Some contributions to AutoML : hyper-parameter optimization and meta-learning

Herilalaina Rakotoarison

université PARIS-SACLAY

Some contributions to AutoML: Hyper-parameter Optimization and Meta-learning

Contributions à AutoML: optimisation des hyper-paramètres et méta-apprentissage

Thèse de doctorat de l'université Paris-Saclay

École doctorale n° 580, Sciences et Technologies de l'Information et de la Communication (STIC)
Spécialité de doctorat: Informatique
Graduate School : Informatique et sciences du numérique
Référent : Faculté des sciences d'Orsay

Thèse préparée dans l'unité de recherche **LISN (Université Paris-Saclay, CNRS)** et **Inria Saclay-Île-de-France (Université Paris-Saclay, Inria)**, sous la direction de **Marc SCHOENAUER**, Directeur de recherche, le co-encadrement de **Michèle SEBAG**, Directrice de recherche.

Thèse soutenue à Paris-Saclay, le 27 juin 2022, par

Herilalaina RAKOTOARISON

Composition du jury

Nicolas COURTY Professeur, Université Bretagne Sud	Président
Frank HUTTER Professeur, Albert-Ludwigs-Universität Freiburg	Rapporteur & examinateur
Frédéric SAUBION Professeur, LERIA, Université d'Angers	Rapporteur & examinateur
Rémi BARDENET Chargé de recherche, CNRS, Université de Lille	Examinateur
Joaquin VANSCHOREN Professeur assistant, Eindhoven University of Technology	Examinateur
Marc SCHOENAUER Directeur de recherche, INRIA Saclay	Directeur de thèse

THESE DE DOCTORAT
NNT : 2022UPASG044

université PARIS-SACLAY

ÉCOLE DOCTORALE
Sciences et technologies de l'information et de la communication (STIC)

Contributions à AutoML: optimisation des hyper-paramètres et méta-apprentissage
Mots clés: AutoML, séléction des modèles, optimisation d'hyper-paramètre, meta-apprentissage

Résumé: Cette thèse présente trois principales contributions afin d'améliorer l'état de l'art de ces approches AutoML. Elles sont divisées entre deux thèmes de recherche: l'optimisation et meta-apprentissage. La première contribution concerne un algorithme d'optimisation hybride, appelé Mosaic, qui exploite les méthodes MCTS et optimisation bayésienne pour résoudre respectivement la sélection des algorithmes et la configuration des hyperparamètres. L'évaluation, conduite à travers le benchmark OpenML 100, montre que la performance empirique de Mosaic surpasse ceux des systèmes d'AutoML de l'état de l'art (Auto-Sklearn et TPOT). La deuxième contribution introduit une architecture de réseau neuronal, appelée Dida, qui permet d'apprendre des descripteurs de données invariants à la permutation de colonnes et d'exemples. Deux tâches (classification des patchs et prédiction des performances) sont considérées lors de l'évaluation de la méthode. Les résultats de Dida sont encourageants comparés à ceux de ses concurrents (Dataset2 vvec et DSS). Enfin, la troisième contribution, intitulée Metabu, vise à surmonter les limites de Dida à opérer sur de vrais jeux de données d'AutoML. La stratégie de Metabu comporte deux étapes. Tout d'abord, une topologie idéale de ces jeux de données, basée sur les meilleurs hyperparamètres, est définie. Puis, une transformation linéaire d es descripteurs manuels est apprise pour les aligner, selon un critère de transport optimal, avec la représentation idéale. Les comparaisons empiriques montrent que les descripteurs Metabu sont plus performants que les descripteurs manuels sur trois problèmes différents (évaluation du voisinage des jeux de données, recommandation d'hyperparamètres, et initialisation d'un algorithme d'optimisation).

Title: Some contributions to AutoML: Hyper-parameter Optimization and Meta-learning
Keywords: AutoML, model selection, hyper-parameter tuning, meta-learning

Abstract: This thesis proposes three main contributions to advance the state-of-the-art of AutoML approaches. They are divided into two research directions: optimization (first contribution) and meta-learning (second and third contributions). The first contribution is a hybrid optimization algorithm, dubbed Mosaic, leveraging Monte-Carlo Tree Search and Bayesian Optimization to address the selection of algorithms and the tuning of hyper-parameters, respectively. The empirical assessment of the proposed approach shows its merits compar ed to Auto-sklearn and TPOT AutoML systems on OpenML 100. The second contribution introduces a novel neural network architecture, termed Dida, to learn a good representation of datasets (i.e., meta-features) from scratch while enforcing invariances w.r.t features and rows permutations. Two proof-of-concept tasks (patch classification and performance prediction tasks) are considered. The proposed approach yields superior empirical performance compared to Dataset2Vec and DSS on both tasks. The third contribution addresses the limitation of Dida on handling standard dataset benchmarks. The proposed approach, called Metabu, relies on hand-crafted meta-features. The novelty of Metabu is two-fold: i) defining an "oracle" topology of datasets based on top-performing hyper-parameters; ii) leveraging Optimal Transport approach to align a mapping of the hand-crafted meta-features with the oracle topology. The empirical results suggest that Metabu meta-feature outperforms the baseline hand-cr afted meta-features on three different tasks (assessing meta-features based topology, recommending hyper-parameters w.r.t topology, and warm-starting optimization algorithms).

Remerciements

En guise de reconnaissance, je tiens à témoigner mes sincères remerciements à mes directeurs de thèse: Marc et Michèle, qui m'ont à la fois donné la liberté sur les différents thèmes à explorer, mais aussi ont été très disponibles pour les conseils. Ce fut un réel plaisir d'avoir effectué ma thèse sous votre encadrement.

Je remercie également Nicolas Courty, Frank Hutter, Joaquin Vanschoren, Rémi Bardenet et Frédéric Saubion, d'avoir accepté de faire partie de mon jury de thèse.

Je voudrais exprimer ma profonde reconnaissance à tous mes co-auteurs: Andry, Antoine, Baptiste, Carola, Gabriel, Gwendoline, Jeremy, Laurent, Louisot, Pak, et Olivier.

Je tiens aussi à remercier amplement, sans exception, tous les membres de l'équipe TAU. J'ai passé 5 ans très enrichissantes et sympathiques, qui ont sans doute contribué à la réussite de cette thèse.

Je voudrais tout particulièrement remercier Zhengying pour les discussions et collaborations durant la thèse, Omar pour les discussions interminables par téléphone, et mon cher binôme Yaohui pour nos anciennes et futures collaborations.

Je tiens à dire ma gratitude à tous mes amis Malagasy en France et à Madagascar, qui m'ont toujours encouragé durant ma thèse.

Et enfin, je suis extrêmement reconnaissant à ma famille: mes parents, ma soeur et mon frère, pour leur support durant toutes mes années études.

Synthèse

Malgré les succès des algorithmes d'apprentissage statistique à résoudre de nombreuses tâches complexes, le choix et la configuration de ces modèles restent un problème difficile en pratique. Cette nécessité de choix et configuration s'élargit à toutes les étapes du traitement allant du nettoyage des données à l'entraînement de modèle. L'approche AutoML (Automated Machine Learning) [Hutter et al. 2019] a suscité énormément d'intérêt dans la communauté de recherche durant les dernières décennies afin de surmonter ce problème. Cette thèse présente trois principales contributions afin d'améliorer l'état de l'art de ces approches AutoML. Elles sont divisées entre deux thèmes de recherche: l'optimisation (première contribution) et meta-apprentissage (deuxième et troisième contributions).

La première contribution concerne un algorithme d'optimisation hybride, appelé Mosaic, qui exploite les méthodes MCTS et optimisation bayésienne pour résoudre respectivement la sélection des algorithmes et la configuration des hyperparamètres. L'évaluation, conduite à travers le benchmark OpenML 100, montre que la performance empirique de Mosaic surpasse ceux des systèmes d'AutoML de l'état de l'art (Auto-Sklearn [Feurer et al. 2015a] et TPOT [Olson et al. 2016]).

La deuxième contribution introduit une architecture de réseau neuronal, appelée Dida, qui permet d'apprendre des descripteurs de données invariants à la permutation de colonnes et d'exemples. Deux tâches (classification des patchs et prédiction des performances) sont considérées lors de l'évaluation de la méthode. Les résultats de Dida sont encourageants comparés à ceux de ses concurrents (Dataset2vvec [Jomaa et al. 2021] et DSS [Maron et al. 2020]).

Enfin, la troisième contribution, intitulée Metabu, vise à surmonter les limites de Dida à opérer sur de vrais jeux de données d'AutoML. La stratégie de Metabu comporte deux étapes. Tout d'abord, une topologie idéale, basée sur les meilleurs hyperparamètres, de ces jeux de données est définie. Puis, une transformation linéaire des descripteurs manuels est apprise pour les aligner, selon un critère de transport optimal, avec la représentation idéale. Les comparaisons empiriques montrent que les descripteurs Metabu sont plus performants que les descripteurs manuels sur trois problèmes différents (évaluation du voisinage, recommandation d'hyperparamètres, et initialisation d'un algorithme d'optimisation).

Contents

List of Figures

List of Tables

Introduction

ARTIFICIAL Intelligence (AI) is ever more present in numerous real-life contexts, such as marketing [Brei 2020, Dzyabura and Yoganarasimhan 2018], healthcare [Bhardwaj et al. 2017, Wiens and Shenoy 2018], and transportation [Zantalis et al. 2019, Tizghadam et al. 2019]. However, the pervasive deployment of AI remains in its infancy. Numerous research papers from conferences such as NeurIPS, ICML, AAAI, and ICLR continue to make discoveries in the field of AI. These discoveries yield a broader understanding of the theoretical and empirical proprieties of approaches toward AI while also reducing their computational complexities. Furthermore, tech companies are recently devoting more resources to implementing recent AI advancements to solve real-life problems.

The emergence of Machine Learning (ML) is among the main reasons for the recent success of AI. ML covers any method that learns from data, experiences, or interactions. It has gained significant interest in the research community for myriad reasons. For example, the current technological infrastructure and existing social network platforms ease the collection and storage of data at an exponential rate. In this context, world-renowned magazines [Forbes 2018, Economist 2017] argue that data is the new oil of the 21st century, and only companies that can efficiently exploit information will remain competitive. Fortunately, ML proposes intelligent strategies to mine the available data by identifying patterns to support domain-level decisions or learning recurring tasks for further automation. Another reason for increased interest in AI comes from advancements in computing power, which allow the adoption of ML models. Modern-day computing units (e.g., CPUs and GPUs) are improving rapidly and, therefore, becoming more efficient and accessible for companies.

However, the unprecedented success of ML models comes at the cost of the complexity of choosing a suitable model. In the last fifty years, researchers have proposed a wide variety of ML models, each one having its strengths and limitations. A key challenge for adopting ML involves correctly choosing the model that best fits the problem at hand. A traditional ML experiment often extends to additional steps such as data preparation, cleaning, and setting hyper-parameters. The overall processing steps are called *pipelines* throughout the rest of the document.

In practice, researchers and data scientists rely on their experiences over similar problems to find the most promising pipeline. While it allowed tuning state-of-the-art AI models [Krizhevsky et al. 2012, Silver et al. 2016, 2017, Senior et al. 2020], this manual approach is a tedious and error-prone task due to the enormous possibilities of experiment settings. AutoML (Automated Machine Learning) aims thus at addressing this limitation by automating the search process. Within the AutoML context, three strategies are proposed in this thesis to improve the efficiency of the search over the existing approaches.

Automated Machine Learning: AutoML

AutoML is a hot topic in AI, situated at the intersection of Machine Learning and Optimization. It is a subfield of the long-dated research area of Algorithm Selection (AS). AS was first tackled by Rice in 1976 in his seminal work *The Algorithm Selection Problem*, paving the way for a large body of works (Chapter 1).

The AutoML research has received incredible interest from the AI community over the last two decades. It is reflected by the successes of AutoML workshops (from 2014 to 2021) and international AutoML challenges [Guyon et al. 2015, Escalante et al. 2020]; all confirm the growing tendency of AutoML papers and interests.

The early AutoML competitions focus on tabular datasets, leading to the development of Auto-Sklearn [Feurer et al. 2015a]. After that, Liu et al. [2018b] organized further challenging tasks to tackle various domains, including computer vision [Liu et al. 2019] and speech processing [Wang et al. 2020]. Recently, Baz et al. [2021] proposed a competition on Meta-Learning to learn through a sequence of ML tasks. These competitions played a crucial role in developing robust practical and theoretical AutoML systems [Feurer et al. 2015a, 2018, Lim et al. 2019, Baek et al. 2020].

If numerous AutoML systems are available in open-source [Feurer et al. 2015a, Olson et al. 2016, Thornton et al. 2013, Mohr et al. 2018, Gijsbers and Vanschoren 2019], they are often targeted for research purposes, operating on standard dataset benchmarks [Dua and Graff 2017, Vanschoren et al. 2014]. Several challenges thus need to be addressed to fulfill the promise of AutoML to the best extent possible [Blom et al. 2021].

Technical challenges of the AutoML problem

The critical components of AutoML are two-fold. On the one hand, AutoML relies on an optimization algorithm to search for the optimal ML experiment setting. On the other hand, it requires learning from previously seen tasks to speed up the optimization.

The optimization part, the core of AutoML, is the Hyper-Parameter Optimization (HPO) [Feurer and Hutter 2019]. The HPO problem involves a noisy, expensive, and black-box optimization problem over a structured search space. Another critical challenge of HPO is to enforce the generalization to the hold-out test instances of the dataset. Despite these difficulties, however, several Black-Box Optimization algorithms (e.g., Bayesian Optimization, Evolutionary Algorithms, and Planning algorithms) remain appropriate for addressing the HPO problem.

AutoML can also leverage knowledge from previous similar tasks to speed up the optimization (e.g., warm-starting HPO algorithms). This strategy is called Meta-Learning, as it requires learning on a task level. In practice, Meta-Learning is shown to drastically reduce the computational cost of HPO [Feurer et al. 2015b]. A

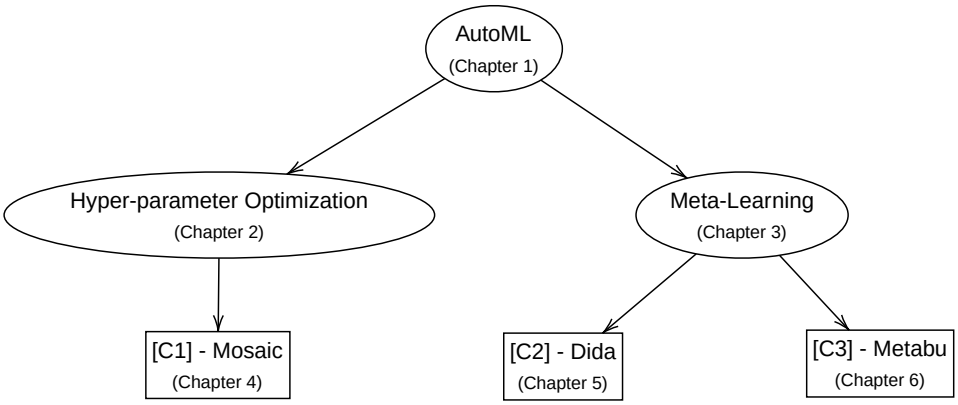


Figure 1: Outline of the thesis. The contributions are depicted by square.

primary challenge of Meta-learning is the lack of a clear definition of task similarity to support the learning. Nevertheless, the literature provides various approaches to estimate this similarity. A first line of research is to learn task similarity during the HPO, solely relying on the evaluated pipelines [Fusi et al. 2018, Yang et al. 2019]. A second research direction is to map a task into a set of descriptors, termed meta-features [Rivolli et al. 2022], which are further leveraged to assess the similarity between tasks. However, if the latter approach showed promising results to describe optimization problems [Xu et al. 2008, Kotthoff 2016], its efficiency for describing machine learning tasks is limited [Misir and Sebag 2017]. One of the purposes of this thesis is to address this limitation.

Outline of the Thesis

As illustrated in Figure 1, this thesis presents three contributions addressing issues both in the Hyper-Parameter Optimization and Meta-Learning sides. It is organized as follows.

Part I focuses on the formal background of the AutoML problem. Starting with the motivation and context of the work, it then presents an overview of the Algorithm Selection domain and, afterward, an introduction to AutoML (Chapter 1). The state-of-the-art methods for HPO (Chapter 2) and Meta-Learning (Chapter 3) are then described. Further, Parts II and III present the three contributions of the thesis (details bellow). Lastly, this manuscript concludes with a summary of the contributions and a discussion of the perspectives and future works direction.

The contributions are separated into two parts: one contribution for HPO (Part II) and two contributions for Meta-learning (Part III). They are described as follows.

[C1] Monte Carlo Tree Search for Algorithm Configuration (Part II, Chapter 4). This contribution, entitled *Automated Machine Learning with Monte-Carlo Tree Search* [Rakotoarison et al. 2019], was published at the *Twenty-Eighth International Joint Conference on Artificial Intelligence*. It addresses the complexity of learning over structured search space induced by the sequence of choice required to build an ML pipeline. Concretely, a pipeline describes the dependencies of the processing steps, from data preparation to the training algorithm, to yield an end-to-end ML experiment. The fundamental idea of this chapter is to propose a hybrid algorithm: (a) a Monte-Carlo Tree Search (MCTS) strategy to handle the algorithm selection part, (b) and a Bayesian Optimization (BO) algorithm to deal with tunning the hyper-parameters. The proposed approach, dubbed Mosaic, thus inherits the advantages of BO as being sample efficient and MCTS suitable for the combinatorial nature of pipeline selection.

[C2] Distribution-Based Invariant Deep Networks for Learning Meta-Features (Part III, Chapter 5). As mentioned earlier, Meta-Learning uses task similarity to reduce the computational cost of an HPO running a new task. For example, it can exploit knowledge from the most identical previously seen task. The efficiency of the Meta-Learning, thus, critically depends on the distance metric used to compare tasks. This chapter is concerned with defining the task similarity with the help of meta-features. While current state-of-the-art meta-features still rely on hand-made meta-features, this work considered a novel perspective of learning them. Mainly, the contribution is a Neural Network architecture, dumbed Dida, that handles tasks as input and outputs meta-features. Since ML tasks have varying dimensions with invariance proprieties, the primary difficulty is accommodating such constraints into a neural network. This work, entitled *Distribution-based invariant deep networks for learning meta-features*, is available as a preprint paper [De Bie et al. 2020].

[C3] Learning meta-features for AutoML (Part III, Chapter 6) This contribution, called *Learning meta-features for AutoML* [Rakotoarison et al. 2021], will appear at the *Tenth International Conference on Learning Representations*. It is a follow-up on the previous contribution, mainly to mitigate Dida limitations. Those limitations concern three barriers restraining the adoption of the learned meta-features for AutoML. First, the most significant task benchmark available [Bischl et al. 2017] is insufficient to learn meta-features. Second, Dida does not treat general tabular data because it does not handle data quality issues such as categorical variables and missing values. The latter concerns depreciated its relevance to the general AutoML tasks. Third, the target variable (meta-features suitable for AutoML) is unavailable hence needs to be constructed in advance. The proposed approach Metabu intends to pave the issues mentioned above to learn task meta-features for AutoML.

Part I

Background and State-of-the-art

1 - Formal Background

AutoML is attracting considerable interest in the research community to make machine learning algorithms more robust and support the deployment of these ML algorithms into real production scenarios. This work aims to advance current AutoML approaches to achieve this objective.

This first chapter introduces the formal background of the AutoML domain to allow the reader to situate the contributions presented in the upcoming chapters. Firstly, Section 1.1 presents the context and motivation behind AutoML. Then, Section 1.2 reviews the AutoML acknowledged mother-field, namely Algorithm Selection. Finally, Section 1.3 formally introduces AutoML, focusing on the optimization problem tackled throughout this manuscript.

1.1 . Context & Motivation

1.1.1 . Context

Automated Machine Learning (AutoML) builds upon the fields of Algorithm Selection (AS) and Algorithm configuration (AC) techniques to respectively select and tune machine learning pipelines for a given task. In this context, ML *pipeline* refers to the sequence of all the processing steps, from data preparation to training ML model, yielding an end-to-end training of an ML model. Therefore, it involves parameters; for simplicity, all pipeline parameters are called hyper-parameters.

The considered ML tasks include supervised learning (classification, regression), unsupervised learning (clustering), and reinforcement learning problems. This thesis focuses on supervised learning, specifically the single label (binary and multi-class) classification problem.

On the one hand, the algorithm *selection* is the process of choosing one out of a set of possibilities, such as selecting the optimal learning algorithm from a collection of classifier models. The Algorithm Selection problem was first formalized in Rice [1976]. It also pointed out various applications of AS, ranging from estimation to artificial intelligence. Section 1.2 provides a detailed review of AS for completeness.

On the other hand, the algorithm *configuration* is the approach to set hyper-parameter values, e.g., tuning the regularization parameter C of the SVM model [Boser et al. 1992]. Note, however, that, since the performance of an algorithm tightly depends on its hyper-parameters, it is not uncommon for researchers to combine AS and AC within the same optimization process (more in Section 2.2).

1.1.2 . Motivation

One of the primary motivations of AutoML is to handle the overwhelming task of choosing an ML algorithm and configuring its hyper-parameters. Indeed, numerous recent studies suggest that machine learning algorithms dominate the

broader field of AI, to name a few, ranging from computer vision [Krizhevsky et al. 2012], driving cars [Bojarski et al. 2016], playing games [Silver et al. 2016, 2017] to learning protein structure [Senior et al. 2020]. However, these successes and breakthroughs were only obtained by carefully choosing the learning model and its hyper-parameter values. The purpose of AutoML thus is to delegate the time-consuming and expertise-demanding procedures of algorithm selection and configuration to the machine.

Another motivation of AutoML is to address, to some extent, the shortage of experienced data scientists, opening the room for non-experts to build high-performance machine learning models. For example, it allows researchers from other domains (e.g., medicine and climate change) to benefit from ML at its best in their respective research fields.

From a theoretical point of view, AS and AC have gained ever more attention since the publication of the No-free lunch theorem (NFL). In a nutshell, this theorem states that all (optimization or ML) algorithms perform equally when considering their performance expectation over a uniform distribution on the set of possible problem instances.[1] This NFL theorem, which was proved for black-box optimization [Wolpert and Macready 1995] and later for supervised machine learning [Wolpert 1996], thus establishes that there is no point in finding a universal algorithm in the above sense and paves the way toward developing portfolios of algorithms and selecting the appropriate ones depending on the problem at hand.

1.2 . Overview on Algorithm Selection

Prior to its application in machine learning, AS was broadly applied in several domains such as Travelling Salesman Problem (TSP) [Kotthoff et al. 2015], Satisfiability Problem (SAT) [Xu et al. 2008], Mixed-Integer Programming (MIP) [Hutter et al. 2010, Xu et al. 2011] to Constraint Programming [Loth et al. 2013]. We refer the interested reader to the recent literature reviews of Kotthoff [2016] and Kerschke et al. [2019], which describe the foundations and up-to-date results of applying AS to optimization problems.

Rice [1976] formalizes the AS problem as a procedure to learn a mapping from problem space $\mathcal{I}$ to the algorithm space $\mathcal{A}$; i.e., associating a problem instance to its optimal algorithm. Rice's formalization, also known as Per-Instance Algorithm Selection (PIAS), is defined in Definition 1 and illustrated in Figure 1.1.

