, and A. Dengel. Prediction of classifier training time including parameter optimization. In *Annual Conference on Artificial Intelligence*, pages 260–271. Springer, 2011.

[39] M. Reif, F. Shafait, and A. Dengel. Meta-learning for evolutionary parameter optimization of classifiers. *Machine learning*, 87:357–380, 2012.

[40] R. Sibson. Slink: an optimally efficient algorithm for the single-link cluster method. *The computer journal*, 16(1):30–34, 1973.

[41] Q. Sun, B. Pfahringer, and M. Mayo. Towards a framework for designing full model selection and optimization systems. In *Multiple Classifier Systems: 11th International Workshop, MCS 2013, Nanjing, China, May 15-17, 2013. Proceedings 11*, pages 259–270. Springer, 2013.

[42] C. Thornton, F. Hutter, H. H. Hoos, and K. Leyton-Brown. Auto-weka: Combined selection and hyperparameter optimization of classification algorithms. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 847–855, 2013.

[43] J. Vanschoren, J. N. Van Rijn, B. Bischl, and L. Torgo. Openml: networked science in machine learning. *ACM SIGKDD Explorations Newsletter*, 15(2):49–60, 2014.

A Meta-features

Table 11 shows the meta-features used for clustering datasets in this paper. Each column denotes the features for a different task type.

Binary Target	Multi-Class Target	Continuous Target
nr_cat	nr_cat	nr_cat
nr_num	nr_num	nr_num
num_to_cat_ratio	num_to_cat_ratio	num_to_cat_ratio
cor_mean	cor_mean	cor_mean
cor_sd	cor_sd	cor_sd
cov_mean	cov_mean	cov_mean
cov_sd	cov_sd	cov_sd
iqr_mean	iqr_mean	iqr_mean
iqr_sd	iqr_sd	iqr_sd
kurtosis_mean	kurtosis_mean	kurtosis_mean
kurtosis_sd	kurtosis_sd	kurtosis_sd
max_mean	max_mean	max_mean
max_sd	max_sd	max_sd
mean_mean	mean_mean	mean_mean
mean_sd	mean_sd	mean_sd
median_mean	median_mean	median_mean
median_sd	median_sd	median_sd
min_mean	min_mean	min_mean
min_sd	min_sd	min_sd
missing_mean	missing_mean	missing_mean
missing_sd	missing_sd	missing_sd
best_node_mean	best_node_mean	best_node_mean
nr_bin	nr_bin	nr_bin
outliers	outliers	outliers
sd_mean	sd_mean	sd_mean
sd_sd	sd_sd	sd_sd
nr_feat	nr_feat	nr_feat
nr_inst	nr_inst	nr_inst
attr_to_inst_ratio	attr_to_inst_ratio	attr_to_inst_ratio
skewness_mean	skewness_mean	skewness_mean
skewness_sd	skewness_sd	skewness_sd
variance_mean	variance_mean	variance_mean
variance_sd	variance_sd	variance_sd
attr_conc_mean	attr_conc_mean	attr_conc_mean
attr_conc_sd	attr_conc_sd	attr_conc_sd
class_conc_mean	class_conc_mean	attr_ent_mean
class_conc_sd	class_conc_sd	attr_ent_sd
attr_ent_mean	attr_ent_mean	kurtosis_target
attr_ent_sd	attr_ent_sd	bayesian_ridge_mean
eq_num_attr	eq_num_attr	skewness_target
joint_ent_mean	joint_ent_mean	linear_regr_mean
joint_ent_sd	joint_ent_sd	target_corr_mean
random_node_mean	random_node_mean	target_corr_sd
naive_bayes_mean	naive_bayes_mean	random_node_mean
worst_node_mean	worst_node_mean	worst_node_mean
linear_discr_mean	linear_discr_mean	-
majority_class_size	majority_class_size	-
minority_class_size	minority_class_size	-
-	nr_class	-

Table 11: Meta-features used in this paper.

B Datasets

Table 12, Table 13 and Table 14 show the data sets from the AutoML benchmark that are used for validation and testing in this study.

