EINDHOVEN UNIVERSITY OF TECHNOLOGY

Department of Mathematics and Computer Science
Data Mining Research Group

Master's Thesis

MLTA: a Meta-Learning Toolbox for AutoML

Towards Increased Adoption of Meta-Learning in AutoML

Leighton J.P.C. van Gellecom

Supervisors:
Dr. Ir. Joaquin Vanschoren
Prabhant Singh, MSc

Committee Members:
Dr. Ir. Joaquin Vanschoren
Dr. Mike Holenderski
Alexis Cvetkov-Iliev, PhD

Final version

Eindhoven, Saturday 12th August, 2023

Abstract

AutoML promises to transform the current imperative machine learning paradigm into a declarative paradigm, abstracting away machine learning model development. Key to the development of high-performing AutoML is meta-learning, or learning-to-learn, which aims to reuse prior task experience to learn new tasks more quickly. Meta-learning solutions can consist of a variety of components, including data (and model) similarity measures, characterization methods and the meta-learning model itself. However, selecting these components is difficult because they are hard to implement and evaluate robustly – we cannot directly assess their performance due to a lack of ground truth values. Therefore, designing and developing meta-learning solutions is challenging and requires implementations to behaviorally assess their performance. Yet, current AutoML-directed meta-learning research is plagued by issues concerning reproducibility, accessibility, and comprehensibility limiting the adoption of meta-learning in AutoML systems. In this thesis we propose a Meta-Learning Toolbox for AutoML (MLTA) to easily construct, analyze and compare existing and novel meta-learning approaches and components. The toolbox supports AutoML practitioners in adopting meta-learning for AutoML and may also be valuable for meta-learning related research fields including continual learning, multi-task learning and transfer learning. To provide the toolbox with modularity and extensibility properties we devise an AutoML-directed meta-learning framework and formalize the meaning of its main components. Furthermore, we perform a literature analysis on meta-learning components to identify and compare state-of-the-art approaches, incorporating the most promising approaches in MLTA. To showcase MLTA's utility we use it to perform a case study on warm-starting pipeline search. To that end we create a metadataset of nearly 1 million evaluations, which we make publicly available. Moreover, our case study suggests that (AutoML-directed) meta-learning may be most appropriate for settings with difficult-to-evaluate datasets and time restrictions: we were unable to improve AutoML's performance, but we can decrease the amount of time to find appropriate pipelines. Lastly, we suggest follow-up work – employing MLTA – to explore the effect of meta-learners, the AutoML optimization procedure, their exploration-exploitation trade-offs and the metadataset in warm-starting AutoML.

Preface

This thesis concludes my Master's Program Data Science & AI at the Eindhoven University of Technology (TU/e). I would like to thank my supervisor Joaquin Vanschoren and daily supervisor Prabhant Singh for providing their guidance, feedback and support during the execution of this thesis despite their busy schedules. Their insights have helped me to shape and achieve this work. Moreover, I would like to thank the committee members Joaquin Vanschoren, Mike Holenderski and Alexis Cvetkov-Iliev for taking the time to read, assess and discuss my work. In addition to them I would like to thank my co-students for providing valuable insights and discussions. Lastly, I want to thank my partner, friends and family for their unconditional support, motivating me to get the best out of my thesis. Without these people my thesis would not have been possible.

Eindhoven, Saturday 12th August, 2023
Leighton van Gellecom

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This introductory chapter provides the thesis' main concepts and their importance in Section 1.1. Therein, we introduce AutoML, meta-learning, how they relate, and provide an example of a meta-learning solution and associated challenges. The main concepts of this thesis are formalized and discussed in depth in Chapter 2. In Section 1.2 we formulate the research problem and detail the research objectives which are within the overarching research theme of facilitating increased adoption of meta-learning in AutoML. In Section 1.3 we give an overview of the thesis' contents addressing the research aim.

1.1 Motivation and Challenges

AutoML Machine learning (ML) concerns models that learn from data. Currently, most machine learning follows an imperative paradigm, requiring manual specification of model development steps. To make machine learning more accessible, imperative ML can be transformed into declarative ML. In the declarative paradigm, one only needs to specify the desired input and output, without any of the steps in-between. Going from imperative to declarative ML is where AutoML comes in. Though the goals and outcomes of AutoML are manyfold (see Section 2.1), the main interest is, as the name suggests, in automation of machine learning. Thus, AutoML promises to achieve declarative ML through abstraction of model development.

AutoML has become a vast research field, with popular open-source contributions [54, 36, 48], and multiple companies provide AutoML services[1]. However, for now, most AutoML systems are not end-to-end and are unable to rapidly and consistently achieve expert-level, human, performance in [146]. Moreover, sometimes, configuring an AutoML system may be more complex than configuring a ML model. Therefore, more research is necessary to fulfill the AutoML promise of declarative ML. A critical aspect of any AutoML system is efficiently developing ML models because the model development search space is huge. For instance, we can specify the model class, and characterize how it should learn, from differently processed data all with varying contexts. As explained, automation of machine learning – especially model development – is far from trivial. It gives rise to the question: "can we learn to efficiently develop learning models?"

[1] Google cloud AutoML (`https://cloud.google.com/automl`), Amazon Sagemaker (`https://aws.amazon.com/machine-learning/automl`), Databricks AutoML (`https://www.databricks.com/product/automl`) and Microsoft AutoML (`https://www.microsoft.com/en-us/research/project/automl/`).

Meta-Learning One research field in particular is trying to answer that question: meta-learning. Meta-learning is closely connected to the definition of learning: "an increase, through *experience*, of problem-solving ability" [130]. Meta-learning, or learning-to-learn, covers any type of (machine) learning using prior task experience to learn new tasks more quickly [124].

Meta-learning covers many different challenges, all with a fundamental notion of reusing previously learned information. Typically, meta-learning research builds upon the assumption that similar tasks are useful as previous experience. Reusing prior experience is especially important for AutoML, whose search space explodes when considering full ML pipelines; the authors of AutoWeka-MCPS [112] enlarge the search space from 22.000 to 820 billion.

The *assumption* that previously-seen similar tasks are useful for current learning tasks, is solidified through similar findings in cognitive science [93, 77]. However, with automation in mind, we distantiate from cognitive science by distinguishing between different forms of information: quantitative and qualitative information. Humans can express similarity through qualitative information, but automation is more feasible for quantitative information. Therefore, to enable identifying useful experiences, similarity must be quantified, which is the goal of similarity measures. Likewise, we could employ (quantifiable) characteristics of previous and to-be-solved tasks and their solutions – i.e. dataset and ML model characterizations (see Section 2.2) – to aid learning a novel task. Quantification of task similarity and characterization of tasks and solutions are fundamental to meta-learning, which in turn is suitable to solve challenges in AutoML – meta-learning allows to more efficiently explore the search space.

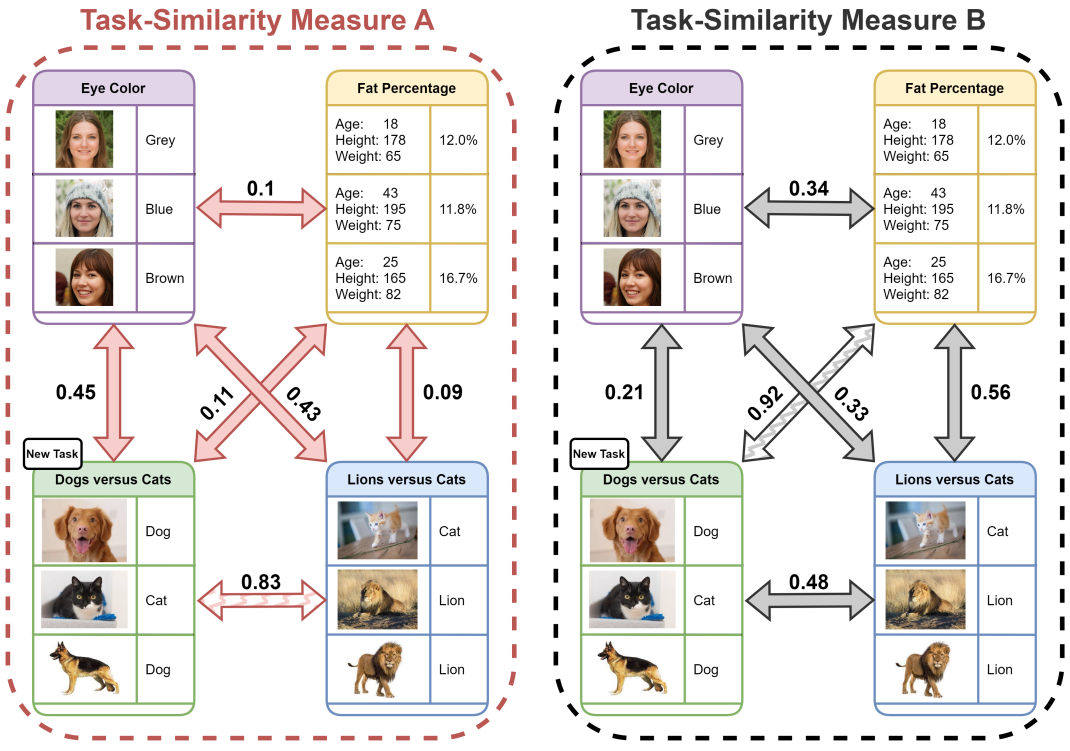

Figure 1.1: An example of a meta-learning application utilizing two *hypothetical* dataset-similarity measures, A and B, on four tasks. The values near the arrows indicate the *made-up* similarity between the datasets. This example is created for illustrative purposes, contemporary similarity measures hardly support multiple data modalities. The humans shown do not exist, they are created with thispersondoesnotexist.com.

Meta-Learning Example To make the main concepts and their usage more concrete we utilize Figure 1.1 illustrating a *simplified and fictional* meta-learning application and the role of similarity. Later, in Section 2.2 we provide a more elaborate discussion of meta-learning approaches and their components. Figure 1.1 displays two *hypothetical* task-similarity measures, A and B, assigning similarity values to pairs of the following four distinct tasks: (1) classifying eye color based on images; (2) predicting the fat percentage using tabular data; (3) classifying dog versus cat images; (4) classifying lion versus cat images. The goal of a task-similarity measure is to quantify the similarity between pairs of tasks. The obtained similarity values can be used for knowledge transfer (meta-learning) applications.

Consider the meta-learning scenario transferring knowledge, say the choice for machine learning algorithm, from previous tasks to a new task. Suppose we want to learn a new task, the dogs-versus-cats classification task. In addition, we have previously solved the three aforementioned tasks: eye color, fat percentage, and lions versus cats. Task-similarity measure A identifies the lion-versus-cat task as the most similar to our current task with a hypothetical similarity value of 0.83. Instead of learning what algorithm is most suitable for our new task, we could employ a simple meta-learning strategy: *use the best-performing algorithm from the most similar task.*

The ML algorithm from the previously-learned lions-versus-cats task likely transfers well to our new dogs-versus-cats task, in turn accelerating the learning process. On the other hand, task-similarity measure B identifies the fat percentage task to be the most similar to our dogs-versus-cats task with a similarity value of 0.92. Due to the task's varying data modality – tabular versus image data – we do not expect successful knowledge transfer. Hence, task-similarity measure B likely identifies and transfers an algorithm poorly suited to the new task, consequently yielding inferior performance compared to learning the new task from scratch – a phenomenon referred to as negative transfer [110].

Thus, similarity measures – and other meta-learning components – can influence the performance of meta-learning applications but selecting these components is challenging; we cannot directly measure their performance since there is *no* ground truth. Typical meta-learning applications consist of multiple difficult-to-select components whose performance may be dependent on the application and available prior knowledge. Our simple example already required specification of two interacting components: the similarity measure, and the meta-learner. However, meta-learning approaches may use varying types of components (see Section 2.2.1 and 2.2) further complicating the development of a meta-learning application. Due to its dependence on the application and prior knowledge, and the lack of ground truth, we must resort to the implementation of and experimentation with meta-learning approaches.

There exist at least four benefits to using meta-learning and each can be framed as a cost reduction through more efficient learning, requiring less for similar performance. The benefits include reducing costs in experimentation, data collection, training, and missed opportunities. Experimentation costs could be reduced by requiring fewer design options – meta-learning can exclude or recommend them – to achieve satisfactory performance. Data collection, including annotation, costs can be reduced because we may need to collect fewer observations for equal performance – meta-learning may find a better solution within a given time frame. Less data for similar performance may lead to reduced training costs because training time for ML algorithms typically scales with the amount of data. Lastly, for an equal amount of data, effectively using meta-learning may lead to a more suitable model – for instance higher accuracy-related performance – in turn reducing opportunity costs.

1.2 Formulation of the Problem and Objectives

In the previous subsection, we introduced meta-learning solutions and correspondingly established their complexity and explained their potential benefits. Despite an abundance of AutoML systems [83] and meta-learning approaches for AutoML (see Section 2.2), only a limited amount of open-source AutoML systems incorporate meta-learning, examples being Auto-Sklearn 1.0 [36], Auto-Sklearn 2.0 [34], Auto-PyTorch Tabular [145] and AlphaD3M [28], with the first 3 stemming from a single research group.

As a main challenge for this thesis, we identify *the limited adoption of meta-learning for AutoML*, and we attribute it to issues concerning *reproducibility, accessibility and comprehensibility*, which unfortunately reinforce each other.

- Existing AutoML-directed meta-learning works tend to be irreproducible and inaccessible because their specifics are not discussed and their implementations are not provided in a well-maintained and documented way.
- Additional reproducibility issues are caused by approaches' dependence on varying sets of prior experiences – in the form of a metadataset – which are often not made publicly accessible.
- With varying metadatasets it is challenging to compare meta-learning approaches, limiting comprehension and reducing the impact of any related meta-learning work.
- Though there is increased research towards meta-learning components, there is no clear overview nor comparison of them, again limiting comprehension and accessibility.
- Implementations for approaches and components are overly tailored to the specifics of their use-case. Subroutines, such as computing dataset characterizations or similarity, are overly dependent on their interpretation, the meta-learning algorithm, and metadatasets' materialization. Due to implementations' overdependence on the use-case convenient re-use is diminished, negatively affecting reproducibility and prototyping pace, successively reducing the number of novel ideas and comparisons between them.
- The degree of accessibility is limited due to the lack of open-source initiatives providing unified access to implementations. The lack of such initiatives additionally culminates in time-consuming prototyping and experimentation, a problem especially severe for meta-learning components whose performance can only be established through experimentation due to the lack of a ground truth.

To sum up, current AutoML-directed meta-learning research is plagued by issues concerning reproducibility, accessibility and comprehensibility, stemming from the lack of standardized implementations, difficulty in creating prototypes and comparisons, and the absence of an AutoML-focused meta-learning framework. The lack of reproducibility, accessibility and comprehension limits the adoption of meta-learning for AutoML.

Striving towards increased adoption of meta-learning in particularly AutoML systems, we propose to alleviate the aforementioned issues by introducing **MLTA:** a Meta-Learning Toolbox for AutoML, focusing on tabular data for supervised classification.

MLTA provides a modularized and extensible framework for AutoML-directed meta-learning solutions, standardizing development though its framework, providing unified access to common components, facilitating comparisons and promoting code-reuse through its modularization. The framework and toolbox components are designed agnostically to the AutoML-related task it solves, e.g. warm-starting search, performance prediction or training time estimation. Lastly, we hope that through promoting a pursuit of standardized practices we may inspire the community to arrive at a consensus for comparing meta-learning solutions in the form of a benchmark.

The aforementioned challenge – increased adoption of meta-learning in AutoML – sets the stage for this thesis, but it does not allow assessing its success. Therefore we define a more concrete research objective with respect to MLTA.

- **Research objective:** create a toolbox to easily construct, analyze and compare existing and novel meta-learning approaches and components.

In a broader sense, we thereby aim to *partially* mitigate reproducibility, accessibility and comprehensibility issues (with meta-learning research) to support AutoML practitioners to adopt meta-learning in AutoML. This thesis is most useful for research(ers) in the areas of AutoML, meta-learning and related research areas.

In Section 2.2 we explain the relation of multi-task learning, transfer learning, NAS and continual learning to meta-learning. These, and additional related research fields could employ components which are present in meta-learning systems, making them potential beneficiaries of our toolbox. To support that claim we provide an overview of the usage of similarity measures in the aforementioned research fields in Appendix A. Furthermore, through potential accelerated research towards AutoML systems, there may be a benefit for their users. These can be expert and non-expert ML enthusiasts from companies or completely different research fields.

The proposed toolbox distinguishes itself from other meta-learning toolboxes by its focus on AutoML systems for supervised tasks on tabular datasets. In contrast, current meta-learning toolboxes such as learn2learn [9], NASLib [111] TorchMeta [24] and BOML [76] are mostly directed towards gradient-based methods in NAS, meta-reinforcement learning, and few-shot learning. MLTA differs from works providing meta-learning components for tabular datasets, such as MFE [5], by its broadness focusing on all main components of an AutoML-related meta-learning application. MLTA extends MFE's dataset characterization functionality by additionally including more recent, and potentially higher-performing, dataset characterization methods and corresponding similarity measures. Apart from MLTA's divergent focus, MLTA distinguishes itself from other approaches by explicitly incorporating prior experiences (ML pipelines) in its meta-learning framework.

This thesis shares great similarity with GAMA [44], which aims to support the development of novel ideas for AutoML systems and its components by providing a modularized framework. Like GAMA, our work intends to support AutoML research, but it distinguishes from GAMA by its focus on meta-learning for AutoML – a feature currently not incorporated in GAMA. The need for our toolbox is strengthened by the success of related works such as GAMA, MFE, learn2learn and NASLib.

1.3 Thesis Outline

The remainder of this thesis is structured as follows. In Chapter 2 we explore our problem setting in-depth, detailing AutoML and exploring the state-of-the-art for AutoML-associated meta-learning. Moreover, in Chapter 2, we provide a structured overview of existing approaches for typical meta-learning components, which serves as a basis for Chapter 3 introducing "MLTA", the meta-learning toolbox. In Chapter 3, we compare the state-of-the-art presented in Chapter 2 to decide which components should be provided by the toolbox. In Chapter 4 we showcase the toolbox achieves the research objective by performing a case study – employing MLTA – on warm-starting pipeline search. In Chapter 5 we summarize key take-aways and reflect on our contributions by contextualizing them and assessing their implications.

Chapter 2

Related Work

In this chapter we introduce the main preliminaries of this thesis: AutoML, associated meta-learning solutions and their components. In their respective subsections we provide formulations, where appropriate a formalization, and explain the relation to the thesis' goals. This chapter additionally serves as an overview of state-of-the-art AutoML-related meta-learning works, explaining necessary details for their suitability comparison in Chapter 3.

2.1 AutoML

2.1.1 Problem Setting and Toolbox Scope

In Section 1.1 we already established that AutoML promises to make machine learning declarative, instead of imperative. In an AutoML survey [146] it is explained that AutoML aims to make design decisions in a data-driven, objective, and automated way. The authors exhibit the vast size of the research field by characterizing the numerous challenges in AutoML, they include:

> "automating data collection and experiment design; automating data cleanup and missing data imputation; automating feature selection and transformation; automating model discovery, criticism, and explanation; automating the allocation of computational resources; automating hyperparameter optimization; automating inference; and automating model monitoring and anomaly detection."

The field of AutoML is vast and diverse, as apparent from the many distinct challenges, therefore we need to scope the toolbox by restricting ourselves to AutoML for:

1. Model development,
2. on tabular datasets,
3. in a supervised learning setting,
4. performing classification tasks,
5. considering non-deep-learning ML pipelines (see 2.1.3).

We introduce restriction (5) to preclude deep-learning models and corresponding pipelines, to avoid overlap with the research field of Neural Architecture Search (NAS). It does however not restrict the toolbox' usage in NAS works; the toolbox could support works using both classical and NN-based ML models like Auto-PyTorch Tabular [145].

Restriction (2) concerns the focus on tabular data, whose definition we adopt from Borisov et al. [16], namely: a heterogeneous dataset with a fixed number of features that are either continuous or categorical, typically presented in a table with data points as rows and features as columns. Restricting ourselves to tabular data strengthens the focus on AutoML and avoids overlap with NAS, because tabular data's properties favor more traditional machine learning methods over deep learning [16].

Restriction (1) concerns "model development", which encompasses multiple AutoML challenges, including the automation of: feature selection and transformation; model discovery; hyperparameter optimization and the allocation of computational resources. Such challenges could be partially solved by employing meta-learning systems, which for instance carry out performance predictions or training time estimation. Thus, meta-learning solutions and components facilitated by MLTA can help achieve high-performing AutoML, desirable for a variety of reasons. An AutoML system would enable non-experts to build ML applications [146]. Moreover, in the pursuit of increasing accessibility and democratization, we obtain tools and practices for more systematic and efficient machine learning [52]. An example of such a tool is Hyperband [71], for the challenge of hyperparameter optimization, whose success is demonstrated through its incorporation in popular libraries such as Keras [21].

2.1.2 Problem Formulation

In AutoML we search for solutions, termed configurations, in a configuration search space. AutoML systems vary by their search method and search space, popular frameworks include Autogluon [31], auto-sklearn 1 [36], auto-sklearn 2.0 [33], FLAML [127], GAMA [44], H2OAutoML [69], AutoWeka [48] and TPOT [96]. Thus, the definition of a configuration differs across its usage, but typically a configuration is a choice in the model development search space. A configuration in the full model development search space thus includes any aspect defining a part of the model as a whole, it can include options for hyperparameters, algorithms, pipelines, architectures, and so forth.

A main task of AutoML systems is to find an adequate configuration, typically by searching the configuration space, which is challenging due to multiple factors [35]. First, evaluations can be, and typically are, expensive. This issue worsens over time, with developments of extremely complex models, such as Megatron-Turing NLG containing 530 billion parameters [116]. Moreover, datasets are getting larger, increasing evaluation time even further. Second, the configuration space is often complex and high-dimensional. Furthermore, it may not be clear which dimensions of the configuration are important.

Choosing an adequate configuration is not only challenging but also important. There are numerous findings indicating the effect of configuration choice on performance. For instance, in hyperparameter tuning, performance can be improved over the default settings [81, 95, 113]. Moreover, weaker well-tuned models can outperform stronger poorly-tuned models [82]. Additionally, findings suggest performance depends on configuration choices for the model [95], model and hyperparameters [121], pre-processing techniques [4, 106], and architecture [78, 138].

The exact materialization of what constitutes a configuration, or its search space, depends on the AutoML formulation that is chosen. Some works, like H2OAutoML [69], employ the Combined Algorithm Selection and Hyperparameter Optimization [121] formulation commonly termed CASH. Others, like TPOT [96], additionally include feature engineering steps, and some approaches extend this by including data cleaning steps, e.g. AutoWeka-MCPS [112].

In this thesis, we want to focus on a formulation of AutoML allowing us to provide the most general insights about meta-learning for AutoML. We argue for the formulation of AutoML including the search for preprocessing steps performing feature engineering, algorithm selection and hyperparameter tuning for each of those, a formulation similar to that of the AutoML frameworks TPOT [96] and GAMA [44]. By including preparatory steps in our formulation we align better with the goal of true automation of ML; it is reported practitioners spend the majority of their time in such preparatory steps [102]. This formulation is similar to CASH, but additionally includes preprocessing steps performing feature engineering. On the other hand it searches for pipelines, like approaches additionally taking into account data preparation steps. However, in contrast to those approaches, our formulation does not include data preparation steps because information transferability is less straightforward for them due to their inherent dependence on the dataset.

2.1.3 Problem Formalization

Our formalization of the AutoML problem, presented in Definition 1, follows that of Gijsbers [42] adapting the interpretation of Zoller [146]. It generalizes previous formulations and interprets AutoML as a pipeline creation problem with the objective to minimize the empirical risk of a pipeline according to some evaluation procedure. By interpreting a pipeline as a DAG, the formalization decomposes AutoML by separately considering the pipeline structure and its materialization in the form of algorithms and hyperparameters.

Definition 1 (Pipeline Creation Problem)**.** *Let a set of algorithms $\mathcal{A}$ with an according domain of hyperparameters $\Lambda_{(\cdot)}$, a set of valid pipeline structures G and dataset $\mathcal{D}$ be given. The pipeline creation problem consists of finding a pipeline structure in combination with a joint algorithm and hyperparameter selection that minimizes the loss*

$$(g, \boldsymbol{A}, \boldsymbol{\lambda})^* \in \underset{g \in G, \boldsymbol{A} \in \mathcal{A}^{|g|}, \boldsymbol{\lambda} \in \Lambda}{\operatorname{argmin}} \mathcal{R}(\mathcal{P}_{g,\boldsymbol{A},\boldsymbol{\lambda}}, \mathcal{D}) \tag{2.1}$$

within a given resource budget B.

where:

- $g \in G$ is a graph from the set of all valid graphs,
- $\boldsymbol{A} \in \mathcal{A}^{|g|}$ is a vector specifying for each node in graph g the algorithm from the set of algorithms $\mathcal{A}$,
- $\boldsymbol{\lambda} \in \Lambda$ is a vector specifying the hyperparameter configuration of each algorithm from the set of all possible configurations Λ,
- B is a resource budget, for instance in time or iterations.
- $\mathcal{P}_{g,\boldsymbol{A},\boldsymbol{\lambda}}$ is the pipeline defined by its algorithm(s), hyperparameter(s) and structure.
- $\mathcal{R}$ is the risk of pipeline $\mathcal{P}_{g,\boldsymbol{A},\boldsymbol{\lambda}}$ on dataset $\mathcal{D}$.

The AutoML formalization, through its interpretation as a DAG-based pipeline creation problem, is rather broad. Therefore, to align the formalization with the focus of our work we impose one further restriction for this thesis. For the search space of the pipeline creation problem we explicitly do not consider algorithms in $\mathcal{A}$ performing data preparation steps, e.g. those performing formatting, encoding and imputation. However, this need not imply pipelines lack these steps, they may for instance be fixed or rule-based. We simply state that we exclude data preparation algorithms from the search space because, with meta-learning in mind, they are difficult to transfer due to their inherent dependence on the dataset. We impose no further restrictions on G, B, Λ and $\mathcal{A}$ nor the evaluation procedure estimating $\mathcal{R}$, because these specifications are relevant for a particular AutoML system or use-case, not for the meta-learning toolbox introduced in this thesis.

2.2 Meta-Learning

In Section 1.1 we discussed meta-learning, or learning-to-learn, and its relation to AutoML. A brief recap, meta-learning is "the science of systematically observing how different machine learning approaches perform on a wide range of learning tasks, and then learning from this experience, or meta-data, to learn new tasks much faster than otherwise possible. Additionally, meta-learning approaches typically assume similar tasks to be useful previous experiences.

Identifying the main concepts in the definition of meta-learning and its assumptions yields a list of main components for a meta-learning system: a set of prior experiences (metadataset), machine learning approaches (configurations), a learning task (dataset) and learning the new task (meta-learner). In the forthcoming subsections we discuss methods related to these main components in more depth, giving an overview of the state-of-the-art, except for the metadataset because we are unaware of any works explicitly investigating it.

This preliminary section is most related to a meta-learning survey [124], but distinguishes itself by including more recent approaches and the explicit focus on and distinction between AutoML-related meta-learning components. Before exploring the main components of meta-learning systems we outline the field of meta-learning and its related fields.

The broad definition of meta-learning allows it to include lots of different approaches or even entire research fields such as transfer learning or Neural Architecture Search (NAS). Though current meta-learning research is mostly focused on applications in deep learning [50], the research field is far broader. According to [124] meta-learning approaches are typically part of one of three categories, they either learn from model evaluations, task properties, or prior models. The latter refers to transfer learning, which concerns approaches that account for differences in two distributions [61], commonly termed a source and task distribution. It accounts for these differences using knowledge transfer techniques. Transfer learning can be seen as a form of meta-learning where previously learned models are transferred.

Besides transfer learning and NAS, the field of meta-learning is related to multi-task learning and continual learning. Multi-task learning (MTL) is an approach using information in the training signals of related tasks; utilizing a shared task representation [17]. Like meta-learning, MTL aims to use information learned from other tasks. MTL uses task information by jointly learning tasks, while meta-learning learns tasks separately. Hence, in MTL you learn from a task by learning it directly, while in meta-learning you learn from an already completed task.

Continual learning considers a setup to learn continually from an infinite stream of data trying to gradually extend knowledge and use it for future learning [23]. From the definition of continual learning, we point out the connection to meta-learning and multi-task learning. As with meta-learning, continual learning aims to reuse learned information. As with multi-task learning, continual learning learns from more than one task, though not simultaneously. Distinguishing it from MLT and meta-learning, continual learning not only aims to learn from other tasks, but to keep on learning.

2.2.1 Characterization Methods and Similarity-Measures

In this subsection we discuss preliminary work on configuration characterizations, dataset characterizations and dataset similarity measures, which are common meta-learning components typically interoperating with a meta-learner (see Section 2.2.2).

A dataset characterization is a description of the most important or typical characteristics of a dataset or its distinctive nature. Comparably, a configuration characterization describes the distinctive nature or most important or typical characteristics of a configuration. Though similar to a dataset characterization, a dataset similarity measure solely quantifies the similarity or dissimilarity between datasets. In this thesis, we discriminate between dataset characterizations and similarity measures because their materialization and usage vary despite many works simultaneously providing solutions for both.

Typically a dataset characterization takes the form of a vector (or matrix), while similarity measures represent the (dis)similarity between datasets using a scalar value, i.e. a similarity value. Their usage varies with some meta-learners requiring the characteristics of a dataset, while others only employ its relation to other datasets (see Section 2.2.2). Distinguishing between properties, a dataset characterization method can operate in isolation of other datasets while a similarity measure cannot.

As mentioned, many works provide solutions for both a dataset characterization method and a similarity measure. After all, a dataset characterization method providing fixed-size vectorized representations of datasets allows to be extended to a similarity measure by computing the similarity between characterizations. Please note the ambiguity of the term "dataset *characterization*" — either referring to a dataset's distinctive nature *or* its characteristics — captures its possible extension well. However, a characterization method is not per se extensible because a characterization's dimensionality may depend on properties of the dataset it characterizes.

A characterization method extended to be a similarity measure benefits from a key property of a characterization method: a characterization exists in isolation of other datasets. This property implies a characterization-based similarity measure can be computed efficiently with respect to the number of datasets; to compute all N^2 pair-wise similarity values between N datasets we only need to execute a characterization method N times. Contrarily, a similarity measure typically needs to be executed N^2 times to compute all N^2 pair-wise similarity values.