Definition 1 (Optimal decision in PIAS)**.** *Let $\mathcal{I}$ and $\mathcal{A}$ respectively denote a set of problem instances and a set of algorithms. Then, given $p : \mathcal{I} \times \mathcal{A} \mapsto \mathbb{R}$, a loss function to be minimized, the optimal decision in PIAS is a pair of a*

[1] Note that this does not preclude the existence of an optimal e.g. pipeline in the context of a given task domain.

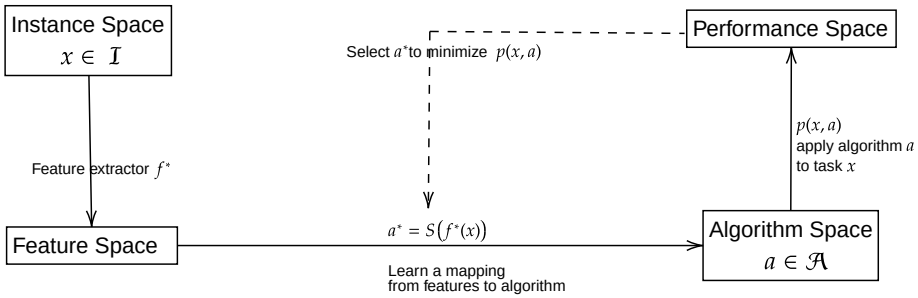


Figure 1.1: The continuous edges represent the original formalization of Algorithm Selection as in Rice [1976].

selector S and feature extractor f^* that when combined minimizes the loss p for all instances $x \in \mathcal{I}$.
$\forall (x, a) \in \mathcal{I} \times \mathcal{A}$, $p(x, S(f^*(x))) \leq p(x, a)$.

PIAS involves two sub-tasks. Firstly, each problem instance is associated with a set of descriptors noted $f^*(x)$, computed from a feature extractor function f^*. Then, the selector S exploits the description $f^*(x)$ to determine the best algorithm $S(f^*(x))$, with optimal performance in the sense of the considered loss function.

Kotthoff [2016] discusses various building blocks involved in the AS problem, chiefly: designing algorithm portfolios, learning a surrogate model of the performance $p(x, a)$, and constructing the feature extractor f^*. The following subsections briefly discuss these three building blocks.

1.2.1 . Portfolio optimization

Researchers and practitioners are both convinced that an algorithm is unlikely to perform best for all problem instances. As said, the idea of a universal algorithm goes against the NFL theorem [Wolpert and Macready 1997, Wolpert 1996].[2] The mainstream approach to overturn the NFL is to design a portfolio of algorithms, a finite set of algorithms. Ideally, algorithms in the portfolio are diverse enough to cover the landscape of the problem space. Souravlias et al. [2021] presents a recent survey of portfolio optimization and its related challenges.

AS suitably handles the selection of an algorithm in a portfolio, as the latter is a finite set of algorithms. However, the AC problem instead considers a set of configurations, the size of which might be infinite (due to continuous hyper-parameters) or at least exponentially increasing with the number of hyper-parameters. A critical issue with algorithm portfolios is that their performance depends strongly on the considered problem instance; typically, the "best on average" algorithm on the portfolio is not necessarily the best one on problem instance $\mathcal{I}$.

[2]Although the NFL relies on the rather unrealistic assumption of a uniform distribution on the set of all possible problem instances.

Ideally, one requires a perfect joint representation of algorithms and problem instances to:

(i) Cluster the instance space;

(ii) Select the optimal algorithm for each cluster to define a portfolio with good coverage.

Some works [Yap et al. 2020, Smith-Miles and Lopes 2012] followed this line of research in combinatorial optimization problems, although they did not apply their results to Algorithm Selection for ML algorithms.

1.2.2 . A Machine Learning approach to surrogate model learning

As illustrated in Figure 1.1, algorithm selectors usually leverage feedback value such as the observed performance, gathering more information about the current task, which thereafter fed to the selector S to support AS. Most authors [Xu et al. 2008, Hoos et al. 2015] introduce a *performance model* estimating the performance $p(f^*(x), a)$, and delegate the learning of the performance model to mainstream machine learning algorithms. Under the assumption of a good enough feature extractor f^*, the performance model predicting $p(f^*(x), a)$ exploits a (meta-) dataset composed of pairs $\{(f^*(x), a)); p(f^*(x), a))\}$ and proceeds exactly as in supervised learning.

Along this line, ML-based selector approaches allow the offline exploitation of a performance database (involving triplets: instance $f^*(x)$; algorithm a; associated performance $p(f^*(x), a)$). This approach can leverage earlier run experiments and adapt the model in case of changes in the algorithms or in the problem instance distribution.

In practice, the performance model and AS come in two modes. In the first mode, the learned model is the selector itself: $S : \mathcal{I} \mapsto \mathcal{A}$

In the second mode, one learns the performance model defined on pairs of instance-algorithm:

$$\hat{p} : \mathcal{I} \times \mathcal{A} \mapsto \mathbb{R}$$

that is thereafter used to select the most promising algorithm: $S(x) = \text{argmax}_a \, \hat{p}(x, a)$.

In the former case, S is a classifier, where each algorithm corresponds to a class; this approach is arguably best suited to portfolios. In the latter case, the performance model is more flexible as it can predict any continuous value criterion such as runtime, loss, or performance.

An alternative solution to these learning approaches is to construct S as a set of static hand-made rules without a learning component. The selection rule is thus based solely on the instance features. A recent benchmarking paper [Meunier et al. 2021, Liu et al. 2020a] based on this idea shows that the rule-based selected algorithm outperforms the portfolio algorithms. The comparison was carried out

from various optimization problems ranging from academic benchmarks to real-world applications.

1.2.3 . Instance features

Instance features play a crucial role in AS. For example, a "perfect" representation of instances would support an optimal recommendation. Let the Euclidean distance based on the features allows to identify the nearest instances to the current instance problem. Assuming the neighborhoods based on this representation were "perfect", one could pick the optimal algorithm for the current instance as the best algorithm for its nearest neighbor. Overall, instance features define a computable vector representation of every instance problem. The interested reader is referred to [Kerschke et al. 2019] for a survey of the feature sets adopted for various optimization problems.

1.3 . Automated Machine Learning (AutoML)

As said, AutoML involves an Algorithm Selection component aimed to select an ML algorithm to handle the ML problem instance. When not specified, a problem instance refers to a dataset in the remainder: a set of samples, each described with features values and target label to be predicted.

AutoML also involves an Algorithm Configuration component, referred to as Hyper-Parameter Optimization. The AC component aims to configure an end-to-end and trainable machine learning experiment, or **pipeline**, defined as a sequence of processing algorithms and their associated hyper-parameters.

The building blocks of AS introduced in Section 1.2 all apply to the AutoML domain:

- Algorithm portfolio (Section 1.2.1) is leveraged by various works on ML pipelines recommendation [Misir and Sebag 2017, Yang et al. 2019, Fusi et al. 2018].
- Surrogate performance model (Section 1.2.2) is also standard in AutoML, mainly to speed up hyper-parameter optimization algorithms [Bergstra et al. 2011, Swersky et al. 2014, Feurer et al. 2015a, Fusi et al. 2018].
- Instance features (Section 1.2.3), commonly termed **meta-features** for clarity (and make the distinction with dataset features), are of high interest in AutoML especially to transfer knowledge across problem instances (datasets).

The following sub-sections detail all the definitions, challenges, and prerequisites.

1.3.1 . Problem Statement

As illustrated on Rice's diagram (Figure 1.1), AutoML proceeds to find an optimal pipeline in the sense of a predefined criterion, w.l.o.g. a loss function to be minimized. This goal can be formalized as an optimization problem (Definition 2) on the whole configuration space Θ, searching for the optimal pipeline $\theta_z^* \in \Theta$ for dataset z.

Definition 2 (AutoML)**.** *Let z be a dataset (w..l.o.g. a binary or multiclass classification problem) and z_{train} and z_{valid} two disjoint subsets of z. Let Θ be the space of machine learning pipelines. The following optimization problem defines AutoML on a dataset z:*

$$\text{Find } \theta_z^* \in \arg\min_{\theta \in \Theta} L(\theta, z_{train}, z_{valid}), \tag{1.1}$$

where L denotes a loss function to assess the ML pipeline θ (trained on z_{train}) on z_{valid}.

Regardless of the type of ML task examined (e.g., classification or regression) and the pipeline space, AutoML involve two basic and essential components: an optimization algorithm (Chapter 2) and a Meta-Learning method for learning across tasks (Chapter 3). The two AutoML components mentioned above, like the above formalization, are agnostic w.r.t. the type of ML task. Nevertheless, the rest of the manuscript focuses on classification problems.

1.3.2 . Technical issues from the AutoML problem

The optimization problem defined in Equation 1.1 presents some challenges, being noisy, structured, black-box, and expensive. They are discussed below.

Noisy optimization

The noise observed in the objective function L has many sources. A first source might be the randomness of ML algorithms, e.g., the initialization in a neural network. Such an issue is handled in practice by fixing the random seed.
A second source may come from the sampling of the training, validation, and test sets from the whole dataset z (e.g., cross-validation split).
A third source is the noise of the optimization algorithm itself (e.g., see Shang et al. [2019], especially when the noise is not gaussian).[3]

These noises can be straightforwardly handled, as done in various AutoML systems [Feurer et al. 2021a, Thornton et al. 2013, Olson et al. 2016] and benchmarking papers [Balaji and Allen 2018, Gijsbers et al. 2019], by averaging the performance obtained over multiple independent runs (with varying dataset splits). Nevertheless, in counterpart, this procedure linearly increases the cost of the AutoML process.

[3] Addressing this algorithmic noise is out of the scope of the presented work.

Structured Optimization

An end-to-end AutoML system involves all components of a complete machine learning experiment, ranging from data pre-processing through feature selection to training and ensembling models.

The search space Θ represents the set of possible machine learning pipelines encompassing the union of the space of feasible algorithms with the domain of their hyper-parameters. It thus includes a mix of binary,[4] categorical and continuous variables. For example, Feurer et al. [2015a] considers 40 categorical and 66 real continuous hyper-parameters for binary classification tasks.

Operating directly on Θ is hardly feasible. First, finding the algorithmic components present in pipeline z^*, that is, optimizing the binary variables in Θ, defines a hard combinatorial, NP-hard problem. Second, the exploration of Θ must account for the dependencies among its variables, reflecting the structure of ML pipelines. In other words, the value of some variables controls the relevance of some other variables. For example, a polynomial SVM kernel comes with two specific hyper-parameters; the fact that an algorithmic component is present implies that its hyper-parameters are relevant.

In practice, Θ is defined from the set of algorithm candidates, each with the domain space of their respective hyper-parameters, selected by the human expert. The pipeline structure is tackled by considering another formalization of the search space (as Bayesian Optimization does not directly handle structured search space). Alternatively, another solution is to leverage structure-aware optimization algorithms (such as Evolution Strategies). We return to this issue in Chapter 2.

Black-Box Optimization (BBO)

BBO aims at optimizing a function f without exploiting (or having access to) its analytical definition and computational implementation. BBO can only compute the value $f(x)$ for each input x. In particular, BBO does not use the derivatives of f. The AutoML objective function L (Equation 1.1) defines such a BBO problem, as the value of L for a given dataset and configuration can only be computationally estimated.

Another interesting AutoML approach is to rely on bi-level optimization. It proceeds with a proper formalization of the search space, enabling the use of gradient-based optimization to AutoML [Liu et al. 2018a, Franceschi et al. 2018]. This idea will not be covered in this work, however.

[4] In pipeline $\theta \in \Theta$, the binary variable associated to each pipeline component takes the value $true$ iff this component is part of θ.

Expensive Optimization

According to a recent survey presented by Blom et al. [2021], one of the main obstacles to deploying AutoML approaches in real-life production is their prohibitive computational cost. Indeed, given a Black-box function f, BBO proceeds by computing $f(x)$ for all candidates x, making the overall computationally demanding. Various strategies were proposed to address this issue and further speed up the search. In particular, multi-fidelity strategies [Swersky et al. 2014, Li et al. 2017, Klein et al. 2017] were founded to be incredibly effective. They rely on an approximate but inexpensive estimation of the objective function, to accelerate the optimization while controlling the model complexity or the size of datasets. The approaches to reducing the training time are discussed further in Section 2.2.

Generalization Perspective

As formalized in the statistical learning theory [Vapnik 2000], the essential objective for learning a model is to achieve good performances in expectation. The sought solution thus is to learn a pipeline, based on its only performance on the training and validation sets, that would perform well on a holdout test set (unseen during the optimization), demonstrating that they do not overfit.

AutoML solutions are particularly prone to over-fitting as they require many lookups to the validation score. This overfitting issue is acknowledged as a critical issue; still, the AutoML literature does not agree on how to address this issue. Benchmarking papers [Zöller and Huber 2021, Gijsbers et al. 2019] also raise this issue as one of the causes of the decrease in performance on subset tasks when optimizing for a long time budget. Researchers often rely on cross-validation [Allen 1974, Geisser 1975] scores to minimize the risk of over-fitting.

1.3.3 . Evaluating AutoML systems

The fair comparison of AutoML systems requires that all competitors operate in the same search space Θ and are evaluated along with the same benchmarking procedure.

The search space

As said, the choice of the search space Θ is usually left to the human expert (or encapsulated in the considered algorithm portfolio) to make it a tractable bounded search space. Nevertheless, this choice can eventually affect the difficulty of the optimization.

At the time of writing, the choice of Θ depends on the application domain. Currently, deep learning models are dominating the field of computer vision, NLP, and speech recognition. For these application domains, a strong preference is given to Neural Architecture Search (NAS) over standard machine learning models. As an

example, in DARTS [Liu et al. 2018a], the AutoML problem is formalized as a two-level optimization problem, where the first level aims to learn the interconnection of a set of small networks and the second level aims to determine the respective weight of each of these small networks.

The presented research aims to achieve AutoML for tabular data, which motivates our choice to consider mainstream machine learning and pre-processing algorithms.

Benchmark datasets

Standard practice evaluating and comparing AutoML relies on open-source dataset benchmarks such as UCI [Dua and Graff 2017] and OpenML [Vanschoren et al. 2014].

For the sake of a fair and tractable assessment, we only consider curated and medium-size benchmarks: OpenML CC18 [Bischl et al. 2019] and OpenML 100 [Bischl et al. 2017].

While OpenML contains 3,448 datasets at the time of writing, many have data quality issues, such as datasets with constant features. Some datasets are too big or ill-conditioned, entailing a large SVM running time. Some datasets are also deprecated versions of the others, which may create a risk of over-optimistic evaluation. Because of these issues, Bischl et al. [2019] built OpenML CC-18, a curated benchmarking suite for AutoML, succeeding OpenML 100 [Bischl et al. 2017]. As far as we know, OpenML CC-18 is the largest curated tabular dataset benchmark available for AutoML.

2 - Hyper-parameter Optimization

This chapter focuses on one of the two core tasks of AutoML, referred to as Hyper-Parameter Optimization, that consists in setting the hyper-parameters to optimize the performance (Equation 1.1). The other core task, namely Meta-Learning, will be described in Chapter 3.

This chapter is structured as follows. After reviewing the approaches of the HPO literature [Feurer and Hutter 2019] in Section 2.1, we situate HPO w.r.t the general AutoML problem (Section 2.2) and give an overview of the major AutoML systems (Section 2.3). The chapter finally presents the AutoML benchmarking methodology (Section 2.4).

2.1 . State-of-the-art of Hyper-Parameter Optimization approaches

The early methodology used to set hyper-parameters relies on manually picking hyper-parameter values along a trial and error procedure, and this methodology still is commonly used. However, it faces severe limitations: in terms of domain knowledge to judiciously sample good hyper-parameter values for the problem instance at hand; and in terms of both human and computational time requirements. HPO algorithms thus aim to address these limitations. Figure 2.1 highlights several HPO approaches, which are discussed in the following sub-sections.

2.1.1 . Mainstream Approaches

The most straightforward HPO approaches are the grid search and random search strategies.

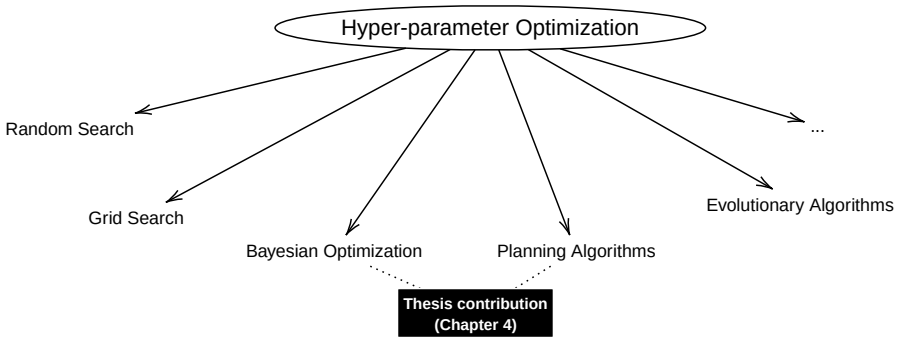


Figure 2.1: Non-exhaustive list of state-of-the-art approaches for HPO.

The grid search (GS) strategy considers a (manually defined) finite set of values for each hyper-parameter. The set of candidate pipelines is defined by considering all combinations of hyper-parameter values. All pipelines are then evaluated in a sequential or parallel manner. Then, the final recommendation to the user is the best configuration in the sense of the considered evaluation metric.

The efficiency of GS depends on the considered hyper-parameter grid values. Recently, Ndiaye et al. [2019] proposed exploiting the objective function's theoretical properties to define an optimal grid of hyper-parameters. Nevertheless, the cost and the performance exponentially increase with the number of hyper-parameters. Furthermore, a severe limitation of the GS approach is when only a few of the hyper-parameters are critical [Bergstra et al. 2011, Hutter et al. 2019].

Compared to GS, Random Search (RS) only requires the hyper-parameter domain spaces to be defined. It proceeds by *uniformly* sampling candidate configurations from the specified hyper-parameter domains. Despite its simplicity, RS performs well on expensive settings, e.g., Neural Networks [Bergstra et al. 2011]. Hence, RS is commonly used as a baseline on numerous HPO and AutoML papers [Hutter et al. 2011, Bergstra et al. 2011, Feurer et al. 2015a, Olson et al. 2016, Thornton et al. 2013]. Note that different versions of RS can be formulated depending on the sampling strategy. For instance, instead of using uniform sampling, researchers experimented with other space-filling sampling methods [Bousquet et al. 2017, Cauwet et al. 2020], enforcing the diversity of sampled pipelines.

A key strength of GS and RS approaches is the ease of parallelization, even more so as the emergence of high computing infrastructures supports the deployment and study of parallel methods in academia. For example, [Li et al. 2020, Cauwet et al. 2020] show the merits of a massive random search approach compared to Bayesian Optimization and Evolutionary Algorithms on various HPO tasks.

2.1.2 . Bayesian Optimization

The celebrated Bayesian Optimization (BO) approach [Mockus 1989, Brochu et al. 2010, Frazier 2018] is an optimization algorithm tailored for black box and expensive optimization problems under limited computational resources, thus well suited to HPO. BO uses an auxiliary probabilistic model, also termed surrogate model, to guide the search. The surrogate model is meant to model the optimization objective and estimate the modeling uncertainties; both are leveraged during the optimization.

A notable implementation is the Sequential Model Based-Optimisation (SMBO), which is commonly used to achieve HPO (Alg. 1). It proceeds as follows. Iteratively (for a total number T of iterations, governing the optimization cost), the surrogate model M is learned from the observed performances (line 6), then used to choose a promising new hyper-parameter configuration (line 3), that is evaluated afterward (line 4). The function A, termed Acquisition Function, encapsulates the selection procedure of the next hyper-parameter to evaluate.

Bayesian Optimization commonly uses a Gaussian Process [Rasmussen and

Algorithm 1: Sequential Model-Based Optimization (SMBO)

input : initial surrogate model M_0, number of iterations T, dataset z, acquisition function A, loss function L, hyper-parameter space Θ

output: Recommended solution θ^*

$\mathcal{H} \leftarrow \emptyset$
for $t \leftarrow 1$ **to** T **do**
  $\theta_t \leftarrow \text{argmax}_{\theta \in \Theta} A(M_{t-1}, \theta)$
  Evaluate $L(\theta_t, z)$
  $\mathcal{H} \leftarrow \mathcal{H} \cup \{(\theta_t, L(\theta_t, z))\}$
  Fit M_t on $\mathcal{H}$
end
$(\theta^*, l^*) \leftarrow \text{argmin}_{(\theta,l) \in \mathcal{H}} l$

Williams 2005] as a surrogate model for its soundness and efficiency in terms of both performance prediction and uncertainty estimation. GP, however, suffers from two limitations. Firstly, it does not scale up w.r.t. the number of samples and the dimension of the search space. Secondly, it is well defined on continuous domains only.

The acquisition function that is commonly used is *Expected Improvement* (EI) [Jones et al. 1998], defined as follows:

$$A(M, \theta) = \mathbb{E}[max(L_{min} - M(\theta), 0)] \tag{2.1}$$

with L_{min} be the best performance so far and $M(\theta)$ the estimated loss of hyper-parameter θ according to the surrogate model M.

Wessing and Preuss [2017] states that the success of EI (to tackle expensive optimization problems on a low computational budget) is related to its ability to identify multiple local optima regions. This experimental finding might explain the adoption of EI in the HPO context, as HPO usually admits a number of local optima.

The SMBO approach comes in various modes in the state-of-the-art, which differ in the definition of acquisition function A and the choice of surrogate model M. A non-exhaustive list of open-sourced BO algorithms is presented below.

- SMAC [Hutter et al. 2011, Lindauer et al. 2022] uses Random Forest [Breiman 2001] as a surrogate model with Expected Improvement as an acquisition function. The Random Forest model addresses a core limitation of GP in handling mixed type domain values (e.g., real, categorical, or integer hyper-parameter). Moreover, it drastically reduces the computational complexity both for the training and the inference.

- HyperOpt [Bergstra et al. 2011], instead of modeling directly the performance $M(\theta)$, fits two density distributions for good $\mathbb{P}(\boldsymbol{\theta} = \theta | L(\theta) < \tau)$ and bad $\mathbb{P}(\boldsymbol{\theta} = \theta | L(\theta) > \tau)$ hyper-parameters, with τ a user defined threshold. These distributions are then constructed using a 1-dimensional Parzen Windows density estimation algorithm. A tree structure is introduced to cope with conditional hyper-parameters, hence the term Tree Parzen Estimator (TPE) [Bergstra et al. 2011, 2013]. Note that this algorithm also handles mixed-type variables while having low complexity of training and inference.

- [Snoek et al. 2012] modify the mainstream BO to be better suited to HPO. Firstly, a new kernel function carefully crafted for HPO is proposed. Secondly, it considers the training cost when maximizing the acquisition function (EI per second). Since the internal surrogate model still is a Gaussian Process, it inherits both the advantages and limitations of GPs.

Eggensperger et al. [2013] conducted an empirical benchmarking study on popular BO algorithms, including SMAC, TPE and [Snoek et al. 2012]. The lessons learned from this study are that the GP-based BO [Snoek et al. 2012] tends to outperform SMAC and TPE on low dimensional problems, while TPE yields better performance on higher dimension search space, possibly including conditional hyper-parameter.