Dataset	Instances	Features	Set
bank-marketing	45211.0	17.0	validation
Bioresponse	3751.0	1777.0	validation
Click_prediction_small	39948.0	12.0	validation
PhishingWebsites	11055.0	31.0	validation
ozone-level-8hr	2534.0	73.0	validation
ada	4147.0	49.0	validation
pc4	1458.0	38.0	validation
MiniBooNE	130064.0	51.0	validation
churn	5000.0	21.0	validation
gina	3153.0	971.0	validation
adult	48842.0	15.0	test
kc1	2109.0	22.0	test
Satellite	5100.0	37.0	test
Internet-Advertisements	3279.0	1559.0	test
madeline	3140.0	260.0	test
philippine	5832.0	309.0	test
kick	72983.0	33.0	test
Amazon_employee_access	32769.0	10.0	test
blood-transfusion-service-center	748.0	5.0	test
Australian	690.0	15.0	test
credit-g	1000.0	21.0	test
kr-vs-kp	3196.0	37.0	test
jasmine	2984.0	145.0	test
qsar-biodeg	1055.0	42.0	test
christine	5418.0	1637.0	test
wilt	4839.0	6.0	test
numerai28.6	96320.0	22.0	test
nomao	34465.0	119.0	test
phoneme	5404.0	6.0	test
sylvine	5124.0	21.0	test

Table 12: Specifics of the binary classification datasets from the AutoML Benchmark.

Dataset	Instances	Features	Set
cmc	1473.0	10.0	validation
okcupid-stem	50789.0	20.0	validation
eucalyptus	736.0	20.0	validation
car	1728.0	7.0	validation
first-order-theorem-proving	6118.0	52.0	validation
helena	65196.0	28.0	validation
micro-mass	571.0	1301.0	validation
yeast	1484.0	9.0	test
dna	3186.0	181.0	test
cnae-9	1080.0	857.0	test
steel-plates-fault	1941.0	28.0	test
GesturePhaseSegmentationProcessed	9873.0	33.0	test
Diabetes130US	101766.0	50.0	test
shuttle	58000.0	10.0	test
mfeat-factors	2000.0	217.0	test
vehicle	846.0	19.0	test
jannis	83733.0	55.0	test
connect-4	67557.0	43.0	test
segment	2310.0	20.0	test
jungle_chess_2pcs_raw_endgame_complete	44819.0	7.0	test
wine-quality-white	4898.0	12.0	test
fabert	8237.0	801.0	test

Table 13: Specifics of the multi-class classification datasets from the AutoML Benchmark.

Dataset	Instances	Features	Set
pol	15000.0	49.0	validation
MIP-2016-regression	1090.0	145.0	validation
house_sales	21613.0	22.0	validation
tecator	240.0	125.0	validation
black_friday	166821.0	10.0	validation
OnlineNewsPopularity	39644.0	60.0	validation
SAT11-HAND-runtime-regression	4440.0	117.0	validation
space_ga	3107.0	7.0	validation
wine_quality	6497.0	12.0	test
yprop_4_1	8885.0	252.0	test
elevators	16599.0	19.0	test
boston	506.0	14.0	test
Moneyball	1232.0	15.0	test
Brazilian_houses	10692.0	13.0	test
abalone	4177.0	9.0	test
us_crime	1994.0	127.0	test
topo_2_1	8885.0	267.0	test
diamonds	53940.0	10.0	test
sensory	576.0	12.0	test
socmob	1156.0	6.0	test
Mercedes_Benz_Greener_Manufacturing	4209.0	377.0	test
colleges	7063.0	45.0	test
house_prices_nominal	1460.0	80.0	test
quake	2178.0	4.0	test
house_16H	22784.0	17.0	test

Table 14: Specifics of the regression datasets from the AutoML Benchmark.

C Full Grid Search Results

Chapter 6 shows the results for the selected candidate of our defined grid. Here, we present the results of the entire grid on the validation set for each task. Table 15, Table 17 and Table 19 show the results using the max policy for binary classification, multi-class classification and regression respectively. Similarly, the results using the stable policy can be seen in Table 16, Table 18 and Table 20.