To measure the (dis)similarity between two equally-sized vector-based characterizations we can chose among multiple approaches, including the cosine similarity or Euclidean distance. A similarity measure employing the former assumes the difference in angles captures the similarity well, while the latter assumes distances do. It is unclear which approach is most appropriate, the authors of Task2Vec [1] show the dataset characterization's norm encodes the datasets' difficulty, while another work [87] shows the angles between their dataset characterizations are preserved when varying meaningless changes in training factors. These findings highlight more research

is needed to choose approaches for measuring the (dis)similarity between characterizations in a well-substantiated manner. To be precise in our terminology and transparent in our classification of approaches we provide formal definitions of the previously discussed concepts.

Definition 2 (Dataset Characterization Method)**.** *A dataset characterization method* $d : \mathcal{D} \mapsto \mathcal{C}_{\mathcal{D}} \in \mathbb{R}^{n \times m}$ *with* $n, m \in \mathbb{Z}^+$ *maps a dataset to a characterization* $\mathcal{C}_{\mathcal{D}}$ *representing* $\mathcal{D}$*'s main characteristics. Where a dataset* $\mathcal{D} = \{(\boldsymbol{x}_i,\ \boldsymbol{y}_i)\}_{i=1}^{N}$ *consists of feature vectors* $\boldsymbol{x}_{\cdot} \in \mathbb{R}^d$ *having* $d \in \mathbb{Z}^+$ *features and target vectors* $\boldsymbol{y}_{\cdot} \in \mathbb{R}^k$ *having* $k \in \mathbb{Z}^+$ *dimensions.*

Typically $k = 1$ for a target vector $\boldsymbol{y}_{\cdot} \in \mathbb{R}^k$ having $k \in \mathbb{Z}^+$ dimensions, but not always, e.g. with multi-label classification $k > 1$. Note we do not include a property for a characterization specifying it uniquely identifies a dataset, to allow for a broad overview. Whether this encoding-like property would be preferable for dataset characterization methods is unknown.

Definition 3 (Pipeline Configuration Characterization Method)**.** *A pipeline configuration characterization method* $c : (g, \boldsymbol{A}, \boldsymbol{\lambda}) \mapsto \mathcal{C}_{\mathcal{C}} \in (\mathbb{R} \cup B^*)^{n \times m}$ *with* $n, m \in \mathbb{Z}^+$ *maps a pipeline configuration to a characterization* $\mathcal{C}_{\mathcal{C}}$ *representing its main characteristics. A pipeline configuration* $(g, \boldsymbol{A}, \boldsymbol{\lambda})$ *is a tuple consisting of a pipeline structure, corresponding algorithms and hyperparameters with* $g \in G$ *is a graph from the set of all valid graphs,* $\boldsymbol{A} \in \mathcal{A}^{|g|}$ *is a vector specifying for each node in graph* g *the algorithm from the set of algorithms* $\mathcal{A}$*, and* $\boldsymbol{\lambda} \in \Lambda$ *is a vector specifying the hyperparameter configuration of each algorithm from the set of all possible hyperparameter configurations* Λ*. Let* $B = \{a, b, .., y, z\} \cup \{0, 1, ..., 8, 9\}$ *be the alphabet consisting of the union of all letters in the Latin-script alphabet and all single digits in the decimal system. Let a string* $s \in B^*$ *be a sequence of characters in* B *where* B^* *denotes the Kleene star on alphabet* B*.*

In Definition 3 we do not specify which configuration characteristics are captured by the pipeline configuration method. The lack of its specification permits methods only capturing part of the configuration, e.g. those neglecting the hyperparameter configuration or the pipeline structure. This impreciseness is deliberate, for it allows a broad spectrum of methods to be consistent with the definition. Moreover, we are deliberately ambiguous with respect to the realization of characterization, it can be a vector or matrix consisting of real-valued numbers and/or strings (a sequence of characters). The broad definition is necessary, because currently only few such methods exist (see 2.2.1.3) and we must avoid steering development in a particular direction by tailoring the definition to a specific method. Intuitively, a pipeline configuration characterization method is akin a dataset characterization method, but for configurations. Therefore, we postulate the possibility to have configuration-similarity measures, extending their corresponding characterizations. We are however not aware of any such works and will thus not consider them in this preliminary chapter.

Definition 4 (Dataset-Similarity Measure)**.** *A dataset-similarity measure* $s : \mathcal{D}_1 \times \mathcal{D}_2 \mapsto \mathbb{R}$*, is a mapping from two datasets* $\mathcal{D}_1$ *and* $\mathcal{D}_2$ *to a real number indicative of the (dis)similarity between the datasets. Where the two datasets are given as* $\mathcal{D}_1 = \{(\boldsymbol{x}_{1,i},\ \boldsymbol{y}_{1,i})\}_{i=1}^{N_1}$ *with feature vectors* $\boldsymbol{x}_{1,\cdot} \in \mathbb{R}^{d_1}$ *having* $d_1 \in \mathbb{Z}^+$ *features and target vectors* $\boldsymbol{y}_{1,\cdot} \in \mathbb{R}^{k_1}$ *having* $k_1 \in \mathbb{Z}^+$ *dimensions, and* $\mathcal{D}_2 = \{(\boldsymbol{x}_{2,i},\ \boldsymbol{y}_{2,i})\}_{i=1}^{N_2}$ *with feature vectors* $\boldsymbol{x}_{2,\cdot} \in \mathbb{R}^{d_2}$ *having* $d_2 \in \mathbb{Z}^+$ *features and target vectors* $\boldsymbol{y}_{2,\cdot} \in \mathbb{R}^{k_2}$ *having* $k_2 \in \mathbb{Z}^+$ *dimensions.*

Our formalization of a dataset-similarity measure (Definition 4) allows for datasets with varying number of instances, features and target dimensionality. The subsequent overview also includes measures with additional restrictions, which are indicated if present. Observe that in our definition of a dataset-similarity measure we restrict ourselves to pair-wise measures. We claim most interesting applications of similarity measures consider pair-wise similarities, accordingly in this thesis we only consider pair-wise similarity measures. Thus, a dataset similarity measure quantifies the similarity or dissimilarity between exactly two datasets.

In the overview of dataset similarity measures we focus on measures relying on both targets and features, because we have restricted ourselves to a supervised setting (see Restriction 2). It is possible to quantify the similarity between datasets with and without using targets, but there are considerable differences between the power and applicability of these measures as showcased in [1, 7]. For instance, measures incorporating targets are suited to distinguish between two datasets with equal features but varying targets. In contrast, measures which do not take into account targets cannot distinguish between different tasks using the same features. On the other hand, those measures' non-reliance on targets can be a strength, it makes them suitable for unsupervised settings. In this preliminary chapter, we do not discuss similarity measures solely relying on features because research towards them is limited and due to our supervised setting.

Transferability measures Transferability measures are different from, but very much related to, dataset-similarity measures. Transferability measures, also known as transferability metrics, quantify or predict the amount of transferability between a source and a target task [2], e.g. the performance difference of a model on to the source task and target task. Similarity and transferability measures differ on two main aspects, their broadness and their availability of ground truth information. Transferability measures are less broadly applicable than dataset-similarity measures because their inputs (may) differ. Typically, transferability measures include source models in their input, whereas similarity measures take *only* datasets as input. Moreover, some transferability measures impose additional restrictions on their input. Often the source models are assumed to be neural networks, because they allow for the transferability of their structure and parameters. Transferability measures are model-oriented and more specific due to additional restrictions on the input. This makes transferability measures suited for specific use cases like transferring only certain parts of a neural network [104], or ranking checkpoints to transfer from [73]. Transferability measures only relying on data to characterize the dataset similarity can be framed as dataset-similarity measures, yet not all similarity measures are transferability measures. Unlike similarity measures, the development of transferability measures is guided by ground truth. That is, we can compare transferability measures on their performance in transfer learning. For similarity measures there is no ground truth, one can only evaluate their use for downstream applications such as a meta-learning application. Transferability measures can be used in meta-learning applications, but these are typically not oriented towards tabular AutoML. Therefore, we do not provide an elaborate overview of transferability measures. However, due to their resemblance with similarity measures, possibly influencing their development, we give a brief overview. Popular transferability measures include LEEP [90], LogMe [141] and GBC [98]. These methods measure how well a model transfers to a target task or dataset. To estimate the transferability they use the source model to embed the target data. Other transferability measures include [10, 118, 51, 3, 73, 119], for an elaborate discussion on transferability metrics consider [2, 120].

Up until now we have introduced configuration characterizations, dataset characterizations and similarity measures, what follows are overviews of them (see Sections 2.2.1.3, 2.2.1.2 and 2.2.1.1). In these overviews we identify the state-of-the-art and provide corresponding explanations to introduce their main aspects. Thus, assessing their aspects to determine their suitability for meta-learning is not part of this preliminary chapter, rather, this is carried out in Chapter 3 (detailing the meta-learning toolbox).

One particular approach is excluded from the overview which does not conform to the provided Definitions 2, 3 and 4 of a configuration characterization method, nor a characterization method or similarity measure. The approach [29] computes a language-based embedding of the metadata as a representation of the task. We exclude the approach because the manifestation of metadata is greatly dependent on human interpretation. Note that datasets are also products of human interpretation, but they tend to be more objective, specific, complete, consistent, and less language dependent.

2.2.1.1 Dataset Similarity Measures

As mentioned in Section 2.2.1, characterization methods with fixed dimensionality – n, m remain equal in $\mathcal{C}_{\mathcal{D}} \in \mathbb{R}^{n \times m}$ – allow to be extended to a characterization-based similarity measure. Therefore, we do not explicitly treat these approaches in this section.

OTDD Optimal transport dataset distances (OTDD) [7] can be considered a model-agnostic task-dissimilarity measure, an instantiation of s. Thus, computing the OTDD, i.e. s, does not involve training a model. The OTDD is a hybrid Euclidean-Wasserstein distance, it possesses the properties of a distance metric. Hence, the output of OTDD is inversely related to the similarity between datasets $\mathcal{D}_1, \mathcal{D}_2$. OTDD treats the distance between two samples as a sum of their feature-based and label-based distance (see Figure 2.1). The method assumes to have access to a metric to compute the feature-based distance. For the label-based distance, OTDD models labels as distributions over feature vectors, therefore the label sets can be completely disjoint. The distance between the label distributions is determined by computing the statistical divergence between their feature-based distributions. Optimal transport is used as the measure for statistical divergence. Then, the distance s between the two datasets can be determined by computing the optimal coupling between its samples. Computing both the feature and label-based distances is computationally expensive, therefore the authors propose to approximate the label distribution as a Gaussian distribution. Although this computational trick speeds up computation, OTDD still scales cubically in the number of features, and quadratically in the number of samples. Despite OTDD's computational burden, the paper does not compare it to baselines to justify the computational complexity.

2.2.1.2 Dataset Characterization Methods

Meta-Feature-Based Characterization Perhaps the most straightforward dataset characterization method d is to compute $\mathcal{C}_D$ using meta-features: an approach similar to the featurization of datasets, but featurizing datasets themselves instead of samples therein to capture dataset's main characteristics. Thus, featurization is employed at one higher level of abstraction (dataset over sample therein) and therefore the featurized representation of the dataset is termed meta-features.

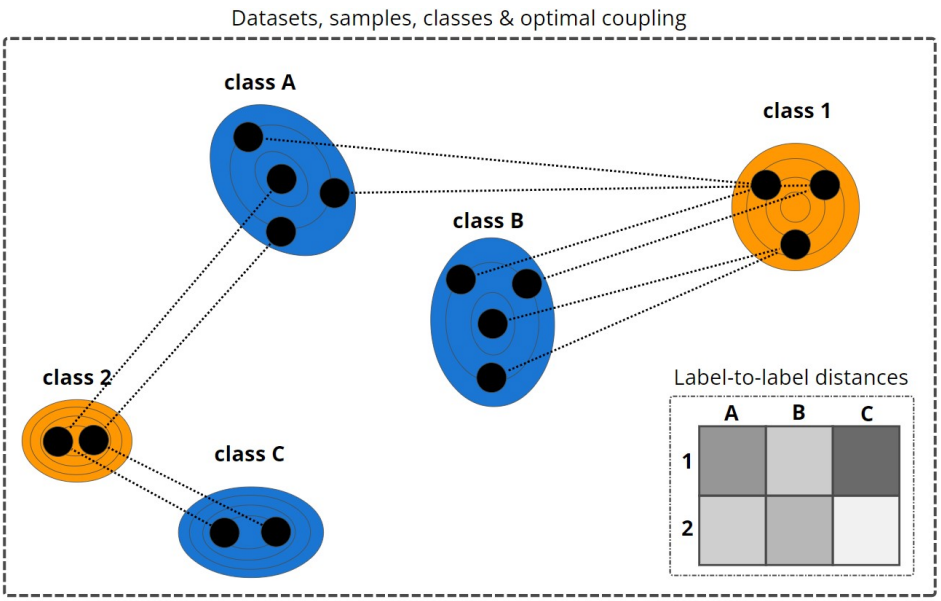

Figure 2.1: OTDD [7] example depicting two *made-up* 2-dimensional datasets with their classes: dataset blue with classes A, B and C; and dataset orange with classes 1 and 2. Data points are depicted as black dots, within their Gaussian-approximated label distribution, whose color depicts the dataset they belong to. The lower right corner depicts the inter-dataset label-to-label distances (darker is larger) which are computed using the Gaussian approximations, which together with a ground metric on features allows to compute the optimal coupling between the datasets' data points (grey dashed lines). Observe dataset blue contains 10 data points while dataset orange contains half of that. Thus, if the mass from one blue point only transfer to exactly one orange point (which happens in the example but need not be the case) there must be 2 blue data points connected to each orange data point. Also note the datasets' label sets are completely disjoint, a key property of OTDD available due to labels being represented by the Gaussians approximating their feature-based distributions.

More formally, a meta-feature f is a function $f : \mathcal{D} \mapsto \mathbb{R}^k$ that upon application to a dataset $\mathcal{D}$ returns a set of k values characterizing the dataset [108]. Moreover, these meta-features are often hand-crafted to be predictive of the performance of machine learning models applied to dataset $\mathcal{D}$.

We reuse the terminology and formalization from the authors of [108], whom distinguish between meta-features and characterization measures because a characterization measure $m : \mathcal{D} \mapsto \mathbb{R}^{k'}$ results in k' values, with k' potentially depending on the dataset itself. Yet, using meta-features as a dataset characterization requires fixed dimensionality (see Definition 2). Therefore, we must map each characterization measure to a fixed-dimensional meta-feature using a summarization function $\sigma : \mathbb{R}^{k'} \mapsto \mathbb{R}^k$; frequently used summerization functions include the mean, minimum, maximum, skewness and kurtosis. Then, a meta-feature-based dataset characterization $\mathcal{C}_\mathcal{D} \in \mathbb{R}^{n \times m}$ is a collection of meta-features $f(\mathcal{D}) = \sigma(m(\mathcal{D}, h_m), h_s)$ with h_m and h_s being hyperparameters for m and σ respectively. The dimensionality of $\mathcal{C}_\mathcal{D}$ is fixed with $m = 0$ and n being the sum of the dimensionality k of each meta-feature after applying σ.

The selection of meta-features is a notoriously difficult task; there are many meta-features for many different tasks whose selection may be context dependent. Within our supervised classification setting, meta-features can be categorized in different groups: simple features easily extracted and general in nature; statistical features capturing statistical properties; information-

theoretic features capturing the amount of information and complexity; model-based features derived from properties of models; landmarking features capturing performance on simple and fast learners; and other features. For an extensive discussion of possible meta-features for classification tasks consider [108, 5].

Popular and typically well-performing collections of meta-features for supervised classification settings include the ones proposed by Feurer et al. [39] and Wistuba et al. [132], whose specific meta-features are listed in Appendix C. The former contains simple, information-theoretic, statistical, PCA-based and landmarking meta-features which together with specified summarization functions result in a dataset characterization containing 46 values. The latter is composed of statistical and information-theoretic meta-features which together with specified summarization functions yield a dataset characterization composed of 22 values. The latter collection of meta-features and summarization functions is a subset of the former, and thus requires fewer computational resources. Perhaps a counter-intuitive finding: the subset outperforms the superset [55].

Task2Vec Task2Vec [1] characterizes a dataset $\mathcal{D}$ as $\mathcal{C}_{\mathcal{D}} \in \mathbb{R}^n$ in a behavioral way, by observing a neural network's parameter sensitivity to the data in $\mathcal{D}$. Given a dataset $\mathcal{D}$ consisting of images, Task2Vec computes a fixed-size embedding – n, m remain equal – using a pre-trained neural network, which is referred to as a probe network. Prior to computing $\mathcal{C}_{\mathcal{D}}$, the classification layer of the probe network is re-trained on the given task. The embedding computation d builds on the behavioral intuition that neural networks' activations of the data characterize the data itself. $\mathcal{D}$ is passed through the probe network to estimate (the diagonal elements of) the Fisher Information Matrix corresponding to the probe network's filter parameters. Collecting these estimates in a vector yields $\mathcal{C}_{\mathcal{D}}$, and thus n is equal to the number of diagonal elements in the FIM. Any $\mathcal{C}_{\mathcal{D}}$ is of fixed size, because Task2Vec uses the same probe network (architecture) for all embeddings. The authors find that the norm of $\mathcal{C}_{\mathcal{D}}$ encodes the task's complexity. There are two dataset-similarity measures s based on the Task2Vec embedding, the symmetric Task2Vec distance, and the asymmetric Task2Vec "distance". Both distance measures take two Task2Vec embeddings ($\mathcal{C}_{\mathcal{D}}$) as input and produce a scalar similarity value. However, the asymmetric Task2Vec distance takes the complexity of the first task into account. Therefore, it is asymmetric and not a true distance metric. Moreover, the asymmetric distance introduces hyperparameter α, weighting the influence of complexity. Due to the inclusion of complexity and α, the asymmetric distance can be negative.

Dataset2Vec Another dataset characterization method employing a pre-trained model is Dataset2Vec [55]. Dataset2Vec learns a meta-feature extractor $d : \mathcal{D} \mapsto \mathcal{C}_{\mathcal{D}} \in \mathbb{R}^n$ by training a neural network, see Figure 2.2. The last layer of the neural network represents the learned meta-features, together they form a fixed-size embedding of $\mathcal{D}$ dataset. Neural networks require input consisting of equal size and therefore Dataset2Vec interprets datasets as hierarchical sets, specifically single predictor-target pairs, to achieve schema-agnosticism. To learn a meaningful embedding, the neural network is trained on a novel task with abundant data: the authors propose a patch identification task, where the neural network must predict whether two patches are sampled from one dataset or two distinct datasets. A patch consists of a subset of instances, features, and targets from a given dataset. In performing the patch identification task, Dataset2Vec learns features to discriminate the datasets, in turn yielding the meta-feature extractor. Instead of explicitly learning meta-features for each pair of datasets, the authors

pre-trained the network on the patch identification task for a large collection of classification datasets. This pre-trained network extracts the general, previously learned, meta-features that compose the fixed-size task embedding $\mathcal{C}_\mathcal{D}$ by averaging the meta-features over the number of batches specified. Using the pre-trained network implies $n = 32$ and thus $\mathcal{C}_\mathcal{D} \in \mathbb{R}^{32}$.

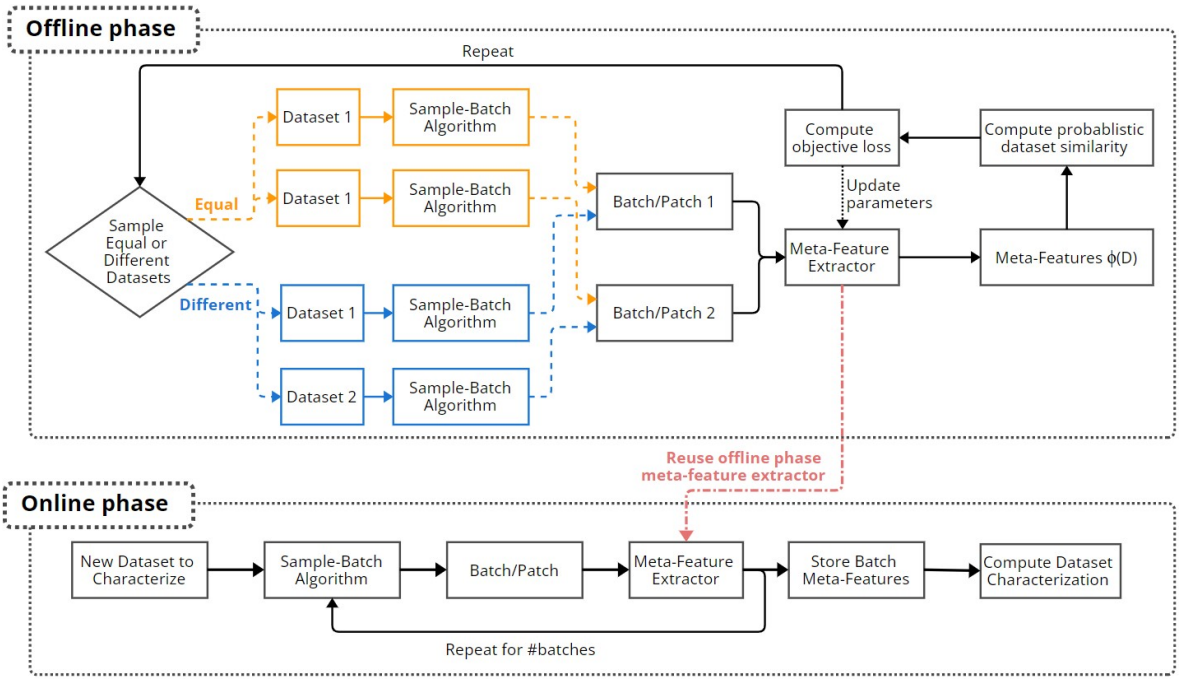

Figure 2.2: Illustration of the Dataset2Vec [55] framework to characterize datasets. The top part of the figure depict the offline phase, in which the meta-feature extractor is trained. The bottom part depicts the online phase in which a new dataset is characterized using the pre-trained extractor. Solid arrow lines indicate the always-executed flow of steps within a phase, and dashed arrows indicate possible execution flow. The red semi-dashed arrow highlights the interaction between the offline and online phase.

DIDA DIDA [22] uses distribution-based invariant neural networks to learn a dataset characterization $\mathcal{C}_\mathcal{D} \in \mathbb{R}^n$. DIDA uses the coordinates in the latent space of the last neural network layer as $\mathcal{C}_\mathcal{D}$, thus yielding a fixed-dimensional vector of learned meta-features. The authors propose a distribution-based invariant layer as the first layer, or the first two layers, of the neural network. The invariant layer is built on top of aggregated interaction functionals, themselves employing a four-dimensional aggregator, aggregating all feature and label values. By using the invariant layer as the first layer in the proposed DIDA component the authors achieve invariance with respect to labels and features, but at high computational expense. To reduce the computational burden of the invariant layer, the authors propose to only extract and aggregate information from the neighborhood of samples. Meta-features are learned akin to Dataset2Vec: a Siamese architecture consisting of DIDA components is trained on a patch identification task. The Siamese network is optimized through a loss defined using the distance between the embeddings of two DIDA components, each taking in one dataset. Then, after training, the last layer of the DIDA component represents the learned meta-features. Thus, using DIDA to compute a dataset-similarity value s involves training a neural network, with a

considerable amount of parameters. The paper claims DIDA outperforms Dataset2Vec, but the evaluation to support this claim is weak. The only comparison to Dataset2Vec is in the patch identification task, which is not the goal of the learned meta-features, merely a tool to learn them. The paper evaluates DIDA on a downstream application but lacks a comparison to DataSet2Vec.

WTE The Wasserstein Task Embedding (WTE) [75] is an optimal-transport-based dataset embedding method $d : \mathcal{D} \mapsto \mathcal{C}_{\mathcal{D}} \in \mathbb{R}^{n \times m}$ and the Euclidean distance between the embeddings produces an associated dataset-similarity value s. The WTE takes inspiration from both Task2Vec [1] and OTDD [7], it creates a dataset embedding – Task2Vec inspiration – based on solving an OT problem – OTDD inspiration. Like OTDD, WTE is model-agnostic and free of training, capable of handling (partially) disjoint label sets. Like other embedding-based dataset-similarity measures, WTE alleviates the problem of expensive pairwise computation by employing embeddings as task representations. The WTE associated similarity correlates highly with the OTDD distance while reducing the computational cost from quadratic to linear with respect to the number of tasks. Like OTDD, WTE represents labels as their feature-based distributions, using Gaussian approximations to speed up computation. Additionally, WTE embeds these label distributions through multi-dimensional scaling (MDS), a non-linear dimensionality reduction approach preserving pairwise distance. Afterward, the MDS-based label embeddings are concatenated to the feature vectors, yielding the updated samples. Tasks' updated samples are embedded using a, previously proposed, Wasserstein embedding method. The Wasserstein embedding creates an embedding for probability distributions, for which the Euclidean distance between them approximates the 2-Wasserstein distance. Hence, the distance between tasks is the 2-Wasserstein distance between their updated samples, which is approximated by the squared Euclidean distance between the embeddings.

2.2.1.3 Configuration Characterization Methods

RankML's Configuration Characterization The authors of [66] introduce RankML, a meta-learner employing dataset and configuration characterizations, using their novel ML pipeline representation approach. This configuration characterization method represents the topology of the ML pipeline as a sequence of words. Following our formalization (see Definition 1), each $a \in \boldsymbol{A}$ and its accompanying hyperparameters in λ are represented by a unique fixed-length hash, with a sequence thereof composing the configuration characterization $\mathcal{C}_{\mathcal{C}} \in (B^*)^n$. See Figure 2.3 for example configuration characterizations representing their respective pipeline. Note that the method employs the hyperparameters $\boldsymbol{\lambda}$ in pipeline $(g, \mathbf{A}, \boldsymbol{\lambda})$ to characterize the pipeline while simultaneously yielding a compact pipeline representation. Adversely, the unique hashes composing the compact pipeline representation together with a huge search space, imply many different elements in the pipeline representation, which may be challenging for a meta-learner to distinguish. The pipeline configuration is created using the following rules:

1. Define the configuration characterization's dimensionality as the longest pipeline to characterize.
2. Hash each node (algorithm a and its accompanying hyperparameters in λ) in the pipeline in reverse order (from output to input).

3. Each node in g must be hashed before its inputs.
4. If there are parallel sub-pipelines, the longest sub-pipeline is processed first, ties are broken randomly.
5. All pipeline characterizations have equal dimensions, pad with hashes representing "blanks" if necessary.

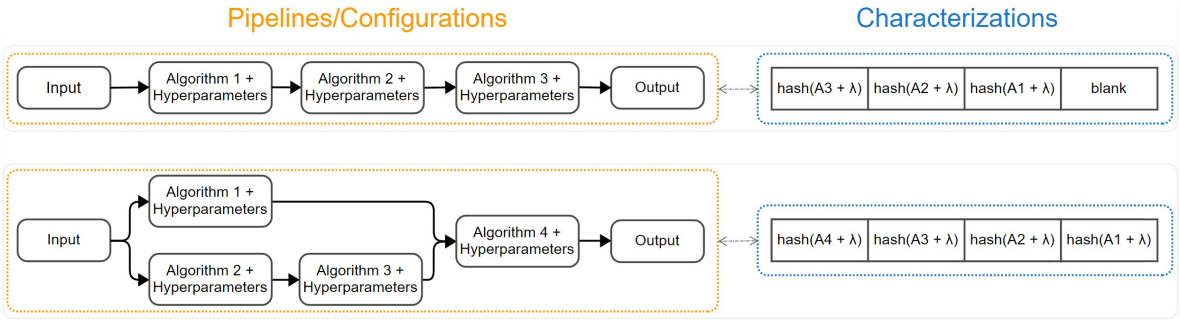

Figure 2.3: Example depicting configurations (i.e. pipeline structure g with its algorithms $\boldsymbol{A}$ and accompanying hyperparameters λ) and RankML's corresponding characterizations following aforementioned rules (1, 2, 3, 4, 5). The example is modified from the RankML paper [66].

Decomposition-Based Configuration Characterizations Probabilistic Matrix Factorization [41] (PMF) is a collaborative-filtering-inspired AutoML approach considering a matrix $\mathbf{Y} \in \mathbb{R}^{N \times D}$ composed of dataset-pipeline evaluations for N pipelines and D datasets. PMF seeks a low-rank decomposition $\mathbf{Y} \approx \mathbf{XW}$ with $\mathbf{X} \in \mathbb{R}^{N \times Q}$ and $\mathbf{W} \in \mathbb{R}^{Q \times D}$ with Q denoting the dimensionality of the learned latent space. The goal of PMF is to automate the selection and tuning of pipelines consisting of a pre-processing step and a predictive model, but in creating the low-rank decomposition it additionally learns to embed pipelines in the latent space purely from their performance in $\mathbf{Y}$. The learned matrix $\mathbf{X}$ thus contains the N configuration characterizations $\mathcal{C}_{\mathcal{C}} \in \mathbb{R}^n$ as Q-dimensional vectors in the learned latent space, note $n = Q$. The approach is computationally demanding, because it requires decomposing a high-dimensional matrix with PCA, where the computational cost grows cubically with the number of encountered datasets and quadratically with pipelines (whose configuration space is infinite).

Like PMF, TensorOBOE [137] uses a low rank decomposition for pipeline search, but on tensors instead of matrices. The 6-order error tensor ϵ contains prior pipeline-dataset evaluations, where each direction represents a different aspect, namely: datasets, imputers, encoders, standardizers, dimensionality reducers and estimators. We refrain from repeating elaborate technical details in this high-level overview, for more details consider Section 3.3 of the paper. In short, the tensor is decomposed using Tucker decomposition to yield one core tensor and six factor matrices (one per direction in ϵ). The columns (1 for each matrix) together embed a pipeline, which composes the configuration characterization $\mathcal{C}_{\mathcal{C}} \in \mathbb{R}^{n \times m}$ with $m = 6$. Clearly, computing the Tucker decomposition of a 6-order tensor is even more computationally expensive than computing a PCA-based matrix factorization, for details on the time complexity of low-rank Tucker decompositions consider [80]. Though the time complexity may differ per solver it is safe to say that the required computational resources are beyond that of AutoML

systems, with TensorOBOE requiring a 128 core machine with 1056GB of memory for 215 small-sized datasets and 23.424 pipelines.

2.2.2 Meta-Learners

A meta-learner is a component performing knowledge transfer from prior to new tasks in a meta-learning system. In performing knowledge transfer a meta-learner may require information regarding the learning task or prior solutions. Therefore, we can consider three groups of AutoML-related meta-learners, namely;

- Similarity-based meta-learners which employ dataset similarity.
- Characterization-based meta-learners which employ dataset or configuration characterizations.
- Dataset-agnostic meta-learners which do not depend on datasets.