2.1.3 . Evolutionary Algorithms

The active research area of Evolution Algorithms (EA), also referred to as population-based algorithms, is concerned with Black-Box optimization problems. Formally, EAs include algorithms based on the evolution of a population of solutions. The evolution is achieved through operators (mutation, crossover, selection) remotely inspired by the Darwinian "survival of the fittest" ideas.

The main two trends in EAs are Genetic Algorithms (GAs) [Mitchell 1996] and Evolution Strategies (ES) [Beyer and Schwefel 2002].

EAs proceed iteratively: an initial population (i.e., a set of initial solutions) is used to create a new generation of solutions by applying mutation and recombination rules over the initial population. Next, the selection rule is applied to construct a new population, refining individuals upon the initial and generated populations. These two steps are repeated until the optimal solution is reached or the training budget is exhausted.

Both GAs and ES are widely applied to HPO. Genetic Algorithms are well suited to the optimization of design structure. A notable application of GAs is the automatic design of neural network architecture, also known as Neuroevolution [Stanley et al. 2019]. NEAT [Stanley and Miikkulainen 2002, Stanley et al. 2009, Risi and Stanley 2012, Miikkulainen et al. 2019] is an example of a prominent Neuroevolution algorithm. A promising application of GAs Real et al. [2020] aims to discover machine learning pipelines from scratch (see Section 2.3). Along the

same line, a search for ML pipelines using context-free grammar is presented by Marinescu et al. [2021].

While GAs mainly handle binary or discrete spaces, Evolution Strategies (ES) is restricted to fixed-size real value space. A successful ES-based optimization algorithm is the Covariance Matrix Adaptation Evolution Strategy (CMA-ES) [Hansen 2005]. The authors of [Loshchilov and Hutter 2016] apply CMA-ES to tune hyper-parameter of deep network models, improving state-of-the-art HPO algorithms such as SMAC and TPE.

Like Random and Grid Search, EAs can be parallelized straightforwardly; their efficiency in parallel mode is widely observed in the literature [Salimans et al. 2017, Jaderberg et al. 2017, Conti et al. 2018, Gaier and Ha 2019]. Furthermore, EAs can also exploit information gathered from previous evaluations (as in any sequential HPO algorithms).

2.1.4 . Bandit & planning Algorithms

Multi-armed bandit

Multi-armed bandits (MABs) [Lattimore and Szepesvári 2020] pertain to the field of Reinforcement Learning [Sutton and Barto 1998], aimed to learn policies yielding an optimal action in each state. MABs consider the single-state RL problem, facing a finite discrete action set, with stochastic bounded action values. MABs were formalized as tackling a sequential decision problem under uncertainty since the early 1930s [Thompson 1933, Robbins 1952, Bather and Chernoff 1967]. Its main applications include A/B testing, resource allocation, and ads placement (not exhaustive list).

Numerous algorithms were devised to address the MAB problem, e.g., for continuous actions [Bubeck et al. 2011] or a large number of actions [Woodroofe 1979, Langford and Zhang 2008]. In the scope of the presented research, i.e., when considering an algorithm selection problem, the MAB algorithm most commonly used is UCB1 (Upper Confidence Bound) [Auer et al. 2002a]. Each algorithm corresponds to an arm; the associated reward is its (noisy) performance.

In Gagliolo and Schmidhuber [2010], the authors propose GAMBLETA, a bandit method to select an optimal algorithm from a portfolio of SAT solvers. The specificity of this method is to leverage contextual information [Auer et al. 2002b] for the bandit algorithm to transfer knowledge across a set of SAT problems.

In Fialho et al. [2008], the authors likewise propose a bandit-based rule selector for an evolutionary algorithm where the novelty lies in the definition of the reward.

Another area of research aims to adapt bandit to algorithm configuration, i.e., HPO. Notably, Shang et al. [2019] proposes a setting to handle an infinite set of hyper-parameters. The proposed method proceeds by maintaining a portfolio of sampled hyper-parameters; at each iteration, the algorithm decides whether to add a new hyper-parameter in the portfolio or consider a previously sampled hyper-parameter, and this hyper-parameter is evaluated.

Another powerful bandit method designed for HPO is the highly cited Hyperband algorithm proposed by Li et al. [2017, 2018]. Unlike previous approaches, Hyperband leverages a bandit algorithm for allocating the overall resource budget across a set of running evaluations; in practice, poorly performing hyper-parameters are discarded early to save resources and attribute them to the best-performing ones.

Planning Algorithms

Another strategy is to sequentially handle the choice of hyper-parameter values (as opposed to the former setting, where the algorithm and the hyper-parameters values are picked simultaneously, allowing to assess the pipeline performance instantly). The issue of such sequential approaches is that the performance can only be measured when all hyper-parameter values are determined: the feedback is delayed. This setting falls into the category of planning problems, aiming to optimize a path (here, the sequence of hyper-parameter values) to maximize the final performance (here, that of the pipeline).

Casting the AutoML into a planning problem opens the room for many planning algorithms to be applied to HPO. One of the solutions is to represent the search space as a tree, with a path representing a pipeline. Any tree search algorithm can thus be considered, such as depth-first search and best-first search [Wever et al. 2018a,b, Mohr et al. 2018]. One of the main contributions of this thesis is to adapt the Monte-Carlo Tree Search (MCTS) [Kocsis and Szepesvári 2006], by combining the tree-structured extension of multi-armed bandit algorithms with Bayesian Optimization, to AutoML (Chapter 4).

A critical limitation of tree-structured representations is that they only handle hyper-parameters with discrete and small size domains. In particular, the domain of continuous hyper-parameters must be discretized.

Along the same line, formalizing AutoML as a sequential decision problem (selecting each hyper-parameter value) makes it a Reinforcement Learning (RL) problem [Sutton and Barto 1998]. Formally, an incomplete pipeline is viewed as a state, and only final states (complete and trainable pipelines) are associated with the pipeline performance reward. RL aims to learn a policy, associating an action to each state and thus navigating among states; an optimal policy is such that the final reward is maximal. An RL approach is based on using Machine Learning algorithms on sequential examples; for example, [Zoph and Le 2017] used an LSTM [Hochreiter and Schmidhuber 1997] to build neural architectures.

2.2 . AutoML as a Hyper-Parameter Optimization problem

Hyper-Parameter Optimization is primarily defined as the only problem of tuning hyper-parameters. AutoML instead considers selecting and tuning larger machine learning experiments, from the choice of data preparation to the learning

algorithm, where each algorithm is associated with specific hyper-parameters. Traditional Hyper-Parameter Optimization algorithms thus require some adaptation to handle the entire AutoML problem.

The idea is to consider a particular algorithm as a hyper-parameter. This representation thus corresponds to a structured and conditional search space, where the previous choices condition the possible options. However, considering this vast and complex pipeline space entails a non-negligible increase in the computational cost. For this reason, substantial research focused on reducing the training cost of HPO to speed up AutoML.

Combining Algorithm Selection and Algorithm Configuration (CASH) Note that the formal definition of AutoML (Equation 1.1), i.e., finding the optimal configuration $\theta^* \in \Theta$ that minimizes the defined loss function, with Θ be the space of observable configurations, already includes the HPO task as Θ both covers the set of algorithms and the space of their hyper-parameters.

The so-called *Combined Algorithm Selection and Algorithm Configuration* (CASH) approach is commonly used in practice [Thornton et al. 2013, Feurer et al. 2021a].

Let an ML pipeline x involve a fixed ordered sequence of ℓ components such as data pre-processing, feature selection, and learning algorithms. At the i^{th} decision step, some algorithm $a_i \in \mathcal{A}_i$ is selected (with $\mathcal{A}_i$ the finite set of possible algorithms at i^{th} step). Denoting $\Theta(a_i)$ the (possibly varying dimension) space of hyper-parameters associated with a_i, the eventual pipeline is described as $x = (a_1, \theta_1), \ldots (a_\ell, \theta_\ell)$, with $\theta_i \in \Theta(a_i)$. Given a ℓ-size pipeline structure, we denote the overall hyper-parameters of the search space as

$$\Theta = \bigcup_{(a_1,\ldots a_\ell) \in \mathcal{A}_1 \times \ldots \times \mathcal{A}_\ell} (a_1, \Theta(a_1)) \times \ldots \times (a_\ell, \Theta(a_\ell))$$

CASH is thus formally covered by the framework of Equation 1.1, where the Θ space is defined as above, encompassing the whole pipeline space.

As said, the domain of a hyper-parameter can be real-valued (such as learning rate), integer-valued (such as the number of layers), binary (for the example, whether to use early stopping or not), or categorical (such as the selection of a learning algorithm).

Reducing computational cost Arguably, the most popular strategy for reducing the computational cost of HPO relies on multi-fidelity approaches, that is, using cheap estimations of the final performance of hyper-parameters. The overall computational budget governs the admissible model complexity through, e.g., limiting the number of trees for Random Forest, the number of examples for SVM, and the number of iterations to any iterative ML algorithm (e.g., neural network). Such a multi-fidelity approach is Hyperband, already cited [Li et al. 2017]. First,

it uniformly samples hyper-parameter domains; then evaluations are subject to a limited resource budget, allowing to discard poor-performing hyper-parameters earlier and thus allocate more resources to promising ones. Finally, the selection process is repeated until one hyper-parameter is retained. This simple strategy showed its merits in optimizing expensive ML models like Deep Neural Network and SVM compared to standard BO algorithms such as TPE [Bergstra et al. 2011] and SMAC [Hutter et al. 2011]. Further improvements proposed by Falkner et al. [2018] and Awad et al. [2021] rely on using respectively Bayesian Optimization and Differential Evolution (as opposed to uniform sampling) in the sampling step, significantly speeding up the search.

Another strategy is to predict the eventual performance associated with a configuration, based on learning curves modeling, and discard unpromising runs in early steps. For instance, Swersky et al. [2014] uses the training history to decide whether to pause the training of a configuration or resume a previously considered training. Likewise, Domhan et al. [2015] model learning curves (using a list of parametric functions), and use the beginning of the learning curve associated with a configuration to estimate whether it is likely to outperform the best configuration so far. Most interestingly, Klein et al. [2017] learns two surrogate models: one for modeling hyper-parameters performance and one for modeling the training cost (depending on the size of the considered training set). The strategy consists of training the most promising configurations with larger training sets, based on a trade-off between the expected gain of performance and the computational cost.

2.3 . State-of-the-art AutoML Systems

This section presents a non-exhaustive list of the prominent AutoML systems, structured after their internal optimization algorithm.

In the realm of Bayesian Optimization-based AutoML there are Auto-Sklearn [Feurer et al. 2015a, 2021a], Auto-Weka [Thornton et al. 2013], and Auto-Prognonis [Alaa and Schaar 2018]. Auto-Sklearn and Auto-Weka are based on SMAC [Hutter et al. 2011], a Random Forest-based B0. Compared to Auto-Weka, Auto-Sklearn involves extra components: a meta-learning strategy to initialize the search and an ensembling strategy to provide a more robust prediction.

Auto-Prognonis uses a GP-based surrogate model with a structured kernel to account for the complex configuration search space. It involves the same extra components as Auto-Sklearn; the ensembling strategy is achieved using Bayesian model averaging, for the sake of explainability.

Another powerful AutoML system is sc Hyperopt-Sklearn [Komer et al. 2014], which uses TPE as a surrogate model; it does not have meta-learning and ensembling components.

Evolution-based AutoML is as commonly used as BO-based AutoML in the literature. A primary advantage of evolutionary algorithms over BO approaches is

on handling structured search spaces naturally without specific adjustments. For instance, TPOT [Olson et al. 2016] and GAMMA [Gijsbers and Vanschoren 2019] use Genetic Programming to evolve compound ML pipelines (preprocessing, feature construction, and model building methods) while enjoying the parallelizable nature of Genetic Programming. Along the same line, de Sá et al. [2017] uses a grammar formalization of the ML search experiment space and designs a grammar-based algorithm for the optimization. Chen et al. [2018] proposes another evolution-based AutoML which instead of optimizing a single machine learning pipeline, focuses its search on finding a combination of pipelines that provides optimal performance overall. More recently, the AutoML-Zero [Real et al. 2020] intends to discover ML pipelines from scratch (without a predefined template as in Auto-Sklearn or Auto-Weka).

A divide-and-conquer strategy proposed by Liu et al. [2020b] consists of decomposing the initial AutoML problem into several sub-problems to reduce the number of variables and address the issue of mixed variable types.

Another category of AutoML systems leverages planning and reinforcement learning methods. In this category, the most notable AutoML is ML-PLAN [Mohr et al. 2018] that formalizes AutoML as a graph problem and leverages tree-search algorithms (e.g., Hierarchical Task Network) with random roll-outs to find the optimal path (i.e., optimal pipeline). Similar approaches were proposed [Kietz et al. 2012, Nguyen et al. 2014] to handle AutoML in data mining tasks.

As said, a severe limitation of the tree-structured approach is that they hardly deal with continuous domains and require the discretization of continuous hyper-parameters. One of the main contributions presented in this manuscript addresses this limitation by hybridizing MCTS and Bayesian optimization, to handle mixed-type variable hyper-parameters.

2.4 . Benchmarking HPO and AutoML algorithms

As said (Section 1.3.3), the benchmarking of AutoML systems presents fairly technical specifics. In particular, it must enforce the same experimental setting for all candidates, considering the same search space and resources budget. The difficulty here is that there is no general agreement in the research community about the environment and search space that should be considered. Typically, the abovementioned state-of-the-art AutoML systems (e.g., Auto-Sklearn, Auto-Weka, Auto-Progronis, TPOT) do not describe their search space.

Nevertheless, some researchers [Balaji and Allen 2018, Gijsbers et al. 2019, Zöller and Huber 2021] conducted benchmarking over the existing AutoML approaches and analyzed the results despite this difficulty. They define a unified framework for AutoML systems and consider a curated subset of the problems in the OpenML benchmark [Vanschoren et al. 2014].

The evidence from these empirical results suggests that *there exists no AutoML*

system that consistently outperforms all others.[1] These results were inspected to determine whether the poor performance is due to over-fitting or difficulty of the optimization. No general conclusion was drawn as the failures seem to be dataset-specific.

This negative result establishes that a broader and more realistic dataset benchmark is needed to push the AutoML analysis further and understand the patterns of difficulty. Note that finding such patterns is significantly related to extracting relevant descriptive features of datasets, i.e., designing meta-features (Chapter 3).

The benchmarking of HPO systems is much more advanced than for AutoML, with quite a few good platforms. For instance, Eggensperger et al. [2021] presents a platform including an extensive set of HPO problems, together with several existing HPO algorithms. The search space and resource budget are fixed for each problem, enforcing a fair comparison between the candidates.

Note that another Black-box benchmarking platform, Nevergrad [Rapin and Teytaud 2018], also considers HPO problems. However, unlike Eggensperger et al. [2021], Nevergrad involves a broader set of optimization algorithms but fewer HPO problems.

[1] Of course, this claim reminds the famed No Free Lunch theorem [Wolpert and Macready 1995]. A key difference however is that the set of tasks considered here is far from being uniform on the space of all tasks.

3 - Meta-learning

In this manuscript, we adopt the tentative definition of meta-learning proposed by Brazdil and Giraud-Carrier [2018] and Vanschoren [2019], that is, *the science of learning from previous experiences, which can be any information gathered from the same task or another task*. Meta-learning is a long open problem, gaining increasing attention in AI in the last decade. This is because the ability to learn from previous tasks is still lacking in mainstream AI approaches, while it is crucial to achieving human-level intelligence.

This chapter provides an overview of meta-learning research and situates the contributions of the thesis w.r.t. the existing research directions. The chapter is structured as follows. First, Section 3.1 motivates the domain of meta-learning and its motivations. Then, a brief survey of the state-of-the-art, describing the meta-learning research spectrum, is introduced in Section 3.2. Finally, Section 3.3 focuses on our main topic of interest, namely the definition and usage of meta-features.

3.1 . Context and Motivations

Meta-learning defines a *learning to learn* research perspective. It thus operates on a higher level compared to mainstream machine learning [Liu 2021]. Formally, while ML algorithms primarily handle a specific task (i.e., a dataset), meta-learning is concerned with learning and transferring knowledge across tasks. Task and dataset will be used interchangeably in this chapter. Meta-learning paves the way toward continual, a.k.a. lifelong learning for AI agents. It is also very relevant to AutoML systems as it typically yields a better initialization of the AutoML search, and enforces the transfer of knowledge across different tasks.

The ultimate goal of meta-learning is to be capable of learning and adapting itself to a sequence of tasks (possibly but not necessarily related), like a human being.

Meta-learning clearly is among the most challenging tasks faced by Machine Learning. In our opinion, a critical difficulty comes from the lack of formalization and tools for representing ML tasks. While an ML task consists of a dataset sampled from some distribution on a feature and label space, the design of a rigorous representation and a reliable and tractable similarity function, enabling tasks comparison, is still an open problem. Note that tasks usually involve different input and output dimensions.[1] Indeed, in order to share knowledge between two

[1] Even in the case of distributions defined on spaces of same dimensions, distances among distributions such as the Kullback Leibler divergence or the optimal transport raise issues related to the ill-definedness of KL divergence in the general case, or the computational cost of optimal transport [Ganin et al. 2016].

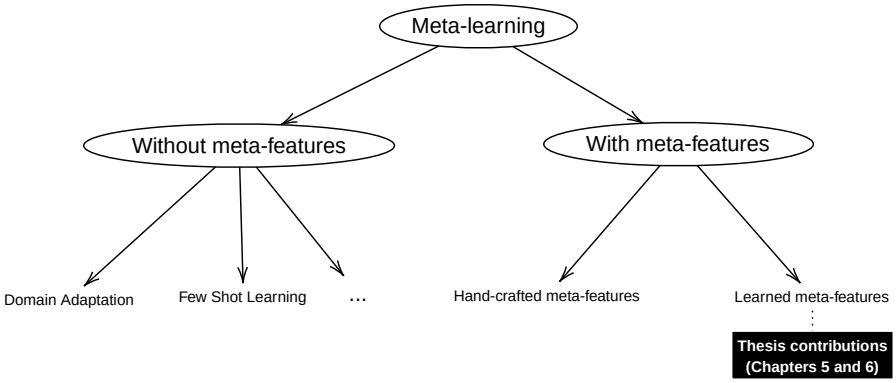


Figure 3.1: Overview of meta-learning approaches, inspired from Vanschoren [2019].

tasks, it is most desirable to assess their similarity. A key challenge of meta-learning thus is to propose such a representation, a similarity or metric, to leverage knowledge from previous tasks and allow for the lifelong learning of the agent.

Most generally, Meta-Learning is a crucial issue for the principled deployment of AutoML systems. HPO is a black-box, expensive, and hard optimization problem, as discussed and illustrated in the previous chapter. According to a few optimization papers [Glorot and Bengio 2010, Kazimipour et al. 2013], the success of the search and the ability to find reasonable solutions critically depend on the initialization of the search. In practice, the initialization of the HPO search is often addressed along with simple heuristics: selecting a set of generally promising configurations, based on the AutoML archive, then running them on the current task to initialize the performance model (more in Section 3.2.2). Despite its simplicity, this approach can significantly boost the performance of Auto-Sklearn [Feurer et al. 2015a] compared to a naive or random initialization.

The meta-learning application abovementioned relies on the transfer of knowledge among tasks. For this reason, a fundamental issue for meta-learning and efficient transfer is to build (formally define or learn) a similarity measure among the said tasks. The simplest way to define such similarity is to embed the space of instance tasks $\mathcal{I}$ (the set of datasets; see Figure 1.1) into a metric space. Formally, a set of d descriptors named meta-features is defined as computable functions from $\mathcal{I}$ onto $\mathbb{R}$. The embedding defined from $\mathcal{I}$ onto $\mathbb{R}^d$ thus is used, setting the (dis)similarity of two tasks as the Euclidean distance of their images in $\mathbb{R}^d$. Two of the three contributions of this thesis are concerned with designing new meta-features, supporting the meta-learning facets of AutoML.

3.2 . Literature Review

A survey by Vanschoren [2019] structures the meta-learning domain in three directions. The first research direction focuses on meta-learning based on leveraging hyper-parameter performances on a benchmark. The second research direction concerns learning to adapt a model learned from an early task to a new task. The last research direction aims to exploit dataset meta-features in order to share knowledge across tasks, e.g., based on the similarity defined from the meta-features. Indeed, these research directions are complementary, and some AutoML systems might benefit from a combination of such approaches.

For the sake of consistency with the rest of the document, the state-of-the-art presented below is structured into two categories: meta-learning without meta-features and meta-learning with dataset meta-features. Figure 3.1 provides a high-level overview of both research directions.

3.2.1 . Meta-learning without meta-features

Meta-learning without meta-features can proceed in various ways. Most approaches proceed by leveraging the performance of hyper-parameters to identify similar tasks, based on the intuition that if hyper-parameters tend to behave similarly on two tasks, then these tasks are similar. In other words, the hyper-parameter performances are used *in lieu* of meta-features. This similarity is leveraged after the task similarity (based on the hyper-parameter performance) is identified. The HPO problem on the considered task can benefit from all information attached to the neighbor tasks.

This strategy is followed in Collaborative Filtering based algorithm recommendations, illustrated by Misir and Sebag [2017]. Formally, given an extensive archive reporting the hyper-parameter performances on a dataset benchmark, matrix decomposition is used to extract a latent representation of each dataset. This latent representation is used to define a similarity, and ultimately the optimal hyper-parameters for the similar tasks are recommended to the current task. Note that it is an iterative process since the representation of the dataset changes as new hyper-parameters are evaluated (and matrix decomposition is achieved anew). Following Misir and Sebag [2017], OBOE [Yang et al. 2019] incorporates additional constraints on the optimization to encourage the recommendation of a cheap and informative hyper-parameter at the beginning, yielding a better performance model with reduced computational complexity. Along the same lines, Fusi et al. [2018] use a probabilistic version of matrix factorization.

Other approaches are based on surrogate models, where each dataset is associated with a surrogate model predicting the configuration performance. Along the same lines as above, two datasets are expected to be similar if and only if their surrogate models are similar. Feurer et al. [2021b], Wistuba et al. [2018] proceed as follows. Each known dataset is associated with a surrogate model expressed as a Gaussian Process [Rasmussen and Williams 2005]. Then, another surrogate model for the current (test) dataset is constructed and updated along with the HPO iterations. Importantly, the final surrogate model of interest is the ensemble

of all surrogate models (associated to known and current datasets), where each model is weighted according to its similarity with the surrogate model of the test task.

Yet another approach is based on multi-task learning. Swersky et al. [2013] and Springenberg et al. [2016] simultaneously tackle multiple *similar* tasks and share the related information. In practice, they consider surrogate models that can handle multiple tasks, namely Swersky et al. [2013] use multi-task Gaussian Processes, whereas Springenberg et al. [2016] use Bayesian Neural Networks. The main requirement for this approach is that the considered tasks must be sufficiently similar after the user's expertise.

Another meta-learning approach relies on a global and static HPO strategy for all tasks, determined by mining existing performance databases. In other words, one aims to determine the configuration with the best performance expectation over all tasks; the specifics of the current task are not considered. A simple strategy is to rank all stored hyper-parameters according to some criterion, and recommend the top-ranked hyper-parameters for any new dataset. The performance of this strategy depends on the considered criterion. For instance, Abdulrahman et al. [2018] introduce a criterion to account for accuracy and runtime, allowing an important speed up. Instead of recommending the same set of hyper-parameters for all tasks, van Rijn and Hutter [2018] leveraged 25.000 OpenML experiments to identify important hyper-parameters and propose a prior distribution on each of them. Similarly, [Pfisterer et al. 2021, Rijn et al. 2018] propose to learn default values of the hyper-parameters by exploiting the OpenML database.