	$p=4$	$p=4$	$p=4$	$p=4$	$p=8$	$p=8$	$p=8$	$p=8$	$p=16$	$p=16$	$p=16$	$p=16$
DataSet ID	$\alpha=3$	$\alpha=5$	$\alpha=8$	$\mathcal{R}$	$\alpha=3$	$\alpha=5$	$\alpha=8$	$\mathcal{R}$	$\alpha=3$	$\alpha=5$	$\alpha=8$	$\mathcal{R}$
1461	0.67	0.508	0.702	0.774	0.691	0.666	**0.779**	0.774	0.709	0.696	**0.779**	0.774
4134	0.869	0.868	-	0.872	0.877	0.871	0.875	0.872	0.877	0.872	**0.882**	0.872
42733	0.686	0.686	0.682	0.69	**0.705**	0.689	0.693	0.69	**0.705**	0.689	0.698	0.697
4534	0.995	0.995	**0.996**	0.995	0.995	0.995	**0.996**	0.995	**0.996**	**0.996**	**0.996**	0.995
1487	0.873	0.874	0.865	0.868	0.876	0.874	0.871	0.878	0.882	0.876	0.873	**0.883**
41146	0.909	0.911	0.909	0.912	0.909	0.911	0.917	0.912	**0.918**	0.916	0.917	**0.918**
1049	0.941	0.947	0.948	**0.949**	0.946	0.947	**0.949**	**0.949**	0.947	0.947	**0.949**	**0.949**
41150	0.981	0.981	-	0.984	0.981	0.982	0.983	0.984	0.984	0.983	**0.985**	0.984
40701	0.918	0.920	0.914	0.911	0.921	0.921	0.914	0.919	0.921	**0.922**	0.920	0.920
41158	0.984	0.985	0.988	**0.989**	0.988	0.987	0.988	**0.989**	0.988	0.987	0.988	**0.989**

Table 15: Grid search results on binary classification tasks using a 33.33% holdout of the AutoML Benchmark. The values represent the maximum ROC AUC calculated over the portfolio using 10-fold cross-validation.

	$p=4$	$p=4$	$p=4$	$p=4$	$p=8$	$p=8$	$p=8$	$p=8$	$p=16$	$p=16$	$p=16$	$p=16$
DataSet ID	$\alpha=3$	$\alpha=5$	$\alpha=8$	$\mathcal{R}$	$\alpha=3$	$\alpha=5$	$\alpha=8$	$\mathcal{R}$	$\alpha=3$	$\alpha=5$	$\alpha=8$	$\mathcal{R}$
1461	0.975	**0.997**	0.984	0.964	0.955	0.948	0.954	0.883	0.946	0.942	0.916	0.904
4134	0.625	0.749	-	0.746	0.623	**0.873**	0.686	0.68	0.655	**0.873**	0.809	0.65
42733	0.997	**0.999**	0.997	0.994	0.990	0.997	0.993	0.994	0.989	0.997	0.993	0.992
4534	**0.999**	**0.999**	**0.999**	0.998	**0.999**	**0.999**	**0.999**	0.998	**0.999**	**0.999**	**0.999**	0.998
1487	**0.998**	0.997	**0.998**	0.997	0.997	0.997	0.996	0.995	0.993	0.996	0.997	0.994
41146	**0.999**	0.998	**0.999**	0.995	0.999	0.994	0.994	0.996	0.994	0.990	0.990	0.993
1049	0.998	**0.999**	0.997	**0.999**	0.997	0.998	0.996	0.998	0.996	0.996	0.997	0.998
41150	0.625	0.625	-	0.749	0.562	0.750	0.625	0.684	0.593	**0.812**	0.78	0.654
40701	**0.999**	**0.999**	0.997	0.998	0.998	0.998	0.997	0.996	0.998	0.998	0.996	0.997
41158	0.625	0.750	0.625	0.997	0.624	0.873	0.562	0.808	0.655	**0.874**	0.624	0.715
Average	0.884	0.911	0.760	0.944	0.874	0.943	0.880	0.903	0.882	**0.948**	0.910	0.890

Table 16: Grid search results on binary classification tasks using a 33.33% holdout of the AutoML Benchmark. The values represent the stability values of the portfolios as calculated by the formula provided in Chapter 5.