Thus, a meta-learner could simultaneously be similarity- and characterization-based. In the following subsections we give an overview of the state-of-art meta-learners categorized according to our previously defined groups. Meta-learners employing multiple components simultaneously are discussed in the section whose component's usage is most pronounced.

2.2.2.1 Similarity-Based Meta-Learners

Similarity-based direct configuration transfer Configuration transfer applications reuse configurations from (similar) previous tasks for the current task. In reusing previous configurations an inductive bias is transferred, reducing the effort to choose among a vast amount of possible configurations. The simplest similarity-based configuration transfer approach is to use the best-performing configuration from the most similar task. Though this likely yields a non-optimal configuration, its application in an AutoML tool for time-series data [64] shows promising configurations can be identified. An extension of this approach is to evaluate several recommended configurations [124], transferring the best evaluated configuration – an approach we disregard due to our scoping.

Similarity-based direct configuration transfer for warm-starting optimization Besides directly using other tasks' configurations, there are also opportunities to use them indirectly by warm-starting optimization for the current task. The goal of warm-starting is to increase convergence rate and/or improve final performance by selecting configurations from previous tasks to yield a high-quality initialization for the optimization procedure. In this setup, the role of a dataset-similarity measure is to identify the prior tasks whose configurations are transferred. In doing so, it is assumed that similar tasks have similar high-performing configurations.

One approach, as presented in [39], is to use the t best configurations from the d most similar tasks, yielding $t \cdot d$ configurations for initialization. We need to be mindful of the value $t \cdot d$, because it directly influences the exploration-exploitation trade-off of AutoML systems. If $t \cdot d$ is too small, we may under-exploit and could have found better configurations through a better initialization. If $t \cdot d$ is too large, we waste the exploration potential of the optimization procedure. In the extreme case, with $t \cdot d$ being the total amount of evaluations, we again yield

a direct configuration transfer mentioned previously. Not only $t \cdot d$ influences this trade-off, but also the individual choices for t and d; a higher t resembles more exploitation whereas a higher d promotes exploration.

To the best of our knowledge, there is no consensus on choosing t and d, nor are there other approaches to incorporate similarity in selecting configurations for warm-starting pipeline optimization. There is however much related work on warm-starting hyperparameter optimization (HPO) using similarity information, with approaches for Bayesian Optimization [38, 123, 59] and CMA-ES [94] showing performance increases. Since pipeline optimization is more complex than HPO, warm-starting pipeline optimization is especially promising; prior work [39] noticed complex problems benefit most from warm-starting Bayesian optimization.

A critical reader may have noted that the discussed approach [39] is applied on HPO, while we excluded meta-learners for sub-problems from the discussion's scope. The meta-learning approach is included because it has been successfully applied in warm-starting search for entire pipelines in subsequent work [37]. For that reason, similar meta-learning approaches (e.g. [132]) without such follow-up works are not discussed.

Similarity-based search space pruning for warm-starting optimization Apart from direct configuration transfer, there are other approaches for warm-starting optimization procedures based on dataset similarity. We can transfer knowledge not only by promoting certain parts of the search space, but also by excluding parts thereof. This core idea – pruning the search space for an AutoML system's optimization procedure based upon a similarity measure – is investigated in [92] for the AutoWeka4MCPS [112] AutoML system. The core idea is not novel, for it had already been explored in [131], but the extension to a more complete setting is, i.e. warm-starting full pipeline search instead of HPO.

Following our notation (see Definition 1), one approach [92] prunes the search space (initially $|\mathcal{A}| = 30$) through excluding all but the k best algorithms $a \in \mathcal{A}$, thus yielding $|\mathcal{A}| = k$. Interestingly, the paper compares this meta-learning approach in a setting employing dataset similarity – only considering the performance on the most similar datasets – and in a dataset-agnostic setting, but it can only affirm the similarity-based approach to be significantly better than the baseline – therefore it is discussed here instead of Section 2.2.2.3).

It is worth noting the dataset-similarity measure is modular with respect to the meta-learning approach. The authors opt for a meta-feature-based dataset-similarity measure solely employing landmarking meta-features, begging the question whether these results can be improved with more diverse meta-features or learned dataset characterizations.

The experimental evaluation explores the effect of choosing k, which we can consider a exploration-exploitation trade-off parameter, with decreasing k exploiting more and increasing k exploring more. The findings indicate that among $k \in \{1, 4, 8, 10, 19\}$ setting $k = 10$ is optimal using the similarity-based approach. Interestingly, the choice for k and thus the exploration-exploitation trade-off, differs for the dataset-agnostic and dataset-similarity-based meta-learning approaches, with the higher-performing approach preferring a higher value for k. This discombobulating finding suggests increased meta-learner performance need not imply favoring exploitation over exploration in warm-starting AutoML search. To further complicate the trade-off choice, the value for k depends on the subsequent optimization procedure's time budget, with a higher time budget allowing for more exploration.

2.2.2.2 Characterization-Based Meta-Learners

RankML RankML [66] ranks machine learning pipelines using a XGBoost-based ranker trained on dataset- and configuration characterizations together with their associated performance, see Figure 2.4. The pipeline configuration is characterized through a novel representation of its topology (see Section 2.2.1.3) while descriptive and correlation-based meta-features characterize the data. RankML employs distinct XGBoost ranker models per task – regression or classification – and evaluation metric. By default RankML ranks all pipelines in the metadataset on the novel task to recommend the top ranked pipeline without the need for evaluation. However, better performance is achieved by selecting, through evaluation, the best-performing pipeline from a set of top-ranked pipelines. If RankML recommends the best-evaluated top-4-ranked pipeline, its performance is on par with TPOT [96] and Auto-Sklearn [36], but at reduced computational cost.

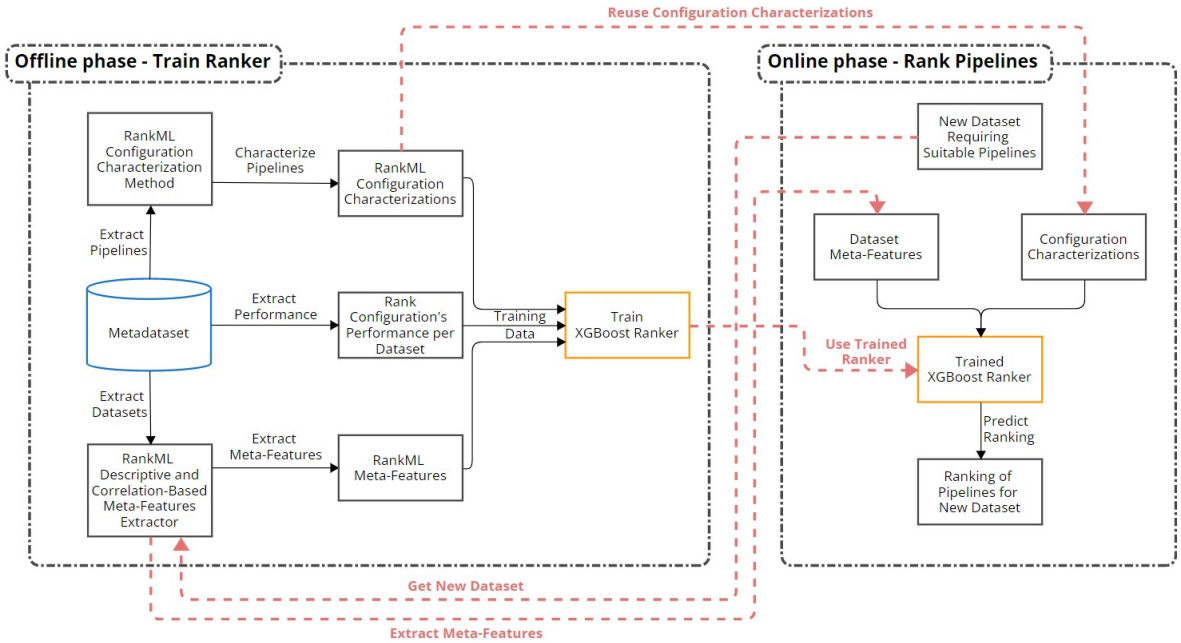

Figure 2.4: Illustration of the RankML [66] meta-learner, arrows illustrate the steps and red arrows highlight interactions between the offline and online phase. The left part of the figure depicts the meta-learner offline phase in which the XGBoost-based ranker is trained using dataset and configuration characterizations, with their associated ranks. Datasets are characterized using descriptive and correlation-based meta-features, pipelines are characterized using the RankML configuration characterization method (see Figure 2.3), and their performance is ranked per dataset. The right part of the figure displays the online phase, in which the metadataset pipelines (candidate pipelines) are ranked for a new dataset using the trained ranker from the offline phase. The input for the ranker consists of a characterization of the new dataset and the configuration characterizations previously computed in the offline phase.

Similar to many meta-learners, a disadvantage of RankML is its poor ability to extrapolate beyond the metadataset; RankML solely ranks previously-created pipelines, it does not originate them. The authors acknowledge this weakness, but argue a sufficiently large metadataset (i.e. their 313.488 pipeline evaluations) partially mitigates the issue. Note the RankML meta-learner need not be tailored to specific dataset nor configuration characterization methods.

Correspondingly, according to the authors, investigating whether the RankML setup benefits from more recent dataset characterizations (see Section 2.2.1.2) is an interesting research direction.

The use of RankML in AutoML systems is investigated in MetaTPOT [67], specifically exploring the incorporation of a ML pipeline ranking model in the synchronous evolutionary AutoML system TPOT. The work shows that MetaTPOT decreases TPOT runtime without negatively affecting performance. MetaTPOT does not exhibit improved accuracy-related performance, which we postulate may be attributed to a limited and non-uniform incorporation of pre-ranked individuals in warm-starting the evolutionary search procedure.

2.2.2.3 Dataset-Agnostic Meta-Learners

Portfolio building Portfolio building is a dataset-angostic meta-learning approach introduced in [32] and incorporated in auto-sklearn 2.0 [34] to replace the original meta-features based approach. The approach uses prior evaluations to create a portfolio $\mathcal{P}$ – an ordered set of pipelines $(g, \mathbf{A}, \lambda)$ – of a specified size $|\mathcal{P}|$ that is expected to perform well, regardless of the new dataset. The approach is intended to warm-start a subsequent optimization procedure, but with $|\mathcal{P}|$ as a parameter this need not be the case (e.g. $|\mathcal{P}| = 1$).

In an offline setting, the approach commences by searching the best configuration for each dataset $d \in \mathcal{D}$ to create the set of candidate pipelines $\mathcal{C}$ for the to-be-built portfolio. Each configuration $c \in \mathcal{C}$ is evaluated on each dataset $d \in \mathcal{D}$ using a cross-validation procedure to yield performance matrix $\mathbf{P}$, a $|\mathcal{D}| \times |\mathcal{D}| = |\mathcal{C}| \times |\mathcal{D}|$ dimensional matrix. Performance matrix $\mathbf{P}$ is transformed to error matrix $\mathbf{E}$ with equal dimensionality by computing the normalized regret across all datasets.

Portfolio $\mathcal{P}$ is built by employing greedy submodular function minimization [62] on $\mathbf{E}$, with a process as follows. The portfolio is built iteratively by greedily selecting configurations one by one: in each iteration, we add the candidate $c \in C$ to portfolio $\mathcal{P}$ such that the portfolio's generalization error is minimized. The generalization error is the sum of the regret on each dataset for its highest-performing configuration in $|\mathcal{P}|$. If multiple candidates reduce the generalization error equally – with a special, yet not uncommon, case being no reduction – then the first evaluated candidate is added to $\mathcal{P}$. The portfolio building procedure stops when the portfolio size is reached, which is upper-bounded by $|\mathcal{D}|$.

In a warm-starting scenario, portfolio size $|\mathcal{P}|$ acts as an exploration-exploitation parameter, with increasing values exploiting more, and decreasing values exploring more. If we were to generalize beyond the introduced framework – considering $\mathcal{C}$ to not only consist of the best configuration per dataset d, but the k best configurations – then k would act as a trade-off parameter similar to $|\mathcal{P}|$. Moreover, in this generalized setting the upper bound on the portfolio size increases by factor k.

2.2.2.4 Scoping

A meta-learner can perform a wide variety of tasks including those in few-shot learning, reinforcement learning and deep learning [49]. The scope of our meta-learner discussion concerns approaches specifically operating on tabular datasets (see Restriction 2) such that these can be used for tabular AutoML systems. The aforementioned application fields tend to be neural-network driven and therefore tailored to data modalities different from tabular data, causing their meta-learning approaches to be excluded from the current literature review.

In the overview we focus on works that most closely follow our formulation and formalization of the AutoML problem (see Section 2.1), considering the *pipeline* creation problem. Meta-learners solely considering a subproblem of the pipeline creation problem, i.e. only selecting the pipeline structure, or its algorithms or hyperparameters are out-of-scope because their incorporation in complete-pipeline AutoML systems is less straightforward. Likewise, meta-learning approaches crafted solely for data preparation [142, 13, 46, 63, 14] or feature generation [56] are excluded from this overview. For similar reasons, we disregard meta-learning and AutoML approaches opting for multiple stages, such as a bi-level scheme [144], two-stage approach [103] or even more stages as in [85]. The latter distinction dismisses meta-learning approaches within multi-stage settings, including [147, 86, 105].

Lastly, we exclude approaches that explicitly rely on new or specific prior dataset-pipeline evaluations, because it limits their incorporation in AutoML systems. Despite a fine line, we argue works (iteratively) gathering new evaluations are better classified as AutoML systems instead of meta-learning approaches. Compared to meta-learning approaches, entire AutoML systems are limited with respect to their modularity, causing their exclusion from this overview.

We thus exclude approaches developed with a specific evaluation scheme in mind, that being prior to pipeline synthesis or during. Exclusions include approaches performing a matrix decomposition such as Probabilistic Matrix Factorization (PMF) [41] and TensorOBOE [137] because they require pipeline evaluations for a novel dataset and gather new evaluations thereafter. Additionally, PMF explicitly assumes the metadataset is composed of a set of previously defined possible pipelines. Other exclusions are the meta-reinforcement-learning-based AlphaD3M [28] and a phased performance-based pipeline planner P4ML [45], because both require pipeline evaluations in their setup.

To summarize we exclude any meta-learning approach: not operating on tabular data, specifically crafted for pipeline sub-problems, considering multi-phase setups, gathering new evaluations or imposing metadataset assumptions. Our scope is supported by a desire for a modular meta-learning framework (see Section 1.2), AutoML research increasingly focusing on complete ML pipelines (for tabular data) [146], and pre-existing literature already comparing meta-learning approaches for subproblems [58].

Chapter 3

MLTA: a Meta-Learning Toolbox for AutoML

In this chapter we detail the proposed Meta-Learning Toolbox for AutoML (MLTA). In Section 3.3 we present the modular structure and functionality of the toolbox, detailing which meta-learning approaches can be built using its main components, for which we provide a primer in Section 3.1. In Section 3.2 we employ the preliminary knowledge from Section 2.2 – especially the overviews on dataset characterization methods, dataset similarity measures, and configuration characterization methods (Section 2.2.1) and meta-learners (Section 2.2.2) – to select meta-learning components to incorporate in MLTA and compare in the experimental evaluation (see Chapter 4). In addition to that, we perform experiments analyzing the run time for dataset characterization methods in Section 3.3.3.

3.1 MLTA High-Level Overview

This section provides a brief primer of the meta-learning framework adopted by MLTA to help contextualize and understand the subsequent meta-learning component selection (Section 3.2), a more elaborate explanation is provided in Section 3.3.1. Consider Figure 3.1, a meta-learner depends on meta-learning components – e.g. a dataset-similarity measure, dataset characterization method, or configuration characterization method. The meta-learner is evaluated on a dataset using evaluation and metadatabase functionality, the latter emulating typical AutoML functionality, such as providing metadata to train the meta-learner.

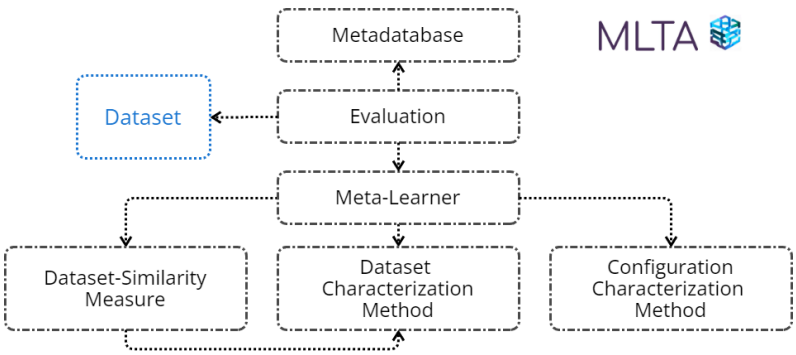

Figure 3.1: A high-level overview of MLTA's main components and their interactions.

3.2 Meta-Learning Component Selection

In this section we compare and select approaches from the preliminary Chapter 2 to incorporate in MLTA, sometimes generalizing beyond their initial interpretation. We explicitly defer from re-explaining the approaches, because they are already explained in Chapter 2. The approaches to compare are few in number and relatively well-fitting due to extensive scoping in the preliminary chapter, guided by Restrictions 1, 2, 3, 4, 5 and the aim for a modular framework. Nonetheless, we need to select – per meta-learning component – which approaches are to be incorporated in the toolbox. This selection is guided by a comparison on a set of key dimensions, namely: applicability, scalability and utility. The materialization of these properties for the different meta-learning components – dataset characterization methods, dataset similarity measures, configuration characterization methods and meta-learners – will be discussed in their respective subsections. To permit comparing approaches, we argue for the most relevant characteristics of each dimension under consideration, which we term aspects of each dimension. It is important to note that for all components there is no one-size-fits-all approach; what constitutes the most suitable method may be dependent on the problem at hand.

3.2.1 Dataset Characterization Method Selection

For the dimension of applicability we argue for the aspects of data modality and task dependency. The scalability dimension's aspect concerns scalability with respect to the data. The last dimension, utility, has the main aspects of robustness, parameterization and effectiveness.

Applicability The applicability – the ability to compute a dataset characterization – depends on its input, namely the data representing a task to be solved. A dataset may vary widely, on lots of different facets, but possibly, the most informative and limiting aspect of a dataset is its modality. We consider only the most common: image data, tabular data, and text data. We aim for a method that is at least applicable on tabular data, without restrictions on its schema. The second aspect of applicability concerns the task dependency. We aim for methods supporting both binary and multi-class classification tasks, where the additional applicability on regression tasks would be welcome.

Scalability Scalability concerns the ability to perform the method efficiently and in a tractable way, involving base costs and costs scaling with dataset size. We deem a dataset characterization method scalable, with respect to the data, if its computational time complexity is at most linear with respect to the number of samples, features, and outcomes.

Utility The utility dimension is concerned with the use of a dataset characterization method or its output. We distinguish between aspects concerning the production of the output, or the output itself. Given equal input, in the form of datasets, there are two aspects influencing the production of the output of a task-similarity measure: the measures' robustness and its parameterization. Additionally, the quality of the dataset characterization is important. However, with a lack of ground truth we can only assess it in a behavioral way: the effectiveness. Due to the need for a behavioral/experimental evaluation of effectiveness we exclude it from the comparison.

The degree of robustness of a dataset characterization method involves: invariance to sample order; invariance to feature order (if the feature order is independent of the outcome),

and a stable output given equal input. A robust method achieves stable characterizations invariant to meaningless changes in the input. Hence, for a dataset characterization method to be robust it must be invariant to the permutation of the datasets' samples, because sets are *unordered* collections of items. Additionally, when features are independent of each other, the measure should be invariant with respect to the permutation of features, but only if the permutation is consistent across all samples. Robustness also involves stability of the output when there is no change in the input. A non-deterministic method is not necessarily undesirable, as long as its output is stable.

Types of parameterization come in two forms, namely methods that are parametric or non-parametric. Parametric methods are more difficult to use, because the output and its effectiveness depend on user input. Therefore, especially in the context of automation, non-parametric dataset characterization methods are preferred.

In the remainder of this subsection, we assess the dataset characterization methods presented in Section 2.2.1.2 on the aforementioned dimensions and their aspects, for a brief overview consider Table 3.1. Subsequently, we use these aspects to compare and select approaches for MLTA.

Meta-Feature-Based Dataset-Characterization Method The properties of meta-feature-based dataset characterizations depend on the collection of meta-features used. They can be applicable to multiple data modalities – including text, image and tabular data – and multiple tasks such as regression, classification and clustering. Moreover, they can be scalable and fast to compute – mostly simple meta-features – and robust without the need for parameterization. Contrarily, they can also be computationally demanding, tailored to a data modality and task type, require parametererization and lack robustness.

For comparison sake, we determine the properties of the common meta-feature collections by Feurer et al. [39] and Wistuba et al. [132] previously discussed in Section 2.2.1.2 and detailed in in Appendix C. These meta-features are applicable to image and tabular datasets in binary and multi-class classification settings.

Both collections tend to be computationally undemanding, but this depends on the dataset at hand. Being composed of simple and low-complexity statistical meta-features, the Wistuba collection is scalable with respect to the number of instances, features and outcomes – i.e. the data. Contrarily, the Feurer meta-feature collection requires the more computationally demanding linear discriminant analysis (LDA) and principal component analysis (PCA), whose computational time-complexity can grow poorly depending on the solver, but is always non-linear for either features, or instances or outcomes. Moreover, the Feurer meta-features include landmarking models such as k-nearest neighbors, growing quadratically in the number of samples. Thus, due to the incorporation of PCA-based, LDA-based and landmarking-based meta-features the Feurer meta-feature collection is not scalable with respect to the data, potentially leading to a poor run time on larger datasets.

Regarding utility, both meta-feature and summarization function collections do not require further parameterization. The robustness for the Wistuba meta-features is excellent due to a deterministic computation of the meta-features, invariant with respect to meaningless permutations in the instances, features and outcomes. Contrarily, the Feurer meta-features are less robust due to its additional PCA-based, LDA-based and landmarking-based meta-features. Both the tree-based landmarking models and LDA are non-deterministic methods, with the latter *possibly* being sensitive to the data at hand [60]. Therefore it is debatable whether the

Feurer meta-features produce a stable dataset characterization. However, many of these features are invariant with respect to meaningless permutations in instances, features and outcomes. Therefore, we opt for classifying the Feurer-meta-features-based dataset characterization to be semi-robust; seemingly there are no prevalant robustness issues, though an experimental evaluation is required to confirm this.

Task2Vec [1] Task2Vec's use of a probe network determines some of its properties. First of all, the probe network is considered a parameter, with the paper itself showing considerable performance differences based on its choice. The CNN-based probe network(s) also determine the data modality to be images, with a requirement for their size. Hence, the approach may be less applicable to images that cannot be properly scaled to the pre-defined resolution. Moreover, the probe networks considered are specifically pre-trained on a classification task. The authors argue the method also computes meaningful embeddings for regression if the probe network performs a regression task, but the experimental evaluation thereof is lacking.

Task2Vec is not highly robust for several reasons involving the probe network. Computing the embedding involves re-training the probe network's classification layer. Therefore, the order of the instances influences the computation of the embedding. Moreover, neural network training and the estimation of the Fisher Information Matrix (FIM) are non-deterministic, possibly resulting in an unstable characterization.

The computation of the embedding involves both re-training on the task and FIM estimation. The computational time complexity for estimating the FIM is independent of the number of samples, their resolution, and the outcomes. Moreover, the time complexity for training a neural network grows linearly in the number of samples and constant in the number of outcomes. Therefore, although the base cost may be high, Task2Vec scales well with increasing data size.

Dataset2Vec [55] Dataset2Vec is most suited for tabular datasets because it learns the relation between single predictor-target pairs. Due to this setup, Dataset2Vec likely yields poor performance on other data modalities. For instance, for image data Dataset2Vec would consider the relation between specific pixels and the target, thereby completely disregarding the spatial structure necessary for meaning. Due to the setup of the pre-trained neural network, Dataset2Vec is only suited for classification datasets. To the best of our knowledge, there currently is no effort extending Dataset2Vec to a regression problem setting.

Because Dataset2Vec only needs to pass a specified number of batches through the pre-trained network to compute the characterization it is scalable with respect to the data.

Dataset2Vec is feature, sample and outcome invariant because it uses the coordinates in the pre-trained network's latent space directly, which only depends on the batches. However, the batching procedure causes the method to be non-deterministic, possibly yielding unstable characterizations. Luckily, this issue is mitigated by setting a seed for the batching procedure. Therefore, we can still claim that DataSet2Vec is a robust method. However, the characterization, and and especially its quality, depend on the number of batches that are passed through the pre-trained network. Though the size of the batches is determined by the pre-trained network, the number of batches is a parameter, with a default value of 10 in the original work. Unlike Task2Vec, the neural network employed by Dataset2Vec is not a parameter, because Dataset2Vec directly uses coordinates in the learned pre-trained neural network's latent space as a dataset representation.

DIDA [22] As shown in its paper's experimental evaluation, architectural variants using DIDA components can be applied to both regression and classification tasks. In essence, DIDA can be applied to any dataset with features and outcomes. However, because it does not take into account spatial structure, it is best applied to tabular datasets.

DIDA does not rely on a pre-trained model to extract meta-features, they need to be learned and thus computed each time. The computational time complexity scales quadratically in the number of samples, and therefore DIDA is not scalable with respect to the data. Though DIDA follows a similar setup as Dataset2Vec – characterizing a dataset using coordinates in a trained network's latent space – it is not scalable because it does not employ a pre-trained model during dataset characterization inference, it needs to be trained.

The main selling points of the approach are its invariance properties with respect to the data. Moreover, confidence estimates in the experimental evaluation suggest that the output is sufficiently stable. We consider DIDA to be robust due to its invariance properties and the (experimentally shown) characterization stability. Lastly, DIDA is parametric because learning meta-features for one dataset requires training on a patch identification task, involving another dataset to be specified.

WTE [75] Due to a setup similar to OTDD, the WTE dataset characterization can be computed for tabular data, image data and embedded text data. Similarly, due to explicitly modeling labels the method's applicability is restricted to classification tasks.

The computational time complexity of the WTE characterization method grows cubically with respect to the number of samples, it is thus not scalable with respect to the data.

Unlike OTDD – inheriting its invariance properties from the optimal-transport setup – WTE is not invariant to permutations of its samples. Recall WTE computes label embeddings by multidimensional scaling (MDS), MDS is not invariant to sample permutation and therefore neither is WTE. Whether the characterization is stable is debatable, we only know the associated similarity value correlates highly with OTDD, which itself is stable as shown in their experimental evaluation. Thus to claim stability of the characterizations one must assume that a stable similarity value implies a stable characterization and that a highly correlated similarity value implies stability thereof. We deem both these assumptions strong and not at all obvious and therefore, together with not satisfying invariance, we classify the WTE approach to not be robust. Lastly, we note that the dimensionality of the MDS output – which encodes the label distribution – is a parameter with a default value of 10.

Comparison and Selection The comparison and selection are guided by approaches' aspects in Table 3.1. First of all, we consider the applicability dimension; we can compare the methods on their applicable data modality: we exclude Task2Vec, because it only operates on image data. Next we consider the task dependency, which does not support the exclusion of any characterization method; all methods support both binary and multi-class classification.

The following dimension under consideration is scalability. We can distinguish approaches based on whether they need to train/learn a model *on the spot*, either for learning a dataset characterization or not. Approaches learning dataset characterizations are to hand-crafted meta-features as neural networks are to classical features, that is: they learn a way to represent data instead of hand-crafting a representation.

	Applicability		Scalability	Utility	
Char. Method	Data modality	Task dependency	Data	Robustness	Parametrization
✓ WistubaMF [132]	Tabular, Image	Classification	✓	✓	✓ None
✓ FeurerMF [39]	Tabular, Image	Classification	✗	∼	✓ None
✓ Dataset2Vec [55]	Tabular	Classification	✓	✓	Number of batches
Task2Vec [1]	✗ Image	Classification, Regression	✓	✗	probe network, α : complexity
DIDA [22]	Tabular	Classification, Regression	✗	✓	Supplementary dataset
WTE [75]	Image, Tabular, Embedded Text	Classification	✗	✗	MDS dimensionality
Sim. Measure					
OTDD [7]	✗ Image, Embedded Text	Classification	✗	✓	Feature distance metric
WTE [75]	✗ Image, Embedded Text	Classification	✗	✗	MDS dimensionality

Table 3.1: Comparison of dataset characterization methods and dataset similarity measures on applicability, scalability, and utility. Note each fixed-dimensional dataset characterization method – all but WTE – has an associated characterization-based dataset similarity measure with the displayed properties. WTE is presented twice because its aspects differ across its uses. A green check mark for an aspect of the dimension indicates the corresponding approach attains its properties in desirable fashion, a red cross implies it does not, and an orange tilde represents semi-attainability. A checkmark next to an approach indicates its inclusion in MLTA.

Like training neural networks, learning dataset characterizations seems to be a computationally intensive task, none of these approaches are scalable. Note Dataset2Vec is scalable even though it has learned a dataset characterization, because it employs a pre-trained network and therefore need not learn the dataset characterization on the spot. In similar fashion, by using a pre-trained network, Task2Vec also achieves scalability with respect to the data.

Perhaps surprisingly, we do not exclude approaches based on their poor scalability with respect to the data. In contrast to a poor utility or applicability, a poor scalability need not imply an infeasible method, though it does limit the scenario's for which it is feasible. In other words, an unscalable method may still be used for smaller datasets. Therefore, we do not exclude Feurer meta-features, DIDA and WTE purely on this aspect.

Next, we consider the utility dimension for the remaining approaches: Wistuba Meta-Features (WistubaMF), Feurer Meta-Features (FeurerMF), Dataset2Vec, DIDA and WTE. Observe we can distinguish the approaches on parameterization by whether or not they learn a dataset characterization. We exclude WTE based on its poor robustness in combination with its poor scalability and the need for parameterization (and it is poorly suited for characterization-based meta-learners – it does not produce fixed-size characterizations). We exclude DIDA due

to its daunting parameterization: a difficult-to-choose supplementary dataset for which – in the original work – there is no sensible default nor a proper argument to select it. The remaining approaches attain sufficient utility, because they do not have poor robustness (FeurerMF is not clear), nor difficult-to-choose parameters. We do not exclude Dataset2Vec based on the parameterization aspect, because the original work provides a sensible and well-performing default value which we need not alter.