Last but not least, a hot Meta-learning trend is based on learning from prior models. The goal of Transfer Learning is to adapt a model learned from a source task into a model suited to a target task [Pan and Yang 2010]. This topic gains some momentum, particularly in the field of deep neural networks, targeting the adaptation to new (and rare) classes, referred to as Few Shot Learning [Finn et al. 2017, Snell et al. 2017, Doersch et al. 2020].

3.2.2 . Meta-learning with dataset meta-features

As already said, a meta-feature is a function or a computable procedure associating a real value to a dataset. A set of d meta-features thus defines a vectorial representation in $\mathbb{R}^d$ characterizing every dataset. A detailed discussion about the main meta-features in the literature is presented in Section 3.3.1. A recent literature review of meta-features for machine learning is also presented in Rivolli et al. [2022].

By embedding the set of datasets into the metric space $\mathbb{R}^d$, meta-features naturally induce a metric on the dataset space. Indeed, some abovementioned approaches (Section 3.2.1) also aim to define a metric or dissimilarity on the dataset space, using hyper-parameter performance as meta-features. The difference is that meta-features are supposed to be inexpensive compared to hyper-parameter performance. Some approaches combine both strategies [Fusi et al. 2018], using

both meta-features and hyper-parameter performance to define a dissimilarity on the dataset space.

Such a (dis)-similarity is used to support an HPO method, either during the initialization phase or during the optimization search.

More precisely, the similarity is used during the initialization phase to warm-start HPO algorithms. For instance, in AutoSkLearn [Feurer et al. 2015b], the authors initialize a Bayesian Optimization process as follows:

- In the sense of the Euclidean distance defined from their meta-features, the nearest neighbors of the current task are retrieved.
- For each nearest neighbor, its optimal hyper-parameter configuration θ is launched on the current task, and the associated performance $r(\theta)$ is stored;
- The surrogate model associated with the current task is initialized from the pairs $(\theta, r(\theta))$.

This surrogate model is used to warm-start AutoSkLearn in Feurer et al. [2015a], and to achieve cold-start in the recommendation approach proposed by Fusi et al. [2018] and Misir and Sebag [2017].

Meta-features can also be used to learn parameterized surrogate models. For instance, Klein et al. [2017] learn surrogate models parameterized from the hyper-parameter values *and* the dataset size, supporting a multi-fidelity approach (where the estimated performance depends on both the configuration and the number of samples in the dataset) and reducing the computational complexity. In Bardenet et al. [2013], a single global surrogate model is learned and parameterized from both hyper-parameter and meta-feature values, facilitating the sharing of information across tasks.

3.3 . Dataset Meta-features for Meta-learning

This section focuses on the meta-features *per se*, either hand-crafted by experts or learned from data. Two of our contributions lie in the field of meta-feature learning.

3.3.1 . Hand-crafted meta-features

This section reviews the principal categories of hand-crafted meta-features commonly used for AutoML. Unsurprisingly, a large amount of information can be extracted from the dataset; depending on how they are computed, the meta-features are divided into several categories, following Vanschoren [2019], Rivolli et al. [2019], Alcobaca et al. [2020].

Statistical meta-features include all descriptive statistics of the dataset: the number of examples/features/classes [Michie et al. 1994]; the ratio of

target classes [Lorena et al. 2019]; the number of categorical and numerical features [Engels and Theusinger 1998]; sparsity [Salama et al. 2013], and mean/variance/kurtosis coefficient of features [Ali and Smith-Miles 2006, Engels and Theusinger 1998].

Information-theoric meta-features include the relationship between the target variable and feature variables of the dataset, such as the average mutual information of each feature with the target variable [Kalousis and Hilario 2000, Castiello et al. 2005], and target class entropy [Michie et al. 1994].

Geometric-based meta-features capture the geometry of points (where a dataset is viewed as a set of points in Euclidean space), including the clustering of points [Vilalta 1999], the distribution of classes [Ho and Basu 2002], and the complexity of the classification [Peng et al. 2002, Lorena et al. 2019].

Landmark meta-features describe datasets by leveraging ML model performances [Bensusan and Giraud-Carrier 2000, Pfahringer et al. 2000]. For the sake of tractability, only inexpensive models are considered in general, including logistic regression, latent Dirichlet allocation, decision trees, and one-nearest neighbor algorithm.

Model-based meta-features cover all information that can be extracted from a trained ML model. For instance, they can refer to the size of branches in a decision tree algorithm, the importance of variables [Agresti 2002], or the impurity of trees in Random Forest [Bensusan et al. 2000].

3.3.2 . Learning dataset meta-features

A last strategy consists in learning meta-features from a benchmark (set of datasets with reported performances for quite a few hyper-parameter configurations each). As said, the contributions presented in Chapters 5 and 6 fall in this category.

To our best knowledge, this strategy was less explored, mainly in the case of tabular datasets, due to its complexity and the shortage of (meta)-data. On the one hand, learning (meta)-features requires a sufficient number of (meta)-samples, here datasets. However, the largest *curated* dataset benchmark OpenML CC-18 [Bischl et al. 2017], yielding 72 binary and multi-class classification datasets, is insufficient for a learning purpose. Indeed the complete OpenML benchmark includes a few thousand datasets; unfortunately, many of those are deprecated versions of others, and some are too limited (e.g., involving a single feature).

On the other hand, the learning setup relevant to learning meta-features is still far from being clearly formalized. In vague terms, meta-features are good if and only if they efficiently support an AutoML process.

The state-of-the-art currently includes two approaches for learning meta-features from a benchmark. In Sun and Pfahringer [2013], meta-features are learned to estimate whether a given algorithm A outperforms an algorithm B, for A and B ranging in a set of landmark models. The meta-feature associated to each pair (A, B) is learned as a decision tree based on the hand-crafted meta-features.

In Jomaa et al. [2021], Kim et al. [2018a], the sought meta-features are learned by training neural networks. In [Kim et al. 2018b], a Siamese network is trained on the top of the hand-crafted meta-features, where the loss is defined by requiring that the learned meta-features of two datasets are similar if their top configurations are similar. In Jomaa et al. [2021], neural networks taking sets of samples as input are considered. Formally, the sought NN is trained to determine whether two patches of data (each defined by a subset of features and samples) are extracted from the same dataset; the meta-features are defined as the nodes in the last NN layer. Note that the meta-features thus do not take into account the hyper-parameter performances on each dataset.

The contribution presented in Chapter 5 takes inspiration from Jomaa et al. [2021], with two main extensions. The first extension consists in setting the meta-feature learning problem in the rigorously defined framework of distributional neural networks [De Bie et al. 2019]. The second extension regards the goal (learning loss) considered to learn the meta-features.

As will be discussed, the main limitation of this contribution regards the shortage of benchmark data used to train the distributional NN. Hence, several (meta)-data augmentation are considered; still, NN training requires a sufficient amount of information that is hardly available in the context of AutoML.

The second contribution, presented in Chapter 6 addresses the above limitation by restricting the search space for the learned meta-features, and only considering linear combinations of the hand-crafted meta-features. As will be experimentally shown, this restriction not only results in significant improvements but also sheds some light on the relevance of hand-crafted meta-features in the context of a particular learning algorithm.

Part II

Hyper-Parameter Optimization

4 - Automated Machine Learning with Monte-Carlo Tree Search

This chapter describes the first contribution of this thesis. As said (Section 1.3), AutoML tackles a black-box, structured and expensive optimization problem (Equation 1.1). Our first contribution focuses on the Hyper-Parameter Optimization problem involved in AutoML. The mixed (structured and parametric) optimization is handled using a hybrid HPO approach, mixing Monte Carlo Tree Search (MCTS), to handle the structured aspects, and Bayesian Optimization (to be sample efficient). The resulting approach is dubbed *Monte-Carlo Tree Search for Algorithm Configuration* (Mosaic).

The chapter is organized as follows. Section 4.1 first discusses the position of the problem and advocates the use of a hybrid approach. Section 4.2 introduces the formal background and presents MCTS, for the sake of self-containedness. Section 4.3 gives a detailed overview of the proposed Mosaic approach. The experimental setting and the goals of experiments are presented in Section 4.4. Finally, Section 4.5 reports on the empirical validation of Mosaic on the OpenML benchmark suite and the Scikit-learn portfolio.

4.1 . Position of the problem

A key difficulty of the AutoML optimization problem lies in the structure of the search space: an ML pipeline is a series of selected modules or components (algorithms), and a vector of hyper-parameters (possibly of varying dimension) is attached to each component. The AutoML task thus combines a combinatorial optimization problem (selecting the components of the pipeline structure) and a parametric optimization problem (optimizing the hyper-parameters of each selected component). The nature of the former optimization problem (finding pipeline structure) is arguably very different from the latter one (tuning hyper-parameters). This suggests that an algorithm best suited to structure optimization may be less efficient to achieve hyper-parameter tuning, and vice-versa.

Note that most AutoML approaches tackle both problems using a single optimization approach technique (CASH, Section 2.2). The originality of Mosaic is to use specific optimization approaches, one for each problem, and to tightly couple them (Section 4.3).

Formally, the combinatorial optimization of the pipeline structure is tackled as a sequential decision process, and Monte-Carlo Tree Search (MCTS) [Kocsis and Szepesvári 2006] is adapted to solve this sequential problem efficiently. On the other hand, the celebrated Bayesian optimization (BO) approach [Mockus 1989] efficiently handles expensive black box optimization problems, and it has been used

in particular in the context of hyper-parameter tuning [Hutter et al. 2011, Bergstra et al. 2011, Bardenet et al. 2013, Swersky et al. 2014].

Taking the best of both worlds, Mosaic combines MCTS and BO to tackle the AutoML problem efficiently. The coupling of both approaches is enforced as MCTS and BO share a single surrogate performance model, used to guide the BO search for the hyper-parameter optimization and the MCTS search for the pipeline structure.

4.2 . Formal background

After formalizing AutoML as a sequential optimization problem, this section presents the Monte-Carlo Tree Algorithm [Kocsis and Szepesvári 2006] for the sake of self-containedness. Its adaptation to the context of the *per-instance* AutoML problem is last described.

4.2.1 . AutoML as a Sequential Decision Problem

Following the CASH formalization (Section 2.2), the search space for ℓ-size ML pipelines is noted $X = \bigcup_{(a_1,\ldots a_\ell)\in\mathcal{A}_1\times\ldots\times\mathcal{A}_\ell}(a_1,\Theta(a_1))\times\ldots\times(a_\ell,\Theta(a_\ell))$, where: $\mathcal{A}_i$ is the finite set of i-th pipeline components, and $\Theta(a_i)$ is the hyper-parameters space of component a_i. Algorithm and component are used interchangeably in the remainder of the chapter.

The straightforward formalization of AutoML as a sequential decision problem consists of considering the sequence of decisions, selecting each pipeline component and its hyper-parameters according to a fixed ordered sequence of ℓ decisions. Examples of such decisions include the choice of the data pre-processing, feature selection, and learning algorithms.

A k-pipeline structure (k-ps) is a k-tuple $\mathbf{s} = (a_1,\ldots a_k) \in \mathcal{A}_1\times\ldots\times\mathcal{A}_k$, with $k \leq \ell$. Given a k-ps $\mathbf{s}$, any $\mathbf{x}\in X$ with same first k decisions as $\mathbf{s}$ is said to be compatible with $\mathbf{x}$ (noted $\mathbf{s}\preccurlyeq\mathbf{x}$) and the subset of pipelines compatible with $\mathbf{s}$ is noted $X(\mathbf{s}) = \{\mathbf{x}\in X; \mathbf{s}\preccurlyeq\mathbf{x}\}$.

A default distribution $\mathcal{D}$ is defined on X, involving a uniform distribution on all $\mathcal{A}_i$ and, conditionally to the selected a_i, uniform distribution on the (bounded) $\Theta(a_i)$. The default distribution on $X(\mathbf{s})$ is defined similarly.

4.2.2 . Monte-Carlo Tree Search

Monte-Carlo Tree Search (MCTS) is a tree-structured extension of the multi-armed bandit (MAB) algorithm [Lattimore and Szepesvári 2020]. As said (Section 2.1), a MAB is concerned with single state decision problems, aimed to select the action among a finite set of action that gets the best reward in expectation. In contrast, MCTS handles sequential decision problems, where a sequence of decisions is needed to get a reward. In other words, MCTS is best suitable for planning problems or games, as was amply demonstrated for the game of Go [Silver et al. 2016].

MCTS handles sequential decision making along a tree-structured approach, where each decision at any step is managed by a MAB algorithm. The relations among actions are expressed through the tree structure. Each (completed) tree path corresponds to a solution, list of decisions and is associated with a reward (or feedback score). The final outcome of MCTS is an optimal path of the tree space, representing the optimal sequence of decisions.

In most cases (e.g. the game of Go) the tree-structured space X has a high branching factor; thus, considering an exhaustive search strategy is intractable. Therefore, instead of brute-forcing, MCTS iteratively explores the tree space while gradually biasing the exploration toward the most promising regions of the search space. Formally, MCTS iteratively proceeds as follows. Each iteration, corresponding to a tree-walk (Figure 4.1), involves four phases [Gelly and Silver 2007]:

Down the MCTS tree: The first phase traverses the MCTS tree from the root node. In each (non-leaf) node $\mathbf{s}$ of the tree, the next node $\mathbf{s}.a$ to visit is selected among the child nodes of $\mathbf{s}$ classically using the multi-armed bandit Upper Confidence Bound criterion [Kocsis and Szepesvári 2006]:

$$\text{select } \arg\max_a \left\{ \hat{\mu}_{\mathbf{s}.a} + C_{ucb} \sqrt{\frac{\log n(\mathbf{s})}{n(\mathbf{s}.a)}} \right\} \tag{4.1}$$

with $\hat{\mu}_{\mathbf{s}.a}$ the average reward gathered over all tree-walks with prefix $\mathbf{s}.a$, $n(\mathbf{s})$ (resp. $n(\mathbf{s}.a)$) the number of visits to node $\mathbf{s}$ (resp. node $\mathbf{s}.a$), and C_{ucb} a problem-dependent constant that controls the exploitation *vs* exploration trade-off;

Expansion: When arriving at a leaf node, a new child node is added. The choice of the new node can be guided using, e.g., *Rapid Action Value Estimate* [Gelly and Silver 2011]. Following the Progressive Widening strategy [Couëtoux et al. 2011], the number of considered options is gradually extended with the number of visits n_i to the current node. Formally, when the integer value of $n(s,a)^{PW}$ is incremented, a new value is considered, with PW the coefficient of progressive widening (usually 1/2).

Playout: After the expansion phase, a playout strategy is used to complete the tree-walk until reaching a terminal node and computing the associated reward. A simple playout strategy is to choose the remaining nodes uniformly.

Back-propagation: The reward is back-propagated along the current path, incrementing $n(\mathbf{s})$ for all visited nodes and updating the value of each node s, noted $\hat{\mu}_s$, accordingly.

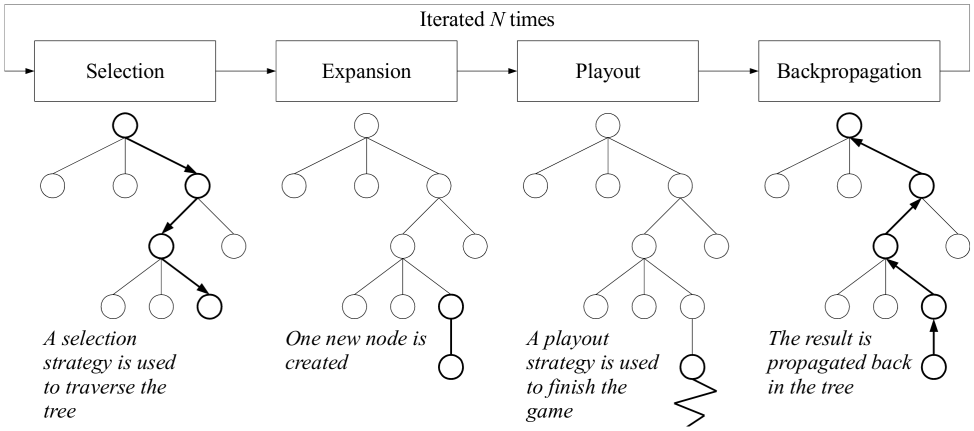


Figure 4.1: Monte-Carlo Tree Search: each iteration includes four phases, from [Chaslot et al. 2008]

4.2.3 . Per-instance AutoML

This section briefly situates the main AutoML components that will be handled in the MCTS search.

Surrogate model-based optimization (SMBO). As said (Section 2.1), most approaches rely on a *surrogate model* $\widehat{\mathcal{F}}$ of the objective function $\mathcal{F}$, iteratively exploiting $\widehat{\mathcal{F}}$ to make a decision and updating $\widehat{\mathcal{F}}$ on the basis of the current reward. At step t, surrogate model $\widehat{\mathcal{F}}_t : X \mapsto \mathbb{R}$ is learned from the set $\{(\mathbf{x}_u, \mathcal{F}(\mathbf{x}_u)), u = 1 \ldots t\}$ gathering the previously selected pipelines and their associated performances.

Surrogate models are often exploited along *Bayesian optimization* (BO) [Mockus 1989]. Formally, if model $\widehat{\mathcal{F}}_t$ yields an estimate of the performance for any given $\mathbf{x}$ and the confidence of this estimate, the most promising $\mathbf{x}^*_{t+1}$ is determined by maximizing the acquisition function, e.g., Expected Improvement (EI) [Jones et al. 1998] compared to the current best value $\mathcal{F}(\mathbf{x}^*_t)$.

The main difficulty lies in the structure of space X. In all generality, this space includes categorical variables (e.g., the name of the pre-processing or ML algorithms) and continuous or integer variables, the number and range of which depend on the value of the categorical variables (e.g., the hyper-parameters of the retained algorithms). Diverse surrogate model hypothesis spaces were considered to cope with the structured of the search space: *Sequential Model-based Algorithm Configuration* (SMAC) [Hutter et al. 2011] uses Random Forests [Breiman 2001]; [Bergstra et al. 2011] use a *Tree-structure Parzen Estimator*.

Another issue is the distribution used to sample the configuration space to optimize the acquisition function. For instance, Auto-Sklearn, as it uses SMAC, considers a small number of configurations close to the best-so-far pipelines, augmented with a large number of uniformly sampled pipelines.

Search initialization Several approaches are used to address the initialization of the search process, long known to be critical for ill-posed optimization problems [Glorot and Bengio 2010, Kazimipour et al. 2013]. Such approaches include Meta-learning strategies (Chapter 3), leveraging knowledge from similar previous tasks and selecting the initial candidates $\mathbf{x}_u$'s as the best configurations for these previous tasks.

Specifically regarding SMBO, what matters is the accuracy of the surrogate model *in the worth part of the search space*; this accuracy is governed by the selection of the $\mathbf{x}_u$'s. In Auto-Sklearn.MetaLearning for instance, the $\mathbf{x}_u$'s are selected based on an archive $\{(\mathbf{z}_i, \mathbf{x}_i)\}$ where the meta-feature vector $\mathbf{z}_i$ describes the i-th dataset and $\mathbf{x}_i$ is the best-known pipeline for this dataset. Letting $\mathbf{z}$ denote the meta-feature vector associated with the current dataset, its nearest neighbors in the archive (in the sense of the Euclidean distance on the meta-feature vector space) are computed, and the $\mathbf{x}_i$s associated with these neighbors are used by Auto-Sklearn as first configurations [Feurer et al. 2015a].

Model ensembling The merits of ensemble learning are long known in terms of accuracy and robustness. Along this line, an ensemble of ML models is often used in AutoML instead of a single model, taking advantage of the fact that the sequence of solutions found by an AutoML process can be exploited in the spirit of ensemble learning. [Caruana et al. 2004] propose a simple and efficient procedure to compute an optimal ensemble from a set of models. This approach is adapted to the AutoML context as follows.

Starting with an empty set S, iterate over the pipelines and add it into S if only if the weighted sum of the pipelines improves the validation score. Then, leveraging the same strategy, Auto-Sklearn.Ensemble iteratively recomputes the optimal ensemble each time a new configuration is launched, yielding a new model.

4.3 . MCTS-aided Algorithm Configuration

This section details how Mosaic tackles the combinatorial and the parametric optimization problems at the core of AutoML, respectively concerned with the selection of the algorithms in the pipeline, $\mathbf{a} \in \mathcal{A}$, and the tuning of their hyper-parameters, $\theta(a_i) \in \Theta(a_i)$ for each algorithm a_i in $\mathbf{a}$.

4.3.1 . Two intertwined optimization problems

Along the mainstream CASH formalization (Section 2.2), the difficulty comes from the fact that the abovementioned optimization problems do not have the same nature and search spaces.[1] However, handling them in a separate way raises

[1] Furthermore, the optimization of $\theta(a_i)$ is of varying dimension, possibly depending on the value of some coordinates in $\theta(a_i)$, e.g. the number of neural layers controls the dimension of the neural layer size.

a key issue: The optimization objective is non-separable. Formally, the marginal performance of a_j depends on all other $a_k, k \neq j$ and on $\theta(\mathbf{a})$. Likewise, the marginal performance of $\theta(a_j)$ depends on all a_k and $\theta(a_k)$ for $k \neq j$.

The naive approaches, e.g. optimizing $\theta(\mathbf{a})$ for every considered $\mathbf{a}$, or estimating the performance of $\mathbf{a}$ from a few samples of $\theta(\mathbf{a})$, are intractable for computational reasons.

Mosaic addresses this challenge along an original hybrid approach, tackling both structural and parametric optimization problems using two coupled strategies. MCTS is used to tackle the structural optimization of $\mathbf{a}$ and Bayesian optimization is used to tackle the parametric optimization of $\theta(\mathbf{a})$. The coupling of MCTS and BO is achieved as they both rely a single surrogate model $\widehat{\mathcal{F}}$ on the overall pipeline space X, learned and maintained using all computed performances $\mathcal{F}(\mathbf{x}_u = (\mathbf{a}_u, \theta(\mathbf{a}_u)))$, with $\mathcal{F}$ be the true performance function.

The difference between Mosaic and Auto-Sklearn (respectively most other AutoML approaches) is that the combinatorial optimization part in Mosaic is based on MCTS as opposed to BO (resp., their own optimization methods).

4.3.2 . Partial surrogate models

This subsection details the surrogate models involved in Mosaic.

Regarding the combinatorial optimization of the pipeline structure with MCTS, the difficulty is to estimate the performance of an incomplete structural pipeline $\mathbf{s}$, where only part of the modules are selected. This partial approximate performance is estimated through a surrogate performance model $Q_{\widehat{F}}$, which is derived from $\widehat{\mathcal{F}}$. The $Q_{\widehat{F}}$ performance is used during the expansion and play-out steps, allowing the selection of promising pipelines to guide the completion of $\mathbf{s}$. Finally, during the back-propagation step, the true performance of the evaluated pipeline is used to refine the value of the tree-walk $\mathbf{s}$.