	$p=4$	$p=4$	$p=4$	$p=4$	$p=8$	$p=8$	$p=8$	$p=8$	$p=16$	$p=16$	$p=16$	$p=16$
DataSet ID	$\alpha=3$	$\alpha=5$	$\alpha=8$	$\mathcal{R}$	$\alpha=3$	$\alpha=5$	$\alpha=8$	$\mathcal{R}$	$\alpha=3$	$\alpha=5$	$\alpha=8$	$\mathcal{R}$
23	0.923	0.919	0.924	0.959	0.923	0.919	0.924	0.959	**0.902**	0.910	0.924	0.959
42734	0.572	0.571	0.573	0.791	**0.569**	**0.569**	0.570	0.791	**0.569**	**0.569**	0.570	0.646
188	1.267	1.216	**0.864**	9.619	1.221	1.216	**0.864**	3.735	0.870	0.865	**0.864**	3.070
40975	0.273	0.288	0.278	14.692	0.273	0.286	0.274	0.793	0.270	0.249	**0.207**	0.549
1475	1.258	1.244	1.251	10.795	1.242	1.244	1.242	10.795	1.242	**1.240**	1.241	4.819
41169	-	-	2.952	**2.896**	-	-	2.952	**2.896**	3.335	3.317	2.908	**2.896**
1515	-	-	**0.396**	32.445	0.428	0.442	**0.396**	5.242	0.397	0.437	**0.396**	5.242

Table 17: Grid search results on multi-class classification tasks using a 33.33% holdout of the AutoML Benchmark. The values represent the minimum log loss calculated over the portfolio using 10-fold cross-validation.

DataSet ID	$p = 4$ $\alpha = 3$	$p = 4$ $\alpha = 5$	$p = 4$ $\alpha = 8$	$p = 4$ $\mathcal{R}$	$p = 8$ $\alpha = 3$	$p = 8$ $\alpha = 5$	$p = 8$ $\alpha = 8$	$p = 8$ $\mathcal{R}$	$p = 16$ $\alpha = 3$	$p = 16$ $\alpha = 5$	$p = 16$ $\alpha = 8$	$p = 16$ $\mathcal{R}$
23	0.281	0.366	0.282	0.067	0.266	0.362	0.200	**0.046**	0.199	0.227	0.246	0.051
42734	0.040	0.167	0.141	**0.001**	0.117	0.192	0.089	0.068	0.188	0.152	0.099	0.132
188	**0.088**	0.34	0.249	0.375	0.295	0.313	0.315	0.218	0.408	0.355	0.32	0.227
40975	0.265	0.170	**0.052**	0.175	0.288	0.180	0.09	0.396	0.070	0.136	0.261	0.202
1475	0.051	0.092	**0.026**	0.313	0.094	0.082	0.056	0.225	0.165	0.079	0.115	0.455
41169	1	1	**0.375**	**0.375**	1.	1.	0.625	0.438	0.413	0.449	0.446	0.469
1515	1	1	0.053	**0.004**	0.253	0.254	0.005	0.030	0.132	0.13	0.034	0.053
Average	0.389	0.448	0.168	0.187	0.330	0.340	0.197	0.203	0.225	0.218	**0.217**	0.227

Table 18: Grid search results on multi-class classification tasks using a 33.33% holdout of the AutoML Benchmark. The values represent the stability values of the portfolios as calculated by the formula provided in Chapter 5. Best performing configuration with $p = 16$ is highlighted.

DataSet ID	$p = 4$ $\alpha = 3$	$p = 4$ $\alpha = 5$	$p = 4$ $\alpha = 8$	$p = 4$ $\mathcal{R}$	$p = 8$ $\alpha = 3$	$p = 8$ $\alpha = 5$	$p = 8$ $\alpha = 8$	$p = 8$ $\mathcal{R}$	$p = 16$ $\alpha = 3$	$p = 16$ $\alpha = 5$	$p = 16$ $\alpha = 8$	$p = 16$ $\mathcal{R}$
201	41.182	41.856	39.779	40.32	40.543	39.768	39.779	31.845	31.845	31.922	31.922	**7.376**
43071	31208.159	31169.929	31169.334	31132.936	**28883.452**	31169.929	30678.909	28903.36	**28883.452**	29024.277	29005.065	28903.36
42731	356940.061	262414.766	**248885.402**	355909.641	270815.395	262414.766	**248885.402**	288373.359	270815.395	262414.766	**248885.402**	286080.127
505	3.972	3.751	4.165	9.144	3.972	3.751	4.165	8.098	**2.635**	**2.635**	**2.635**	**2.635**
41540	**4645.942**	4652.866	4652.024	4721.973	**4645.942**	4652.866	4652.024	4721.973	**4645.942**	4652.866	4652.024	4677.689
42724	11093.86	10979.282	11096.346	11084.422	10997.95	10979.282	11084.13	10983.078	**10953.811**	10979.282	10978.704	**10953.811**
41980	2537.652	2604.625	2351.699	2537.433	2136.192	2537.814	**2010.112**	2125.287	2136.192	2074.159	**2010.112**	2125.287
507	**1818.843**	1859.963	1871.278	2538.124	**1818.843**	1823.436	1864.482	2133.769	**1818.843**	1823.436	1850.551	2133.769