By comparing the dataset characterization approaches on their main dimensions and their corresponding aspects we have selected the following methods for incorporation in MLTA: WistubaMF, FeurerMF and Dataset2Vec. We note that both WistubaMF and FeurerMF are strongly related methods, and could be considered different instantiations of a meta-feature-based characterization method. Nonetheless, we include both approaches because they satisfy our inclusion criteria. Moreover, since the meta-features in WistubaMF are a subset of those in FeurerMF we are eager to compare them in an experimental setting (see Chapter 4).

3.2.2 Dataset Similarity Measure Selection

The aspects are akin to the aspects of a dataset characterization method – which should not surprise with the existence of characterization-based similarity measures. The main difference is on the scalability dimension; for dataset similarity measures it is important that they are scalable with respect to the number of datasets to compute the similarity value for. In addition to that the remaining aspects are different because they now need to be applied to different input – two datasets instead of one – and a different output – a similarity value instead of a characterization. Therefore, we need not explain why the remaining aspects are suitable for a dataset similarity measure, we only need to explain what constitutes the scalability dimension.

Scalability concerns the ability to perform the method efficiently and in a tractable way, involving both the base cost of performing the similarity measure once, and scalability to multiple tasks, thus performing it multiple times. We deem a dataset similarity measure scalable with respect to the data, if its computational complexity is constant with respect to the number of samples, features, and outcomes for *both datasets*. Moreover, similarity measures should be scalable with respect to the number of datasets, because it is important for similarity-based meta-learners to compare the similarity among many datasets. We deem a dataset similarity measure scalable with respect to the number of datasets, if the amount of computational resources involved in computing all pairwise similarities among a group of datasets grows approximately linearly.

An ideal dataset similarity measure should be applicable to two, possibly different, datasets, each allowing for any combination of data modality and task type. In addition to that it should be scalable with respect to the data and number of datasets. Lastly, the computation of the similarity value should be robust and without requiring the specification of difficult-to-select parameters.

In the remainder of this subsection we assess the dataset similarity measures presented in Section 2.2.1.1 on the aforementioned dimensions and their aspects. We additionally include all possible characterization-based dataset similarity measures (see Section 2.2.1). Recall, these characterization-based similarity measures simply compute the (dis)similarity directly from the characterization and are therefore only possible if their corresponding characterization method produces fixed-size characterizations. This consideration excludes WTE to have a characterization-based similarity measure because its characterization dimensionality is dataset

dependent. All characterization-based dataset similarity measures are scalable with respect to the number of datasets because for N datasets we only need to compute N characterizations to compute N^2 pair-wise similarity values. All other aspects are directly inherited from the corresponding characterization method, see the top part of Table 3.1.

OTDD [7] Several of OTDD's aspects follow from it assuming access to a distance metric for feature vectors. Firstly, it makes the distance metric for feature vectors a parameter. Secondly, because of abstracting the distance metric away, it makes the method feature invariant. Thirdly, it disallows the direct application of OTDD on any dataset without a proper distance metric for feature vectors. However, distance metrics for vectors of equal size are well-known and therefore OTDD can be applied to datasets with feature vectors of equal dimensions. This restriction does prohibit most practical applications to tabular datasets, because their feature vectors typically vary in dimensionality.

Considering the suitable data modality, OTDD is applicable to text-based and image-based data, but the text needs to be encoded and the images need to be rescaled to equal resolution. Hence, we deem OTDD to be applicable with respect to images and embedded text because their equal dimensionality between datasets is not a strong assumption. Contrarily, assuming equally dimensional features for tabular datasets is a strong assumption, violating desired schema agnosticism, which leads us to exclude it from OTDD's applicable data modalities. Regarding task dependency, OTDD explicitly models label distributions, therefore it is only applicable to classification tasks.

Regarding scalability, OTDD with the computational speed-up still scales cubically in the number of features and quadratically in the number of samples. Therefore, it is far from scalable with respect to the data and thus best used on datasets with a small number of samples and features. Moreover, it implies that images must be of low resolution and text embeddings must have low dimensionality. OTDD computes a pair-wise distance between two datasets without an intermediary task representation, such as an embedding or characterization. Therefore, OTDD must be fully recomputed for each pair of datasets, disallowing it to scale well with respect to the number of datasets.

Regarding utility, the authors show that the stability of OTDD's output increases with an increasing number of samples, with it also being stable overall. Furthermore, due to the optimal transport setup, the method is sample and feature invariant. Because OTDD attains invariance properties and because its output is stable it can be considered robust.

WTE [75] Due to a similar setup as OTDD, WTE is only applicable for image data and embedded text because it requires equally-sized features across datasets. It thus suffers from similar drawbacks as OTDD: it cannot handle tabular data, images need to have equal resolution and text embeddings need to be of equal size. Moreover, as with OTDD, WTE is only applicable for supervised classification tasks because it explicitly models labels.

The WTE does *not* produce fixed-size dataset characterizations, because the dimensionality of the Wasserstein Embedding is related to the dimensionality of the dataset's features. Therefore, as mentioned before, it *cannot* have a characterization-based similarity measure by applying, for instance, cosine similarity on it. However, WTE still supports computing the similarity value with a time complexity constant with respect to the number of datasets by using its inner dataset representation combined with the optimal-transport setup. WTE scales cubically in the number of samples and is thus not scalable with respect to the data. However,

since the paper's experimental evaluation shows the computational cost scaling linearly with the number of datasets, WTE is scalable with respect to the number datasets.

The utility dimension's aspects are as before due to computing the inner characterization, therefore we only repeat it briefly. WTE cannot be considered a robust similarity measure, because it does not attain invariance properties. Moreover it is debatable whether the produced similarity value is stable. Lastly, the MDS's dimensionality is considered a parameter.

Comparison and Selection The comparison and selection are guided by the approaches' aspects in Table 3.1. The only aspect not present in the table is the scalability with respect to the number of datasets, which only OTDD cannot attain. Due to the strong connection among the aspects for dataset similarity measures and dataset characterization methods we can re-use the latter's comparison as part of the current comparison, i.e. we include characterization-based similarity measures corresponding to the previously selected dataset characterization methods – WistubaMF, FeurerMF and Dataset2Vec.

Despite recognizing its great effectiveness potential, we must exclude OTDD because it does not support tabular data well and scales poorly with respect to the number of datasets (and the data). In contrary to dataset characterization methods we do exclude approaches based on scalability, but with respect to the number of datasets. Unlike unscalability with respect to the data, unscalability with respect to the number of datasets limits the suitability of the method in all applicable scenarios – we can adapt the meta-learning approach per dataset, but we do *not* wish to adapt the metadataset.

Furthermore, we must exclude WTE because it is not applicable to tabular data, does not scale well with the data size and lacks robustness. We do not exclude the remaining approaches since they are scalable with respect to the number of datasets; they produce fixed-size characterizations allowing for easy (dis)similarity computation by applying cosine similarity or Euclidean distance.

Concluding, the dataset similarity measures to be included in MLTA are the characterization-based similarity measures using the WistubaMF-, FeurerMF- and Dataset2Vec characterization methods. We highlight that the selected collection of approaches is akin to the dataset characterization method selection – all the selected dataset similarity measures are characterization-based measures whose characterization method is incorporated in MLTA – because the aspects we selected them on are strongly connected.

3.2.3 Configuration Characterization Method Selection

The materialization of the dimensions for a configuration characterization method is akin to that of a dataset characterization method, but tailored to a configuration instead of a dataset.

Applicability concerns requirements for the ability to compute a configuration characterization. Recall from Definition 1 a pipeline configuration $(g, \mathbf{A}, \boldsymbol{\lambda})$ consists of its structure g, algorithms $\mathbf{A}$ and hyperparameters $\boldsymbol{\lambda}$. We compare configuration characterization methods on the type of graph they can characterize, with options for a tree with a branching factor of 1, or a DAG. Regarding the remaining components $\mathbf{A}, \boldsymbol{\lambda}$ we can simply indicate whether the characterization method incorporates them. Due to limited knowledge concerning configuration characterizations we do not prefer methods incorporating particular parts thereof; a method not incorporating hyperparameters may still be useful.

Scalability concerns the ability to perform the characterization of configurations efficiently and in a tractable way. The key aspect here is whether the method scales well with the number of configurations to characterize, because metadatasets may be composed of evaluations for many distinct configurations. A configuration characterization method is scalable if its computational time complexity grows linearly with the number of configurations to characterize.

Utility is concerned with the use of a configuration characterization method or its output. Again, we argue for the aspects of parameterization and robustness, but interpret the latter differently. Unlike datasets, configurations do not have meaningless changes associated to them, therefore we are no longer interested in invariance properties. A configuration characterization is robust if its characterization is stable given the same configuration. We, again, stress the importance of the characterization quality – i.e. effectiveness – but exclude it, because it requires experimental evaluation.

RankML [66] Due to its rule-based parsing of the pipeline structure, RankML is able to characterize any pipeline structure g. Moreover, it can distinguish between pipelines with varying algorithms and hyperparameters because it uses a hash-based approach. Furthermore, the setup is sufficiently flexible to also allow for only including algorithms, though this is not the default approach. A drawback of the approach is that it requires the specification of the maximal number of nodes in g for any pipeline in the metadataset. This limitation is however not severe since we typically know which pipelines to characterize a priori to characterizing them. The RankML configuration characterization method is thus applicable on any pipeline $(g, \mathbf{A}, \boldsymbol{\lambda})$.

RankML creates a hash per node in graph g within constant time, therefore the computational time complexity with respect to *one* configuration grows linearly with respect to the number of nodes in g. Because RankML characterizes configurations one by one it also scales linearly with respect to the number of distinct pipelines in the metadataset.

The RankML configuration characterization method is a deterministic approach, given the same collection of pipelines to characterize. In a meta-learning scenario it is safe to assume that the maximal size of the pipelines in the metadataset does not constantly change, therefore we consider RankML to be robust.

Decomposition-Based Configuration Characterizations In this paragraph we consider two approaches simultaneously, namely PMF [41] and TensorOBOE [136], because their approach for pipeline characterization is almost equal, resulting in the same aspects. The beauty and perhaps greatest flaw in using decomposition-based configuration characterizations is its sole dependence on the evaluations, therefore they are applicable on any configuration. Moreover, the methods allow specification of the configuration characterization dimensionality.

Both approaches are not scalable with respect to the number of pipelines to characterize. The simpler matrix-based-decomposition approach is already quite computationally demanding, requiring the decomposition of a high-dimensional matrix with PCA, where the computational cost grows cubically with the number of encountered datasets and quadratically with pipelines (whose configuration space is infinite). With the tensor-based decomposition having worse time complexity it is safe to say that decomposition-based approaches are far from scalable with respect to the number of pipelines to characterize.

The sole dependence on the evaluations implies the configuration characterizations depend on the datasets for which the configurations were evaluated. In addition to that, it is questionable whether the characterization is of high quality if pipelines have little evaluations. Furthermore, due to the evaluation dependence characterizations may not be robust to changes in the evaluation collection. Yet, with every evaluation in an AutoML system it changes, and thus configurations must be re-characterized frequently with a computationally costly decomposition. Another potential cause for instability is that the evaluations of pipelines themselves tend to vary. Concluding, decomposition-based configuration characterizations depend on the collection of datasets and the attained evaluation score of configurations thereon. Therefore, we do not deem these approaches to be robust.

Comparison and Selection The comparison and selection of configuration characterization approaches can be brief with only three considered approaches, and two of them sharing aspects due to their great similarity. We exclude the decomposition-based approaches, PMF and TensorOBOE, from the incorporation in MLTA because they are not scalable with the number of pipelines to characterize. Moreover, we cast doubts on the robustness of these methods. Fortunately, the remaining approach, RankML Pipeline Representation, attains the desirable aspects and is therefore included in MLTA.

3.2.4 Meta-Learner Selection

In this subsection we select meta-learners for MLTA guided by a comparison on: their applicability regarding knowledge transfer scenarios; their scalability with respect to the metadataset and amount of knowledge transferred; and their utility in terms of parameterization.

Applicability We compare meta-learners' applicability with respect to their knowledge transfer, desiring a broadly applicable approach for both time-restricted (directly retrieving configurations) and warm-starting scenarios.

Scalability For a meta-learner to be scalable, its online phase time complexity should grow at most linearly with the amount of knowledge transferred, e.g. the number of configurations to be recommended or the number of algorithms disregarded from the search space. Additionally, the computational time-complexity should grow at most linearly with the number of configurations and datasets present in the metadataset.

Utility As with the other meta-learning components we desire the meta-learner to be high-performing, – i.e. effective – but exclude it from the comparison due to the need for an experimental evaluation. In spite of that, we will assess the parameterization of each approach, preferring approaches that do not require difficult to choose parameters.

Similarity-Based Direct Configuration Transfer This assessment regards the meta-learning approaches that directly transfer configurations from the most similar datasets [39, 64], either being one configuration or a collection thereof. We can simultaneously assess them because the former approach generalizes the latter. Because these approaches directly transfer configurations they can be used in both time-restricted settings (solely requiring pipelines recommendations) and warm-starting scenarios.

These approaches compute the similarity between the new dataset and any other dataset in the metadataset, considering all of their configuration scores once. Therefore, the time complexity grows linearly with the number of datasets and configurations in the metadataset, and the number of configurations that are transferred – i.e. these approaches are scalable.

The generalization of the approaches requires the specification of the number of datasets to transfer from and the number of configurations to transfer from it. These *two* parameters influence the exploration-exploitation trade-off, where in a warm-starting scenario the optimization budget additionally influences this choice. The prior works have shown that setting both of these parameters to 1 or both to 5 yields fruitful results. Therefore the utility of this approach is debatable, it requires the specification of non-trivial parameters but empirical results suggest promising default values.

Search Space Pruning [92] The approach only allows for pruning search spaces for a subsequent optimization procedure. Therefore it is applicable for warm-starting but not for time-restricted settings, yielding a limited applicability.

This approach assumes a pipeline error represents a sampled error from an algorithm a in pipeline $(g, \mathbf{A}, \boldsymbol{\lambda})$. Then, the offline phase consists of computing the mean-error statistics to create a ranking for the top algorithms, which can be dataset specific. These dataset specific rankings are identified using a similarity value, which needs to be computed for all datasets, for which only the top k components are selected. Therefore, this approach scales linearly with the number of datasets, or constantly in the dataset agnostic setting. The time complexity is constant with respect to the number of configurations and number of components k transferred to the new search space.

The approach requires the specification of parameter k to trade-off exploration and exploitation, whose choice may depend on (the size of) the search space, and the subsequent optimization procedure's computational budget. Ergo, this approach's utility is limited with a difficult-to-choose parameter.

RankML [66] The meta-learner in the RankML paper is an XGBoost-based ranker model which produces a ranked list of pipelines in the metadataset for the new dataset. Therefore, this approach can both be applied to time-restricted scenarios – e.g. by selecting the top-ranked pipeline, or the top evaluated pipeline – and warm-starting.

The ranker model is trained in the offline phase to later be used for inference in the online phase. For inference it must compute the dataset characterization for *only* the new dataset. Then, it can predict for each pipeline in the metadataset what its ranking score is. Due to this setup the meta-learner scales well, its online phase time complexity is constant with respect to the number of datasets and evaluations in the metadataset, and linear with respect to the number of distinct pipelines in the metadataset. Concluding, we deem the RankML meta-learner to scale well.

The RankML meta-learner is a parametric approach because it requires the specification of the hyperparameters for the XGBoost-based ranker model. These hyperparameters could be tuned, though this a computationally intensive task. Fortunately, the original paper provides the most important hyperparameters' tuned values, these can be used as sensible defaults. Unlike the other approaches, the RankML meta-learner does not have parameters that allow for trading off exploration with exploitation. Overall, due to sensible defaults for its key parameters, we attribute RankML semi-utility.

Portfolio Building [32] The dataset-agnostic portfolio building meta-learner constructs a portfolio consisting of configurations in its offline phase. The initial interpretation is that this portfolio can be used for warm-starting. However, because the pipelines are added one by one to the collection – based on their reduction of the portfolio's estimated regret – we can interpret the portfolio as an ordered set, with the configurations ordered by their suitability. This allows for recommending the best configurations directly for time-restricted scenarios. For the aforementioned reasons we conclude that portfolio building attains high applicability.

Due to its dataset agnosticism, portfolio building's online phase is computationally inexpensive, it simply recommends the previously constructed portfolio or a specified part thereof. Therefore little computation is necessary, yielding constant time complexity with respect to the number of datasets and configurations in the metadataset and the number of configurations transferred. Hence, portfolio building's online phase is very cheap, computationally speaking, and certainly scalable.

Portfolio building, in its initial interpretation, only needs specification of the portfolio size. This parameter influences the exploration-exploitation trade-off in a warm-starting scenario and may therefore be difficult to select, even though empirical evaluations seem fruitful for a size of 16. However, in a time-restricted setting the parameter is chosen more easily. In a generalized setting we could also parameterize the number of configurations per dataset to consider during portfolio construction. Because, in its initial interpretation, the difficult to select trade-off parameter has an upperbound and a well-performing reference value we deem that portfolio building attains semi-utility.

Comparison and Selection We compare and select meta-learners on their applicability, scalability and utility. All meta-learners may have issues with their utility; all but RankML require the specification of difficult-to-select parameters to trade-off exploration and exploitation, and RankML requires computationally intensive and scenario dependent tuning. Because we cannot exclude all meta-learners we do not exclude any approach based on its utility.

Though the dataset-agnostic methods require the least computational resources during the online phase, all approaches scale sufficiently well. Accordingly, we need not exclude any meta-learner based on their scalability.

All but the search space pruning meta-learner [92] are widely applicable. Essentially, the search space pruning meta-learner is an antagonist to all other approaches, because it disregards a certain part of the search space instead of promoting it. Therefore, it cannot be used for time-restricted settings requiring configurations to be explicitly formed. Accordingly, we exclude it from the initial composition of the toolbox, but future work may still incorporate it since its application to warm-starting is promising.

Concluding, we incorporate the similarity-based direct configuration transfer [39, 64]; RankML [66]; and portfolio building [32]. We point out that, according to the previously defined meta-learner categorization (see Section 2.2.2), we yield three differently categorized meta-learners: a similarity-based, a characterization-based and a dataset-agnostic meta-learner.

This allows for comparing types of meta-learners (see Chapter 4), which is important because it is currently unclear which type of approach is to be preferred. For instance, it is uncertain whether dataset-agnostic approaches outperform similarity-based approaches; the paper for the portfolio building and search-space-pruning meta-learners [92, 32] produce conflicting conclusions. The latter finds a similarity-based variant outperforms a dataset-agnostic variant of its setup, with the former concluding the opposite.

3.3 Toolbox Overview

In this section we present the modular framework and functionality of MLTA. In Subsection 3.3.1 we explain the modular framework, detailing the overall structure and its components. In Subsection 3.3.2 we consider the realization of the framework: we discuss the implementation of selected approaches (see Section 3.2) and three additional baselines, perform dataset characterization runtime experiments, and give an overview of MLTA's provided meta-learning approaches.

3.3.1 Framework

As may have become clear from prior sections, the meta-learning framework consists of the following components: meta-learners, dataset characterization methods, dataset similarity measures, configuration characterization methods, evaluation methods and a metadatabase. The framework is depicted in Figure 3.2, detailing the interactions between components.

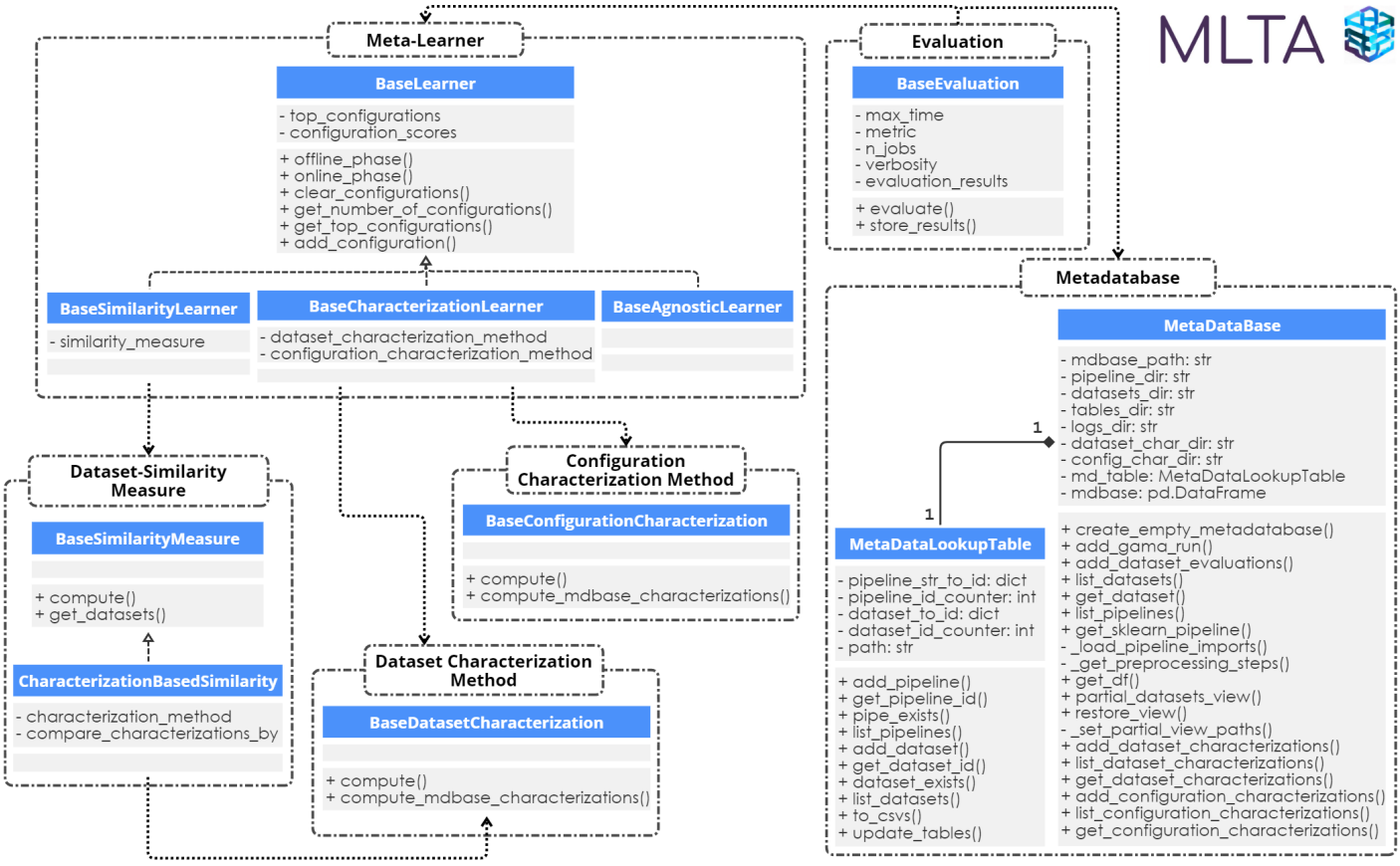

Figure 3.2: A depiction of the MLTA framework, showing the relation between the following main components: meta-learner, dataset-similarity measure, configuration characterization method, dataset characterization method, evaluation and metadatabase.

From the illustration it is apparent a meta-learning approach is modular, with its required components depending on the type of meta-learner. A similarity-based meta-learner inheriting from `BaseSimilarityLearner` functions with any dataset-similarity measure inheriting from

`BaseSimilarityMeasure`. Therefore, the similarity-based meta-learner also functions with any characterization-based similarity measure inheriting from `CharacterizationBasedSimilarity`. Even the characterization-based dataset-similarity measure itself is modular – truely showcasing MTLA's modularity – and functions with any dataset characterization method inheriting from `BaseDatasetCharacterization`. Likewise, any characterization-based meta-learner inheriting from `BaseCharacterizationLearner` functions with all instances inheriting from `BaseConfigurationCharacterization` and `BaseDatasetCharacterization`.

A meta-learner is evaluated using an evaluation method inheriting from `BaseEvaluation`, which interacts with a metadatabase (`MetaDataBase` instance) and a meta-learner (instance inheriting from `BaseLearner`). The interaction with a metadatabase is necessary for accessing the required metadataset, which may be composed of configuration-dataset evaluations but also dataset- and configuration characterizations. Thus, the metadatabase emulates functionality otherwise implemented by an AutoML system, but it also interacts with a `MetaDataLookupTable` instance for in-memory look-ups. The tools (implementations inheriting from the aforementioned classes) following this framework are treated in Section 3.3.2, and later used in a warm-starting case study in Chapter 4.

3.3.2 Functionality

3.3.2.1 Component Implementation

In this subsection we discuss details regarding the implementation[1] of previously selected meta-learning components (see Section 3.2) according to the earlier presented framework (see Section 3.3.1). We highlight components whose implementation required the specification of suppplementary details. Furthermore, we present two additional meta-learners and one configuration characterization method as baselines incorporated in MLTA.

The Dataset2Vec [55] framework is implemented such that pre-trained networks can be used. The original work, for its cross-validation experimentation, trained five meta-feature extractor networks which we allow to choose as a parameter, with a default for the first trained network. Moreover we allow for specifying the number of batches as a parameter, but with a default value of 10. An example using Dataset2Vec is shown in Figure 3.3.

We have implemented a generalization of the dataset-agnostic portfolio building meta-learner [32]. Specifically, we allow for multiple evaluations per dataset in the performance matrix through the `n_pipes` parameter, such that these can be used in the creation of the portfolio. This enlarges the possible portfolio size by, at most, a factor `n_pipes`. Therefore – building on the ordered set interpretation of the portfolio – we allow to specify the number of configurations to be gathered by the online phase's `total_n_configs` parameter, which is upper-bounded by the offline phase's `portfolio_size`.

MLTA provides the implementation of the generalized similarity-based direct configuration transfer approaches as the Top Similarity meta-learner, akin to approaches discussed in Section 2.2.2.1. The Top Similarity meta-learner directly transfers configurations from prior tasks based on their similarity, where the most similar prior tasks are preferred. To generalize beyond the discussed approaches the Top Similarity parameterizes the number of datasets to transfer from and the number of configurations to transfer from it.

The RankML [66] configuration characterization method has been implemented following the original work, but we need to explain design decision for unspecified details. We let the

[1] See `https://github.com/leightonvg/mlta` for the toolbox.

```python
from mlta.dataset_characterization import PreTrainedDataset2Vec
from mlta.metadatabase import MetaDataBase

# load metadatabase from "metadatabase_openml18cc" directory
mdbase = MetaDataBase(path="metadatabase_openml18cc")
# get dataset with id 0 in pd.DataFrame format
X, y = mdbase.get_dataset(dataset_id=0, type="dataframe")

# initialize the characterization method
d2v_char_method = PreTrainedDataset2Vec(split_n=0, n_batches=10)
# compute the characterization for dataset with id 0
d2v_char = d2v_char_method.compute(X, y)

# compute dataset2vec characterizations for entire mdbase and store them accordingly
d2v_chars = d2v_char_method.compute_mdbase_characterizations(mdbase=mdbase)
mdbase.add_dataset_characterizations(characterizations=d2v_characterizations, name="
    dataset2vec_split0_10batches")

# retrieve computed characterizations from metadatabase, possibly specifying datasets
mdbase.get_dataset_characterizations(characterization_name="
    dataset2vec_split0_10batches")
mdbase.get_dataset_characterizations(characterization_name="
    dataset2vec_split0_10batches", dataset_ids=[0])
```

Figure 3.3: Code example for dataset characterizations using Dataset2Vec and the metadatabase functionality. The example assumes to have an already created metadatabase in the directory "metadatabase_openml18cc". Other dataset characterization methods operate similarly.

hash represent the entire primitive (algorithm including hyperparameters) and not merely the algorithm. The hash is created using the MD5 hashing algorithm [107] on a byte representation – employing UTF-8 encoding [139] – of the full string representation of a scikit-learn [99] pipeline's step. We use the empty string to represent the 'blank' hash and only insert it after other hashes. Given that hash collisions are very unlikely when applying MD5 on the number of distinct primitives in typical metadatasets, these design decisions have as a consequence that every distinct configuration in the metadataset has a unique characterization. Moreover, this interpretation of the framework satisfies its desired deterministic property.

Also the RankML [66] meta-learner needs further specification, especially for incorporation in our modular framework. We use the scikit-learn compatible XGBoost-Ranker implementation from the Python-based xgboost package [20] with the approximate trees as specified in [19]. We train this model using the hyperparameter values provided in the paper – using defaults otherwise – on a pairwise ranking objective. For training we assign the ranks $\{0, 1, ..., N_C\}$ using an evaluation-metric-based ranking, where N_C is the number of configurations used for training.

Furthermore, though not necessary when using the RankML configuration characterization method, we need to specify the behavior when multiple configurations are assigned equal ranking scores, but we cannot recommend all of them. In such a setting we select the configurations based on their metadataset performance, instead of the ranking scores. Specifically, we select configurations by their minimal average normalized deviation from the metadata datasets' best evaluation score, averaging over all dataset evaluations the configuration has been evaluated for. This additional specification is only necessary due to our modular setup; we want to be able to use configuration characterization methods that do not necessarily produce unique characterizations.

We additionally parameterize the specification of the candidate models used during ranker-model inference, and the number of evaluations used during training. The original work used all distinct configurations in the metadataset as candidate models. Accordingly, we set this option as a default, but we allow to specify `max_n_models` – the maximal number of configurations that are considered *per dataset* in the metadatabase, selected on their maximal performance. This option is necessary for reducing the online phase time consumption in time-restricted settings utilizing a vast metadataset. Unlike other meta-learners, RankML does not have parameters for trading off exploration and exploitation, but this is possible with the introduction of `max_n_models`. In Figure 3.4 we give a code example for creating and using the RankML meta-learner to highlight the modular framework of MLTA.

```
from mlta.metadatabase import MetaDataBase
from mlta.dataset_characterization import WistubaMetaFeatures
from mlta.configuration_characterization import RankMLPipelineRepresentation
from mlta.metalearners import TopXGBoostRanked

# initialize metadatabase from "metadatabase_openml18cc" directory
mdbase = MetaDataBase(path="metadatabase_openml18cc")

# create meta-learner
metalearner = TopXGBoostRanked(dataset_characterization_method=WistubaMetaFeatures(),
    configuration_characterization_method=RankMLPipelineRepresentation(mdbase=mdbase))

# offline phase: train the model on entire mdbase but dataset with id 0,
# online phase: recommend 25 configurations for dataset with id 0.
mdbase.partial_datasets_view(datasets=[0])  # remove all info w.r.t. dataset 0
metalearner.offline_phase(mdbase=mdbase)
mdbase.restore_view()
X, y = mdbase.get_dataset(dataset_id=0, type="dataframe")
metalearner.online_phase(X, y, max_time=300, evaluate_recommendations=False, metric="
    neg_log_loss", total_n_configs=25, max_n_models=500)
top25_configs = metalearner.get_top_configurations()

# 2nd option: use pre-computed dataset & configuration characterizations in mdbase.
metalearner.clear_configurations()  # remove configurations from prior online phase
mdbase.partial_datasets_view(datasets=[0])  # remove all dataset 0 info, also chars
metalearner.offline_phase(mdbase=mdbase, dataset_characterizations_name="
    wistuba_metafeatures", configuration_characterizations_name="
    rankml_pipeline_representation")
mdbase.restore_view()
X, y = mdbase.get_dataset(dataset_id=0, type="dataframe")
metalearner.online_phase(X, y, max_time=300, evaluate_recommendations=False, metric="
    neg_log_loss", total_n_configs=25, max_n_models=500)
top25_configs = metalearner.get_top_configurations()
```

Figure 3.4: Code example for a characterization-based meta-learner depicting MLTA's modularity. It uses the RankML [66] meta-learner, with Wistuba et al. meta-features [132] as dataset characterization method, and the RankML configuration characterization method. For the online phase we specify we desire 25 configurations to be recommended, without evaluating them on the new dataset, in at most 5 minutes time. Additionally, we specified we only want to consider the 500 top-performing configurations per dataset as candidate models. The second example shows how to use pre-computed and stored dataset and configuration characterizations to avoid expensive offline phase computation. The example assumes to have an already created metadatabase in the directory "metadatabase_openml18cc" with dataset and configuration characterizations added. Other meta-learners function similarly.