For $k < \ell$, let $\mathbf{s}$ be a k-ps, and let $\mathbf{s}.a$ denote the $(k+1)$-ps built from $\mathbf{s}$ by selecting a as $(k+1)$-th decision. Then the surrogate $Q_{\widehat{F}}$ is defined as:

$$Q_{\widehat{F}}(\mathbf{s}, a) = \mathbb{E}_{\mathbf{x} \sim \mathcal{D}[X(s.a)]}\left(\widehat{\mathcal{F}}(\mathbf{x})\right) \approx \frac{1}{n_s}\sum_{j=1}^{n_s} \widehat{\mathcal{F}}(\mathbf{x}_j) \tag{4.2}$$

estimated from a number n_s ($n_s = 100$ in the experiments) of configurations sampled in $X(\mathbf{s}.a)$.[2]

A probabilistic selection policy π can then be built from $Q_{\widehat{F}}$, with:

$$\pi(a|s) = \frac{\exp\left(Q_{\widehat{F}}(s,a)\right)}{\sum_{b \in \mathcal{A}_k} \exp\left(Q_{\widehat{F}}(s,b)\right)} \tag{4.3}$$

Taking inspiration from Silver et al. [2016], this policy is used to enhance the MCTS selection rule (below).

[2]Note that, since the purpose of $Q_{\widehat{F}}(\mathbf{s}, a)$ is to estimate the importance of an algorithm a, other alternative approaches Hutter et al. [2014], van Rijn and Hutter [2018] can be adopted to compute this estimate more efficiently.

4.3.3 . The Mosaic algorithm

The Mosaic algorithm is described in Alg. 2, following the general MCTS scheme (Section 4.2.2), where the main four phases are modified as follows.

Down the MCTS tree (selection) In a non-leaf node $\mathbf{s}$ of the MCTS tree, with $\mathbf{s}$ a k-ps, the child node a is selected in $\mathcal{A}_k$ using the AlphaGo Zero criterion:

$$\underset{a}{\operatorname{argmax}} \left(\overline{Q}(\mathbf{s}, a) + C_{ucb} * \pi(a|\mathbf{s}) * \frac{\sqrt{n(\mathbf{s})}}{1 + n(\mathbf{s}.a)} \right) \tag{4.4}$$

where $\overline{Q}$ is the median[3] of $\mathcal{F}(\mathbf{x})$ for all $\mathbf{x}$ in $X(\mathbf{s}.a)$, $\pi(a|\mathbf{s})$ is defined by Eq.(4.3), $n(\mathbf{s})$ is the number of times s was visited, and C_{ucb} is the usual constant controlling the exploration *vs* exploitation trade-off (with default value .6).

Expansion In a leaf node $\mathbf{s}$ of the MCTS tree, with $\mathbf{s}$ a k-ps, the child node a in $\mathcal{A}_k$ that maximizes the surrogate performance $Q_{\widehat{F}}(\mathbf{s}, a)$ is added to the MCTS tree.

Playout Letting $\mathbf{s}$ be the (possibly complete) k-ps, a full pipeline $\mathbf{x}$ with $\mathbf{s} \preccurlyeq \mathbf{x}$ is defined using a sampling playout strategy. Three sampling strategies are considered:

A a configuration is sampled according to the default distribution $\mathcal{D}(X(\mathbf{s}))$;

B a local search around the best recorded pipeline $(\mathbf{a}^*, \theta^*)$ in $X(\mathbf{s})$ is achieved and the best configuration according to $\widehat{\mathcal{F}}$ is retained;

C a number of configurations is sampled after $\mathcal{D}(X(\mathbf{s}))$ in $X(\mathbf{s})$, together with a few configurations sampled via a local search around $(\mathbf{a}^*, \theta^*)$, and the sample $\mathbf{x}$ that maximizes the Expected Improvement of $\widehat{\mathcal{F}}$ is retained. This strategy is similar as in SMAC [Hutter et al. 2011].

In all cases, the true performance $\mathcal{F}(\mathbf{x})$ of the retained configuration is computed.

Early experiments were conducted to assess these strategies, showing that: strategy A is slow and prone to overfitting; strategy B causes a loss of diversity of the considered pipelines, eventually resulting in a poor surrogate performance model $\widehat{\mathcal{F}}$. Hence only the third strategy C is considered thereafter: the sampled configurations include n_r ($n_r = 1,000$ in the experiments) configurations sampled from default distribution $\mathcal{D}(X(\mathbf{s}))$, augmented with pipelines nearest[4] to $(\mathbf{a}^*, \theta^*)$.

[3] The average was also considered, giving very similar results, except in rare cases of heavily failed runs.

[4] Formally, one selects every $(\mathbf{a}', \theta')$ such that either $\mathbf{a}' = \mathbf{a}^*$ and θ' differs from θ^* by a single hyper-parameter value; or $\mathbf{a}'$ differs from $\mathbf{a}^*$ by a single decision and θ' is the default hyper-parameter vector $\theta(\mathbf{a}')$.

Algorithm 2: Mosaic Vanilla

Procedure `Selection(`$\mathbf{s}$`)`
  input : Incomplete pipeline $\mathbf{s}$.
  let a_{last} be the last algorithm in $\mathbf{s}$
  while a_{last} *is a non-leaf node of the MCTS tree* **do**
    $a \leftarrow$ Select child node of a_{last} using Equation 4.4
    return `Selection(`$\mathbf{s}.a$`)`
  end
  return $\mathbf{s}$

Procedure `Expansion(`$\mathbf{s}$`)`
  input : Incomplete pipeline $\mathbf{s}$.
  return $\operatorname{argmax}_a Q_{\hat{F}}(\mathbf{s}, a)$

Procedure `Playout(`$\mathbf{s}$`)`
  input : Incomplete pipeline $\mathbf{s}$.
  let S be a set of complete pipelines drawed from $\mathcal{D}[X(\mathbf{s})]$
  `/* Neighbors(`$\mathbf{x}$`) outputs the neighbors of pipeline` $\mathbf{x}$. `*/`
  $N \leftarrow$ `Neighbors(`$\mathbf{x}^*_{\mathbf{S}}$`)`, with $\mathbf{x}^*_{\mathbf{S}} \in X(\mathbf{s})$ best pipeline seen so far compatible with $\mathbf{s}$
  return $\operatorname{argmax}_{x \in S \cup N}$ `EI(`x`)`

Procedure `Mosaic(`T, d`)`
  input : Number of iterations T, dataset d.
  for t *in* $\{1..T\}$ **do**
    $\mathbf{s} \leftarrow$ `Selection(`$\emptyset$`)`
    $a \leftarrow$ `Expansion(`$\mathbf{s}$`)`
    $\mathbf{x} \leftarrow$ `Playout(`$\mathbf{s}.a$`)`
    Train pipeline $\mathbf{x}$ on dataset d and observe performance r
    $n(a) \leftarrow 1; \overline{Q}(\mathbf{s}, a) \leftarrow r$
    foreach $a \in ancestors(\mathbf{s})$ **do**
      Update $\overline{Q}$ at node a with r
      $n(a) \leftarrow n(a) + 1$
    end
  end

Back-propagation Performance $\mathcal{F}(\mathbf{x})$ is back-propagated up the tree along the current path, and the $\overline{Q}$ value attached to each node of the path is updated. Example $(\mathbf{x}, \mathcal{F}(\mathbf{x}))$ is added to the surrogate training set, and the surrogate performance model $\widehat{\mathcal{F}}$ is trained anew.

Stopping criterion The algorithm stops after the computational budget is exhausted (one hour per dataset in the experiments).

4.3.4 . Initialization and Variants

The order of the decisions in the structural pipeline is key to the optimization: while MCTS yields asymptotic optimality guarantees, the discovery of good decisions can be very significantly delayed due to poorly informative or unlucky starts [Coquelin and Munos 2007]. For this reason, the order of decisions in the structural pipeline is fixed once for all, with the first decision made at the root node of the tree being the choice of the learning algorithm (associated with a default complete pipeline).

Mosaic**.Vanilla** The initialization proceeds as follows: For each learning algorithm ($\mathbf{s} = (a)$ with $a \in \mathcal{A}_1$), its default complete pipeline is launched, together with κ ($= 3$ in the experiments) other pipelines sampled from $X(\mathbf{s})$, and their associated performances are computed. The initial surrogate model $\widehat{\mathcal{F}}$ is trained from the set of all such $(\mathbf{x}, \mathcal{F}(\mathbf{x}))$ and $Q_{\widehat{\mathcal{F}}}(\emptyset, a)$ is initialized for a in $\mathcal{A}_1$.

Mosaic**.MetaLearning** borrows Auto-Sklearn its better informed initialization, where the first 25 configurations are the best recorded ones for each of the nearest neighbors of the current dataset, in the sense of the meta-feature distance. The next configurations are selected as in Mosaic.Vanilla, and the actual search starts thereafter.

Mosaic**.Ensemble** is similar to Mosaic.Vanilla, but returns the compound model defined as a weighted sum of the models computed along the AutoML search [Caruana et al. 2004], using an online ensemble building strategy as in Feurer et al. [2015a].

4.4 . Experimental Setting

This section details and discusses the experiments conducted to validate Mosaic.

4.4.1 . Goals of experiment

The goal of experiments is two-fold: (i) to assess the efficiency of Mosaic compared to baselines; (ii) to investigate the relative impacts of Mosaic variants, and its sensitivity w.r.t. its own hyper-parameters.

Comparison w.r.t to baselines. The empirical validation of Mosaic firstly aims to assess its performance compared to Auto-Sklearn [Feurer et al. 2015a], that consistently dominated other systems in the international AutoML challenges [Guyon et al. 2015]. The other AutoML system used as baseline is the evolutionary optimization-based[5] TPOT (v0.9.5) [Olson et al. 2016].

Analysis of Mosaic **hyper-parameters and variants.** The second goal of experiments is to better understand the specifics of the AutoML optimization problem. A first issue regards the exploration *vs* exploitation trade-off on the structural *vs* parametric subspaces, and the respective merits of MCTS and Bayesian optimization on the structured space.
A second issue regards the impact of the MetaLearning initialization. MCTS is notorious to achieve a consistent though moderate exploration, which as said might slow down the search in case of unlucky early choices. A smart initialization procedure aims to mitigate such hazards.

4.4.2 . Experimental setting

Search space For the sake of a fair comparison, Auto-Sklearn and Mosaic are compared on the same AutoML search space, defined from the *scikit-learn* library [Pedregosa et al. 2011]. Both Auto-Sklearn and Mosaic search spaces involve 16 ML algorithms, 13 pre-processing methods, 2 categorical encoding strategies, 4 missing values imputation strategies, 6 rescaling strategies and 2 balancing strategies (Table A.1-A.4, Appendix A.1). The size of the structured search subspace is 6,048 (due to dependencies). The overall parametric search space has dimensionality 147 (93 categorical, 32 integer, and 47 continuous hyper-parameters), all managed through the ConfigSpace library [Lindauer et al. 2019]. Each hyper-parameter ranges in a bounded discrete or continuous domain. For each configuration $\mathbf{x} = (\mathbf{a}, \theta(\mathbf{a}))$, $\theta(\mathbf{a})$ involves a dozen scalar hyper-parameter on average.

Mosaic **hyper-parameters** Mosaic shares the hyper-parameters of SMAC (therefore Auto-Sklearn), and involves 3 additional hyper-parameters: the number n_s of samples to compute $Q_{\widehat{F}}$ (Equation 4.2, with default value $n_s = 100$), the C_{ucb} weight controlling the exploration *vs* exploitation tradeoff (Equation 4.4), with default value $C_{ucb} = 1.3$, and the coefficient of progressive widening PW controlling the branching factor of the MCTS tree, with default value $PW = 0.6$. The SMAC hyper-parameters (shared with Mosaic) include: the number n_r of uniformly sampled configurations, and variance $\epsilon = .2$ for the local search used to tune the acquisition function of the BO.

[5] AlphaD3M [Drori et al. 2019] and AutoStacker [Chen et al. 2018] could not be considered due to the lack of a public code.

Computational resources Computational runtimes are all measured on an AMD Athlon 64 X2, 5GB RAM.

Benchmark suite All considered AutoML systems are assessed on the OpenML 100 repository [Bischl et al. 2017], including 100 binary and multi-class classification problems (each with a training and a test sets). The overall computational budget is set to 1 hour for each dataset. For all systems, every considered configuration $\mathbf{x}$ is launched to learn a model from 70% of the training set with a cut-off time of 300 seconds, and performance $\mathcal{F}(\mathbf{x})$ is set to the model accuracy on the other 30%. After 1 hour, for each system the best configuration $\mathbf{x}^*$ is launched to learn a model on the whole training set and its performance on the (unseen) test set is reported. The system performance on this dataset consists of the performance (averaged over 10 independent runs) and its standard deviation.

For each dataset, the performances achieved by all systems are ranked (the lower the better). The main performance indicator associated to each system in the following is its average rank over all datasets.

As the rank indicator might be blurred when many systems and their variants are considered together, duels between pairs of systems (Mosaic.X against Auto-Sklearn.X, where X ranges in *Vanilla, Meta-Learning, Ensemble, Meta-Learning+Ensemble*, Section 4.3.4), are considered. The actual performance (accuracy) of the best confgurations will also be reported for a more in-depth discussion.

4.5 . Empirical Validation

4.5.1 . Comparison with baselines

For the sake of a fair comparison, the assessment is carried out separately for Mosaic vanilla and its variants.

Vanilla variants The comparative performances of Vanilla Auto-Sklearn, TPOT and Mosaic *vs* computational time are displayed on Figure 4.2 (see also Figure 4.4-a), showing that the hybrid optimization used in Mosaic clearly improves on the Bayesian optimisation-only used in Auto-Sklearn (and on the evolutionary optimization-only used in TPOT), for whichever computational resources.

A complementary perspective on the respective performances of Mosaic and Auto-Sklearn in terms of the predictive accuracy of the best configurations is displayed on Figure 4.3. According to a Mann-Whitney-Wilcoxon test with 95% confidence, and if considering the median performance, Mosaic significantly outperforms Auto-Sklearn on 21 datasets out of 100; Auto-Sklearn outperforms Mosaic on 6 datasets out of 100.

Mosaic improves on Auto-Sklearn on 35 other datasets (though not in a statistically significant way), and the reverse is true on 18 datasets. Both are equal

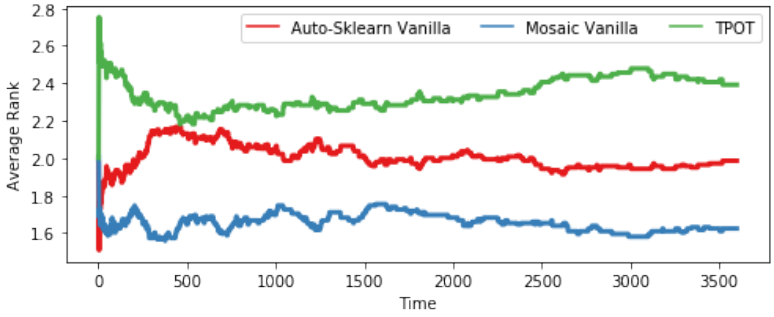


Figure 4.2: Average performance ranks (lower is better) on OpenML-100 *vs* CPU time of the Vanilla versions of Mosaic (bottom), Auto-Sklearn (middle), and TPOT (top). Better seen in color.

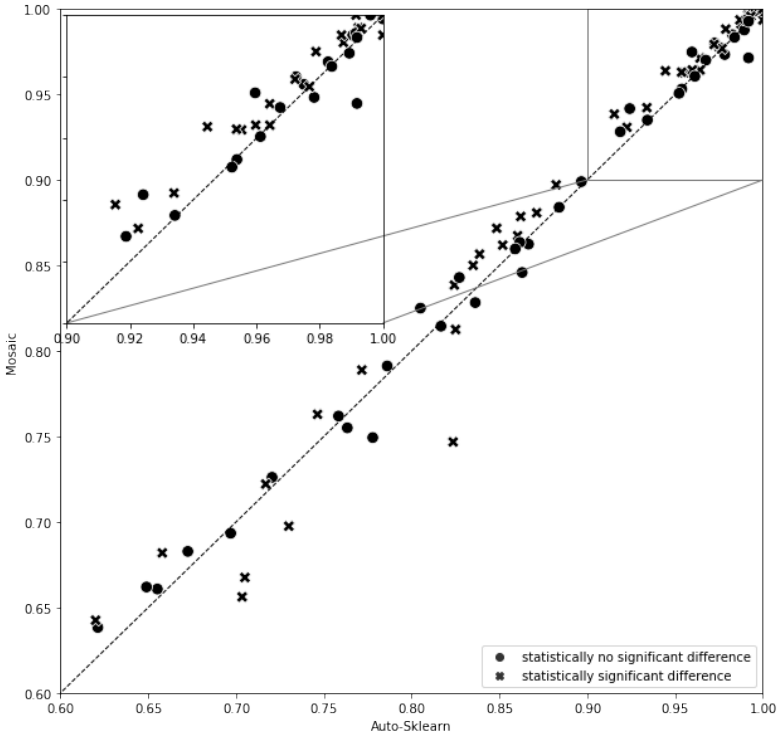


Figure 4.3: Performance of Mosaic (y-axis) versus Auto-Sklearn (x-axis) on OpenML-100. Datasets for which the difference is statistically significant (resp. insignificant) after Mann Whitney Wilcoxon test with confidence 5% are represented with a $\times$ (resp $\bullet$).

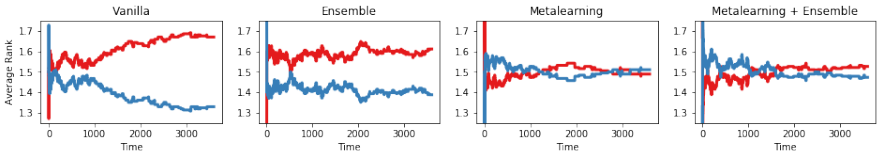


Figure 4.4: Average performance rank (lower is better) on OpenML-100 *vs* CPU time of the different variants of Mosaic (bottom curve on all plots but the *Metalearning*) and Auto-Sklearn.

	Stat. significant			Insignificant		
	Mc > As	Mc < As	Mc = As	Mc > As	Mc < As	Mc=As
Vanilla	21	6	10	35	18	8
Ensemble	11	12	6	38	16	15
MetaLearning	15	14	8	24	23	14
MetaL+Ens.	15	17	2	24	19	21

Table 4.1: Per dataset comparison statistics of the median performance between Mosaic and Auto-Sklearn variants, with Mann-Whitney-Wilcoxon test confidence level of 5% or not.

on 18 datasets and both systems crashed on 2 datasets.

MetaLearning and Ensemble variants The impacts of the MetaLearning and Ensemble heuristics are displayed on Figure 4.4. The difference noted for the Vanilla variants (with Mosaic mostly dominating Auto-Sklearn) is less visible for the Ensemble variants. For the MetaLearning variants and MetaLearning + Ensemble variants, the difference between Auto-Sklearn and Mosaic is no longer statistically significant.

A closer inspection of the results reveals that the best Auto-Sklearn configuration is nearly always among the initial ones: Auto-Sklearn.MetaLearning thus mostly explores the close neighborhood of the initially selected configurations. In the meanwhile, Mosaic more thorough exploration strategy entails a bigger risk of overfitting, discovering configurations with better performance on the validation set, at the expense of the performance on the test set.

For each variant (Vanilla, Ensemble, MetaLearning, and MetaLearning+Ensemble), Table 4.1 reports the number of datasets for which Mosaic outperforms Auto-Sklearn, and vice-versa, indicating whether the difference of performance is statistically significant in the sense of a Mann-Whitney-Wilcoxon test with confidence level 5% on the median performances.

4.5.2 . Assessment of Mosaic variants

Figure 4.5 displays the respective impacts of the Mosaic variants, showing the ranks of the Vanilla, MetaLearning, Ensemble and MetaLearning+Ensemble

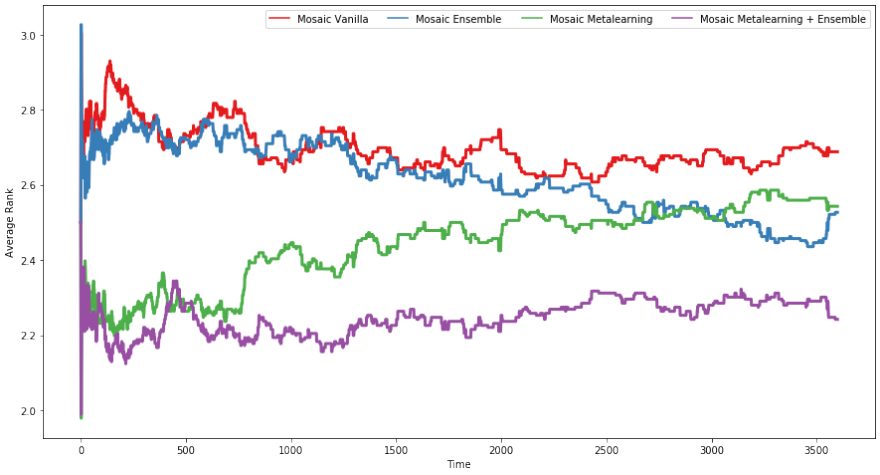


Figure 4.5: Average performance rank (lower is better) of Mosaic variants on OpenML-100.

performances versus time. The main improvement is due to the MetaLearning strategy, yielding a better initialization of the optimization process. Overall, the best variant is the one combining MetaLearning and Ensembling, although the Ensembling variant standalone yields a very moderate improvement on the Vanilla variant.

4.5.3 . Sensitivity of Mosaic hyper-parameter

Complementary experiments are conducted to assess the sensitivity of Mosaic.Vanilla w.r.t. its own hyper-parameters. For computational reasons, only 30 datasets out of 100 are considered, and Mosaic.Vanilla is run 5 times with one hour budget on each dataset.

Figure 5 displays the average rank of Mosaic.Vanilla at the end of the learning curve compared to Auto-Sklearn.Vanilla, for C_{ucb} ranging in $\{.1, .3, .6, 1, 1.3, 1.6\}$ and PW in $\{1, .8, .7, .6, .5\}$. Overall, Mosaic dominates Auto-Sklearn for 24 settings out of 30 (with a rank less than 1.5).

Likewise, the sensitivity w.r.t. hyper-parameter n_s is assessed for $C_{ucb} = 1.3$ and $PW = .6$. Figure 6 displays the average rank vs time of Mosaic.Vanilla for n_s ranging in 50, 100, 500, 1000, showing the low sensitivity of the approach w.r.t. n_s for these (representative) values of C_{ucb} and PW.

4.6 . Partial conclusion

The main contribution of this work is the new Mosaic scheme, tackling the AutoML optimization problem through handling the structural and the parametric optimization problems. The proposed approach is based on a novel coupling of Bayesian Optimization and MCTS strategies, sharing the same surrogate model.