Table 19: Grid search results on regression tasks using a 33.33% holdout of the AutoML Benchmark. The values represent the minimum RMSE calculated over the portfolio using 10-fold cross-validation.

DataSet ID	$p = 4$ $\alpha = 3$	$p = 4$ $\alpha = 5$	$p = 4$ $\alpha = 8$	$p = 4$ $\mathcal{R}$	$p = 8$ $\alpha = 3$	$p = 8$ $\alpha = 5$	$p = 8$ $\alpha = 8$	$p = 8$ $\mathcal{R}$	$p = 16$ $\alpha = 3$	$p = 16$ $\alpha = 5$	$p = 16$ $\alpha = 8$	$p = 16$ $\mathcal{R}$
201	0.371	0.174	0.311	0.241	**0.157**	0.241	0.184	0.170	0.276	0.293	0.276	0.425
43071	**0.042**	0.284	0.251	0.292	0.168	0.223	0.224	0.198	0.133	0.250	0.204	0.154
42731	0.255	0.129	0.336	**0.008**	0.282	0.264	0.413	0.248	0.326	0.366	0.437	0.202
505	0.176	0.137	0.155	0.133	0.145	0.181	0.113	**0.089**	0.150	0.140	0.124	0.094
41540	0.162	**0.032**	0.099	0.250	0.194	0.192	0.183	0.143	0.245	0.193	0.181	0.140
42724	**0.021**	0.101	0.131	0.125	0.105	0.081	0.139	0.149	0.052	0.154	0.151	0.060
41980	**0.024**	0.107	0.170	0.134	0.098	0.158	0.131	0.106	0.100	0.089	0.088	0.059
507	0.021	0.027	**0.003**	0.165	0.030	0.053	**0.003**	0.338	0.099	0.017	0.005	0.201
Average	0.1146	0.1238	0.1662	0.179	0.136	0.1561	0.1611	0.1648	0.1788	0.1801	0.2031	**0.1455**

Table 20: Grid search results on regression tasks using a 33.33% holdout of the AutoML Benchmark. The values represent the stability values of the portfolios as calculated by the formula provided in Chapter 5. Best performing configuration with $p = 16$ is highlighted.

D Additional Results

Table 21 shows the results for the cluster prediction quality experiment on the regression data sets.

DataSet Name	Predicted	Cluster 1	Cluster 2
wine_quality	**0.721**	0.741	0.723
ypro_4_1	**0.029**	-	0.029
elevators	**0.003**	0.003	0.003
boston	**4.150**	4.442	4.202
Moneyball	22.825	**21.143**	22.734
Brazilian_houses	2707.193	**4.677**	2018.178
abalone	2.102	2.098	**2.097**
us_crime	0.137	0.135	**0.134**
topo_2_1	**0.029**	-	0.029
diamonds	**749.898**	762.351	751.754
sensory	0.997	0.990	**0.945**
socmob	**11.948**	14.107	12.663
Mercedes_Benz_Greener_Manufacturing	**8.440**	-	8.503
colleges	0.146	0.146	**0.145**
house_prices_nominal	**25755.142**	27212.059	27175.521
quake	0.198	**0.194**	0.198
house_16H	**31755.646**	41991.149	32889.983

Table 21: ClustervsAll results for regression.

In Chapter 6, we showed the results of an AutoML run for the four defined configurations on a single data set. Here, we present the final results after the time limit provided to GAMA was reached using single best post-processing. Table 22, Table 23 and Table 24 show the results for binary classification, multi-class classification and regression respectively.