Additional Baselines In addition to the selected meta-learning components (see Section 3.2) we present three baselines: Utility Estimate, a similarity-based meta-learner; Average Regret, a dataset-agnostic meta-learner; and Algorithms Propositionalization, a configuration characterization method. These baselines are incorporated in MLTA.

The similarity-based Utility Estimate meta-learner combines similarity values and configurations' evaluation scores to determine a configuration's utility for a new dataset. Firstly, it computes for each configuration the normalized regret on each dataset it is evaluated on, normalizing with respect to the dataset's evaluation scores. Secondly, it computes the similarity value between the new dataset and all datasets in the metadataset. Then, it computes the configuration utility c_u for each configuration in the metadataset by:

$$c_u = \frac{1}{|\mathcal{D}_{eval}|} \sum_{d=0}^{|\mathcal{D}_{eval}|} ((1 - cr_d) \cdot s_d) \tag{3.1}$$

where $\mathcal{D}_{eval}$ is the set of datasets in the metadataset for which the configuration is evaluated, cr_d is the configuration's normalized regret on dataset $d \in \mathcal{D}_{eval}$, s_d is the similarity value between the new dataset and dataset $d \in \mathcal{D}_{eval}$. Note $c_u \in [0, 1]$ if $\forall d \in \mathcal{D}_{eval} : s_d \in [0, 1]$, because $cr_d \in [0, 1]$. The approach suggests the configuration(s) with the highest utility c_u.

We deliberately choose a relatively simple approach to act as a baseline for other similarity-based meta-learners. Observe the approach is akin to the similarity-based direct configuration transfer meta-learner (dubbed Top Similarity) and the dataset-agnostic portfolio building meta-learner, because Utility Estimate tries to minimize the regret for the dataset at hand but by taking into account the similarity to that dataset. Unlike portfolio building, it tries to estimate the utility for one configuration on only one dataset, instead of the entire portfolio on all datasets. Unlike Top Similarity, it does not blindly transfer the configurations from a similar dataset, rather it takes their performance into account. A code example for creating the Utility Estimate meta-learner is illustrated in Figure 3.5.

```
1 from mlta.dataset_similarity import CharacterizationSimilarity
2 from mlta.dataset_characterization import FeurerMetaFeatures
3 from mlta.metalearners import UtilityEstimateLearner
4
5 sim_measure = CharacterizationSimilarity(characterization_method=FeurerMetaFeatures(),
      compare_characterizations_by="cosine_similarity")
6 metalearner = UtilityEstimateLearner(similarity_measure=sim_measure)
```

Figure 3.5: Code example for the baseline similarity-based meta-learner Utility Estimate, using a characterization based dataset similarity measure employing Feurer et al.'s meta-features [39] and cosine similarity.

The Average Regret meta-learner acts as a baseline for dataset-agnostic meta-learners. It is similar to Utility Estimate, but it does not require dataset similarity. It computes the normalized regret per configuration in the metadataset, normalized with respect to the dataset's evaluation scores, and ranks the approaches accordingly. In the online phase the highest-ranked configurations with at least `min_evals` evaluations are recommended. Coming back to the meta-learner comparison (Section 3.2.4), both baselines are applicable, scalable and useful.

Lastly, we present the configuration characterization baseline: Algorithms Propositionalization, which encodes the presence of each possible algorithm $a \in \mathcal{A}$ in the configuration, by setting the corresponding entry to 1 if it is present and 0 otherwise. Additionally, it

encodes the presence of pairs of algorithms, but in an order sensitive way. These two vectors are concatenated to create the configuration characterization. The size of the configuration characterization is thus $|\mathcal{A}| + |\mathcal{A}|^2$, and hence the characterization tends to be sparse. This method is applicable on any pipeline structure g, scales well with the number of datasets, configurations and evaluations in the metadataset and is free from parameters. A drawback of the approach is its relatively large dimensionality which depends on the search space. This baseline is deliberately naive in its approach: given a pipeline $(g, \mathbf{A}, \lambda)$ it only characterizes its algorithms $\mathbf{A}$, not its accompanying hyperparameters λ nor its *full* structure g. Observe this approach implies the configuration characterizations are not unique.

3.3.2.2 Provided Meta-Learning Approaches

Meta-Learning Approach	**Meta-Learner**	**Dataset-Similarity Measure**	**Dataset Characterization**	**Configuration Characterization**
AverageRegret	AverageRegret			
PortfolioBuilding	PortfolioBuilding			
UtilityEstimate-Wistuba	UtilityEstimate	Wistuba Cosine Similarity		
UtilityEstimate-Feurer	UtilityEstimate	Feurer Cosine Similarity		
UtilityEstimate-Dataset2Vec	UtilityEstimate	Dataset2Vec Cosine Similarity		
TopSimilarity-Wistuba	TopSimilarity	Wistuba Cosine Similarity		
TopSimilarity-Feurer	TopSimilarity	Feurer Cosine Similarity		
TopSimilarity-Dataset2Vec	TopSimilarity	Dataset2Vec Cosine Similarity		
RankML-Wistuba	XGBoost-Ranker		Wistuba Meta-Features	RankML
RankML-Feurer	XGBoost-Ranker		Feurer Meta-Features	RankML
RankML-Dataset2Vec	XGBoost-Ranker		Dataset2Vec	RankML
AP-Wistuba	XGBoost-Ranker		Wistuba Meta-Features	Algorithms Propositionalization
AP-Feurer	XGBoost-Ranker		Feurer Meta-Features	Algorithms Propositionalization
AP-Dataset2Vec	XGBoost-Ranker		Dataset2Vec	Algorithms Propositionalization

Table 3.2: Overview of meta-learning approaches we can create using the modularized MLTA framework. The approaches are grouped by their meta-learner categorization, the first two are dataset-agnostic, the following six are similarity-based, and the last six are characterization-based meta-learning approaches. The bold meta-learning approaches are new with the introduction of MLTA, the remaining approaches are re-implementations of prior works.

In this subsection we briefly highlight the power of MLTA's modularity. In Table 3.2 we give an overview of all meta-learning approaches that can be constructed using MLTA's provided meta-learning components. With 5 meta-learners we can create 14 meta-learning approaches using 2 configuration characterization methods, and 3 dataset characterization methods with their accompanying characterization-based similarity measures. Moreover, we can alter the approaches' parameters, such as using Euclidean distance instead of cosine similarity for characterization-based similarity measures. In addition to that, all meta-learners are implemented to have the possibility to evaluate configurations in the online phase. By doing so they return only a selection of the top-performing configurations. We have previously demonstrated that creating these meta-learning approaches requires only few lines of code, see Figures 3.3, 3.4 and 3.5.

Keep in mind – as shown in the dataset characterization method run time experiments – approaches' suitability may vary depending on the task. Unfortunately, we cannot replicate the implemented methods' performances because they are dependent on the original works' metadatasets, which are not made publicly available. However, we are eager to compare these meta-learning approaches on their effectiveness in an experimental setting (see Section 4.2), and evaluate their use for incorporation in an AutoML system (see Section 4.3).

3.3.3 Dataset Characterization Method Run Time Analysis

In this subsection we analyze the run time of the incorporated dataset characterization methods, hoping to validate the scalability claims in the corresponding selection (see Section 3.2.1). Hence, we analyze the run time behavior for Dataset2Vec [55], Wistuba meta-features [132] and Feurer meta-features [39] with respect to datasets' number of samples, features, and outcomes. We execute the dataset characterization methods[2] on OpenML-CC18 [15]: a large (n=72), diverse and curated collection of medium-sized classification datasets with extensive use in AutoML-related research. We perform these analyses because the run time of a dataset characterization method has implications for the run time – and thus viability – of characterization-based meta-learners, but also similarity-based meta-learners employing a characterization-based similarity measure. The results are portrayed in Figure 3.6

None of the scatter plots fit their corresponding line well, rather they violate linear regression assumptions including homoscedasticity and normality of the residuals. Therefore, we cannot confirm the dataset characterization methods scale linearly with respect to the dataset's properties (number of instances, features and outcomes). The results reveal that some datasets require far more time than others. These outliers are annotated and mostly consist of image-based datasets such as MNIST [27] or datasets with lots of features, such as UCI's Internet-Advertisements [65]. These annotated datasets have many samples, features and/or outcomes, which – especially combined – contribute to increased run time. With exceptions for the outliers, the characterization methods seem to have near constant run time due to the distorted scaling for the y-axis. Therefore, we provide an additional illustration (see Figure 3.7) similar to Figure 3.6 but without the annotated data points. These clouds of points seem to better fit their corresponding least-squares line, but the linear regression assumptions are still violated. The additional illustration still depicts greater run times for the Feurer meta-features, but also reveals that only Dataset2Vec requires at least a few seconds for small datasets.

[2] using an AMD Ryzen 5 3600XT CPU

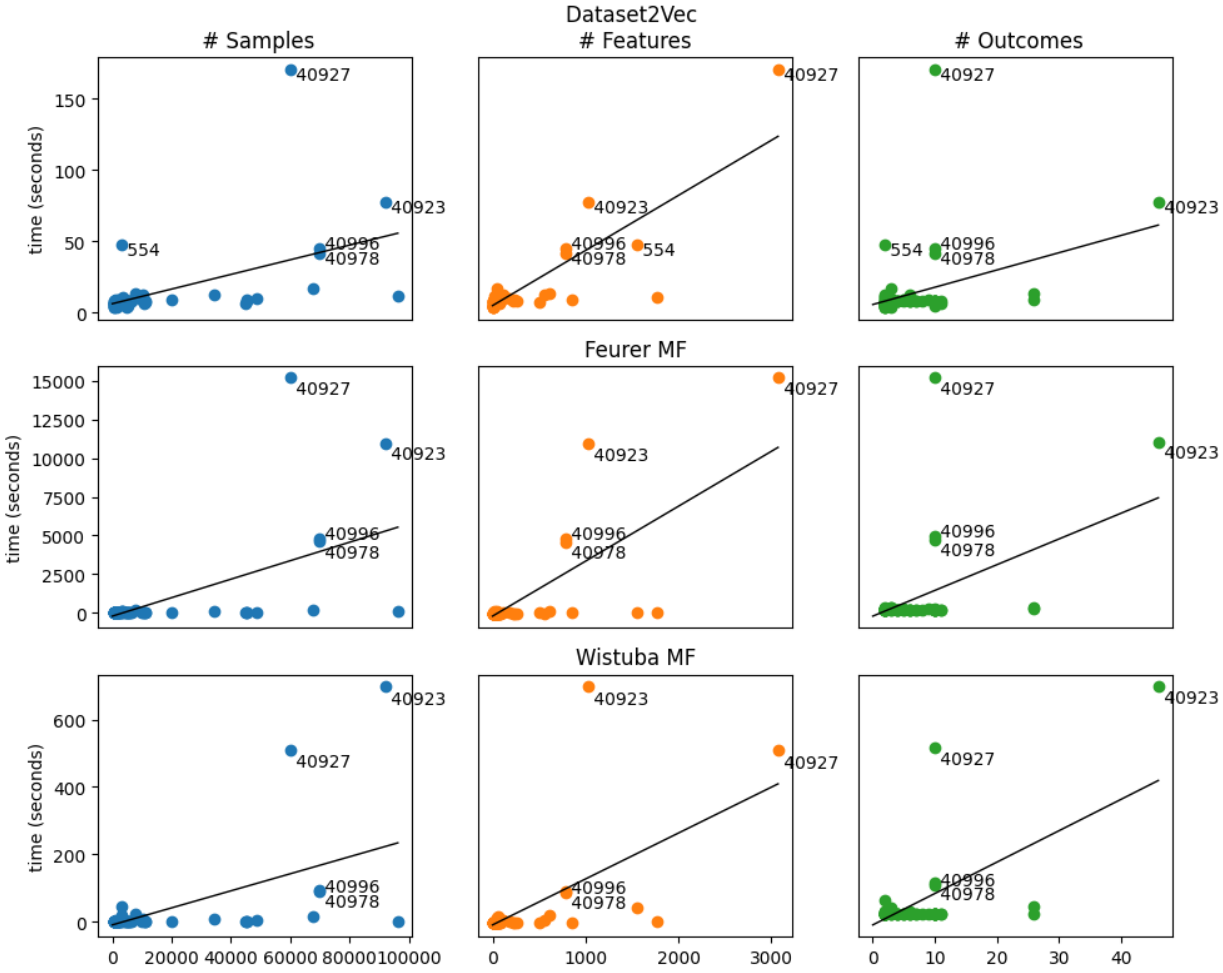

Figure 3.6: Dataset characterization methods' run times plotted against the number of samples, features and outcomes in OpenML18CC [15] datasets. For each cloud of points we depict the best fitting (least-squares) straight line. Outliers are annotated with their dataset id on OpenML.

We must be careful interpreting these plots because the run time for characterizing a dataset depends on all its properties. Unfortunately, this implies we can not deny nor confirm the aforementioned scalability claims. Additionally, it explains why the height for each dataset's point is the same across its horizontally neighboring plots. Despite the impossibility of assessing the scalability in absolute fashion, we are able to compare the approaches to each other. Doing so, both figures indicate the Feurer meta-features are more far computationally expensive, with characterization run times up to 4 hours. Furthermore –as exhibited by its greater run time difference between outliers and non-outliers – the Feurer meta-feature-based dataset characterization is less scalable than the other methods, which partially confirms our prior beliefs (in Section 3.2.1).

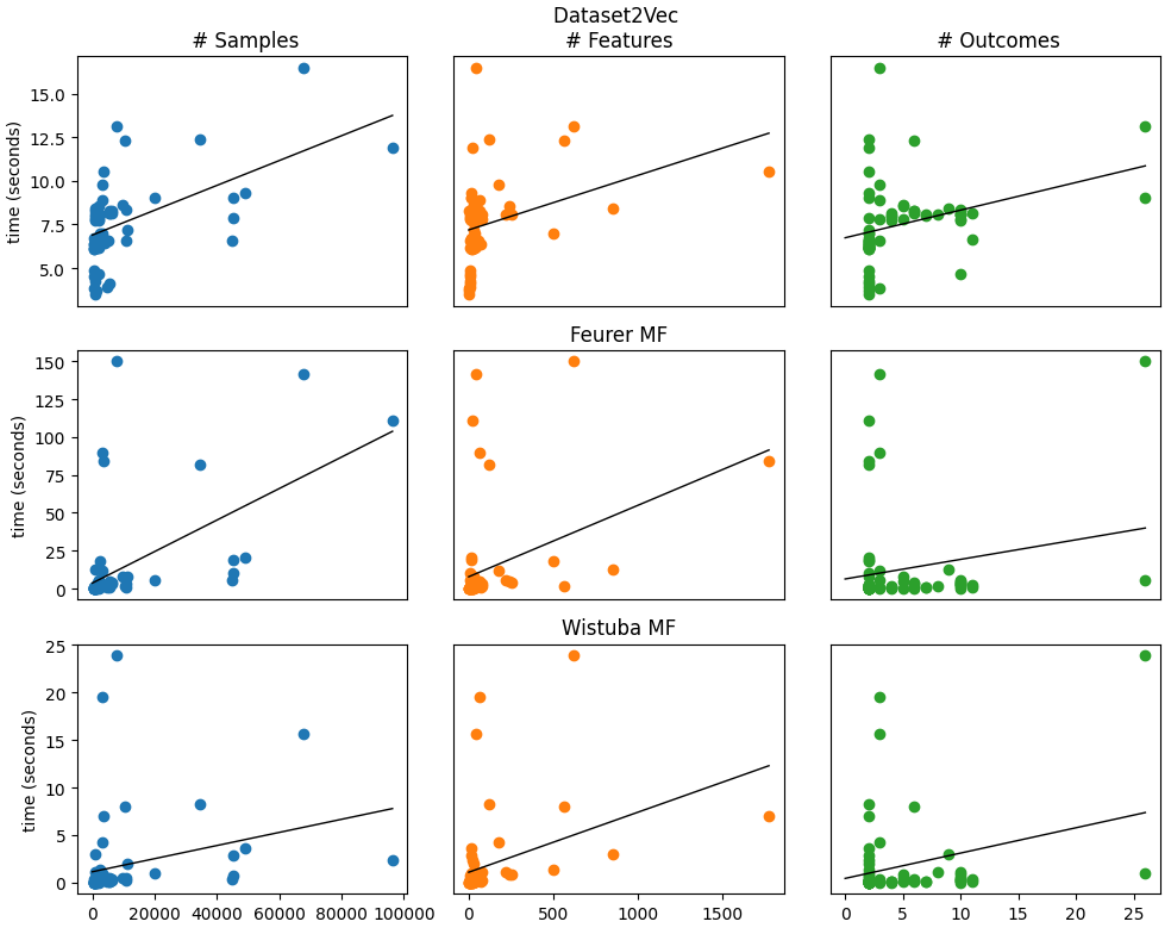

Figure 3.7: Dataset characterization method run times plotted against the number of samples, features and outcomes in OpenML18CC [15] datasets. Similar to Figure 3.6, but excluding its outliers. For each cloud of points we depict the best fitting (least-squares) straight line.

The remaining methods, Dataset2Vec and Wistuba meta-features, seem to be computationally viable, though they also require more time for the annotated datasets. The analysis suggests Dataset2Vec is suitable for datasets with many samples, features and outcomes. Contrarily, the Feurer meta-features are computationally demanding for larger datasets, potentially limiting its applicability thereto. The Wistuba meta-features are far less computationally expensive than the Feurer meta-features, it only struggles with (the two largest) datasets simultaneously having many samples, features, and outcomes.

Additionally, we want to point out that the datasets annotated with id 63 and 27 possess similar properties and exhibit similar behavior, in fact these are the intimately connected MNIST [27] and Fashion-MNIST [134] datasets. Though not the initial intent of the analysis, we do highlight this finding suggests all considered dataset characterization methods are stable in terms of their run time.

Chapter 4

MLTA Case Study: Meta-Learning for Warm-Starting Pipeline Search

In Chapter 3 we introduced the Meta-Learning Toolbox for AutoML (MLTA). In the current chapter we evaluate its utility. In particular, we explore the use of MLTA for warm-starting evolutionary pipeline search, as provided by the AutoML research framework GAMA [44]. Thereby we aim to validate our research objective: create a toolbox to easily construct, analyze and compare existing and novel meta-learning approaches and components.

As mentioned previously (see Section 2.2.2.1), the goal of warm-starting is to attain similar performance at reduced computational cost, or increased performance at similar computational cost, or preferably increased performance at reduced computational cost. To allow for assessing these goals and our research objective we execute the case study in a 2-step approach. We additionally aim to emulate the real-world usage as a prototyping tool, where limited computational resources prohibit the enumeration of all options for computationally demanding experimental evaluations – e.g. warm-starting GAMA.

In the first step we construct MLTA's meta-learning approaches to compare and analyze them in a time-restricted setting. Therein we grant only 5 minutes of online learning to recommend 25 previously-seen pipeline configurations, thus *without* tuning configurations – we simply transfer and evaluate them. The time-restricted analysis results are used to identify the meta-learning approach with the highest potential for warm-starting, without actually performing warm-starting. This allows us to perform the second step: incorporate and evaluate the highest-potential meta-learning approach for warm-starting GAMA.

We stress the goal of this case study is *not* necessarily to improve GAMA's search with warm-starting. Instead, by performing the 2-step case-study we aim to validate our research objective, justifying the claim that we created a toolbox that allows for easily constructing, analyzing and comparing existing and novel meta-learning approaches and components.

The remainder of this chapter is structured as follows. In Section 4.1 we discuss the creation of a set of previous experiences – a metadataset – to be used by the meta-learners, using the metadatabase functionality provided by MLTA. In Section 4.2 we discuss the first step of the case study: an analysis of meta-learning approaches with a time-restricted online phase, therein we employ meta-learning approaches facilitated by MLTA. Having used the time-restricted analysis to select a meta-learner for warm-starting, we warm-start evolutionary pipeline search in Section 4.3. Reproducibility instructions are found in the toolbox documentation [1].

[1] `https://github.com/leightonvg/mlta`

4.1 Metadataset

A large part of creating a meta model is, as with any ML model, the data itself. Many existing works create a metadataset specifically for their own purpose, which is rarely shared. Hence, we need to create a high-quality metadataset ourselves for comparing meta-learning approaches, ideally in a broad manner allowing others to reuse it.

A fully complete metadataset would contain lots of information by exhaustively evaluating each solution in the configuration search space, possibly multiple times for solutions with varying performance, all on many different datasets and performance metrics. It quickly becomes apparent that creating a fully complete metadataset is computationally infeasible. Even more so, in our AutoML formalization (see Definition 1) we do not restrict the size of pipelines, effectively allowing for an infinite pipeline length and hence infinite search space. Fortunately, recent work suggests an opportunistic, and possibly unreliable, set of prior experiences can still be fruitful for guiding AutoML search [92]. Though we cannot and need not construct an exhaustive metadataset, we still care about completeness because it is directly related to the external validity of our case study.

A main consideration for any metadataset is the collection of datasets – or tasks – to consider, aligning with MLTA's focus we only consider classification datasets. For the creation of the metadataset we consider OpenML-CC18 [15]: a large (n=72), diverse and curated collection of medium-sized classification datasets which has been extensively used in AutoML-related research. The next main consideration is which configurations' performance we want to record and on which of the datasets in OpenML-CC18.

This choice is particularly difficult due to an infinite number of configurations and because we need to trade-off the metadataset's completeness with its configurations' performance, which could be dataset dependent. We propose to overcome this difficulty by deferring the choice to AutoML pipeline search, recording the dataset-configuration evaluations it encounters in the process. Pertaining to metadataset creation, existing search methods differ on two main aspects: completeness and performance, which we can trade-off using hyperparameters. But, computational cost prohibits the enumeration all possible search methods and their parameters. Rather, to trade-off the metadataset completeness and its configurations' performance we propose to use multiple distinct search methods – varying on their completeness-performance trade-off – to populate the metadataset with prior experiences, namely:

- Random Search: Randomly pick machine learning pipelines from the search space and evaluate them.
- Asynchronous Evolutionary Algorithm: Evolve a population of machine learning pipelines, drawing new machine learning pipelines from the best of the population.
- Asynchronous Successive Halving Algorithm: A bandit-based approach where many machine learning pipelines iteratively get evaluated and eliminated on bigger fractions of the data.

With random search focusing on completeness, the Asynchronous Successive Halving Algorithm being performance-oriented, and the Asynchronous Evolutionary Algorithm focusing on both aspects. Note, the latter two algorithms depend on their initialization and could benefit from multiple runs but in the creation of the metadataset we only employ them once, arguing the composition of multiple search methods yields sufficient quality, especially considering multiple runs may yield similar results and thus waste limited computational resources.

The metadataset quality depends on the performance of the configurations found during the search methods, whose performance is mainly impacted by the amount of resources dedicated to optimization. The aforementioned search methods to populate the metadataset are incorporated in GAMA [44], which has been shown to achieve satiable performance on the OpenML-18CC datasets given an optimization budget of 1 hour, 32GB of RAM and 8 vCPUs [42]. To create the metadataset we employ GAMA with the aforementioned search methods and optimization budget[2] for each search method separately, using otherwise default hyperparameters. We store these results using MLTA's `MetaDataSet` functionality.

However, we store encountered configurations and their associated negative logistic loss performance only if they are evaluated on a full dataset, thus neglecting many evaluations during the Asynchronous Successive Halving Algorithm. The stored performance estimate for a pipeline configuration is reliable, because it is computed using 5-fold cross-validation. Due to employing multiple search methods, the metadataset may contain multiple evaluations of the same configuration on the same dataset using the same metric, we aggregate these instances by averaging their negative logistic loss score.

Performing the aforementioned process required 216 hours of computational time – with the aforementioned optimization budget – to yield a metadataset composed of 985.475 configuration evaluations and 826.270 distinct pipeline configurations. Though each configuration is evaluated ≈ 1.2 times on average, there are plenty of configurations with numerous evaluations since the distribution is extremely long tailed. Only 66.050 pipelines are evaluated more than once, totalling 225.255 evaluations. In similar fashion, the metadataset is composed of 534 configurations with at least 25 evaluations. The number of evaluations per dataset varies greatly with the most and least evaluated dataset having 76.945 and 21 evaluations respectively (see Appendix B), which varies due to differences in dataset size and associated configuration evaluation time. All non-image datasets in OpenML-18CC have at least 400, but often far more, configuration evaluations. We include these image-based datasets because they can be interpreted in a tabular format, and because they are part of OpenML-18CC, which we do not want to alter unnecessarily.

Another approach for creating a metadataset, requiring no to little compute resources, is to use OpenML [125]. OpenML contains many evaluations (runs) and pipeline configurations (flows), but due to some issues it is not well suited as a source for creating our metadataset:

- The flows, and associated runs, are spread across distinct machine learning frameworks, with varying versions themselves. Because performance may vary across implementations and versions, combining these results is non-trivial.
- With complete ML pipelines in mind, many datasets have an underexplored search space, which may lead to a poor-quality metadataset in terms of completeness. Typically, preprocessing transformations are simple, if there are any. Moreover, apart from a bias towards pipeline structure, there often is a bias towards certain aspects of the pipeline. For instance, there are users contributing many evaluations only exploring the effect of hyperparameters, disregarding algorithm selection for preprocessing and prediction.

As mentioned previously, we desire a metadataset which is as complete as possible – which, unfortunately, conflicts with the aforementioned issues. Therefore, despite containing numerous previous experiences, OpenML is poorly suited for directly creating the metadataset we desire.

[2]We used 8 out of 12 threads of an AMD Ryzen 5 3600 XT CPU

4.2 Time-Restricted Meta-Learning

In this section we use the introduced metadataset (Section 4.1) as previous knowledge for meta-learners whose online phase is time restricted. Evaluating meta-learners in this time-restricted setting has three goals; firstly it aims to compare and analyze meta-learners' performance when employed with limited computational resources; secondly it aims to identify the highest potential meta-learner to later be used in warm-starting pipeline search (see Section 4.3); thirdly it serves as a validation for the research objective: a toolbox to easily construct, analyze and compare existing and novel meta-learning approaches and components.

We construct all meta-learning approaches provided by MLTA's modularized framework (see Table 3.2). For each meta-learning approach we perform a leave-one-dataset-out evaluation on OpenML-CC18: we evaluate each meta-learner on each dataset in OpenML-18CC, using the remaining datasets' metadata as previous experience to learn from. Each meta-learner is given 300 seconds of online phase computation time, and unlimited offline phase time. In the online phase the meta-learner is tasked with recommending 25 pipeline configurations suitable for the left out dataset. The meta-learner does not get access to the dataset and associated metadata it is evaluated on. Nor does it get the possibility to evaluate and tune the recommended evaluations on the new dataset, because that is the job of a subsequent optimization procedure which we do not yet consider. We evaluate the 25 recommended pipeline configurations on negative logistic loss using a 80/20 holdout split on the left out dataset.

We evaluate meta-learners using the MLTA framework (see Figure 3.2). We combine instances indirectly inheriting from `BaseLearner` – the meta-learners – with an evaluation method inheriting from `BaseEvaluation` – a `LeaveOneDatasetOutEvaluation` – and a `MetaDataBase` instance giving access to the metadataset and providing AutoML functionality. Figure 4.1 shows that we can construct, execute and evaluate a meta-learner with few lines of code.

```
1 from mlta.metadatabase import MetaDataBase
2 from mlta.metalearners import PortfolioBuilding
3 from mlta.evaluation import LeaveOneDatasetOutEvaluation
4
5 mdbase = MetaDataBase(path="metadatabase_openml18cc")
6 metalearner = PortfolioBuilding()
7 evaluation_method = LeaveOneDatasetOutEvaluation(validation_strategy="holdout",
      test_size=0.2, n_configs=25, max_time=300)
8 evaluation_results = evaluation_method.evaluate(mdbase, metalearner)
```

Figure 4.1: Code example for evaluating a meta-learner. Due to the MLTA framework we need not bother with evaluation, dataset and metadataset related operations. The example assumes to have an already created metadatabase in the directory "metadatabase_openml18cc". Other meta-learners can be evaluated similarly.

We compare the performance of meta-learning approaches by the negative logistic loss (NLL) of their highest-ranked pipeline configuration recommendation on each dataset and rank the approaches accordingly. To emulate typical limitations in a time-restricted setting we select the highest-ranked configuration by allowing a handful of evaluations. Specifically, we evaluate the 5 configurations ranked highest by the meta-learner and create the meta-learner ranking by the NLL of its top-evaluated pipeline (among the 5 pipelines). Thus, the meta-learners themselves are not allowed to evaluate configurations, but we evaluate their recommended configurations to create the meta-learner ranking. Not only do we belief this

evaluation procedure emulates typical limitations, we belief it may achieve satiable performance; research suggests selecting the meta-learner's best-evaluated pipeline from a handful of its top recommendations yields comparable performance to AutoML [66].

To ground the meta-learner results we include the three one-hour pipeline optimization results from the metadataset (see Section 4.1) utilizing Random Search, the Asynchronous Evolutionary Algorithm (AsyncEA) and Asynchronous Successive Halving Algorithm (ASHA), using their best-evaluated pipeline configuration. Note that the scores for these reference AutoML results are computed using a 5-fold cross-validation and are therefore comparable to the 80/20 hold-out split, though the reliability of the estimates differ. Since the meta-learners do not get the possibility to evaluate and tune the recommended configurations on the new dataset, it would be surprising – but possible – for them to outperform the 1-hour AutoML reference scores with only 5 minutes of online phase time.