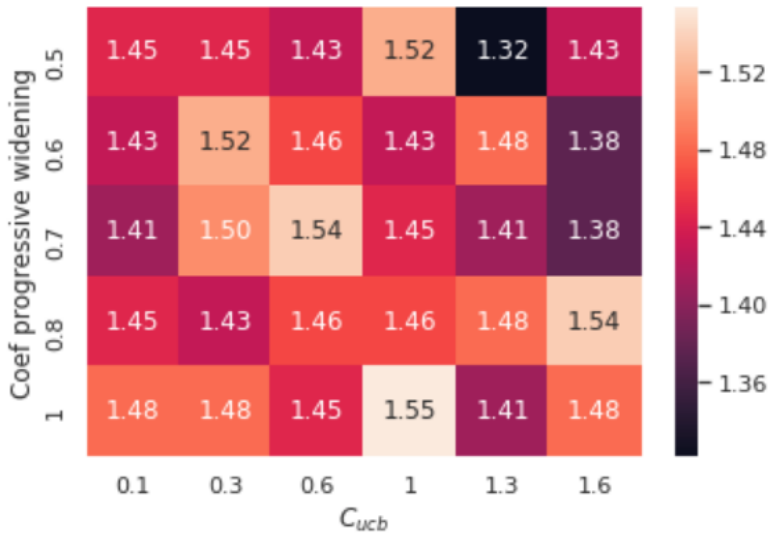


Figure 4.6: Sensitivity study w.r.t. C_{ucb} and coefficient of progressive widening (during expansion phase): Average rank of Mosaic.Vanilla (for $n_r = 100$) w.r.t. Auto-Sklearn.Vanilla (the lower, the better). Better seen in color (the darker the better).

Figure 4.7: Sensitivity study w.r.t. parameter n_s: Average rank of Mosaic.Vanilla (for $C_{ucb} = 1.3$ and $PW = 0.6$) w.r.t. Auto-Sklearn.Vanilla. Better seen in color (Mosaic in blue and Auto-Sklearn in red).

In MCTS the surrogate model is used: i) to estimate, in any node, the average performance of all pipelines below this node, and thus to choose the next node; ii) to choose the optimal hyper-parameters of the pipeline using Bayesian Optimization during the roll-outs.

The empirical validation of the approach demonstrates the merits of Mosaic compared to the challenge winner Auto-Sklearn on the OpenML benchmark suite, at least as long as the Vanilla and Ensemble variants are considered. With the MetaLearning variant however, the difference becomes insignificant as the bulk of optimization is achieved during the initialization phase.

Two lines of research emerge as a perspective for further works.

Better initialization with better meta-features The performance of the MetaLearning variant confirms the importance of the initialization strategy and thus motivates an in-depth study over the meta-features. Concretely, the question becomes that of learning better meta-features than the hand-crafted ones. As discussed (Chapter 3), the learning of meta-features faces critical difficulties (the shortage of dataset benchmarks, the unknown target metric); the next two chapters are devoted to tackling these difficulties.

Variable-length ML pipeline Another interesting line of research is on adapting Mosaic to account for variable-length pipelines [Wever et al. 2018b]. If the MCTS algorithm naturally copes with such a setting, a few limitations still need to be addressed. A formal definition of the variable-length search space is required to ensure that sampled pipelines are admissible (trainable). de Sá et al. [2017], Estevez-Velarde et al. [2019], Marinescu et al. [2021] propose to formalize the search space using grammars [Chomsky 1990]. In their work, Marinescu et al. [2021] compare a greedy exploration of the constructed grammar, dubbed PIPER, with Mosaic. Their results suggest that Mosaic consistently outperforms PIPER during the first hour of training, but then PIPER starts to slightly improve over Mosaic. Incorporating MCTS strategy to search over the pipeline grammar is thus a promising future work.

Another challenge is to adapt the surrogate models in Mosaic to handle the varying input dimension. A promising direction, taking inspiration of the Neural Architecture Search [Zoph and Le 2017], is to learn the surrogate model using recurrent NN, e.g. LSTM [Hochreiter and Schmidhuber 1997].

Part III

Learning Dataset Meta-Features

5 - Distribution-Based Invariant Deep Networks for Learning Meta-Features

The Part III of this manuscript is devoted to meta-learning, complementary to the optimization aspects of AutoML discussed in Part II. While a number of approaches, detailed in Chapter 3, have been deployed to tackle meta-learning, the presented research focuses on learning dataset meta-features. As a second contribution of the thesis, this chapter introduces the *Distribution-based Invariant Deep Architecture* framework (Dida), a neural architecture which operates at the dataset level.

The motivations and preliminaries are discussed in Section 5.1 and 5.2. Section 5.3 details the proposed Dida approach, followed by its theoretical analysis in Section 5.4.1. Finally, the empirical validation of the approach is discussed in Section 5.5.

5.1 . Motivation

At the core of this Part III, one aims to build representations of datasets through *learned meta-features*. Meta-features, meant to represent a dataset as a vector of characteristics, have been mentioned in the ML literature for over 40 years [Rivolli et al. 2022]. A large number of meta-features have been manually designed along the years (detailed in Section 3.3.1, Chapter 3).

However, there exists little evidence that these hand-crafted meta-features accurately capture the underlying joint distribution between datasets and ML performances. It is likely that the set of optimal meta-features depends on the AutoML task and ML algorithm at hand. For example, statistics-based meta-features (e.g. the information gain of a dataset feature), might be more relevant to learning Decision Trees than Support Vector Machines. For these reasons, previous works [Sun and Pfahringer 2013, Jomaa et al. 2021] attempt to learn new sets of meta-features either from scratch or on the top of the hand-crafted meta-features.

The second contribution of the manuscript, detailed in the present chapter, aims to learning dataset meta-features from scratch, that is by processing datasets as in Dataset2Vec [Jomaa et al. 2021]. The challenge is to devise an ML setting accommodating datasets as input while enforcing the invariance proprieties of meta-features w.r.t the features and rows permutations. The proposed Dida architecture addresses the aforementioned challenge using distributional neural nets, as will be detailed in Section 5.2.

5.2 . Preliminaries

Notations $[\![1; m]\!]$ denotes the set of integers $\{1, \dots m\}$. Distributions, including discrete distributions (datasets), are noted in bold font. Sets are noted in capital letters. Vectors are noted in italic, with $x[k]$ denoting the k-th coordinate of vector x.

5.2.1 . Invariant Neural Network architectures

Deep networks architectures, initially devised for structured data such as images and speech, are extended to enforce some invariance or equivariance properties (defined below) [Shawe-Taylor 1993] for more complex data representations. The merit of invariant or equivariant neural architectures is twofold. On the one hand, they inherit the universal approximation properties of neural nets [Cybenko 1989, Leshno et al. 1993]. On the other hand, the fact that these architectures comply with the invariances attached to the considered data representation yields more robust and more general models through constraining the neural weights and/or reducing the number of weights, as examplified by convolutional networks. For instance, when considering point clouds [Qi et al. 2017] or probability distributions [De Bie et al. 2019], the network output is required to be invariant with respect to permutations of the input points. Invariance and equivariance properties are both defined in Definition 3 and 4.

Definition 3. *(Invariance) Let $f : X \mapsto Y$ be a function with input (resp. output) domain X (resp. Y), and σ be an operator defined on X. The function f is said to be invariant under operator σ iff $f(\sigma(x)) = f(x)$ for all x in X.*

Definition 4. *(Equivariance) Let $f : X \mapsto X$ be a function on domain X, and σ be an operator defined on X. The function f is said to be equivariant under operator σ iff $f(\sigma(x)) = \sigma(f(x))$ for all x in X.*

Neural architectures enforcing invariance or equivariance properties were pioneered by [Qi et al. 2017, Zaheer et al. 2017] for learning from point clouds subject to permutation invariance or equivariance. These are extended to permutation equivariance across sets [Hartford et al. 2018]. Characterizations of invariance or equivariance under group actions are proposed in the finite [Ravanbakhsh et al. 2017] or infinite case [Kondor and Trivedi 2018].

On the theoretical side, [Maron et al. 2019, Keriven and Peyré 2019] propose a general characterization of linear layers enforcing invariance or equivariance properties with respect to the whole permutation group on the feature set. The universal approximation properties of such architectures are established in the case of sets [Zaheer et al. 2017], point clouds [Qi et al. 2017], discrete measures [De Bie et al. 2019], invariant [Maron et al. 2019] and equivariant [Keriven and Peyré 2019] graph neural networks. [Maron et al. 2020] presents a neural architecture invariant w.r.t. the ordering of points and their features, handling point clouds.

The novelty of Dida is to handle continuous and discrete probability distributions, extending state-of-the-art approaches dealing with point clouds [Maron et al. 2020, Jomaa et al. 2021]. This extension yields more general approximation results (Section 5.4) based on the weak convergence of distributions. Compared to the set representation, considering datasets as distributions is best suited to capture density related meta-features.

5.2.2 . Problem Definition

Let $\mathbf{z}= \{(z_i) \in \mathbb{R}^d, i \in [\![1; n]\!]\}$ denote a dataset including n labelled samples, where $z_i = (x_i, y_i)$ with $x_i \in \mathbb{R}^{d_X}$ an instance and $y_i \in \mathbb{R}^{d_Y}$ the associated multi-label. With d_X and d_Y respectively the dimensions of the instance and label spaces, let $d \stackrel{\text{def.}}{=} d_X + d_Y$. By construction, $\mathbf{z}$ is invariant under permutation on the sample ordering; it is viewed as an n-size discrete distribution $\frac{1}{n}\sum_{i=1}^{n} \delta_{z_i}$ in $\mathbb{R}^d$ with δ_{z_i} the Dirac function at z_i.

In the following, $\mathbf{Z}_n(\mathbb{R}^d)$ denotes the space of such n-size point distributions, and $\mathbf{Z}(\mathbb{R}^d) \stackrel{\text{def.}}{=} \cup_n \mathbf{Z}_n(\mathbb{R}^d)$ denotes the space of distributions of arbitrary size.

Let $G \stackrel{\text{def.}}{=} S_{d_X} \times S_{d_Y}$ denote the group of permutations independently operating on the feature and label spaces. For $\sigma = (\sigma_X, \sigma_Y) \in G$, the image $\sigma(z)$ of a labelled sample is defined as $(\sigma_X(x), \sigma_Y(y))$, with $x = (x[k], k \in [\![1; d_X]\!])$ and $\sigma_X(x) \stackrel{\text{def.}}{=} (x[\sigma_X(k)], k \in [\![1; d_X]\!])$. For simplicity and by abuse of notations, the operator push forward mapping a distribution $\mathbf{z} = \{z_i, i \in [\![1; n]\!]\}$ to $\{\sigma(z_i), i \in [\![1; n]\!]\} \stackrel{\text{def.}}{=} \sigma_\sharp \mathbf{z}$ is still denoted σ.

Let $\mathbf{Z}(\Omega)$ denote the space of distributions supported on some domain $\Omega \subset \mathbb{R}^d$, with Ω invariant under permutations in G. The goal of this contribution is to define and train deep architectures, implementing functions φ on $\mathbf{Z}(\Omega)$ that are invariant under G, i.e. such that $\forall \sigma \in G, \varphi(\sigma_\sharp \mathbf{z}) = \varphi(\mathbf{z})$.

By construction, a multi-labelled dataset is invariant under permutations of the samples, of the features, and of the multi-labels, in the sense that the results of any learning algorithm do not (should not) depend on the order of samples, features and multi-labels. For the sake of efficiency (notably in terms of number of neural weights), a neural architecture taking multi-labelled datasets should comply with their invariances, i.e. satisfy the sample and feature permutation invariance properties.

5.3 . Distribution-Based Invariant Networks for Meta-Feature Learning

This section describes the core of the proposed Dida architecture, specifically the mechanism of mapping a point distribution onto another one subject to sample and feature permutation invariance, referred to as *invariant layer*. For the sake of readability, the following presentation of the approach and its properties only considers the discrete probability case; the continuous probability case and the proofs in the discrete and continuous cases are presented in Appendix B.3 and B.2.

5.3.1 . Distribution-Based Invariant Layers

The building block of the proposed architecture, the invariant layer meant to satisfy the feature and label invariance requirements, is defined as follows, taking inspiration from De Bie et al. [2019].

Definition 5. *(Distribution-based invariant layers) Let an interaction functional $\varphi : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}^r$ be G-invariant:*

$$\forall \sigma, z_1, z_2 \in G \times \mathbb{R}^d \times \mathbb{R}^d, \quad \varphi(z_1, z_2) = \varphi(\sigma(z_1), \sigma(z_2)).$$

The distribution-based invariant layer f_φ is defined as $f_\varphi : \mathbf{z} = (z_i)_{i \in [\![1;n]\!]} \in \mathbf{Z}(\mathbb{R}^d) \mapsto f_\varphi(\mathbf{z}) \in \mathbf{Z}(\mathbb{R}^r)$ with

$$f_\varphi(\mathbf{z}) \stackrel{\text{def.}}{=} \left(\frac{1}{n} \sum_{j=1}^{n} \varphi(z_1, z_j), \ldots, \frac{1}{n} \sum_{j=1}^{n} \varphi(z_n, z_j) \right) \tag{5.1}$$

By construction, f_φ is G-invariant if φ is G-invariant. The construction of f_φ is extended to the general case of possibly continuous probability distributions by replacing sums with integrals (Appendix B.1).

It is important that f_φ invariant layers (in particular the first layer of the neural architecture) can handle datasets of arbitrary number of features d_X and number of multi-labels d_Y. An original approach is to define φ as follows. Let $z = (x, y)$ and $z' = (x', y')$ be two samples in $\mathbb{R}^{d_X} \times \mathbb{R}^{d_Y}$. Considering two functions (to be learned) $u : \mathbb{R}^4 \mapsto \mathbb{R}^t$ and $v : \mathbb{R}^t \mapsto \mathbb{R}^r$, then φ is obtained by applying v on the sum of $u(x[k], x'[k], y[\ell], y'[\ell])$ for k ranging in $[\![1; d_X]\!]$ and ℓ in $[\![1; d_Y]\!]$:

$$\varphi(z, z') = v \left(\sum_{k=1}^{d_X} \sum_{\ell=1}^{d_Y} u(x[k], x'[k], y[\ell], y'[\ell]) \right) \tag{5.2}$$

Discussion. By construction φ is invariant to both feature and label permutations; this invariance property is instrumental to a good empirical performance (Section 5.5).

Note that (after learning u and v) f_φ can map a n-size dataset $\mathbf{z}$ onto an n-size $f_\varphi(\mathbf{z})$ dataset for any arbitrary n.

As said, f_φ is based on interaction functionals $\varphi(z_i, z_j)$. This original architecture is rooted in theoretical and algorithmic motivations. On the one hand, interaction functionals are crucial components to reach universal approximation results (see Appendix B.3, Theorem 2). On the other hand, the use of local interactions allows to create more expressive architectures; the benefit of these architectures is illustrated in the experiments (Section 5.5). Formally, the principled Dida framework relies on the weak convergence of probability distributions

(the Wasserstein distance), which enables to compare distributions with varying numbers of points or with continuous densities.

Two particular cases are when φ only depends on its first or second input:

- If $\varphi(z, z') = \psi(z')$, then f_φ computes a global 2nd order "moment" descriptor of the input, as $f_\varphi(\mathbf{z}) = \frac{1}{n}\sum_{j=1}^{n} \psi(z_j) \in \mathbb{R}^r$. The first order moment is not accounted for as the dataset is normalized in a pre-processing step.
- If $\varphi(z, z') = \xi(z)$, then f_φ transports the input distribution via ξ, as $f_\varphi(\mathbf{z}) = \{\xi(z_i), i \in [\![1; n]\!]\} \in Z(\mathbb{R}^r)$.

Remark 1. *(Localized computation) The quadratic complexity of f_φ w.r.t. the number n of samples (Equation 5.1) can be reduced in practice by only computing $\varphi(z_i, z_j)$ for pairs z_i, z_j sufficiently close to each other. Layer f_φ thus extracts and aggregates information related to the neighborhood of the samples.*

Remark 2. *(Link to kernels) The interaction functional φ can be thought of in terms of a kernel, however with significant differences: i) in $f_\varphi(z_i)$, the detail of the pairwise interactions $\varphi(z_i, z_j)$ is lost through averaging; ii) φ takes into account labels; iii) φ is learnt.*

5.3.2 . Learning from distributions

Dida distributional neural architecture defined on point distributions, maps a multi-labelled dataset $\mathbf{z} \in \mathbf{Z}(\mathbb{R}^d)$ onto a real-valued vector noted $\mathcal{F}_\zeta(\mathbf{z})$, with

$$\mathcal{F}_\zeta(\mathbf{z}) \stackrel{\text{def.}}{=} f_{\varphi_m} \circ \ldots \circ f_{\varphi_{o+1}} \circ f_{\varphi_o} \circ \ldots \circ f_{\varphi_1}(\mathbf{z}) \in \mathbb{R}^{d_{m+1}} \tag{5.3}$$

where ζ are the trainable parameters of the architecture (below). For simplicity, only the single label case ($d_Y = 1$) is considered in the following.

The first invariant layer is defined from φ_1, mapping pairs of vectors in $\mathbb{R}^d$ ($d_1 = d$) onto $\mathbb{R}^{d_2}$; it is possibly followed by other invariant layers (the impact of using 1 *vs* 2 invariant layers is experimentally studied in Section 5.5). The last o-th invariant layer is followed by a first non-invariant one, defined from some φ_{o+1} only depending on its second argument; it is possibly followed by other standard layers. The functions defined from the neural nodes on the penultimate layer are referred to as meta-features.

The G-invariance and dimension-agnosticity of the whole architecture only depend on the first layer f_{φ_1} satisfying these properties, defined as follows.

$$\varphi_1((x, y), (x', y')) = v(\sum_k u(x[k], x'[k], y, y')) \tag{5.4}$$

with

$$u(x[k], x'[k], y, y') = (\rho(A_u \cdot (x[k]; x'[k]) + b_u), \mathbb{1}_{y \neq y'}) \tag{5.5}$$

$$v(\bullet) = \rho(A_v \cdot \bullet + b_v) \tag{5.6}$$

where ρ is a non-linear activation function, A_u a $(t, 2)$ matrix, $(x[k]; x'[k])$ the 2-dimensional vector concatenating $x[k]$ and $x'[k]$, b_u a t-dimensional vector, A_v a (t, r) matrix and b_v a r-dimensional vector.

Every $\varphi_k, k \geq 2$ is defined as $\varphi_k = \rho(A_k \cdot + b_k)$, with ρ an activation function, A_k a (d_k, d_{k+1}) matrix and b_k a d_{k+1}-dimensional vector. The Dida neural net thus is parameterized by $\zeta \stackrel{\text{def.}}{=} (A_u, b_u, A_v, b_v, \{A_k, b_k\}_k)$, that is classically learned by stochastic gradient descent from the loss function defined after the considered learning task (Section 5.5). The non-linear activation function ρ is set to RELU in the experiments.

5.4 . Theoretical Analysis

This section investigates the properties of invariant-layer based neural architectures, and establishes their robustness w.r.t. bounded transformations of the involved distributions, and their approximation abilities w.r.t. the convergence in law. As already said, the discrete distribution case is considered for the sake of readability; the case of continuous distributions is detailed in Appendix B.1.

5.4.1 . Topology on Datasets

Point clouds vs. distributions. The fact that datasets are preferably seen as probability distributions (as opposed to point clouds) is motivated as including many copies of a point in a dataset amounts to increasing its importance, which usually makes a difference in standard machine learning settings. Accordingly the topological framework used in the following is that of the convergence in law on distributions, with the distance among two datasets measured using the Wasserstein distance. In contrast, the distance among point clouds commonly relies on the Hausdorff distance among sets (see e.g., Qi et al. [2017]). This distance, that is standard for 2D and 3D data involved in graphics and vision domains, however faces some limitations in higher dimensional domains, e.g. due to max-pooling being a non-continuous operator w.r.t. the convergence in law topology.

Wasserstein distance. The standard 1-Wasserstein distance between two discrete probability distributions $\mathbf{z}, \mathbf{z}' \in \mathbf{Z}_n(\mathbb{R}^d) \times \mathbf{Z}_m(\mathbb{R}^d)$ is defined after Santambrogio [2015], Peyré and Cuturi [2019]:

$$\mathrm{W}_1(\mathbf{z}, \mathbf{z}') \stackrel{\text{def.}}{=} \max_{f \in \mathrm{Lip}_1(\mathbb{R}^d)} \frac{1}{n} \sum_{i=1}^{n} f(z_i) - \frac{1}{m} \sum_{j=1}^{m} f(z'_j)$$

with $\mathrm{Lip}_1(\mathbb{R}^d)$ the space of scalar 1-Lipschitz functions on $\mathbb{R}^d$. The G-invariant 1-Wasserstein distance is defined to extend the above and account for the invariance under operators in G:

$$\overline{\mathrm{W}}_1(\mathbf{z}, \mathbf{z}') = \min_{\sigma \in G} \mathrm{W}_1(\sigma_\sharp \mathbf{z}, \mathbf{z}')$$

Accordingly, $\overline{\mathrm{W}}_1(\mathbf{z}, \mathbf{z}') = 0$ iff $\mathbf{z}$ and $\mathbf{z}'$ are equal in the sense of probability distributions up to sample and feature permutations (Appendix B.1).

Lipschitz property. Let $\mathbf{z}^{(k)}$ be a sequence of distributions weakly converging toward $\mathbf{z}$ (noted $\mathbf{z}^{(k)} \rightharpoonup \mathbf{z}$). By construction, $\mathbf{z}^{(k)} \rightharpoonup \mathbf{z}$ iff $\mathrm{W}_1(\mathbf{z}^{(k)}, \mathbf{z}) \to 0$. Map f from $\mathbf{Z}(\mathbb{R}^d)$ onto $\mathbf{Z}(\mathbb{R}^r)$ is said to be continuous iff for any sequence $\mathbf{z}^{(k)} \rightharpoonup \mathbf{z}$, then $f(\mathbf{z}^{(k)}) \rightharpoonup f(\mathbf{z})$. Map f is said to be C-Lipschitz for $\overline{\mathrm{W}}_1$ iff

$$\forall \mathbf{z}, \mathbf{z}' \in \mathbf{Z}(\mathbb{R}^d), \quad \overline{\mathrm{W}}_1(f(\mathbf{z}), f(\mathbf{z}')) \leq C\overline{\mathrm{W}}_1(\mathbf{z}, \mathbf{z}'). \tag{5.7}$$

The C-Lipschitz property entails the continuity of f: if two input distributions are close in the permutation invariant 1-Wasserstein sense, their images by f are close too.

5.4.2 . Continuity Results

Let us assume the interaction functional φ to satisfy the Lipschitz property w.r.t. their first and second arguments ($\forall z \in \mathbb{R}^d, \varphi(z, \cdot)$ and $\varphi(\cdot, z)$ are $C_\varphi -$ Lipschitz.). Then invariant layer f_φ also satisfy the Lipschitz property.

Proposition 1. *Invariant layer f_φ of type (Equation 5.1) is $(2rC_\varphi)$-Lipschitz in the sense of (Equation 5.7).*

A second result regards the case where two datasets $\mathbf{z}$ and $\mathbf{z}'$ are such that $\mathbf{z}'$ is the image of $\mathbf{z}$ through some diffeomorphism τ ($\mathbf{z} = (z_1, \ldots, z_n)$ and $\mathbf{z}' = \tau_\sharp \mathbf{z} = (\tau(z_1), \ldots, \tau(z_n))$. If τ is close to identity, then $f_\varphi(\tau_\sharp \mathbf{z})$ and $f_\varphi(\mathbf{z})$ are close too. More generally, if continuous transformations τ and ξ respectively apply on the input and output space of f_φ, and are close to identity, then $\xi_\sharp f_\varphi(\tau_\sharp \mathbf{z})$ and $f_\varphi(\mathbf{z})$ are also close.