Data set ID	10_min	60_min	10_min_ws	60_min_ws
adult	0.923594	0.925601	0.925924	**0.926511**
kc1	0.801441	0.802823	0.808073	**0.809803**
Satellite	0.994237	0.995585	0.995012	**0.996285**
Internet-Advertisements	0.958102	0.96589	0.964452	**0.96912**
madeline	0.936496	0.916267	0.949869	**0.9547**
philippine	0.86937	0.90229	0.879298	**0.914705**
kick	0.69949	**0.722744**	0.68974	0.713144
Amazon_employee_access	0.85994	0.862141	0.861177	**0.8653**
blood-transfusion-service-center	0.887233	**0.894302**	0.891942	0.894094
Australian	0.941469	0.944124	0.945514	**0.949371**
credit-g	0.810667	0.812714	0.806429	**0.819333**
kr-vs-kp	0.999242	0.998858	0.99899	**0.999403**
jasmine	0.887762	**0.892942**	0.885844	0.891253
qsar-biodeg	0.929744	0.929948	0.929095	**0.930601**
christine	0.748926	0.798384	0.816967	**0.824344**
wilt	0.994606	**0.995251**	0.994453	0.994891
numerai28.6	0.530521	**0.531049**	0.52896	0.52896
nomao	0.971408	0.987089	0.986573	**0.9874**
phoneme	0.965964	0.967799	0.971024	**0.971124**
sylvine	0.988118	0.992922	0.989653	**0.992942**

Table 22: GAMA with and without warm-starting for binary classification data sets on the test split of the AutoML benchmark.

Data	10_min	60_min	10_min_ws	60_min_ws
yeast	1.05374	1.053594	1.049894	**1.048065**
dna	0.118895	0.117023	0.108739	**0.102656**
cnae-9	0.174766	0.162612	0.2103	**0.141187**
steel-plates-fault	0.976342	0.96058	0.961089	**0.936625**
GesturePhaseSegmentationProcessed	1.271437	1.26231	1.316602	**1.254458**
Diabetes130US	0.9079	**0.900035**	0.988155	0.91213
shuttle	0.001544	**0.000506**	0.0009	0.000564
mfeat-factors	0.317156	0.101972	0.107231	**0.098772**
vehicle	0.40636	0.424784	0.413516	**0.406057**
jannis	0.757079	0.820346	0.831641	**0.716671**
connect-4	1.179666	**0.752063**	0.846396	0.824721
segment	0.188393	0.155998	0.165571	**0.155187**
jungle_chess_2pcs_raw_endgame_complete	0.683737	**0.667785**	0.725787	0.691554
wine-quality-white	1.077738	1.072587	1.075761	**1.07178**
fabert	0.779211	**0.778407**	0.95021	0.847913

Table 23: GAMA with and without warm-starting for multi-class classification data sets on the test split of the AutoML benchmark.

Data	10_min	60_min	10_min_ws	60_min_ws
wine_quality	0.723492	**0.721097**	0.722602	0.723707
yprop_4_1	0.028548	0.028492	0.028997	**0.02834**
elevators	0.002494	0.00236	0.00236	**0.00229**
boston	4.989596	3.755377	3.892941	**3.658812**
Moneyball	21.271697	21.232553	21.390523	**21.151408**
Brazilian_houses	1844.120194	5.142092	5.142486	**5.140158**
abalone	2.073261	2.066086	2.081072	**2.057471**
us_crime	0.133131	0.131961	0.131274	**0.130405**
topo_2_1	0.028845	0.028812	0.027065	**0.026042**
diamonds	1106.077502	**1113.3382**	1310.649137	1177.290506
sensory	0.794015	0.791785	0.784353	**0.781132**
socmob	15.299141	15.119294	16.28671	**12.487201**
Mercedes_Benz_Greener_Manufacturing	8.280768	**8.268604**	8.365599	8.31026
colleges	0.150908	0.1492	0.164852	**0.149034**
house_prices_nominal	27697.765827	27028.912379	26045.83275	**25656.378829**
quake	0.188217	0.188247	0.188586	**0.188184**
house_16H	31420.161379	31365.68261	33225.280806	**31131.469199**

Table 24: GAMA with and without warm-starting for regression data sets on the test split of the AutoML benchmark.