As explained in Chapter 3, MLTA allows the pipeline configuration to be evaluated prior to recommending them, since it is a possibility for any of the discussed meta-learners. Because we are interested in warm-starting pipeline optimization we do not evaluate configurations before recommending them, because they will be evaluated within the optimization procedure itself. It is unknown whether evaluating configurations before recommending them for warm-starting pipeline optimization is beneficial, investigating this is out-of-scope for this thesis, but it could be a topic for follow-up investigation.

As previously mentioned, we aim to emulate real-world usage of the toolbox where computational resources are not abundantly available. Yet, this aim does not prohibit the enumeration of all meta-learning approaches in this initial analysis, because each approach is time constrained.

Despite the time-restricted setting, we grant the meta-learner unlimited time during the offline phase because this prior information can be pre-computed and accumulates over time, where AutoML systems can often avoid re-computation, in turn yielding a relatively inexpensive offline phase. For instance, to employ a characterization-based similarity measure we need not compute characterizations for prior tasks, only for the new task, which should be inexpensive compared to the online phase and a potential subsequent pipeline optimization procedure.

The current setup requires $72 \cdot 14 = 1008$ dataset-meta-learner combinations to be evaluated, each requiring their own offline, online and evaluation phase. Thus, we perform the following 1008 times: construct a meta-learner, recommend 25 pipelines with it and evaluate each of them. Therein the typical parallelization while using cross-validation is infeasible, because the meta-learners themselves are allotted the previously mentioned computational resources. Therefore we employ holdout during the leave-one-dataset-out evaluation procedure.

Unfortunately, with the aforementioned setup we cannot adequately assess all approaches due to a data leakage issue: Dataset2Vec employs a pre-trained network trained on UCI classification datasets, with some of them occurring in OpenML-CC18. Even though Dataset2Vec was trained on a different task – dataset similarity prediction – we cannot evaluate it on OpenML-CC18, because its pre-trained network has seen parts of the supposed-to-be test set. We have two options: either exclude UtilityEstimate-Dataset2Vec, TopSimilarity-Dataset2Vec, RankML-Dataset2Vec and AP-Dataset2Vec from the analysis, or alter OpenML-CC18. Instead of preferring one analysis over the other, we provide the results for both analyses in Section 4.2. Thus, we provide the experimental results for the full OpenML-CC18 collection, which excludes the Dataset2Vec approaches. In addition to that, we provide the results for an OpenML-CC18 subset – a selection of 38 out of 72 datasets – excluding Dataset2Vec's pre-training datasets (see Appendix D). Beware that this selection does *not* imply a different metadataset.

Results

Ranks We aim to rigorously compare the ranks using a critical differences (CD) diagram as introduced by Demšar [25]; a CD diagram is a compact visual representation of the approaches' average ranks and significance tests on pairs of approaches with horizontal lines grouping statistically indistinguishable approaches. However, before we are able to produce a CD diagram we must first test whether there is any difference among the meta-learning approaches, for which we employ the non-parametric Friedman test [40]. For both the full OpenML-CC18 collection and the subset we can reject the Friedman test's null hypothesis with $p < 0.0001$ and $p \approx 0.0107$ respectively. We depict the CD diagrams comparing the meta-learning approaches and the pipeline optimization procedures in Figures 4.2 and 4.3. Therein we do not employ Nemenyi's test as proposed by Demšar [25], rather we use a Wilcoxon signed-rank test since it is favored by more recent work [12]. In employing these tests we consider a significance level $\alpha = 5\%$, applying a Bonferroni correction [114] to correct for multiple testing. We correct for multiple testing because otherwise we are expected to create ≈ 14 false positives with $\alpha = 5\%$; in comparing 17 approaches we perform $17 \cdot 16 = 272$ pair-wise tests.

The CD diagram for the OpenML-CC18 subset (Figure 4.2) reveals that we cannot reject equivalence of any meta-learning approach with any baseline. Please note that this does *not* imply these approaches perform equivalently because "absence of evidence is not evidence of absence" [6]. Perhaps surprisingly, we can claim that Portfolio Building outperforms 5 other meta-learning approaches, namely: AP-D2V, AP-Wistuba, UtilityEstimate-Feurer, TopSimilarity-Feurer and RankML-Wistuba. Thus, in context of meta-learner categorization, the dataset-agnostic portfolio building outperforms three characterization-based meta-learners and two similarity-based meta-learners.

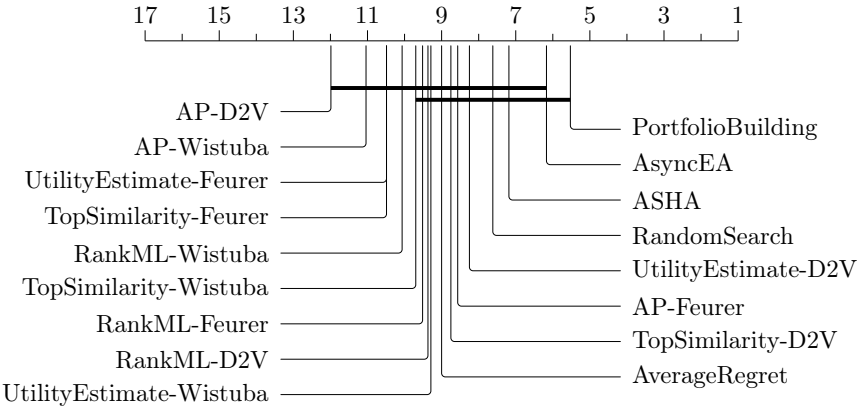

Figure 4.2: Critical differences (CD) diagram comparing meta-learning approaches on the OpenML-CC18 subset. We can create this CD diagram – i.e. perform pairwise tests – because we reject the null hypothesis for the corresponding Friedman test ($p \approx 0.0107$). AsyncEA, ASHA and RandomSearch do not employ meta-learning, they are AutoML reference scores.

The CD diagram for the full OpenML-CC18 collection (Figure 4.3) tells a similar story. Compared to the OpenML-CC18 subset we now have more measurements – which increases the power of our statistical tests – resulting in more statistically significant distinctions among approaches. Now we can confirm that the 1-hour AutoML references outperform a meta-learner, namely AP-Wistuba. It indicates that the combination of a characterization-based

meta-learner with simple dataset- and configuration characterizations may not be the best approach for knowledge transfer. Moreover, this analysis confirms that Portfolio Building is the best meta-learner: with statistical significance we can state that Portfolio Building outperforms all other meta-learners. Yet, we cannot reject Portfolio Building's equivalence from the 1-hour AutoML references. Again we emphasize that "absence of evidence is not evidence of absence" [6], and these results do therefore not mean that Portfolio Building performs equivalently to the 1-hour AutoML references.

Despite varying average ranks, we have found little statistically significant differences between meta-learning approaches. Therefore, we can not claim that certain meta-learning *components* outperform others.

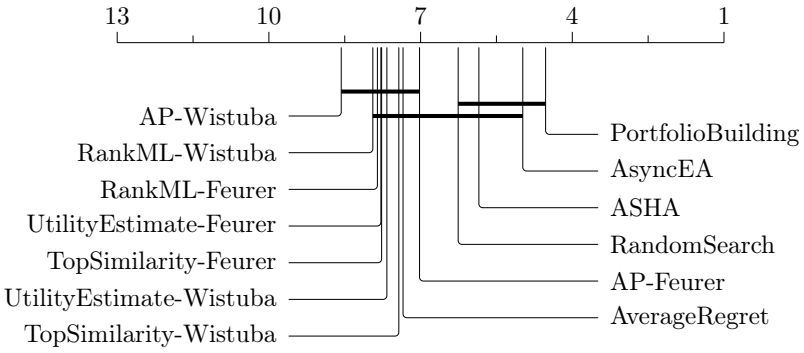

Figure 4.3: Critical differences (CD) diagram comparing meta-learning approaches on the full OpenML-CC18 collection. It excludes the Dataset2Vec-based approaches due to aforementioned data leakage issues. We can create this CD diagram – i.e. perform the pairwise tests – because we reject the null hypothesis for the corresponding Friedman test ($p < 0.0001$). AsyncEA, ASHA and RandomSearch do not employ meta-learning, they are AutoML reference scores.

We visualize the rank distribution for each approach in Figures 4.4 and 4.5. These illustrations do not allow for well-grounded comparisons on average ranks. Rather, they allow for additional qualitative analyses of the rank distributions.

We observe that the approaches' rank distributions are comparable for both figures, i.e. their kernel densities are similar in shape. The shape thereof indicates the approaches' consistency, with more uniform densities indicating less consistency. On this note, we observe some approaches to more consistently rank poorly, including AP-D2V, AP-Wistuba, UtilityEstimate-Feurer, TopSimilarity-Feurer, TopSimilarity-Wistuba – a list very comparable to the approaches outperformed by PortfolioBuilding for the OpenML-CC18 subset (Figure 4.2). Contrarily, both AsyncEA and PortfolioBuilding more consistently receive high ranks. Notably, this consistency is greater for PortfolioBuilding than for any other meta-learning approach, and perhaps some of the AutoML procedures. Lastly, both figures indicate that AsyncEA seems to be more consistent in achieving better ranks than the other AutoML optimization procedures.

Considering the OpenML-CC18 subset (Figure 4.4), we note that none of the metadataset baselines receive the two worst ranks among any dataset. Moreover, AsyncEA never obtains the 3 worst ranks and RandomSearch never attains the best ranking. This behavior is only partially visible for the full OpenML-CC18 collection (Figure 4.5), there only AsyncEA never receives the worst ranking.

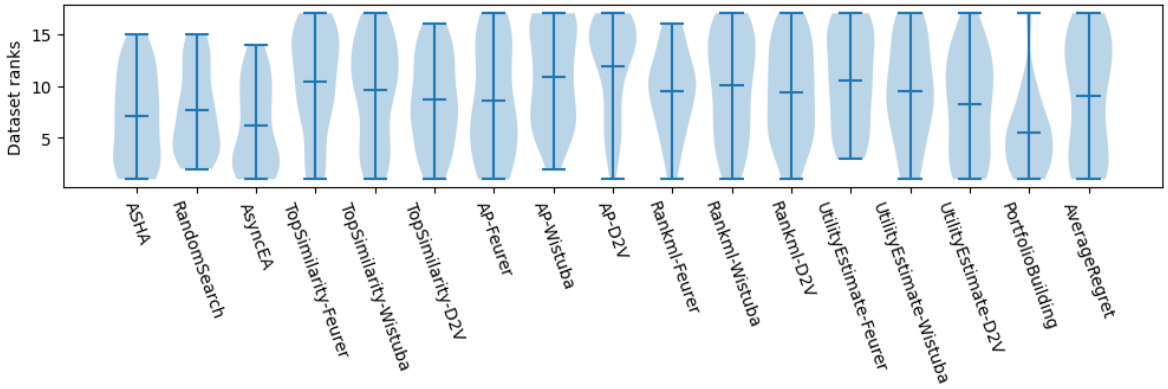

Figure 4.4: Violin plot of the dataset ranks attained on the OpenML-CC18 subset.

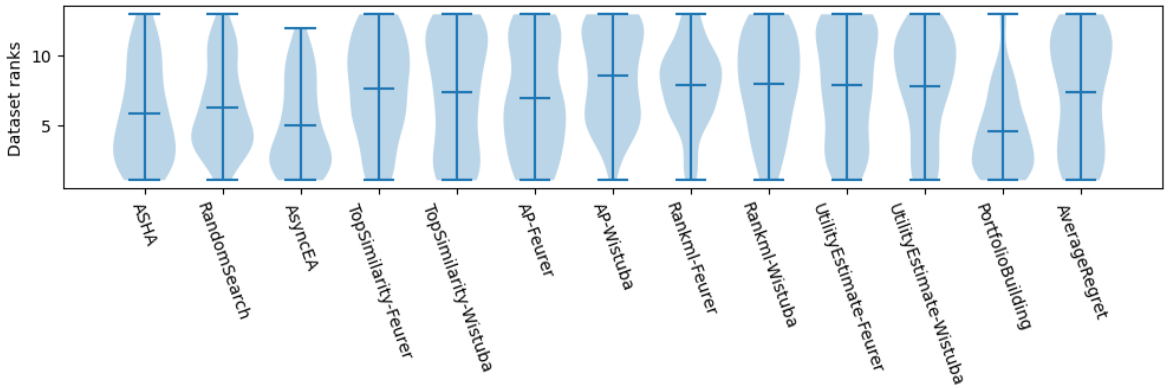

Figure 4.5: Violin plot of the dataset ranks attained on the full OpenML-CC18 collection.

Regret Selecting the meta-learning approach with the highest warm-starting potential is not as simple as taking the method with the highest average rank in the CD diagram, because it only considers the top-ranked recommended pipeline configurations. For warm-starting we provide a larger set of configurations, thus a meta-learner selection could account for it. We are unaware of any work exploring warm-starting potential, that is, extrapolating a meta-learning approach's evaluation results in a time-restricted setting to the final performance obtained by using it for warm-starting AutoML search.

We are not attempting to create such a score for it heavily relies on the exploration-exploitation trade-off in AutoML. Rather, we aim to gain better insights into the performance of the proposed approaches by providing their aggregated regret and standard deviation in Table 4.1. In essence the aggregated regret is the average normalized regret with respect to both datasets and its recommended pipeline configurations, a formalization can be found in Appendix E. The aggregated standard deviation is the standard deviation of the recommendations' evaluation scores, averaged over datasets.

In the context of warm-starting, we consider the aggregated regret to be a proxy for the exploitation focus of a meta-learning approach; a lower regret prioritizes higher evaluation scores and punishes for missing recommendations. Contrarily, the standard deviation is a proxy for the diversity – or exploration in a warm-starting context – of the meta-learning

approach. These values provide an additional – but not leading – analysis into the exploration and exploitation of the presented meta-learning approaches. Note that the regret values are informative in a relative sense instead of absolute; they depend on the performance of the other approaches.

Meta-Learning Approach	Full OpenML-18CC	OpenML-18CC Subset
AverageRegret	**0.1012** ± 0.0992	**0.1366** ± 0.1295
PortfolioBuilding	0.1462 ± **0.2041**	0.1919 ± **0.2437**
UtilityEstimate-Wistuba	0.1484 ± 0.0941	0.2116 ± 0.1165
UtilityEstimate-Feurer	0.1722 ± 0.0997	0.2338 ± 0.1008
UtilityEstimate-Dataset2Vec		0.1975 ± 0.1452
TopSimilarity-Wistuba	0.1559 ± 0.1547	0.2119 ± 0.1851
TopSimilarity-Feurer	0.1601 ± 0.1376	0.2325 ± 0.1601
TopSimilarity-Dataset2Vec		0.1822 ± 0.1662
RankML-Wistuba	0.1761 ± 0.1861	0.2286 ± 0.2093
RankML-Feurer	0.2108 ± 0.1942	0.2817 ± 0.2002
RankML-Dataset2Vec		0.2113 ± 0.2193
AP-Wistuba	0.2146 ± 0.0615	0.2970 ± 0.0770
AP-Feurer	0.1684 ± 0.0564	0.2268 ± 0.0398
AP-Dataset2Vec		0.1687 ± 0.0860

Table 4.1: Aggregated regret and standard deviation of meta-learning approaches on the OpenML-CC18 collection [15] and its subset (Appendix D). The regret is computed according to Equation E.2.

We note that the AverageRegret meta-learner achieves the lowest regret for both the full OpenML-18CC collection and its subset, we claim this is because it recommends configurations based on the aggregated regret. It is a prime example that a lower aggregated regret does not imply a high average rank in the CD diagrams. For warm-starting evolutionary search we seek a meta-learner that has relatively low regret but a relatively high amount of standard deviation.

We remark that all meta-learning approaches including a baseline meta-learning component – Algorithms Propositionalization, UtilityEstimate and AverageRegret – achieve low standard deviation in their recommendations. We hypothesize that the approaches including algorithms propositionalization achieve low variety because the configuration characterization method cannot distinguish well between configurations with varying hyperparameters and pipeline structure. Due to this incapability it is likely many similar pipelines are recommended, which in turn obtain similar evaluation scores. Thus, it seems that for warm-starting AutoML we prefer configuration characterization methods that can distinguish pipelines with varying hyperparameters and structures well. We are not certain why the AverageRegret and UtilityEstimate approaches obtain low standard deviation. A possibility for the UtilityEstimate meta-learners is that the evaluation scores for the most similar dataset are very close to each other. Then, the meta-learner could completely disregard other datasets and only sample configurations from one dataset, whose top configurations are likely similar.

Out of all approaches PortfolioBuilding has the highest standard deviation for both the full OpenML-18CC collection and its subset, while maintaining relatively low levels of regret. In fact, there is no approach that simultaneously has lower regret and higher standard deviation, for both OpenML-18CC variants. We postulate Portfolio Building's supposedly high configuration diversity is achieved by only considering the top performing configuration per dataset, thereby sampling configurations from 25 distinct datasets. The other meta-learning approaches also consider the configurations that did not achieve the best performance on the dataset, and are thus more likely to include configurations from a smaller number of datasets.

Selecting a warm-starting approach We should not select a meta-learning approach for warm-starting purely on one aspect. In that spirit, we compared approaches on their average rank, rank distributions, aggregated regret and standard deviation. Portfolio building outperforms (all) other meta-learning approaches on their average dataset ranks, achieves consistent low ranks on each of the datasets, and seems to have relatively low aggregated regret and high configuration diversity – the latter being especially important for warm-starting evolutionary search. Therefore, we conclude that Portfolio Building is the meta-learning approach with highest potential for warm-starting AutoML search. Though we cannot make strong claims, the analyses seem to indicate that AsyncEA is the highest-performing search method. Therefore we warm-start AsyncEA using Portfolio Building in Section 4.3.

4.3 Warm-Starting Asynchronous Evolutionary Pipeline Search

In this section we aim to further validate our research objective – the creation of a toolbox to easily construct, analyze and compare existing and novel meta-learning approaches and components – by using MLTA for warm-starting AutoML search. However – unlike the previous section – we do not compare meta-learning approaches and components, because warm-starting experimentation requires far more computational resources. In the previous section we have selected the portfolio building meta-learning approach for warm-starting asynchronous evolutionary pipeline search using GAMA, which we will refer to as warm-started AsyncEA from now on. We note that we could warm-start the asynchronous successive halving algorithm in similar fashion, but we cannot warm-start random search.

We allot warm-started AsyncEA one hour of total computation time for pipeline optimization, excluding time spent in metadataset related operations such as data loading. Therein we allow the meta-learner – portfolio building – to use a maximum of 5 minutes for recommending at most 25 pipeline configurations to warm-start AsyncEA, the remaining computation budget (at least 55 minutes) is spent on the pipeline search. The (at most 25) recommended pipeline configurations are incorporated in the initial population consisting of 50 individuals in total, with the remaining (at least 25) individuals being generated randomly.

We have no particular reason to choose for 5 minutes of online phase meta-learning computation time. To the best of our knowledge there are no works that could indicate sensible values, though amounts larger than the optimization budget are clairvoyantly nonsensical. For most meta-learning approaches the amount of time may be a crucial aspect, because it influences the exploration-exploitation trade-off for the subsequent warm-started pipeline optimization. For the current dataset-agnostic portfolio building approach we expect the online phase to almost immediately return the warm-starting individuals – in substantially less than 5 minutes – because we only need to recommend the configurations identified in the offline

phase. Therefore, we do not expect the results *for this meta-learning approach* to depend on the amount of time allotted to it.

We opt for an initial population totalling 50 individuals because that is the default value provided by GAMA. We suppose that replacing half of these individuals is sensible, because we have seen indicators suggesting that portfolio building tends to recommend a relatively diverse set of configurations (see Table 4.1). Yet, we still opt for an initial population partially consisting of random individuals to enforce diversity, because it is a crucial aspect for evolutionary search.

The number of warm-starting individuals also affects the exploration-exploitation trade-off. The work introducing portfolio building [32] provided results suggesting 16 configurations may be a sensible value for warm-starting Bayesian optimization – an AutoML search procedure different from ours. Our setting differs not only on the AutoML search procedure, but also on the initialization because we include random configurations. Therefore, we cannot directly compare our larger number of warm-starting configurations (25) – and thus the exploration-exploitation trade-off – to the number of configurations used in the original work.

To evaluate warm-started AsyncEA we perform an analysis similar to the time-restricted setting, but only on the full OpenML-CC18 collection because we do not warm-start with Dataset2Vec. We perform a leave-one-dataset-out validation on OpenML-CC18: computing the method's negative logistic loss on the left-out dataset using the metadata of all remaining datasets. However, now we evaluate the left-out dataset with a 3-fold cross-validation procedure as compared to hold-out. We include the meta-learning approach in the cross-validation setup – performing the meta-learner on all but the evaluation fold – but this is not required with the current dataset-agnostic meta-learner. Hence, we perform the 1-hour warm-started pipeline search multiple times per dataset – using different parts thereof – and average the results accordingly. We acknowledge that this evaluation setup is still not ideal; we only evaluate the method once on each part of the data, even though the search method is (partially) random in its initialization and search.

We compare the performance of warm-started AsyncEA with the non-warm-started 1-hour optimization procedures – i.e. Random Search, the Asynchronous Evolutionary Algorithm and the Asynchronous Successive Halving Algorithm. We rank the methods per dataset based on their cross-validation-based negative logistic loss scores. To test whether any of the optimization methods performs differently we employ the non-parametric Friedman test; we reject the null hypothesis ($p < 0.0001$), allowing us to perform pair-wise significance tests. Akin the time-restricted setting, we display the search method comparison in a critical differences diagram (see Figure 4.6), employing a Wilcoxon signed-rank test considering a significance level $\alpha = 5\%$ and using a Bonferroni correction.

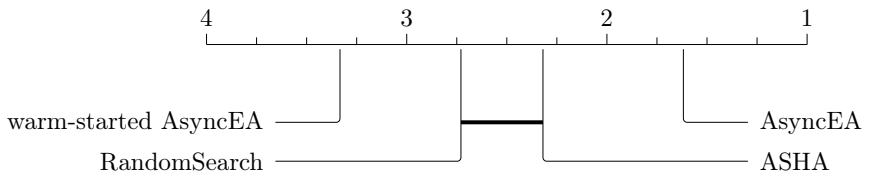

Figure 4.6: Critical differences diagram comparing warm-started and non-warm-started pipeline optimization approaches. We may perform these tests because the corresponding Friedman test is significant ($p < 0.0001$). All procedures have been ran with an equal optimization budget.

We can state that, with a statistically significant difference, the Asynchronous Evolutionary Algorithm (AsyncEA) outperforms both the Random Search and Asynchronous Successive Halving Algorithm (ASHA) on OpenML-CC18. The effect size – the difference between the average ranks, not the loss – is considerable, with the largest difference between AsyncEA and the Random Search algorithm. We stress that despite a larger effect size between AsyncEA and Random Search as compared to AsyncEA and ASHA, we cannot reject the equivalence of Random Search and ASHA. Recall we were unable to reject the null hypothesis – equivalence of these optimization procedures – in the time-restricted analysis. We postulate we can reject the null hypothesis now – and not previously – because we consider fewer approaches, therefore performing fewer statistical tests and thus decreasing the effect of the Bonferroni correction.

Next, to assess the effect of warm-starting, we compare the warm-started AsyncEA results with the non-warm-started AsyncEA results. Surprisingly, it appears that, with a statistically significant difference, warm-starting AsyncEA with portfolio building is detrimental to AsyncEA's performance. Even more so, we observe warm-started AsyncEA – whose non-warm-started variant outperforms other search methods – performs worse compared to both Random Search and ASHA.

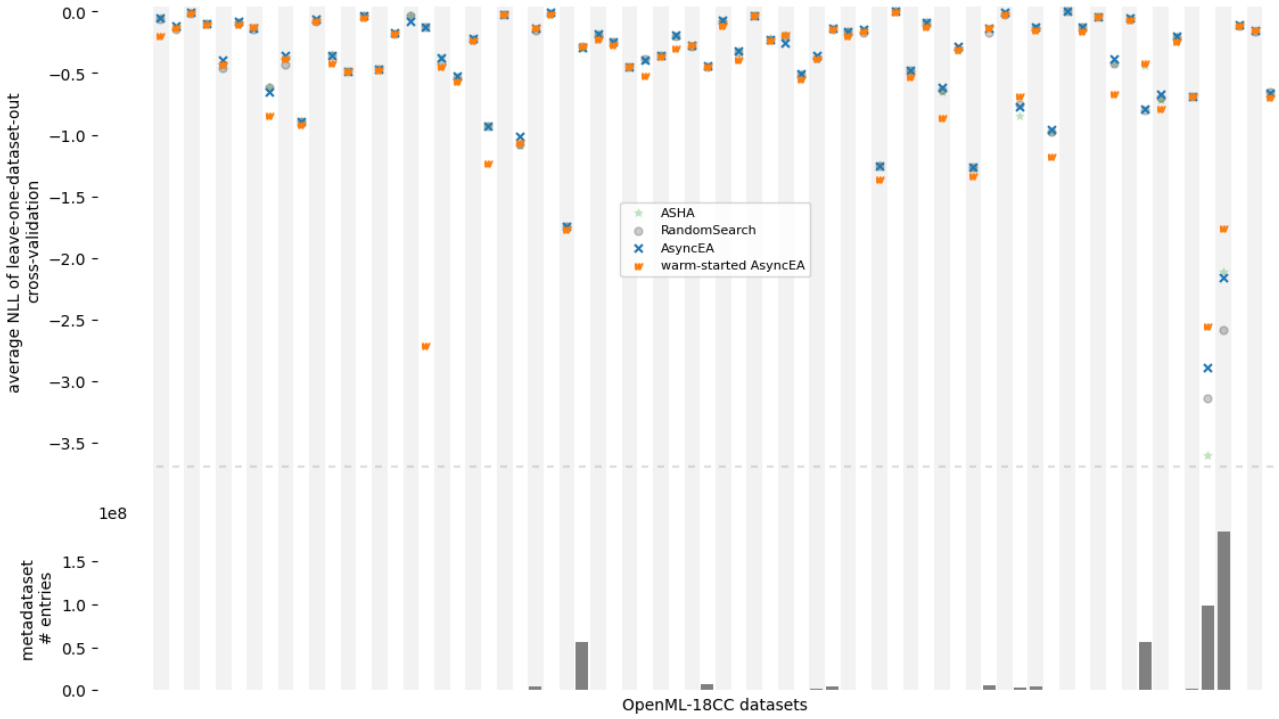

Figure 4.7: Optimization procedures' performance on OpenML-CC18 datasets, and its relation to dataset size. The dataset size is represented by the metadataset # entries (bar chart's y-axis), e.g. the product of the number of samples and the sum of the number of features and outcomes. The performance of optimization procedures' is comprised of the average negative logistic loss attained by the aforementioned cross-validation procedures. The figure excludes two measurements with far smaller negative logistic loss values to prevent distortion of the y-axis. Therefore, some datasets miss only certain optimization procedures' measurements, which indicates those procedures' inferiority.

Figures 4.7 and 4.8 depict the performance of optimization procedures on each of the OpenML-CC18 datasets. These supplementary analyses allow us to compare the effect of warm-starting in a more fine-grained manner by comparing AsyncEA with warm-started AsyncEA on datasets with varying characteristics. In these analyses we include the performance of ASHA and Random Search to contextualize the AsyncEA-based procedures. Again, we stress these analyses should be interpreted with care; the analyses provide additional, suggestive, information concerning the performance and do not lend for performance claims supported by rigorous statistical tests. In Figure 4.7 most optimization procedures achieve difficult-to-distinguish performance on most of the datasets, therefore Figure 4.8 considers only a subset of the measurements.

In addition to performance, Figure 4.7 visualizes the dataset size. Therefrom, it seems there is a relation between dataset size and warm-starting effect: warm-started AsyncEA outperforms AsyncEA on the largest datasets, and vice versa for other datasets. Moreover, on large datasets, warm-started AsyncEA seems to outperform ASHA, which is specifically designed for those larger datasets.

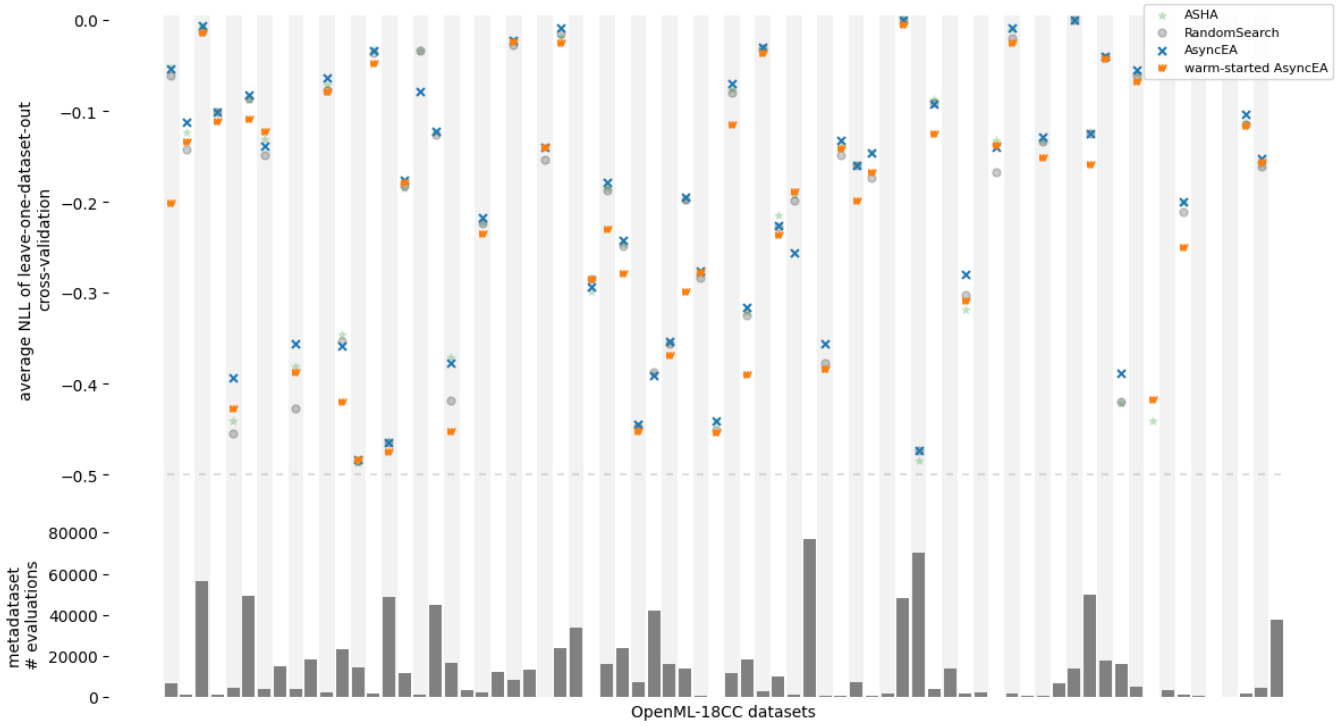

Figure 4.8: Optimization procedures' performance on OpenML-CC18 datasets, and its relation to the number of evaluations. The bar chart displays the ease of performing evaluations on a dataset, using the total number of evaluations in the metadataset as a proxy, e.g. the sum of the number of evaluations performed by ASHA, Random Search and AsyncEA. To precisely depict the relation between performance and the number of evaluations, we exclude optimization procedures achieving less than -0.5 negative logistic loss. Therefore, some datasets miss only certain optimization procedures' measurements, which indicates those procedures' inferiority.