Proposition 2. *Let $\tau : \mathbb{R}^d \to \mathbb{R}^d$ and $\xi : \mathbb{R}^r \to \mathbb{R}^r$ be two Lipschitz maps with respectively Lipschitz constants C_τ and C_ξ. Then, $\forall \mathbf{z}, \mathbf{z}' \in \mathbf{Z}(\Omega)$,*

$$\begin{aligned}
&\overline{\mathrm{W}}_1(\xi_\sharp f_\varphi(\tau_\sharp \mathbf{z}), f_\varphi(\mathbf{z})) \\
&\quad \leq \sup_{x \in f_\varphi(\tau(\Omega))} \|\xi(x) - x\|_2 + 2r \operatorname{Lip}(\varphi) \sup_{x \in \Omega} \|\tau(x) - x\|_2 \\
&\quad \textit{In addition, if } \tau \textit{ is equivariant,} \\
&\overline{\mathrm{W}}_1(\xi_\sharp f_\varphi(\tau_\sharp \mathbf{z}), \xi_\sharp f_\varphi(\tau_\sharp \mathbf{z}')) \leq 2r\, C_\varphi\, C_\tau\, C_\xi \overline{\mathrm{W}}_1(\mathbf{z}, \mathbf{z}')
\end{aligned}$$

Proofs: in Appendix B.2.

5.4.3 . Universal Approximation Results

Lastly, the universality of the proposed architecture is established, showing that the composition of an invariant layer (Equation 5.1) and a fully-connected layer is enough to yield the universal approximation property, over all functions defined on $Z(\mathbb{R}^d)$ with dimension d less than some upper bound D.

Theorem 1. *Let $\mathcal{F} : \mathbf{Z}(\Omega) \to \mathbb{R}$ be a G-invariant map on a compact $\Omega \subset \mathbb{R}^d$, continuous for the convergence in law. Then $\forall \epsilon > 0$, there exists two continuous maps ψ, φ such that*

$$\forall \mathbf{z} \in \mathbf{Z}(\Omega), \quad |\mathcal{F}(\mathbf{z}) - \psi \circ f_\varphi(\mathbf{z})| < \epsilon$$

where φ is G-invariant and independent of $\mathcal{F}$.

Proof: in Appendix B.3.

After Theorem 1, any invariant continuous function defined on distributions with compact support can be approximated with arbitrary precision by an invariant neural network. This result holds for distributions with compact support in $\mathbb{R}^d$ for all $d \leq D$, with D an upper bound on the dimension of the considered distribution supports. The proof (Appendix B.3) involves mainly three steps: (i) an invariant layer f_φ can be approximated by an invariant network; (ii) the universal approximation theorem [Cybenko 1989, Leshno et al. 1993]; (iii) uniform continuity is used to obtain uniform bounds. This result generalizes the universality result established for fixed numbers of dimensions and points [Maron et al. 2020] to the cases of finite distributions of any size n, and continuous distributions.

5.5 . Experimental Validation

The experimental validation is conducted to assess: i) the performance of Dida compared to the state of the art; ii) the merits of the original architecture of invariant layers, based on an interaction functional φ (Equation 5.1).

5.5.1 . Experimental setting

Tasks. The validation is conducted on two tasks, derived from supervised datasets as opposed to standard point cloud benchmarks.

- **Task 1** is a patch identification problem inspired from [Jomaa et al. 2021] aiming to identify if two dataset patches are extracted from a same dataset.
- **Task 2** aims to rank hyper-parameter configurations for a fixed supervised learning algorithm, according to their performance on the considered dataset.

Benchmarks. Three benchmarks are used (Table 5.1): TOY and UCI, taken from [Jomaa et al. 2021], and OpenML CC-18 [Bischl et al. 2019], with data preprocessing detailed in Appendix B.4.1.

Baselines. Three baselines are considered:

- DSS [Maron et al. 2020] is involved with three variants: i) linear invariant layers; ii) non-linear invariant layers; iii) equivariant + invariant layers.

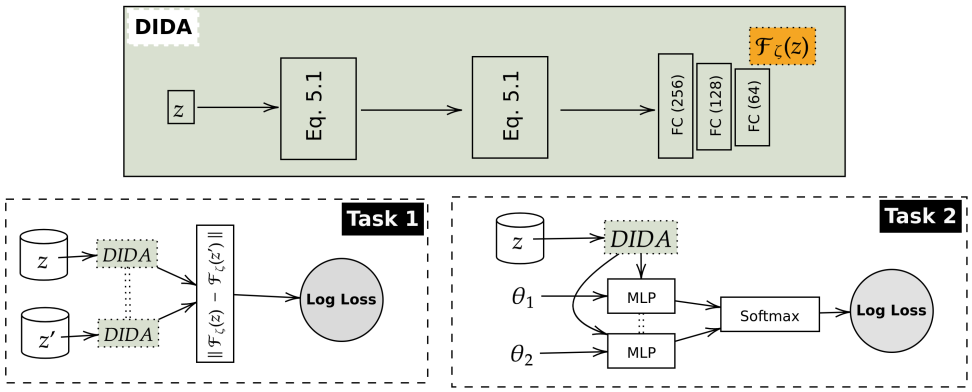


Figure 5.1: Learning meta-features with Dida. (Up) The Dida architecture (FC for fully connected layer). (Bottom left) Task 1: Learning meta-features for patch identification using a Siamese architecture (Section 5.5.2). (Bottom right) Task 2: learning meta-features for ranking hyper-parameter configurations θ_1 and θ_2 (Section 5.5.3).

- Dataset2Vec [Jomaa et al. 2021].
- The last baseline is a function (trained to the task) of the hand-crafted meta-features (HC) (detailed in Table B.2, Appendix B.4.2) [Muñoz et al. 2018].

We implemented the DSS baseline as the code was not available and we re-implemented Dataset2Vec as described in [Jomaa et al. 2021]. DSS is augmented with an aggregator summing over the features in order to accommodate datasets with varying numbers of dimensions. All baseline codes are publicly available at https://github.com/herilalaina/dida for the sake of reproducibility.

Training setups. The same Dida architectures are used for both tasks, involving 1 or 2 invariant layers followed by 3 fully connected (FC) layers (Figure 5.1, left). All experiments run on 1 NVIDIA-Tesla-V100-SXM2 GPU with 32GB memory, using Adam optimizer with base learning rate 10^{-3} and batch size 32. For all considered architectures, meta-features $\mathcal{F}_\zeta(\mathbf{z})$ consist of the output of the penultimate layer, with ζ denoting the trained parameters.

5.5.2 . Task 1: Patch Identification

In Task 1, patches are extracted from datasets and the task consists in predicting whether two patches are extracted from the same dataset. Letting $\mathbf{u}$ denote a dataset with n d-dimensional examples, patch $\mathbf{z}$ is constructed from $\mathbf{u}$, by selecting (uniformly with replacement) n_z examples in $\mathbf{u}$ and considering their description based on d_z features selected uniformly with replacement among $\mathbf{u}$ features. Size n_z and number d_z of features of the patch are uniformly selected (Table 5.1). In

	Datasets			Patches	
	# datasets	# samples	# features	# samples	# features
Toy Dataset	10000	[2048, 8192]	2	200	2
UCI	121	[10, 130064]	[3, 262]	[200, 500]	[2, 15]
OpenML CC-18	72	[500, 100000]	[5, 3073]	[700, 900]	[3, 11]

Table 5.1: Benchmarks and patches characteristics.

Task 1, an example is made of a pair of patches $(\mathbf{z}, \mathbf{z}')$, together with its associated label $\ell(\mathbf{z}, \mathbf{z}')$, set to 1 iff $\mathbf{z}$ and $\mathbf{z}$' are extracted from the same initial dataset **u** and $n_z = n_{z'}$.

For all considered architectures, the parameters are trained using a Siamese architecture (Figure 5.1, bottom-left; Algorithm 3. The learned classifier $\hat{\ell}_\zeta(\mathbf{z}, \mathbf{z}')$ is the softmax $\exp(-\|\mathcal{F}_\zeta(\mathbf{z}) - \mathcal{F}_\zeta(\mathbf{z}')\|_2)$, with $\mathcal{F}_\zeta(\mathbf{z})$ and $\mathcal{F}_\zeta(\mathbf{z}')$ the meta-features computed for $\mathbf{z}$ and $\mathbf{z}'$, where ζ is trained to minimize the cross-entropy loss:

$$\sum_{\mathbf{z},\mathbf{z}'} \ell(\mathbf{z}, \mathbf{z}') \log(\hat{\ell}_\zeta(\mathbf{z}, \mathbf{z}')) + (1 - \ell(\mathbf{z}, \mathbf{z}')) \log(1 - \hat{\ell}_\zeta(\mathbf{z}, \mathbf{z}')) \tag{5.8}$$

Algorithm 3: Patch Identification

Procedure `Task_1`($\mathcal{F}_\zeta$, *bench, N*)
  input : A meta-feature extractor $\mathcal{F}_\zeta$ in {Dida, Dataset2Vec, Deep Sets, DSS, Hand-crafted}, a benchmark $bench$ in {Toy, UCI, OpenML}, and a number of iterations N
  for $i = 1 \ldots .$ **do**
    $\mathbf{z}_1, \mathbf{z}_2, y \leftarrow$ `generate_patches`($bench$)
    $m_1 \leftarrow \mathcal{F}_\zeta(\mathbf{z}_1)$
    $m_2 \leftarrow \mathcal{F}_\zeta(\mathbf{z}_2)$
    Compute loss (Equation 5.8), and update ζ
  end

Table 5.2 reports the empirical results on TOY and UCI datasets. On TOY, Dida with 2 invariant layers, referred to as 2L-Dida behaves on par with Dataset2Vec and DSS. On UCI, the task appears to be more difficult, which is explained from the higher and more diverse number of features in the datasets. The fact that 2L-Dida significantly outperforms all other approaches is explained from the interaction functional structure (Eqs. 5.1, 5.2), expected to better grasp contrasts among examples. Dida with 1 invariant layer (1L-Dida) is much behind 2L-Dida; with a significantly lesser number of parameters than 2L-Dida, the 1L-Dida architecture might lack representational power.

Method	# params	TOY	UCI
Hand-crafted Meta-features	53312	77.05 $\%\pm$ 1.63	58.36 $\%\pm$ 2.64
No-FInv-DSS (no inv. in features)	1297692	90.49 $\%\pm$ 1.73	64.69 $\%\pm$ 4.89
Dataset2Vec (our implementation)	257088	97.90 $\%\pm$ 1.87	77.05 $\%\pm$ 3.49
DSS layers (Linear aggregation)	1338684	89.32 $\%\pm$ 1.85	76.23 $\%\pm$ 1.84
DSS layers (Non-linear aggregation)	1338684	96.24 $\%\pm$ 2.04	83.97 $\%\pm$ 2.89
DSS layers (Equivariant+invariant)	1338692	96.26 $\%\pm$ 1.40	82.94 $\%\pm$ 3.36
Dida (1 invariant layer)	323028	91.37 $\%\pm$ 1.39	81.03 $\%\pm$ 3.23
Dida (2 invariant layers)	1389089	97.20 % $\pm$ 0.10	**89.70** % $\pm$ **1.89**

Table 5.2: Comparative performances on patch identification of Dida, No-FInv-DSS, Dataset2Vec, DSS and functions of hand-crafted meta-features: average and std deviation of predictive accuracy over 10 runs.

Lesion study. A fourth baseline, No-FInv-DSS [Zaheer et al. 2017] only differs from DSS as it is *not* feature permutation invariant; this additional baseline is used to assess the impact of this invariance property. The fact that No-FInv-DSS lags behind all DSS variants, all with similar number of parameters, confirms the importance of this invariance property. Note also that No-FInv-DSS is outperformed by 1L-Dida, while the latter involves significantly less parameters.

5.5.3 . Task 2: Ranking ML confiugrations

Task 2 aims to comparatively assess two vectors of hyper-parameters θ and θ' of a fixed supervised learning algorithm Alg, referred to as configurations of Alg, depending on their performance on a dataset patch $\mathbf{z}$. For brevity and by abuse of language, the performance of a configuration θ on $\mathbf{z}$ is meant for the accuracy of the model learned from $\mathbf{z}$ using Alg with configuration θ, computed using a 3 fold cross validation.

The considered ML algorithms are: Logistic regression (LR), SVM, k-Nearest Neighbours (k-NN), linear classifier learned with stochastic gradient descent (SGD). For each algorithm, a Task 2 problem is defined as follows (Algorithm 4). An example is made of a triplet $(\mathbf{z}, \theta, \theta')$, associated with a binary label $\ell(\mathbf{z}, \theta, \theta')$, set to 1 iff θ' yields better performance than θ on $\mathbf{z}$. Thus, the overall architecture consists of:

- a meta-feature extractor $\mathcal{F}_\zeta(\mathbf{z})$;
- a 2-layer FC network (depending on the considered Alg as they have different configuration spaces) with input vector $[\mathcal{F}_\zeta(\mathbf{z}); \theta; \theta']$

The overall is trained to minimize a cross-entropy loss (Equation 5.8).

In each epoch, a batch made of triplets $(\mathbf{z}, \theta, \theta')$ is built, with θ, θ' uniformly drawn in the algorithm configuration space (Table B.3) and $\mathbf{z}$ a patch of a dataset in the OpenML CC-2018 [Bischl et al. 2019], of size n uniformly drawn in $[700; 900]$

Algorithm 4: Hyper-parameter Ranking

Procedure `Task_2`($\mathcal{F}_\zeta$, *Alg*)
  input : A meta-feature extractor $\mathcal{F}_\zeta$ in {Dida, Dataset2Vec, Deep Sets, DSS, Hand-crafted}, an algorithm Alg in {SGD, SVM, LR, k-NN}.
  $\texttt{NN} \leftarrow$ 2-layer fully connected neural network
  for $i = 1, 2, ...$ **do**
    $\mathbf{z} \leftarrow$ `generate_patch`(*OpenML*)
    Sample (θ, θ'), two configurations of Alg (Table B.3)
    Set binary target y as 1 if $\texttt{accuracy}(\mathbf{z}, \theta) > \texttt{accuracy}(\mathbf{z}, \theta')$ else 0
    Compute loss (Equation 5.8) between y and $\texttt{NN}([\mathcal{F}_\zeta(\mathbf{z}); \theta; \theta'])$
    Update ζ and NN
  end

Method	SGD	SVM	LR	k-NN
Hand-crafted	71.18 %± 0.41	75.39 %± 0.29	86.41 %± 0.419	65.44 %± 0.73
Dataset2Vec	74.43 %± 0.90	81.75 %± 1.85	89.18 %± 0.45	72.90 %± 1.13
DSS (Linear aggregation)	73.46 %± 1.44	82.91 %± 0.22	87.93 %± 0.58	70.07 %± 2.82
DSS (Equivariant+Invariant)	73.54 %± 0.26	81.29 %± 1.65	87.65 %± 0.03	68.55 %± 2.84
DSS (Non-linear aggregation)	74.13 %± 1.01	83.38 %± 0.37	87.92 %± 0.27	73.07 %± 0.77
DIDA (1 invariant layer)	77.31 %± 0.16	84.05 %± 0.71	**90.16** %± **0.17**	74.41 %± 0.93
DIDA (2 invariant layers)	**78.41** %± **0.41**	**84.14** %± **0.02**	89.77 %± 0.50	**78.91** %± **0.54**

Table 5.3: Comparative performances on configuration ranking of Dida, Dataset2Vec, DSS and functions of hand-crafted meta-features: average and std deviation of pairwise ranking performance over 3 runs.

and number of features d in $[3; 10]$. Dida and all baselines are trained using Algorithm 4. Their comparative performances are displayed in Table 5.3, reporting their ranking accuracy. 2L-Dida (respectively 1L-Dida) significantly outperforms all baseline approaches except in the $Alg =$ LR case (resp., in the $Alg = k$-NN case). A higher performance gap is observed for the k-NN case, which is explained as this algorithm mostly exploits the local geometry of the examples.

5.6 . Partial Conclusion

The theoretical contribution presented in this chapter is the Dida architecture, able to learn from discrete and continuous distributions on $\mathbb{R}^d$, invariant w.r.t. feature ordering, agnostic w.r.t. the size and dimension d of the considered distribution sample (with d less than some upper bound D). This architecture en-

joys universal approximation and robustness properties, generalizing former results obtained for point clouds [Maron et al. 2020].

The merits of Dida are empirically and comparatively demonstrated on two tasks defined at the dataset level. Task 2 in particular constitutes a first step toward performance modelling [Rice 1976], as the learned (algorithm-dependent) meta-features support an efficient ranking of the configurations for the current dataset. These meta-features, while requiring circa 4 hours in the considered environment to be learned, are efficiently computed on datasets. On the considered tasks, they improve on the meta-features manually defined in the last 40 years [Muñoz et al. 2018, Rivolli et al. 2022].

Limitations. The proposed Dida approach, however, has two main limitations. Firstly, meta-feature learning, as for any learning setup, relies on tasks with sufficiently many examples to be available. Our early attempts failed due to current (curated) ML benchmarks being not sufficiently representative. Secondly, it is reasonable to think that learned meta-features are specific to the training task. It implies that learning meta-features for AutoML would require the underlying topology over the joint datasets and ML performances spaces, which is not known in practice.

Perspectives. A direct perspective is to investigate the learned meta-features for AutoML use cases. This line of research will be considered in the next chapter, addressing the aforementioned limitations. Another long-term perspective is to investigate the relationships between two datasets, and estimate *a priori* the chances of a successful domain adaptation [Ben-David et al. 2010, Alvarez-Melis and Fusi 2020]. Such a goal requires a large amount of labelled datasets, however, one can explore self-supervised setting to overcome this issue. For instance, bootstrapping output representation as in Grill et al. [2020] is a promising further work.

6 - Meta-Learning for Tabular Data

This chapter presents the third contribution of this manuscript, also devoted to learning meta-features suitable to AutoML problems, focusing on hyper-parameter recommendation or Bayesian Optimization initialization. This approach, called Metabu (Meta-learning for Tabular Data), aims to address the limitations of the Dida approach presented in the previous chapter, relaxing the need for large and representative benchmaks.

Specifically, Metabu leverages Optimal Transport to build a topology on the dataset space, mimicking the topology on the datasets induced from their top hyper-parameter configurations. This topology is used to optimize a linear mapping on the hand-crafted meta-features [Rakotoarison et al. 2021].

The chapter is organized as follows. After presenting the motivations in Section 6.1, the formal background is introduced in Section 6.2, presenting Optimal Transport Cuturi [2013], Peyré and Cuturi [2019]. Section 6.3 gives a detailed overview of Metabu. Section 6.4 reports on the empirical validation of Metabu, and the chapter ends with a partial conclusion.

6.1 . Motivation

A primary motivation of Metabu is to address Dida limitations in order to learn suitable meta-features for AutoML.

Dida limitations are two-faceted. On the one hand, it can hardly handle large and dirty datasets. As a result, Dida empirical experiments only consider patches (instead of datasets) with continuous features (using preprocessing if needed). Such setting hardly handles standard dataset benchmarks such as OpenML [Vanschoren et al. 2014] and UCI [Dua and Graff 2017].

On the other hand, Dida proceeds by training meta-features, which requires sufficiently many datasets in the AutoML benchmark. Unfortunately, the current state-of-the-art curated benchmark OpenML-CC18 has less than a hundred datasets available, which is quite insufficient for training a deep network. This shortage of datasets is all the more blocking as, to our best knowledge, generating *diverse* datasets is a challenging and yet open problem.

Besides these challenges, Dida meta-features, as well as hand-crafted meta-features (except for the landmarking ones), mostly capture statistical features about the datasets. Still, many studies [Feurer et al. 2015a, Misir and Sebag 2017, Fusi et al. 2018] suggest that an efficient AutoML system can hardly rely only on such meta-features. Typically, Auto-Sklearn [Feurer et al. 2015a] relies on Bayesian optimization and iteratively learns and exploits one performance model specific to each dataset; PMF [Fusi et al. 2018] uses a probabilistic collaborative filtering approach, where the cold-start problem is handled as in Auto-Sklearn; OBOE [Yang

et al. 2019] likewise uses a collaborative filtering approach, combined with active learning.

Based on these arguments, the proposed Metabu approach i) builds upon existing meta-features; ii) aims at meta-features defining a reliable *topology* on the dataset space, such that two datasets are close iff the best hyper-parameter configurations for these datasets are close. Capturing the target topology (available for the datasets in the benchmarks only) can support an inexpensive and efficient AutoML strategy: selecting the best hyper-parameter configurations of the nearest neighbor(s) of the current dataset. Moreover, as will be shown in Section 6.4, such meta-features allow one to better understand the dataset space w.r.t. a given ML algorithm, to estimate its intrinsic dimension and appreciate the distribution of the ML benchmark suites in the meta-feature space.

Formally, the Meta-learning for Tabular Data (Metabu) approach casts and tackles the construction of good meta-features – relatively to an ML algorithm $\mathcal{A}$ – as an Optimal Transport (OT) problem [Cuturi 2013, Peyré and Cuturi 2019]. More precisely, two representations of the datasets are considered: the basic one consists of 135 manually designed meta-features; The target one, out-of-reach except for the datasets in the benchmark suite, represents a dataset as the distribution of the hyper-parameter configurations of $\mathcal{A}$ yielding the top performances for this dataset. Optimal Transport is used to find a linear transformation of the basic meta-features, such that the resulting Euclidean distance emulates the Wasserstein distance [Mémoli 2011] on the target representation. Overall, Metabu learns once for all the meta-features aimed to capture the topology and neighborhoods corresponding to the target representation. These meta-features can be computed from scratch for each new dataset.

A main difference w.r.t. e.g. Yang et al. [2019] and Fusi et al. [2018] thus is that no cold-start phase (adjusting the representation of the dataset at hand, through launching new configurations) is needed.

6.2 . Formal Background

The limitations of manually designed meta-features [Caliński and Harabasz 1974, Vilalta 1999, Bensusan and Giraud-Carrier 2000, Pfahringer et al. 2000, Peng et al. 2002, Muñoz et al. 2018, Song et al. 2012, Bardenet et al. 2013, Feurer et al. 2015a·b, Pimentel and Carvalho 2019, Lorena et al. 2019, Rivolli et al. 2022] and those of *learned* meta-features [Jomaa et al. 2021, De Bie et al. 2020] have been respectively detailed in Section 3 and Chapter 5. This section briefly describes the optimal transport methodology used in this chapter to construct new meta-features, and the related works.

Optimal Transport, first mentioned in Chapter 5, enables to compute the distance over datasets using Wasserstein distance. OTDD [Alvarez-Melis and Fusi 2020] uses OT to learn a mapping between datasets over the joint feature and

label spaces.

Let (Ω_x, d_x) and (Ω_y, d_y) denote compact metric spaces, and $\mathbf{x}$ and $\mathbf{y}$ distributions respectively defined on Ω_x and Ω_y. The search space $\Gamma(\mathbf{x}, \mathbf{y})$ is the space of all distributions on $\Omega_x \times \Omega_y$ with marginals $\mathbf{x}$ and $\mathbf{y}$. Let the transport cost function $c : \Omega_x \times \Omega_y \mapsto \mathbb{R}^+$ be a scalar function on $\Omega_x \times \Omega_y$[1].

As said, the Wasserstein distance of $\mathbf{x}$ and $\mathbf{y}$ is defined as:

$$d_W^q(\mathbf{x}, \mathbf{y}) = \min_{\gamma \in \Gamma(\mathbf{x}, \mathbf{y})} \mathbb{E}_{(x,y)\sim\gamma}[c^q(x, y)]^{1/q}$$

with q a positive real number, set to 1 in the following.