Figure 4.8 additionally visualizes the ease to perform evaluations on the dataset, which is to a large extent inversely related to dataset size. Hence, it seems to be the case that warm-started AsyncEA is most competitive on datasets for which evaluations are expensive – confirming the relation visible in Figure 4.7. Figure 4.8 more clearly shows the performance differences of the optimization procedures, supporting the finding that, *in general*, warm-starting AsyncEA with portfolio building is detrimental to AsyncEA's performance. However, these differences tend to be small, noting warm-started AsyncEA is competitive with ASHA and Random Search on datasets whose evaluation is more computationally expensive.

Figures 4.7 and 4.8 suggest when warm-starting AsyncEA with portfolio building is detrimental or desirable. Yet, those analyses do not investigate why our instantiation of portfolio building tends to worsen AsyncEA's final performance. To that end we provide Figure 4.9, depicting the convergence behavior of AsyncEA with and without portfolio building. For most datasets the story is similar: the portfolio initially provides high-performing configurations, but the increase in performance vanishes with the increase in evaluations due to a slower convergence of warm-started AsyncEA.

Moreover, the image-based datasets confirm the following statement: if we can only evaluate a rather limited number of configurations, then warm-starting with portfolio building increases AsyncEA's final performance. Therefore, the image-based datasets favor warm-started AsyncEA, especially since most evaluations yield time outs with GAMA's default search parameters. However, not all findings are consistent with the aforementioned statement. For some datasets, with plenty of evaluations, warm-started AsyncEA is competitive with AsyncEA, outperforming it from time to time. We stress this behavior only occurs for a handful of datasets in OpenML-CC18 datasets. Typically, warm-started AsyncEA improves the performance attained by the portfolio slightly, or not at all.

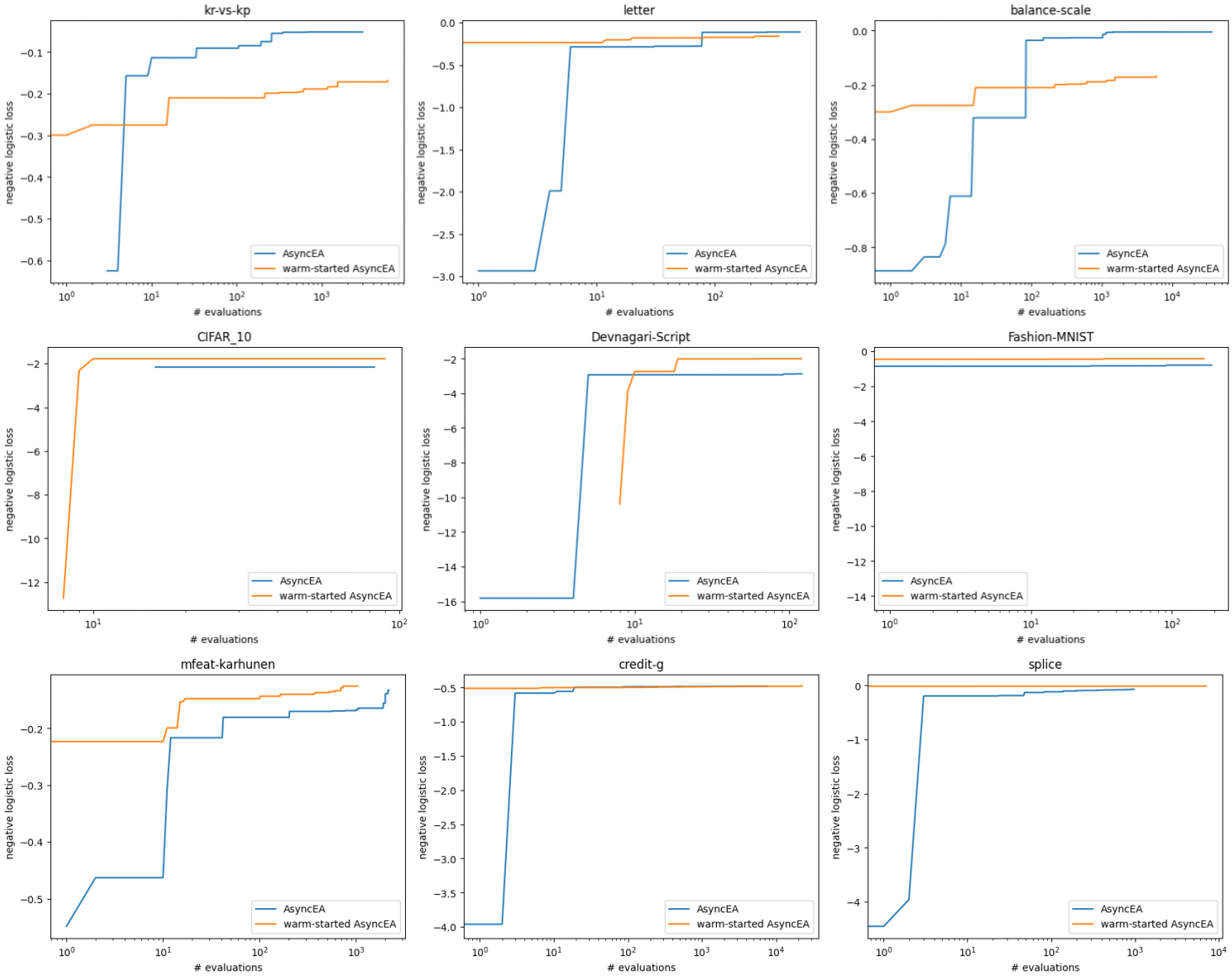

Figure 4.9: Convergence of warm-started and non-warm-started AsyncEA on 9 out of the 72 OpenML-CC18 datasets. The three topmost plots display convergence behavior typical for most OpenML-CC18 datasets. The three middlemost plots depict the convergence behavior on larger datasets with less evaluations. The bottommost plots show three datasets with atypical convergence behavior. All plots present the negative logistic loss of the – until then – top-performing configuration against the log of the number of performed evaluations. The first 25 warm-started AsyncEA evaluations correspond to the portfolio, because the x-axis includes all evaluations. Some line plots only depict the performance starting from a certain number of evaluations, because configurations prior to them could not be evaluated. Causes therefor include configuration incompatibility and resource constraints.

Chapter 5

Conclusion and Discussion

In this chapter we reflect on our contributions by means of a conclusion and discussion. To that end, we start by interpreting our work and relating it to the research objective in Section 5.1. In Section 5.2 we discuss the limitations and implications of our work. Subsequently, in Section 5.3, we suggest avenues for future work.

5.1 Conclusion

The key challenge associated to this work is the limited adoption of meta-learning in AutoML, which is caused by issues concerning reproducibility, accessibility and comprehensibility. Therefore, we aimed to achieve the following research objective: create a toolbox to easily construct, analyze and compare existing and novel meta-learning approaches and components. To that end, we analyzed relevant literature, proposed a modular and extensible framework, selected and implemented promising meta-learning approaches and components, combined them in a meta-learning toolbox adhering to the aforementioned framework, and showed its utility in a warm-starting case study. Therefrom we are able to derive the following key conclusions with respect to our research objective:

- Our publicly available metadataset and toolbox increase the reproducibility and accessibility of AutoML-directed meta-learning approaches and components.
- Our toolbox provides reference solutions to compare novel ideas to, supporting the development and evaluation of novel meta-learning approaches, and thereby the comprehension of their effectiveness.
- Our toolbox facilitates the construction and comparison of meta-learners and meta-learning components in only a few lines of code.
- Our toolbox provides novel meta-learning approaches, in the form of novel baselines and variations on existing state-of-the-art approaches.

We explored the application of the toolbox in varying meta-learning settings in Chapter 4 to show that we can analyze and compare meta-learning approaches and components. The corresponding time-restricted analysis yields the following key conclusions:

- A dataset-agnostic meta-learning approach – portfolio building – is especially powerful, outperforming (most) other meta-learning approaches.

- A characterization-based meta-learner should not use uncomplicated dataset- and configuration characterization methods.

In addition to that, the warm-starting case study provides support for the following claims:

- Warm-starting does not necessarily improve the final performance of AutoML search procedures, contradicting prior studies. Warm-starting success depends on the dataset, especially its size and ease to evaluate.
- Our toolbox allows to construct meta-learning approaches such that they can be incorporated in AutoML procedures.

Lastly, our dataset characterization method runtime analysis suggests:

- The modularized MLTA toolbox allows to not only analyze meta-learning approaches but also the components it consists of.
- Dataset2Vec is computationally tractable regardless of dataset size, while the hand-crafted Wistuba and Feurer meta-features are not, with the latter struggling most with characterizing datasets consisting of either a large number of instances, features or outcomes.

We conclude that the case study, runtime experiments and code examples have shown that we can easily construct, compare and analyze novel and existing meta-learning approaches and components, thereby achieving our research objective. Furthermore, our modularized and extensible approach and publicly available metadataset endorse reproducibility and accessibility of meta-learning research.

5.2 Discussion

5.2.1 Analyses

Dataset2Vec Evaluation The case study was unable to appropriately assess the performance of Dataset2Vec-based meta-learning approaches, because the OpenML-CC18 dataset collection contains datasets Dataset2Vec was pre-trained on. Therefore, we are unable to confirm whether portfolio building is indeed the highest potential meta-learning approach. We note that a similar issue has been experienced before [43]. Our experience confirms it is difficult to assess pre-trained meta-learning approaches veraciously.

Case Study Evaluation We must interpret the case study results with care since the evaluation approaches are not ideal. Computational resource limitations restricted the use of cross-validation in the time-restricted setting and prevented us to perform multiple runs in the warm-starting experiments. Besides decreased fidelity of meta-learning evaluations, computational resource restrictions prohibited a meta-learning comparison for warm-starting. Furthermore, these restrictions contribute to a limited power of the statistical tests, implying there could be performance differences between meta-learning components – in the time-restricted analysis – but that we were unable to detect them.

For instance, we cannot replicate a prior finding that the Wistuba meta-features outperform the Feurer meta-features [55], nor could we confirm that Dataset2Vec outperforms hand-crafted

meta-features. Yet, we expected to find differences between these methods. Firstly, because the prior work suggests it, and secondly because the dataset characterization run time analysis clearly indicates the Feurer meta-features cannot be computed on larger datasets in time-restricted settings. This finding provides support for having limited statistical power in the time-restricted analysis.

Similarly, we were unable to distinguish between characterization-based meta-learning approaches using the baseline algorithms propositionalization configuration characterization method – not including hyperparameters – and the RankML configuration characterization method – including hyperparameters. The lack of finding a statistically significant difference between these approaches does not confirm our intuition that a configuration characterization method should include hyperparameters. Again, we hypothesize this finding may be explained by a limited statistical power.

Time-Restricted Portfolio Building Despite limited statistical power, we were able to confirm Auto-Sklearn 2.0's portfolio building [33] outperforms its meta-feature based predecessor in the time-restricted setting. This finding supports dataset-agnostic meta-learners could be more powerful than dataset-dependent meta-learners, confirming AutoSklearn 2.0's claim [33] and refuting that of others [92]. Alternatively, it may also suggest that, in transferring knowledge, it may be wise to employ configurations from many distinct metadata datasets.

Warm-Starting AsyncEA with Portfolio Building We cannot support prior findings that warm-starting a 1-hour optimization procedure – in our case AsyncEA – with portfolio building improves its performance [33], and we are not certain why. We highlight our settings are different: firstly, in terms of the search space; secondly, we warm-start an evolutionary-based procedure as compared to a Bayesian-optimization-based procedure.

The warm-starting results suggest we should not assume warm-starting approaches perform similarly for varying optimization procedures. Moreover, our findings suggest (AutoML-directed) meta-learning may be most appropriate for settings with difficult-to-evaluate datasets and time restrictions; we were unable to improve AutoML's performance, but we can decrease the amount of time to find appropriate pipelines. Our conclusions support the findings of MetaTPOT [67], which explored the use of RankML [66] in warm-starting the *synchronous* – in contrast to our asynchronous – evolutionary search procedure in TPOT [96].

There are multiple possibilities to explain the performance degradation by warm-starting AsyncEA with portfolio building:

- In contrast to Bayesian-optimization-based optimization our evolutionary optimization procedure – AsyncEA – is designed to evolve, it could be the recommended pipelines belonging to the portfolio were too complex to improve.
- It could be our initialization method – replacing half of the population by a portfolio – did not yield a beneficial exploration-exploitation trade-off, which is influenced by both the proportion of non-random individuals and the portfolio size. After all, the evolutionary algorithm is sensitive to its initialization, especially in high-dimensional search spaces [57] like ours.
- The work introducing portfolio building [32] employed a metadataset consisting of 208 datasets, while ours "only" contains 71 datasets (OpenML-CC18 has 72 datasets, but we performed a leave-one-dataset-out evaluation). Our warm-starting procedure

thus employed approximately one third of the number of datasets, yet we constructed a portfolio 1.5 times as large. Hence, in our setting the proportion of the portfolio size to the number of datasets in the metadataset is almost 5 times as large, which may have caused the inclusion of inappropriate configurations.

- The results matrix – used to greedily compose the portfolio – contains approximately 8 times ($208^2/72^2$) fewer configuration-dataset evaluations. The smaller metadataset may imply that our portfolio is both less diverse – we considered less datasets – and generalizes more poorly – we consider fewer evaluations when composing the portfolio. The warm-starting convergence analysis – showing almost no convergence for warm-started AsyncEA – supports the limited diversity claim most.

Note that the latter two considerations suggests warm-starting an optimization procedure with a meta-learning approach – especially portfolio building – may be intimately related to the size and contents of the metadataset.

Metadataset Creation In creating the metadataset we granted an equal amount of computational resources to each dataset, which resulted in a heavily skewed distribution of the number of evaluations per dataset (recall Appendix B), with less evaluations for larger datasets. This may imply an underexplored search space for the largest datasets – especially CIFAR 10 with only 21 evaluations – limiting the external validity of our analysis and the re-usability of our metadataset.

Likewise, the re-usability of our metadataset is limited by an issue previously experienced by the authors of Oracle AutoML [135]. Namely, the abstraction of knowledge transfer and distillation: the dataset-configuration performance may be dependent on the (version of the) package implementing the configuration and the (version of the) programming language.

Moreover, another issue plagues the re-usability of metadatasets akin to ours: metadatasets based on a benchmarking suite – such as OpenML-CC18 – may not be re-usable if the collection of datasets changes.

5.2.2 Toolbox

MLTA Limitations We stress that unlike comparable meta-learning toolboxes – learn2learn [9], NASLib [111] TorchMeta [24] and BOML [76] – our toolbox is not production ready, because it lacks extensive code tests and documentation nor has it been used to repeat prior results. Note that the latter is impossible in our setting, because it requires access to the original works' metadatasets, which are not available. Moreover, writing test cases – and attaining high code coverage therewith – is possible yet difficult because many of the procedures require time-consuming execution, which deems it out-of-scope for a thesis. We would like to note that, despite the lack of extensive code testing, the case study implicitly tests all meta-learning approaches and components, the metadatabase and the evaluation functionality. We partially mitigate the lack of documentation, because we described the overall functioning of the toolbox in Chapter 3, included a handful of code examples, and wrote docstrings for all core functionality. Therefore, despite not being production ready, the toolbox is still sufficient for its intended use case: supporting research.

5.3 Future Work

Future work could develop benchmarking suites for meta-learning approaches, to solve issues with: re-usability of metadatasets if the dataset collection alters; abstraction of the transfer and distillation of knowledge; and the difficulty in veraciously assessing pre-trained meta-learners.

Future warm-starting research could: study the impact of the metadataset, e.g. in terms of the number of evaluations and datasets; compare meta-learning approaches and AutoML optimization procedures; study the exploration-exploitation trade-off for meta-learning approaches, especially portfolio building, and the effect of AutoML optimization procedures therein. Besides meta-learner specific trade-off parameters, one can study general trade-off parameters, e.g. the number of configurations to recommend and the amount of time for the meta-learner. However, beware the effect may vary per meta-learner, optimization procedure, and the latter's parameters and computational budget. We highlight these follow-up investigations can be executed using our toolbox and, for large parts, the metadataset.

Subsequent work could develop configuration-similarity measures, possibly by extending configuration characterization methods. We envision its usage in measuring and enforcing diversity in warm-starting AutoML systems, which could be crucial for evolutionary-based AutoML frameworks like TPOT [96] or GAMA [44], both during search and in creating the initial population. Moreover, a configuration similarity measure allows for deeper analyses on the exploration-exploitation behavior of meta-learning approaches in warm-starting AutoML.

Future work concerning the toolbox could focus on extending it with: additional and scenario-restricted meta-learning approaches and components; code tests with a high code coverage (at least 90%); a broader scope including regression and unsupervised tasks; and additional run time and robustness analyses of its meta-learning components.

Future work concerning meta-learning components could focus on creating and comparing configuration characterization methods. An interesting avenue would be the creation of a configuration characterization method producing learned representations – akin to Dataset2Vec [55] – but in a computationally tractable fashion – unlike the factorization-based approaches PMF[41] and TensorOBOE [136] – for instance by employing pre-trained models.

Bibliography

[1] A. Achille, M. Lam, R. Tewari, A. Ravichandran, S. Maji, C. C. Fowlkes, S. Soatto, and P. Perona. Task2vec: Task embedding for meta-learning. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 6430–6439, 2019. 11, 13, 16, 18, 28, 30

[2] A. Agostinelli, M. Pándy, J. Uijlings, T. Mensink, and V. Ferrari. How stable are transferability metrics evaluations? In *European Conference on Computer Vision*, pages 303–321. Springer, 2022. 13

[3] A. Agostinelli, J. Uijlings, T. Mensink, and V. Ferrari. Transferability metrics for selecting source model ensembles. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7936–7946, 2022. 13

[4] S. Alam and N. Yao. The impact of preprocessing steps on the accuracy of machine learning algorithms in sentiment analysis. *Computational and Mathematical Organization Theory*, 25(3):319–335, 2019. 8

[5] E. Alcobaça, F. Siqueira, A. Rivolli, L. P. Garcia, J. T. Oliva, and A. C. De Carvalho. Mfe: Towards reproducible meta-feature extraction. *The Journal of Machine Learning Research*, 21(1):4503–4507, 2020. 5, 16

[6] D. G. Altman and J. M. Bland. Statistics notes: Absence of evidence is not evidence of absence. *Bmj*, 311(7003):485, 1995. 52, 53

[7] D. Alvarez-Melis and N. Fusi. Geometric dataset distances via optimal transport. *Advances in Neural Information Processing Systems*, 33:21428–21439, 2020. 13, 14, 15, 18, 30, 32, 78

[8] V. Aribandi, Y. Tay, T. Schuster, J. Rao, H. S. Zheng, S. V. Mehta, H. Zhuang, V. Q. Tran, D. Bahri, J. Ni, et al. Ext5: Towards extreme multi-task scaling for transfer learning. *arXiv preprint arXiv:2111.10952*, 2021. 77

[9] S. M. Arnold, P. Mahajan, D. Datta, I. Bunner, and K. S. Zarkias. learn2learn: A library for meta-learning research. *arXiv preprint arXiv:2008.12284*, 2020. 5, 65

[10] Y. Bao, Y. Li, S.-L. Huang, L. Zhang, L. Zheng, A. Zamir, and L. Guibas. An information-theoretic approach to transferability in task transfer learning. In *2019 IEEE International Conference on Image Processing (ICIP)*, pages 2309–2313. IEEE, 2019. 13

[11] S. J. Bell and N. D. Lawrence. The effect of task ordering in continual learning. *arXiv preprint arXiv:2205.13323*, 2022. 78

[12] A. Benavoli, G. Corani, and F. Mangili. Should we really use post-hoc tests based on mean-ranks? *The Journal of Machine Learning Research*, 17(1):152–161, 2016. 52

[13] L. Berti-Equille. Learn2clean: Optimizing the sequence of tasks for web data preparation. In *The World Wide Web Conference*, pages 2580–2586, 2019. 24

[14] B. Bilalli, A. Abelló, T. Aluja-Banet, and R. Wrembel. Presistant: Learning based assistant for data pre-processing. *Data & Knowledge Engineering*, 123:101727, 2019. 24

[15] B. Bischl, G. Casalicchio, M. Feurer, P. Gijsbers, F. Hutter, M. Lang, R. G. Mantovani, J. N. van Rijn, and J. Vanschoren. Openml benchmarking suites. *arXiv preprint arXiv:1708.03731*, 2017. 44, 45, 46, 48, 55

[16] V. Borisov, T. Leemann, K. Seßler, J. Haug, M. Pawelczyk, and G. Kasneci. Deep neural networks and tabular data: A survey. *IEEE Transactions on Neural Networks and Learning Systems*, 2022. 8

[17] R. Caruana. Multitask learning. *Machine learning*, 28(1):41–75, 1997. 10

[18] S. Chen and H. He. Sera: Selectively recursive approach towards nonstationary imbalanced stream data mining. In *2009 International Joint Conference on Neural Networks*, pages 522–529. IEEE, 2009. 78

[19] T. Chen and C. Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794, 2016. 40

[20] T. Chen, T. He, M. Benesty, and V. Khotilovich. Package 'xgboost'. *R version*, 90:1–66, 2019. 40

[21] F. Chollet et al. Keras. `https://keras.io`, 2015. 8

[22] G. De Bie, H. Rakotoarison, G. Peyré, and M. Sebag. Distribution-based invariant deep networks for learning meta-features. *arXiv preprint arXiv:2006.13708*, 2020. 17, 29, 30, 78

[23] M. De Lange, R. Aljundi, M. Masana, S. Parisot, X. Jia, A. Leonardis, G. Slabaugh, and T. Tuytelaars. A continual learning survey: Defying forgetting in classification tasks. *IEEE transactions on pattern analysis and machine intelligence*, 44(7):3366–3385, 2021. 11

[24] T. Deleu, T. Würfl, M. Samiei, J. P. Cohen, and Y. Bengio. Torchmeta: A meta-learning library for pytorch. *arXiv preprint arXiv:1909.06576*, 2019. 5, 65

[25] J. Demšar. Statistical comparisons of classifiers over multiple data sets. *The Journal of Machine learning research*, 7:1–30, 2006. 52

[26] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009. 78

[27] L. Deng. The mnist database of handwritten digit images for machine learning research [best of the web]. *IEEE signal processing magazine*, 29(6):141–142, 2012. 44, 46

[28] I. Drori, Y. Krishnamurthy, R. Rampin, R. d. P. Lourenco, J. P. Ono, K. Cho, C. Silva, and J. Freire. Alphad3m: Machine learning pipeline synthesis. *arXiv preprint arXiv:2111.02508*, 2021. 4, 24

[29] I. Drori, L. Liu, Y. Nian, S. C. Koorathota, J. S. Li, A. K. Moretti, J. Freire, and M. Udell. Automl using metadata language embeddings. *arXiv preprint arXiv:1910.03698*, 2019. 14

[30] A. El Khatib, M. Nasr, and F. Karray. Accounting for the effect of inter-task similarity in continual learning models. In *2021 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pages 1241–1247. IEEE, 2021. 77

[31] N. Erickson, J. Mueller, A. Shirkov, H. Zhang, P. Larroy, M. Li, and A. Smola. Autogluon-tabular: Robust and accurate automl for structured data. *arXiv preprint arXiv:2003.06505*, 2020. 8

[32] M. Feurer, K. Eggensperger, S. Falkner, M. Lindauer, and F. Hutter. Practical automated machine learning for the automl challenge 2018. In *International Workshop on Automatic Machine Learning at ICML*, pages 1189–1232, 2018. 23, 37, 39, 57, 64

[33] M. Feurer, K. Eggensperger, S. Falkner, M. Lindauer, and F. Hutter. Auto-sklearn 2.0: The next generation. *arXiv preprint arXiv:2007.04074*, 24, 2020. 8, 64

[34] M. Feurer, K. Eggensperger, S. Falkner, M. Lindauer, and F. Hutter. Auto-sklearn 2.0: Hands-free automl via meta-learning. *The Journal of Machine Learning Research*, 23(1):11936–11996, 2022. 4, 23

[35] M. Feurer and F. Hutter. Hyperparameter optimization. In *Automated machine learning*, pages 3–33. Springer, Cham, 2019. 8

[36] 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. 1, 4, 8, 22, 79

[37] 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. 21

[38] M. Feurer, B. Letham, and E. Bakshy. Scalable meta-learning for bayesian optimization. *stat*, 1050(6), 2018. 21

[39] M. Feurer, J. Springenberg, and F. Hutter. Initializing bayesian hyperparameter optimization via meta-learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 29, 2015. 16, 20, 21, 27, 30, 35, 37, 42, 44, 83

[40] M. Friedman. A comparison of alternative tests of significance for the problem of m rankings. *The Annals of Mathematical Statistics*, 11(1):86–92, 1940. 52

[41] N. Fusi, R. Sheth, and M. Elibol. Probabilistic matrix factorization for automated machine learning. *Advances in neural information processing systems*, 31, 2018. 19, 24, 34, 66

[42] P. Gijsbers. Systems for automl research. 2022. 9, 49

[43] 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. 63

[44] P. Gijsbers and J. Vanschoren. Gama: genetic automated machine learning assistant. *Journal of Open Source Software*, 4(33):1132, 2019. 5, 8, 9, 47, 49, 66

[45] Y. Gil, K.-T. Yao, V. Ratnakar, D. Garijo, G. Ver Steeg, P. Szekely, R. Brekelmans, M. Kejriwal, F. Luo, and I.-H. Huang. P4ml: A phased performance-based pipeline planner for automated machine learning. In *AutoML Workshop at ICML*, 2018. 24

[46] J. Giovanelli, B. Bilalli, and A. Abelló. Data pre-processing pipeline generation for autoetl. *Information Systems*, 108:101957, 2022. 24

[47] M. Grobelnik and J. Vanschoren. Warm-starting darts using meta-learning. *arXiv preprint arXiv:2205.06355*, 2022. 77

[48] 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. 1, 8

[49] T. Hospedales, A. Antoniou, P. Micaelli, and A. Storkey. Meta-learning in neural networks: A survey. *IEEE transactions on pattern analysis and machine intelligence*, 44(9):5149–5169, 2021. 23

[50] T. Hospedales, A. Antoniou, P. Micaelli, and A. Storkey. Meta-learning in neural networks: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(9):5149–5169, 2022. 10

[51] L.-K. Huang, J. Huang, Y. Rong, Q. Yang, and Y. Wei. Frustratingly easy transferability estimation. In *International Conference on Machine Learning*, pages 9201–9225. PMLR, 2022. 13

[52] F. Hutter, L. Kotthoff, and J. Vanschoren. *Automated machine learning: methods, systems, challenges*. Springer Nature, 2019. 8

[53] A. S. Iwashita and J. P. Papa. An overview on concept drift learning. *IEEE access*, 7:1532–1547, 2018. 78

[54] H. Jin, Q. Song, and X. Hu. Auto-keras: An efficient neural architecture search system. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1946–1956. ACM, 2019. 1

[55] H. S. Jomaa, L. Schmidt-Thieme, and J. Grabocka. Dataset2vec: Learning dataset meta-features. *Data Mining and Knowledge Discovery*, 35(3):964–985, 2021. 16, 17, 28, 30, 39, 44, 63, 66, 85

[56] G. Katz, E. C. R. Shin, and D. Song. Explorekit: Automatic feature generation and selection. In *2016 IEEE 16th International Conference on Data Mining (ICDM)*, pages 979–984. IEEE, 2016. 24

[57] B. Kazimipour, X. Li, and A. K. Qin. A review of population initialization techniques for evolutionary algorithms. In *2014 IEEE congress on evolutionary computation (CEC)*, pages 2585–2592. IEEE, 2014. 64

[58] I. Khan, X. Zhang, M. Rehman, and R. Ali. A literature survey and empirical study of meta-learning for classifier selection. *IEEE Access*, 8:10262–10281, 2020. 24

[59] J. Kim, S. Kim, and S. Choi. Learning to warm-start bayesian hyperparameter optimization. *arXiv preprint arXiv:1710.06219*, 2017. 21

[60] S.-J. Kim, A. Magnani, and S. Boyd. Robust fisher discriminant analysis. *Advances in neural information processing systems*, 18, 2005. 27

[61] W. M. Kouw and M. Loog. An introduction to domain adaptation and transfer learning. *arXiv preprint arXiv:1812.11806*, 2018. 10

[62] A. Krause and D. Golovin. Submodular function maximization. *Tractability*, 3:71–104, 2014. 23

[63] S. Krishnan, J. Wang, E. Wu, M. J. Franklin, and K. Goldberg. Activeclean: Interactive data cleaning for statistical modeling. *Proceedings of the VLDB Endowment*, 9(12):948–959, 2016. 24

[64] J. J. Kurian, M. Dix, I. Amihai, G. Ceusters, and A. Prabhune. Boat: A bayesian optimization automl time-series framework for industrial applications. In *2021 IEEE Seventh International Conference on Big Data Computing Service and Applications (BigDataService)*, pages 17–24. IEEE, 2021. 20, 35, 37

[65] N. Kushmerick. Internet Advertisements. UCI Machine Learning Repository, 1998. DOI: https://doi.org/10.24432/C5V011. 44

[66] D. Laadan, R. Vainshtein, Y. Curiel, G. Katz, and L. Rokach. Rankml: a meta learning-based approach for pre-ranking machine learning pipelines. *arXiv preprint arXiv:1911.00108*, 2019. 18, 19, 22, 34, 36, 37, 39, 40, 41, 51, 64

[67] D. Laadan, R. Vainshtein, Y. Curiel, G. Katz, and L. Rokach. Metatpot: enhancing a tree-based pipeline optimization tool using meta-learning. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, pages 2097–2100, 2020. 23, 64

[68] C. P. Le, M. Soltani, J. Dong, and V. Tarokh. Fisher task distance and its application in neural architecture search. *IEEE Access*, 10:47235–47249, 2022. 77

[69] E. LeDell and S. Poirier. H2o automl: Scalable automatic machine learning. In *Proceedings of the AutoML Workshop at ICML*, volume 2020, 2020. 8, 9

[70] R. Leite, P. Brazdil, and J. Vanschoren. Selecting classification algorithms with active testing. In *International workshop on machine learning and data mining in pattern recognition*, pages 117–131. Springer, 2012. 79

[71] L. Li, K. Jamieson, G. DeSalvo, A. Rostamizadeh, and A. Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. *The Journal of Machine Learning Research*, 18(1):6765–6816, 2017. 8

[72] Y. Li, D. E. Carlson, et al. Extracting relationships by multi-domain matching. *Advances in Neural Information Processing Systems*, 31, 2018. 77

[73] Y. Li, X. Jia, R. Sang, Y. Zhu, B. Green, L. Wang, and B. Gong. Ranking neural checkpoints. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2663–2673, 2021. 13

[74] H. Liu, K. Simonyan, and Y. Yang. Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055*, 2018. 77

[75] X. Liu, Y. Bai, Y. Lu, A. Soltoggio, and S. Kolouri. Wasserstein task embedding for measuring task similarities. *arXiv preprint arXiv:2208.11726*, 2022. 18, 29, 30, 32

[76] Y. Liu and R. Liu. Boml: A modularized bilevel optimization library in python for meta learning. In *2021 IEEE International Conference on Multimedia & Expo Workshops (ICMEW)*, pages 1–2. IEEE, 2021. 5, 65

[77] J. P. Livesey and J. I. Laszlo. Effect of task similarity on transfer performance. *Journal of Motor Behavior*, 11(1):11–21, 1979. PMID: 15186968. 2

[78] V. Maeda-Gutiérrez, C. E. Galvan-Tejada, L. A. Zanella-Calzada, J. M. Celaya-Padilla, J. I. Galván-Tejada, H. Gamboa-Rosales, H. Luna-Garcia, R. Magallanes-Quintanar, C. A. Guerrero Mendez, and C. A. Olvera-Olvera. Comparison of convolutional neural network architectures for classification of tomato plant diseases. *Applied Sciences*, 10(4):1245, 2020. 8

[79] F. Mahmood, R. Chen, S. Sudarsky, D. Yu, and N. J. Durr. Deep learning with cinematic rendering: fine-tuning deep neural networks using photorealistic medical images. *Physics in Medicine & Biology*, 63(18):185012, 2018. 79

[80] O. A. Malik and S. Becker. Low-rank tucker decomposition of large tensors using tensorsketch. *Advances in neural information processing systems*, 31, 2018. 19

[81] R. G. Mantovani, T. Horváth, R. Cerri, J. Vanschoren, and A. C. de Carvalho. Hyper-parameter tuning of a decision tree induction algorithm. In *2016 5th Brazilian Conference on Intelligent Systems (BRACIS)*, pages 37–42. IEEE, 2016. 8

[82] G. Melis, C. Dyer, and P. Blunsom. On the state of the art of evaluation in neural language models. *arXiv preprint arXiv:1707.05589*, 2017. 8

[83] M. Milutinovic, B. Schoenfeld, D. Martinez-Garcia, S. Ray, S. Shah, and D. Yan. On evaluation of automl systems. In *Proceedings of the ICML Workshop on Automatic Machine Learning*, volume 2020, 2020. 4

[84] S. Mishra, R. Panda, C. P. Phoo, C.-F. R. Chen, L. Karlinsky, K. Saenko, V. Saligrama, and R. S. Feris. Task2sim: Towards effective pre-training and transfer from synthetic data. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9194–9204, 2022. 78

[85] F. Mohr and M. Wever. Naive automated machine learning. *Machine Learning*, pages 1–40, 2022. 24

[86] F. Mohr, M. Wever, and E. Hüllermeier. Ml-plan: Automated machine learning via hierarchical planning. *Machine Learning*, 107:1495–1515, 2018. 24

[87] L. Moschella, V. Maiorca, M. Fumero, A. Norelli, F. Locatello, and E. Rodolà. Relative representations enable zero-shot latent space communication. *arXiv preprint arXiv:2209.15430*, 2022. 11

[88] K. Murugesan and J. Carbonell. Active learning from peers. *Advances in neural information processing systems*, 30, 2017. 77

[89] K. Murugesan, H. Liu, J. Carbonell, and Y. Yang. Adaptive smoothed online multi-task learning. *Advances in Neural Information Processing Systems*, 29, 2016. 77

[90] C. Nguyen, T. Hassner, M. Seeger, and C. Archambeau. Leep: A new measure to evaluate transferability of learned representations. In *International Conference on Machine Learning*, pages 7294–7305. PMLR, 2020. 13

[91] C. V. Nguyen, A. Achille, M. Lam, T. Hassner, V. Mahadevan, and S. Soatto. Toward understanding catastrophic forgetting in continual learning. *arXiv preprint arXiv:1908.01091*, 2019. 77

[92] T.-D. Nguyen, D. J. Kedziora, K. Musial, and B. Gabrys. Exploring opportunistic meta-knowledge to reduce search spaces for automated machine learning. In *2021 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2021. 21, 36, 37, 48, 64

[93] Y. Niv. Learning task-state representations. *Nature neuroscience*, 22(10):1544–1553, 2019. 2

[94] M. Nomura, S. Watanabe, Y. Akimoto, Y. Ozaki, and M. Onishi. Warm starting cma-es for hyperparameter optimization. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 9188–9196, 2021. 21

[95] R. S. Olson, W. L. Cava, Z. Mustahsan, A. Varik, and J. H. Moore. Data-driven advice for applying machine learning to bioinformatics problems. In *Pacific Symposium on Biocomputing 2018: Proceedings of the Pacific Symposium*, pages 192–203. World Scientific, 2018. 8

[96] R. S. Olson and J. H. Moore. Tpot: A tree-based pipeline optimization tool for automating machine learning. In *Workshop on automatic machine learning*, pages 66–74. PMLR, 2016. 8, 9, 22, 64, 66

[97] V. Padmakumar, L. Lausen, M. Ballesteros, S. Zha, H. He, and G. Karypis. Exploring the role of task transferability in large-scale multi-task learning. *arXiv preprint arXiv:2204.11117*, 2022. 77

[98] M. Pándy, A. Agostinelli, J. Uijlings, V. Ferrari, and T. Mensink. Transferability estimation using bhattacharyya class separability. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9172–9182, 2022. 13

[99] 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. 40

[100] A. Pentina and C. H. Lampert. Multi-task learning with labeled and unlabeled tasks. In *International Conference on Machine Learning*, pages 2807–2816. PMLR, 2017. 77

[101] A. Pentina, V. Sharmanska, and C. H. Lampert. Curriculum learning of multiple tasks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5492–5500, 2015. 78

[102] G. Press. Data scientists spend most of their time cleaning data, May 2016. Available at `https://whatsthebigdata.com/2016/05/01/data-scientists-spend-most-of-their-time-cleaning-data/`. 9

[103] A. Quemy. Two-stage optimization for machine learning workflow. *Information Systems*, 92:101483, 2020. 24

[104] J. Raiman, S. Zhang, and C. Dennison. Neural network surgery with sets. *arXiv preprint arXiv:1912.06719*, 2019. 13

[105] H. Rakotoarison, M. Schoenauer, and M. Sebag. Automated machine learning with monte-carlo tree search. *arXiv preprint arXiv:1906.00170*, 2019. 24

[106] G. Ranganathan et al. A study to find facts behind preprocessing on deep learning algorithms. *Journal of Innovative Image Processing (JIIP)*, 3(01):66–74, 2021. 8

[107] R. Rivest. The md5 message-digest algorithm. Technical report, 1992. 40

[108] A. Rivolli, L. P. Garcia, C. Soares, J. Vanschoren, and A. C. de Carvalho. Characterizing classification datasets: a study of meta-features for meta-learning. *arXiv preprint arXiv:1808.10406*, 2018. 15, 16

[109] Y. Ro and J. Y. Choi. Autolr: Layer-wise pruning and auto-tuning of learning rates in fine-tuning of deep networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 2486–2494, 2021. 78

[110] M. T. Rosenstein, Z. Marx, L. P. Kaelbling, and T. G. Dietterich. To transfer or not to transfer. In *NIPS 2005 workshop on transfer learning*, volume 898, 2005. 3

[111] M. Ruchte, A. Zela, J. N. Siems, J. Grabocka, and F. Hutter. Naslib: a modular and flexible neural architecture search library. 2020. 5, 65

[112] M. M. Salvador, M. Budka, and B. Gabrys. Automatic composition and optimization of multicomponent predictive systems with an extended auto-weka. *IEEE Transactions on Automation Science and Engineering*, 16(2):946–959, 2018. 2, 9, 21

[113] S. Sanders and C. Giraud-Carrier. Informing the use of hyperparameter optimization through metalearning. In *2017 IEEE International Conference on Data Mining (ICDM)*, pages 1051–1056. IEEE, 2017. 8

[114] J. P. Shaffer. Multiple hypothesis testing. *Annual review of psychology*, 46(1):561–584, 1995. 52

[115] C. Shui, M. Abbasi, L.-É. Robitaille, B. Wang, and C. Gagné. A principled approach for learning task similarity in multitask learning. *arXiv preprint arXiv:1903.09109*, 2019. 77

[116] S. Smith, M. Patwary, B. Norick, P. LeGresley, S. Rajbhandari, J. Casper, Z. Liu, S. Prabhumoye, G. Zerveas, V. Korthikanti, et al. Using deepspeed and megatron to train megatron-turing nlg 530b, a large-scale generative language model. *arXiv preprint arXiv:2201.11990*, 2022. 8

[117] P. Soviany, R. T. Ionescu, P. Rota, and N. Sebe. Curriculum learning: A survey. *International Journal of Computer Vision*, pages 1–40, 2022. 78

[118] Y. Tan, Y. Li, and S.-L. Huang. Otce: A transferability metric for cross-domain cross-task representations. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 15779–15788, 2021. 13, 79

[119] Y. Tan, Y. Li, S.-L. Huang, and X.-P. Zhang. Transferability-guided cross-domain cross-task transfer learning. *arXiv preprint arXiv:2207.05510*, 2022. 13

[120] A. Thakur, A. Pankajakshan, V. Abrol, and D. A. Clifton. Neural transferability: Current pitfalls and striving for optimal scores. *Available at SSRN 4196999*. 13

[121] 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. 8, 9

[122] E. C. Too, L. Yujian, S. Njuki, and L. Yingchun. A comparative study of fine-tuning deep learning models for plant disease identification. *Computers and Electronics in Agriculture*, 161:272–279, 2019. 78, 79

[123] R. Turner, D. Eriksson, M. McCourt, J. Kiili, E. Laaksonen, Z. Xu, and I. Guyon. Bayesian optimization is superior to random search for machine learning hyperparameter tuning: Analysis of the black-box optimization challenge 2020. In *NeurIPS 2020 Competition and Demonstration Track*, pages 3–26. PMLR, 2021. 21

[124] J. Vanschoren. Meta-learning: A survey. *arXiv preprint arXiv:1810.03548*, 2018. 2, 10, 20, 78

[125] 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. 49

[126] T. Veniat, L. Denoyer, and M. Ranzato. Efficient continual learning with modular networks and task-driven priors. *arXiv preprint arXiv:2012.12631*, 2020. 77

[127] C. Wang, Q. Wu, M. Weimer, and E. Zhu. Flaml: A fast and lightweight automl library. *Proceedings of Machine Learning and Systems*, 3:434–447, 2021. 8

[128] S. Wang, Y. Choi, J. Chen, M. El-Khamy, and R. Henao. Toward sustainable continual learning: Detection and knowledge repurposing of similar tasks. *arXiv preprint arXiv:2210.05751*, 2022. 77

[129] Z. Wang, Z. Dai, B. Póczos, and J. Carbonell. Characterizing and avoiding negative transfer. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 11293–11302, 2019. 79

[130] J. N. Washburne. The definition of learning. *Journal of Educational Psychology*, 27(8):603, 1936. 2

[131] M. Wistuba, N. Schilling, and L. Schmidt-Thieme. Hyperparameter search space pruning–a new component for sequential model-based hyperparameter optimization. In *Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2015, Porto, Portugal, September 7-11, 2015, Proceedings, Part II 15*, pages 104–119. Springer, 2015. 21

[132] M. Wistuba, N. Schilling, and L. Schmidt-Thieme. Two-stage transfer surrogate model for automatic hyperparameter optimization. In *Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2016, Riva del Garda, Italy, September 19-23, 2016, Proceedings, Part I 16*, pages 199–214. Springer, 2016. 16, 21, 27, 30, 41, 44, 83

[133] M. Wistuba, N. Schilling, and L. Schmidt-Thieme. Scalable gaussian process-based transfer surrogates for hyperparameter optimization. *Machine Learning*, 107(1):43–78, 2018. 79

[134] H. Xiao, K. Rasul, and R. Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017. 46

[135] A. Yakovlev, H. F. Moghadam, A. Moharrer, J. Cai, N. Chavoshi, V. Varadarajan, S. R. Agrawal, S. Idicula, T. Karnagel, S. Jinturkar, et al. Oracle automl: a fast and predictive automl pipeline. *Proceedings of the VLDB Endowment*, 13(12):3166–3180, 2020. 65

[136] C. Yang, Y. Akimoto, D. W. Kim, and M. Udell. Oboe: Collaborative filtering for automl model selection. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 1173–1183, 2019. 34, 66

[137] C. Yang, J. Fan, Z. Wu, and M. Udell. Automl pipeline selection: Efficiently navigating the combinatorial space. In *proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 1446–1456, 2020. 19, 24

[138] Z. Ye, A. Gilman, Q. Peng, K. Levick, P. Cosman, and L. Milstein. Comparison of neural network architectures for spectrum sensing. In *2019 IEEE Globecom Workshops (GC Wkshps)*, pages 1–6. IEEE, 2019. 8

[139] F. Yergeau. Utf-8, a transformation format of iso 10646. Technical report, 2003. 40

[140] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson. How transferable are features in deep neural networks? *Advances in neural information processing systems*, 27, 2014. 78, 79

[141] K. You, Y. Liu, J. Wang, and M. Long. Logme: Practical assessment of pre-trained models for transfer learning. In *International Conference on Machine Learning*, pages 12133–12143. PMLR, 2021. 13

[142] F. R. Zagatti, L. C. Silva, L. N. D. S. Silva, B. S. Sette, H. de Medeiros Caseli, D. Lucrédio, and D. F. Silva. Metaprep: Data preparation pipelines recommendation via meta-learning. In *2021 20th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 1197–1202. IEEE, 2021. 24

[143] W. Zhang, L. Deng, L. Zhang, and D. Wu. A survey on negative transfer. *IEEE/CAA Journal of Automatica Sinica*, 2022. 79

[144] Y. Zhang, M. T. Bahadori, H. Su, and J. Sun. Flash: fast bayesian optimization for data analytic pipelines. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 2065–2074, 2016. 24

[145] L. Zimmer, M. Lindauer, and F. Hutter. Auto-pytorch: Multi-fidelity metalearning for efficient and robust autodl. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 43(9):3079–3090, 2021. 4, 8

[146] M.-A. Zöller and M. F. Huber. Benchmark and survey of automated machine learning frameworks. *Journal of artificial intelligence research*, 70:409–472, 2021. 1, 7, 8, 9, 24

[147] M.-A. Zöller, T.-D. Nguyen, and M. F. Huber. Incremental search space construction for machine learning pipeline synthesis. In *Advances in Intelligent Data Analysis XIX: 19th International Symposium on Intelligent Data Analysis, IDA 2021, Porto, Portugal, April 26–28, 2021, Proceedings 19*, pages 103–115. Springer, 2021. 24

Appendix A

Similarity Measure Usage in Meta-Learning Related Research Fields

Continual learning applications There is a mixed view on whether similar tasks help or hurt performance in continual learning. Some research shows improved performance using similar tasks, yet other argues for the use of dissimilar tasks [91].

One approach incorporates similarity information by extending continual learning approaches with a (1%) memory budget based on inter-task similarity [30]. The approach proves to be successful in mitigating catastrophic forgetting in low inter-task similarity settings. Another work proposes to incorporate a similarity detection function to analyze whether a specific task in the past is similar to the current task [128]. Then, the previous task's knowledge is reused to slow down parameter expansion. Other research explored task-driven priors to choose modules for modular neural networks [126]. The method requires specification of the most similar task. Finding the most similar task is necessary because the search space grows exponentially. Once the most similar task is specified, the method evaluates the use of its modules for the current task to improve performance.

Multi-task learning A MTL study [97] shows competitive performance for pre-finetuning a smaller set of related tasks and large-scale multi-task learning. Moreover, the study finds that pre-finetuning on unrelated tasks diminishes performance. Thus, task relatedness, or similarity, is highly relevant for multi-task learning. Selecting an ideal subset of related tasks for multi-task pre-training is difficult [8], perhaps similarity measures help identify such groups of tasks. The incorporation of similarity in MTL has theoretical guarantees showing the benefit of explicit and implicit similarity knowledge [115]. Similarity measures may be suitable to provide the similarity for explicit-similarity MTL approaches.There are multiple works using similarity for weighting empirical loss [100, 89, 88]. Another study presents a multi-task learning approach relying on similarity using only features [72].

Neural Architecture Search (NAS) application Task-aware NAS [68] is a NAS framework employing an asymmetric similarity measure. The similarity measure is used to find the most similar tasks. Subsequently, the architecture search space is reduced by building upon the optimized architectures of, previously solved, similar tasks. Furthermore, there exists an approach to warm-start neural architecture search. More specifically, in [47] differentiable architecture search (DARTS) [74] is warm-started from similar tasks.

Curriculum learning applications Curriculum learning concerns ordering tasks, or data, in a meaningful order to improve performance, without increasing computational costs [117]. One study uses similarity information to effectively order the tasks by taking the shortest task through a task space [11]. Other work uses a similarity-based ordering to decompose a multi-task learning problem into a continual learning problem [101].

Synthetic data applications Another interesting line of research uses similarity information to generate synthetic data. One application is to choose an optimal data augmentation [7]. In data augmentation one applies transformations on the training data to achieve higher generalization performance. The research [7] shows a negative correlation between (augmented) dataset similarity and performance, it thus indicates that similarity measures may be used for ranking data augmentation procedures. Other research shows the use of representations learned by a similarity measure [84]. The approach uses similarity-measure-created representations to create synthetic data for pre-training [84]. The obtained performance is competitive with pre-training on Imagenet [26].

Drift detection and patch identification Concept drift approaches regularly utilize an active drift detection mechanism, informing when the data has changed substantially [53]. In this setup, the role of similarity measures is to help detect when drift occurs. To the best of our knowledge, there is little research establishing the use of similarity information in concept drift learning. We suspect this to be the case because concept drift learning is more concerned with similarities on a more fine-grained level, data, instead of datasets. We do believe that these methods can be adapted to incorporate similarity measures. There is one concept drift study relying on a similarity measure over chunks of data [18]. The study mentions the issue, at the time, of inadequate similarity estimation between chunks of data. Another study is connected to the previous application [22]. The study shows the applicability of similarity information for patch identification: determining whether two data patches are from the same initial dataset.

Transfer Learning The next few paragraphs are dedicated to a challenge in transfer learning, namely source selection for fine-tuning. Though transfer learning applies to a wide variety of models, it is best applied to neural networks because both their structure and parameters can be reused [124]. In transfer learning, we have source tasks and target tasks. Conventionally, source tasks have a remarkably higher cardinality than target tasks. Therefore, in transfer learning, an inductive bias is transferred from a source task to a target task. In the context of neural networks, the inductive bias usually consists of a model architecture and its trained parameters. Typically, there is more than one source task available, each with an associated pre-trained model. A common task in transfer learning is source selection for fine-tuning; a possible similarity measure downstream application.

Fine-tuning is a concept in transfer learning [122]. To understand fine-tuning it helps to think of neural networks consisting of two parts, a feature extractor and a top. The feature extractor extracts features from the raw input and feeds it to the top for classification or regression. There are different fine-tuning approaches, varying the amount the pre-trained model is changed [140]. We can copy the first n layers from the source to the target. The remaining layers of the target model are randomly initialized and trained on the target task. The amount of model change is controlled through n, yielding many fine-tuning options, potentially with different rates of change per layer [109]. Simplifying this choice, typically

one of two approaches is used. Either fine-tune the feature extractor, backpropagating the target task's errors, or freeze the feature extractor, leaving its values unchanged. This choice depends on the similarity between the source and target task. A popular approach is to freeze the feature extraction layers and fine-tune the classification or regression top. Fine-tuning is powerful because it allows for training complex neural networks, with relatively little data, in turn achieving otherwise unachievable high performance [79, 122].

In source selection for fine-tuning, the goal is to select the source task, or model, which after fine-tuning yields the best performance on the target task. An additional goal is to select the source task efficiently, on that account, it is prohibited to continue training the source models on the target task. Similarity measures are well-suited to the source selection task because more similar source and target tasks achieve higher transferability [140]. Even more so, in transfer learning, knowledge transfer yields degrading performance if the source and target are too dissimilar [129, 143]. An interesting extension of source selection is multi-source feature fusion, which has been proven to benefit from similarity information [118]. In multi-source feature fusion, multiple source models are transferred to the target task by merging their inferred features to obtain a fused representation.

Similarity as weight Active testing [70] is an approach to classification algorithm selection employing dataset similarity. Active testing is initialized with a high-performing default and proceeds tournament style. In rounds, it selects and evaluates the configurations most promising to beat the current best solution. In identifying the most promising competitor a similarity measure is used in weighting. The active testing framework for algorithm selection can likely benefit from other, more recent, similarity measures.

Apart from transferring configurations (see Section 2.2.2.1), we can also transfer surrogate models. A surrogate model models the evaluations of the configurations. A straightforward approach to transfer a surrogate model is to transfer the one from the most similar task. A general and powerful approach to transferring surrogate models is to transfer multiple of them [36], potentially weighted by a similarity value [133]. In [133] the authors propose a similarity measure based on landmark vectors, but the overall approach is flexible enough to function with any similarity measure.

Appendix B

Supplemental Material Metadataset

Table B.1: The number of configuration evaluations for each dataset in OpenML18CC.

OpenML Dataset Name	**OpenML ID**	**Number of Evaluations**
ilpd	1480	76945
blood-transfusion-service-center	1464	70066
balance-scale	11	56495
climate-model-simulation-crashes	1467	49823
breast-w	15	49360
diabetes	37	48927
banknote-authentication	1462	48181
tic-tac-toe	50	44684
kc2	1063	42227
dresses-sales	23381	37862
analcatdata_dmft	469	33786
analcatdata_authorship	458	24003
pc3	1050	23880
credit-approval	29	23492
cmc	23	18620
qsar-biodeg	1494	18509
wilt	40983	17997
vehicle	54	17030
car	991	16233
kc1	1067	16084
pc4	1049	15973
mfeat-morphological	18	15248

credit-g	31	14719
MiceProtein	40966	14189
cylinder-bands	6332	14177
pc1	1068	13991
vowel	307	13275
eucalyptus	188	12151
wdbc	1510	11939
spambase	44	11912
phoneme	1489	10011
sick	38	8252
jm1	1053	7515
ozone-level-8hr	1487	7409
kr-vs-kp	3	7112
steel-plates-fault	1504	6641
segment	36	5159
churn	40701	4456
mfeat-fourier	14	4381
mfeat-karhunen	16	4111
mfeat-zernike	22	3890
PhishingWebsites	4534	3874
jungle_chess_2pcs_raw_endgame_complete	41027	3593
electricity	151	3280
wall-robot-navigation	1497	3188
GesturePhaseSegmentationProcessed	4538	2487
satimage	182	2395
optdigits	28	2381
pendigits	32	2032
dna	40670	1872
bank-marketing	1461	1739
texture	40499	1725
first-order-theorem-proving	1475	1630
mfeat-pixel	20	1536
mfeat-factors	12	1475
splice	46	1465
semeion	1501	1308

letter	6	1107
Internet-Advertisements	40978	952
adult	179	925
numerai28.6	23517	905
madelon	1485	898
cnae-9	1468	892
nomao	1486	639
connect-4	1591	534
isolet	300	503
Bioresponse	4134	439
har	1478	411
mnist_784	554	233
Fashion-MNIST	40996	207
Devnagari-Script	40923	114
CIFAR_10	40927	21

Appendix C

Common Meta-Feature Collections

Table C.1: Table depicting two common and well-performing collections of meta-Features for supervised classification tasks, namely the meta-features and summarization functions by Feurer et al. [39] and those by Wistuba et al. [132]. Note that meta-features are grouped by their categorization and that the Wistuba meta-features are a subset of the Feurer meta-features.

Meta-Feature Group	**Feurer Meta-Features**	**Wistuba Meta-Features**
Simple	Number of instances	Number of instances
	Log number of instances	Log number of instances
	Number of classes	Number of classes
	Number of features	Number of features
	Log number of features	Log number of features
	Number of instances with missing values	
	Percentage of instances with missing values	
	Number of features with missing values	
	Percentage of features with missing values	
	Number of missing values	
	Percentage of missing values	
	Number of numeric features	
	Number of categorical features	
	Ratio numerical to categorical	
	Ratio categorical to numerical	
	Dataset dimensionality	Dataset dimensionality
	Log dataset dimensionality	Log dataset dimensionality
	Inverse dataset dimensionality	Inverse dataset dimensionality
	Log inverse dataset dimensionality	Log inverse dataset dimensionality
	Class probability min	Class probability min
	Class probability max	Class probability max
	Class probability mean	Class probability mean
	Class probability std	Class probability std

Information-theoretic	Class entropy	Class entropy
Statistical	Min number of categorical values	
	Max number of categorical values	
	Mean number of categorical values	
	Std number of categorical values	
	Total number of categorical values	
	Kurtosis min	Kurtosis min
	Kurtosis max	Kurtosis max
	Kurtosis mean	Kurtosis mean
	Kurtosis std	Kurtosis std
	Skewness min	Skewness min
	Skewness max	Skewness max
	Skewness mean	Skewness mean
	Skewness std	Skewness std
PCA	PCA 95%	
	PCA skewness first principal component	
	PCA kurtosis first principal component	
Landmarking	One Nearest Neighbor	
	Linear Discriminant Analysis	
	Naive Bayes	
	Decision Tree	
	Decision Node Learner	
	Random Node Learner	

Appendix D

OpenML-CC18 Metadataset Subset

Table D.1: The OpenML18CC subset. Datasets were selected by hand to not overlap with the datasets that were used for training Dataset2Vec [55].

OpenML Dataset Name	**OpenML ID**
climate-model-simulation-crashes	1467
banknote-authentication	1462
kc2	1063
dresses-sales	23381
analcatdata_dmft	469
analcatdata_authorship	458
pc3	1050
qsar-biodeg	1494
wilt	40983
kc1	1067
pc4	1049
mfeat-morphological	18
MiceProtein	40966
pc1	1068
eucalyptus	188
phoneme	1489
jm1	1053
kr-vs-kp	3
churn	40701
mfeat-fourier	14
mfeat-karhunen	16
mfeat-zernike	22
PhishingWebsites	4534
jungle_chess_2pcs_raw_endgame_complete	41027
electricity	151

GesturePhaseSegmentationProcessed	4538
first-order-theorem-proving	1475
mfeat-pixel	20
mfeat-factors	12
Internet-Advertisements	40978
numerai28.6	23517
madelon	1485
cnae-9	1468
nomao	1486
isolet	300
Bioresponse	4134
har	1478
mnist_784	554
Fashion-MNIST	40996
Devnagari-Script	40923
CIFAR_10	40927

Appendix E

Meta-Learning Approaches' Regret Formalization

Let $\mathcal{M}$ be the set of all meta-learning approaches and $\mathcal{D}_E$ the set of all evaluation datasets. Let a meta-learning approach $m_i \in \mathcal{M}$ be tasked with composing a warm-starting vector $\boldsymbol{w}_{i,j} = [(g, \mathbf{A}, \boldsymbol{\lambda})_k,\ (g, \mathbf{A}, \boldsymbol{\lambda})_{k+1},\ ....,\ (g, \mathbf{A}, \boldsymbol{\lambda})_{N_C}]$ consisting of N_C pipeline configurations for dataset $d_j \in \mathcal{D}_E$. Associated to the warm-starting vector $\boldsymbol{w}_{i,j}$ is the evaluation vector $\boldsymbol{e}_{i,j} \in \mathbb{R}^{N_C}$ consisting of the evaluation scores for executing the pipeline configurations in $\boldsymbol{w}_{i,j}$ on dataset d_j, thus $\boldsymbol{e}_{i,j,k}$ denotes the evaluation score for meta-learning approach m_i using its k^{th} recommended configuration on dataset d_j. For normalization we create reference evaluation scores $t_j, b_j \in \mathbb{R}$ for each dataset $d_j \in \mathcal{D}_E$ composed of the top and bottom pipeline configuration evaluation scores respectively, thus

$$t_j = \max_{i,k}(\mathbf{e}_{i,j,k}), \qquad b_j = \min_{i,k}(\mathbf{e}_{i,j,k}) \tag{E.1}$$

If meta-learning approach m_i fails to provide a configuration recommendation $(g, \mathbf{A}, \boldsymbol{\lambda})_k$ in $\boldsymbol{w}_{i,j}$ its entry $\boldsymbol{w}_{i,j,k}$ will be empty, and its corresponding evaluation entry $\boldsymbol{e}_{i,j,k}$ is set to b_j to yield $\dfrac{|t_j - \boldsymbol{e}_{i,j,k}|}{|t_j - b_j|} = \dfrac{|t_j - b_j|}{|t_j - b_j|} = 1$. Assuming the evaluations in $\mathbf{e}$ are from a scoring function (higher is better), we can assign each meta-learning approach m_i its average normalized regret:

$$r_i = \left(\frac{1}{|\mathcal{D}_E| \cdot N_C} \sum_{j=1}^{|\mathcal{D}_E|} \sum_{k=1}^{N_C} \frac{|t_j - \mathbf{e}_{i,j,k}|}{|t_j - b_j|} \right) \in [0, 1] \tag{E.2}$$

which can be interpreted as the average normalized Manhattan distance between the worst possible and the recommended configuration scores, averaged over all datasets. Or in other words it is the normalized regret, averaged over datasets and recommended configurations.