Another OT-based distance is the Gromov-Wasserstein distance (GW) [Mémoli 2011], measuring how well a distribution in $\Gamma(\mathbf{x}, \mathbf{y})$ preserves the distances on both Ω_x and Ω_y, akin a rigid transport between both domains:

$$d_{GW}^q(\mathbf{x}, \mathbf{y}) = \min_{\gamma \in \Gamma(\mathbf{x}, \mathbf{y})} \mathbb{E}_{(x,y)\sim\gamma,(x'y')\sim\gamma}[|d_x(x, x') - d_y(y, y')|^q]^{1/q}$$

.

The Fused Gromov-Wasserstein (FGW) distance [Vayer et al. 2019] combines both the Wasserstein and the Gromov-Wasserstein distances.

Definition 6. *The Fused q-Gromov-Wasserstein distance is defined on $\Omega_x \times \Omega_y$ as follows:*

$$\begin{aligned} d_{FGW;\alpha}^q(\mathbf{x}, \mathbf{y}) = \min_{\gamma \in \Gamma(\mathbf{x}, \mathbf{y})} & (1-\alpha) \underbrace{\left(\int_{\Omega_x \times \Omega_y} c^q(x, y) \mathrm{d}\gamma(x, y) \right)^{\frac{1}{q}}}_{\textit{Wasserstein Loss}} \\ & + \alpha \underbrace{\left(\int_{\Omega_x \times \Omega_y} \int_{\Omega_x \times \Omega_y} |d_x(x, x') - d_y(y, y')|^q \mathrm{d}\gamma(x, y) \mathrm{d}\gamma(x', y') \right)^{\frac{1}{q}}}_{\textit{Gromov-Wasserstein Loss}} \end{aligned} \tag{6.1}$$

$\alpha \in [0, 1]$ is a trade-off parameter: For $\alpha = 0$ (resp. $\alpha = 1$), the fused q-Gromov-Wasserstein distance is exactly the q-Wasserstein distance d_W^q (resp. the q-Gromov-Wasserstein distance d_{GW}^q).

According to [Xu et al. 2019b,a, 2020] the non-convex optimization in Equation 6.1 can be efficiently optimized along an iterative process using proximal gradient method. Concretely, given a current estimate $\gamma^{(j)}$ at j-th iteration, define a

[1] When $\Omega_x = \Omega_y = \Omega$, unless otherwise stated, the transport cost $c(x, y)$ is the Euclidean distance $d(x, y)$.

new objective function $d^{1,j}_{FGW;\alpha}$ similar to Equation 6.1 with a regularization term KL$(\gamma||\gamma^{(j)})$, where KL be the Kullback-Leibler divergence. The later sub-problem is then solved using Sinkhorn Algorithm [Cuturi 2013] yielding the new transport map γ^{j+1}. We refer the reader to Algorithm 2 of Xu et al. [2019b] for a more complete description of the optimization method.

The Wasserstein distance and variants thereof were successfully used to evaluate the "alignment" among datasets, e.g. between the source and the target datasets in the context of domain adaptation [Courty et al. 2017] or transfer learning [Alvarez-Melis and Fusi 2020, 2021]. FGW distance was used to enforce the consistency of the latent space when jointly training several Variational Auto-Encoders [Xu et al. 2020, Nguyen et al. 2020]. Metabu will likewise take inspiration from OT to create a bridge between two representations of the datasets: the basic one, and the target one, critically using both GW and FGW distances.

6.3 . Overview of Metabu

We use the same notations as in the previous chapter: let δ be the Dirac function; distributions are noted in bold font and vectors in italic.

Let $\mathcal{A}$ and $\Theta^{\mathcal{A}}$ respectively denote an ML pipeline and its hyper-parameter configuration space; superscript $\mathcal{A}$ is omitted when clear from the context. Space Θ is embedded into the a-dimensional real-valued space $\mathbb{R}^a$, using a one-hot encoding of Boolean and categorical hyper-parameters. After describing the principle of the approach, some key issues are detailed: the augmentation of the AutoML benchmark to avoid overfitting, and the setting of the number d of the Metabu meta-features, estimated from the intrinsic dimensionality of the AutoML benchmark suite.

6.3.1 . Principle

Intuitively, two representations can be associated with a dataset: The *basic representation* $x \in \mathbb{R}^D$ of a dataset reports the values of the D manually designed meta-features for this dataset. By construction, it can be cheaply computed for any dataset.[2] The *target representation* $\mathbf{z}$ of a dataset is the distribution on the space Θ supported by the configurations yielding the best performances on this dataset. This precious target representation is unreachable in practice, but can be approached after the performances of the models learned with a number of configurations (aka configuration performances) have been assessed. In practice, the configuration performances are only available for a small number n of datasets. The difference between the basic and the target topologies is depicted on Figure 6.1, in Θ space (projected on first two PCA eigenvectors). The later figure suggests that the target representations, built upon the top configurations of datasets, are better suited for AutoML problems.

[2] Only non-expensive landmark meta-features are considered in the following.

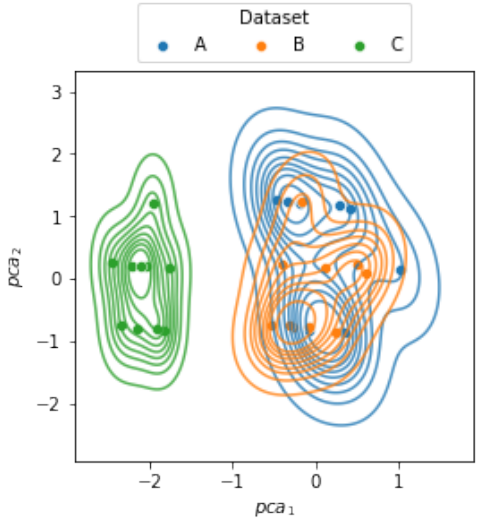


Figure 6.1: Top configurations of datasets A, B, and C, where B, in orange (resp. C, in green) is the nearest neighbor of A w.r.t. target (resp. basic) representation.

In order to build a bridge between both representations, let us consider an intermediate representation, termed *projected target representation*, derived from the target representation ones by mapping each $(z_i)_{1\leq i\leq n}$ on some $u_i \in \mathbb{R}^d$ using a distance-preserving projection, e.g. Multi-Dimensional Scaling (MDS) [Kruskal 1964].

Metabu tackles an Optimal Transport problem so as to learn a mapping $\psi : \mathbb{R}^D \mapsto \mathbb{R}^d$ from the basic representation on the projected target representation space such that the $\psi(x_i)_{1\leq i\leq n}$ are aligned with the u_is in the sense of the q-Fused Gromov-Wasserstein distance (Section 6.2).

In brief, mapping ψ sends the basic meta-feature space on $\mathbb{R}^d$, such that the Euclidean metric on the $\psi(x_i)$ reflects the Euclidean metric on the u_is, itself reflecting the metric on the target z_is.

The descriptive features of the $\psi(x_i)$, referred to as Metabu meta-features, are meant to both be cheaply computable from the basic meta-features, and define a Euclidean distance conducive to the AutoML task.

6.3.2 . Augmenting the AutoML benchmark.

The OpenML CC-18 [Bischl et al. 2019], to our knowledge the largest curated tabular dataset benchmark (that will be used in the experiments), contains $n = 72$ classification datasets; the target representation is available for 64 of them. The shortage of such datasets yields a risk of overfitting the learned meta-features. This challenge is tackled by augmenting the OpenML CC-18 benchmark suite,

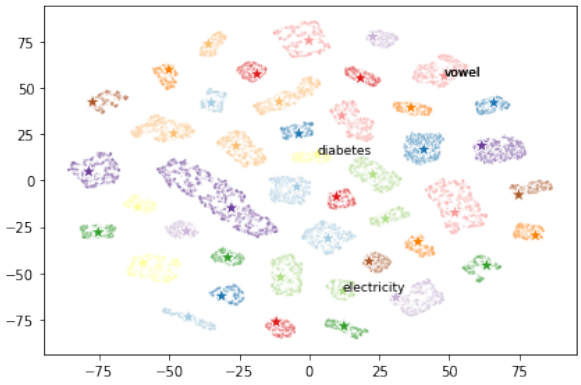


Figure 6.2: 2-D Visualisation of the OpenML datasets in basic representation (legended with a $\star$'s) + their boostrapped augmentations.

using a bootstrap procedure [Efron 1979].[3] The goal is to pave the meta-feature space more densely and more accurately than through e.g., perturbing the basic representation with Gaussian noise.

The visualisation of the augmented benchmark (Figure 6.2, projected using tSNE [Maaten and Hinton 2008] on the basic representation), shows that the datasets built by bootstrapping of some initial dataset E form a cluster close to E (as could be expected as the manually designed meta-features are stable under small stochastic variations), and separated from the clusters generated from other datasets, suggesting that the initial benchmark suite only sparsely paves the basic meta-feature space. Complementary experiments (omitted) with perplexity ranging in [5, 10, 15, 25, 30, 40, 50] show that clusters generated by augmentation of different OpenML datasets keep staying far apart from one another.

6.3.3 . The Metabu algorithm

The algorithm is provided the $p = 1,000 \times n$ training datasets of the benchmark suite, augmented as described above. The Metabu meta-features are constructed in a 3-step procedure, depicted on Figure 6.3 and illustrated in Algorithm 5:

Step 1: *Target representation and Wasserstein distance*. Considering the i-th training dataset, let $\Theta_i \subset \Theta$ denote the set of hyper-parameter configurations with performance in the top-L known configuration performances ($L = 20$ in the experiments).[4]

[3] For each ℓ-size dataset E in the benchmark suite, $K = 1,000$ new datasets $F_1, \ldots F_K$ are generated, where F_i includes ℓ examples selected in E uniformly with replacement. The basic representation of F_i is computed, and its target representation is set to that of E.

[4] Early attempts to define Θ_i in a more sophisticated way, e.g. using t-test to distinguish the "good" configurations from the others, led to an uninformative target representation. A tentative interpretation for this fact is that quite a few OpenML datasets are very easy, leading to retain all configurations for these datasets and

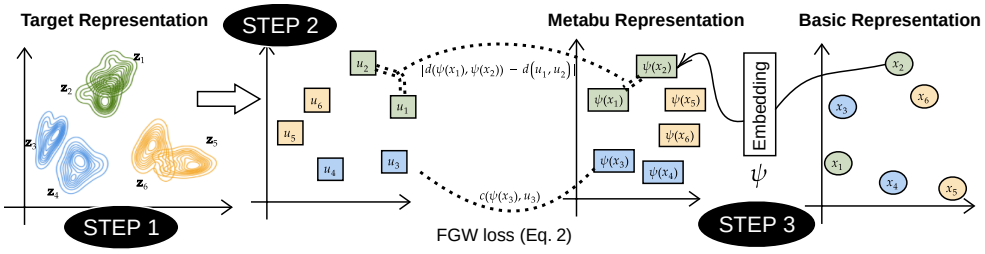


Figure 6.3: From basic to Metabu meta-features using Fused Gromov-Wasserstein. Basic (respectively Metabu) representations are depicted by circles (resp. squares). Target representations are depicted in the rightmost subplot. Neighbor datasets in the target space have same color in all subplots.

Algorithm 5: Learning Metabu meta-features

Data: Set of n training datasets, each represented with its **basic representation** (meta-feature vector) x_i and its **target representation** (set of top 20 hyper-parameters) $\mathbf{z}_i$ for $i = 1 \dots n$.

Result: Embedding layer ψ^*

// Build projected target representation

$C_{i,j} \leftarrow d^1_{\mathcal{W}}(\mathbf{z}_i, \mathbf{z}_j)$ for $i = 1 \dots n, j = 1 \dots n$; /* STEP 1 */

Estimate intrinsic dimension d from matrix C (Alg. 6);

$\mathbf{u} \leftarrow \text{MDS}(C, d)$; /* STEP 2 */

// Learn ψ

$\psi \leftarrow \text{Linear}(135, d)$; /* 135 basic meta-features. */

$\mathbf{x} \leftarrow \frac{1}{p} \sum_{i=1}^{n} \delta_{x_i}$;

$\mathcal{L} \leftarrow$ FGW as defined in Equation 6.1;

$\psi^* \leftarrow \text{ADAM}(\mathcal{L}, \psi_\sharp \mathbf{x}, \mathbf{u})$; /* STEP 3 */

The target representation $\mathbf{z}_i$ of the i-th dataset is the discrete distribution with support Θ_i. The distance $d_W^1(\mathbf{z}_i, \mathbf{z}_j)$ is the 1-Wasserstein distance among distributions (Section 6.2).

Step 2: *Projecting the target representation on* $\mathbb{R}^d$. The second step consists in projecting the $\mathbf{z}_i$s on $\mathbb{R}^d$, where d is identified using an intrinsic dimensionality procedure (details below), using Multi-Dimensional Scaling [Kruskal 1964], such that the distance $d(u_i, u_j)$ approximates the 1-Wasserstein distance $d_W^1(\mathbf{z}_i, \mathbf{z}_j)$ (Figure 6.3, leftmost and second subplots). Note that by construction, the u_is are defined up to an isometry on $\mathbb{R}^d$.

Step 3: *Learning the* Metabu *meta-features*. Let $\mathbf{x} = \frac{1}{p}\sum_{i=1}^{p} \delta_{x_i}$ denote the uniform discrete distribution on $\mathbb{R}^D$ whose support is the set of p datasets using their basic representations.

Let $\mathbf{u} = \frac{1}{n}\sum_{i=1}^{n} \delta_{u_i}$ denote the uniform discrete distribution on $\mathbb{R}^d$ whose support is the set of u_is defined above. The Metabu meta-feature space is built by finding a mapping ψ from $\mathbb{R}^D$ on $\mathbb{R}^d$ that pushes the representation metric on $\mathbb{R}^d$, that is, such that the image of $\mathbf{x}$ via ψ is as close as possible to $\mathbf{u}$, and reflects its topology in the FGW sense (Figure 6.3, rightmost and third subplots).

Formally, let $\psi_\sharp \mathbf{x} \stackrel{\text{def.}}{=} \frac{1}{p}\sum_{i=1}^{p} \delta_{\psi(x_i)}$ be the push-forward distribution of $\mathbf{x}$ on $\mathbb{R}^d$ for a given ψ. The overall optimization problem is to find a mapping ψ^* that minimizes the FGW distance between the $\mathbf{u}$ distribution and the push distribution $\psi^*_\# \mathbf{x}$:

$$\psi^* = \operatorname*{argmin}_{\psi \in \Psi} d_{FGW;\alpha}\left(\psi_\sharp \mathbf{x}, \mathbf{u}\right) + \lambda \|\psi\| \tag{6.2}$$

with λ the regularization weight and $\|\psi\|$ the norm of the ψ function. Note that, as $\mathbf{u}$ and $\psi_\# \mathbf{x}$ are distributions on the same space $\mathbb{R}^d$, the transport cost c is the Euclidean distance on $\mathbb{R}^d$.

In the following, only linear mappings ψ are considered for the sake of avoiding overfitting and facilitating the interpretation of the Metabu meta-features w.r.t. the manually designed meta-features. The norm of ψ is set to the L_1 norm of its weight vector.

Optimization setting. The efficient optimization of Equation 6.2 is achieved using a bilevel optimization formulation.

- For a given ψ, the inner optimization problem consists of minimizing $d_{FGW,\alpha}(\psi_\sharp \mathbf{x}, \mathbf{u})$ (Equation 6.1). Metabu leverages the optimization approach proposed in Xu et al. [2019b,a, 2020], also described in Section 6.2, to efficiently compute $d_{FGW,\alpha}$. In the experiments, the number of iterations for refining γ is set to 10 and Sinkhorn iterations to 5.

blurring the target representation.

- The outer optimization problem consists of optimizing ψ: The transport matrix γ is treated as a constant, and the outer objective function (Equation 6.2) is solved with ADAM optimizer [Kingma and Ba 2015] with learning rate 0.01, $\alpha = 0.5$ and $\lambda = 0.001$.

6.3.4 . Intrinsic dimension of the space of datasets

The main hyper-parameter of Metabu is the number d of meta-features needed to approximate the target representation. Indeed, d depends on the considered algorithm $\mathcal{A}$: the more diverse the target representations associated with datasets, the higher d needs to be. In the other extreme case (all datasets have similar target representations), the AutoML problem becomes trivial. The relation between the intrinsic dimensionality and the difficulty of the AutoML problem is complex, we shall return to it in Section 6.4.4.

To our best knowledge, measuring the intrinsic dimension of the dataset space w.r.t. a learning algorithm has not been tackled in the literature. The approach proposed to do so builds on Levina and Bickel [2005] and [Facco et al. 2017], exploiting the fact that the number of points in a hypersphere of radius r in dimension d increases like r^d. Then d provides a guaranteed approximation of the intrinsic dimensionality of the manifold where the x_is family lives [Facco et al. 2017]. A formal pseudo-code is provided in Algorithm 6.

It is commonplace to say that the good distance between any two items depends on the considered task. The original approach used in Metabu in order to estimate the intrinsic dimensionality of the dataset space, is to set the distance of two datasets to the 1-Wasserstein distance among their target representations.

Algorithm 6: Compute intrinsic dimension as in [Facco et al. 2017]

Procedure `Intrinsic_dim`(X)
  input : A set of points $X = \{x_0, x_1, \ldots, x_m\}$.
  output: Intrinsic dimension d.
  Let $x^{(1)}$ and $x^{(2)}$ be the first and second neighbors of $x \in X$.
  Compute $\mu_i = \frac{||x_i - x_i^{(2)}||^2}{||x_i - x_i^{(1)}||^2}$ for all x_i in X.
  Sort μ values in ascending order through a permutation σ.
  Compute $F(\mu_{\sigma(i)}) = \frac{i}{m}$ for $i = 1 \ldots m$.
  $d \leftarrow$ slope of the linear approximation on $\{(log(\mu_i), -log(1 - F(\mu_i)))|i = 1 \ldots m\}$

6.4 . Experiments

All material (code, data, instructions) is made available as publicly at https://github.com/luxusg1/metabu. Runtimes are measured on an Intel(R) Xeon(R) CPU E5-2660 v2 @ 2.20GHz.

6.4.1 . Experimental Settings

Goals of experiment. The goal of experiments is three-fold. First, we aim to assess the dataset neighborhoods induced by the Metabu meta-features (constructed on the top of the manually designed 135 meta-features from the literature) and the relevance of these dataset neighborhoods w.r.t. the AutoML problem.
The second goal of experiments is to assess the sensitivity of Metabu w.r.t. its own two hyper-parameters, the weight α used to balance the importance of the Wasserstein and Gromov-Wasserstein distances in FGW (Equation 6.1), and the regularization weight λ involved in the optimization of ψ (Equation 6.2). As said, the dimension of meta-features d is automatically determined using Algorithm 6, nevertheless, a complementary experiments investigating the sensitivity w.r.t d is shown in Appendix C.5.

The third goal is to gain some understanding of the dataset landscape, and see whether the Metabu meta-features give some hints into when a given ML algorithm or pipeline does well (its *niche*).

Baselines. The performances are assessed against three baselines: Auto-Sklearn meta-feature set [Feurer et al. 2015b], Landmark [Pfahringer et al. 2000] and SCOT [Bardenet et al. 2013] meta-feature sets. All meta-feature sets are detailed in Appendix C.3. An additional baseline is based on the uniform sampling of the hyper-parameter configuration space, for sanity check.

Tasks. Three tasks are considered to investigate the relevance of the Metabu meta-features. All reported performance values are measured using a Leave-One-Out process over dataset (detailed in Appendix C.1).
$\Rightarrow$ **Task 1**: *Capturing the target topology*.
This task aims to highlight the merits of Metabu meta-features on ranking neighbor datasets (w.r.t the target representation). For each test dataset, one considers its nearest neighbors w.r.t. the target topology (the 1-Wasserstein metric on the target representation), and its nearest neighbors w.r.t. the Euclidean distance on the Metabu and meta-feature sets. The alignment between both ordered lists is measured using the normalized discounted cumulative gain over the first k neighbors (NDCG@k) [Burges et al. 2005], with $5 \leq k \leq 35$. The performance indicator is the NDCG@k[5] averaged on test datasets.
$\Rightarrow$ **Task 2**: *AutoML with no performance model (Initialization)*.
The purpose of Task 2 is to assess the topology constructed in Task 1 with a simple AutoML strategy i.e recommending ML configurations of neighbor datasets.

[5]To recall, $DCG@k = \sum_{i=1}^{k} \frac{2^{r_i}-1}{log(i+1)}$, with $r_i \in \{0,1\}$, indicating if the i-th dataset neighbor is relevant or not w.r.t the ranking of target topology. The NDCG score is then obtained by normalizing with the ideal DCG.

Concretely, for each test dataset and each meta-feature set mf, let $\hat{\mathbf{z}}_{mf}$ be the distribution on the considered hyper-parameter configuration space:

$$\hat{\mathbf{z}}_{mf} = \frac{1}{Z} \sum_{\ell=1}^{10} exp(-\ell)\, \mathbf{z}_\ell$$

where $\mathbf{z}_\ell$ is the target representation of the ℓ-th neighbor of the dataset w.r.t. Euclidean distance on the mf space, and Z a normalization constant. This distribution $\hat{\mathbf{z}}_{mf}$ is thereafter used to iteratively and independently sample the hyper-parameter configurations, and the performances of the learned models are measured. Letting $r(t, mf)$ denote the rank of the test performance[6] associated with meta-feature set mf after t iterations, the performance curves report $r(t, mf)$ for the Metabu and baseline meta-feature sets (plus a uniform hyper-parameter configuration sampler for sanity check), averaged over the test datasets.

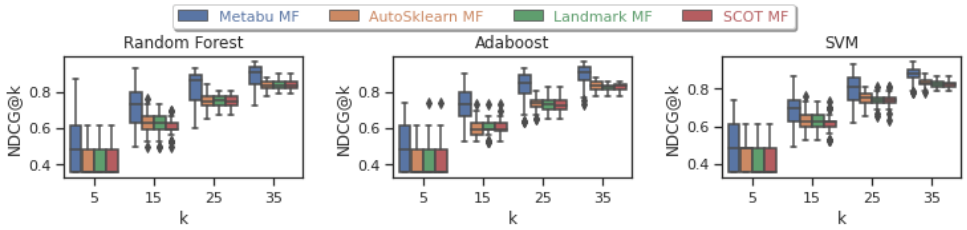
$\Rightarrow$ **Task 3**: *AutoML with performance model (Optimization)*.

AutoML systems based on performance models, such as Auto-Sklearn and PMF, cannot be directly compared with Metabu as they acquire additional information along the AutoML search: they iteratively use a performance model to select a hyper-parameter configuration, and update the performance model using the performance of the selected configuration. In Task 3, the relevance of Metabu meta-features is investigated in that they govern the initialization for Auto-Sklearn and PMF performance models. Precisely, the original meta-feature sets used in Auto-Sklearn and PMF are replaced with Metabu meta-features. Similarly to Task 2, the performance indicator is the rank of the performance obtained by Auto-Sklearn (resp. PMF) using Metabu meta-features to initialize its performance model, noted Metabu+Auto-Sklearn (resp. Metabu+PMF) compared to the original Auto-Sklearn (resp. PMF) implementation and the uniform baselines.

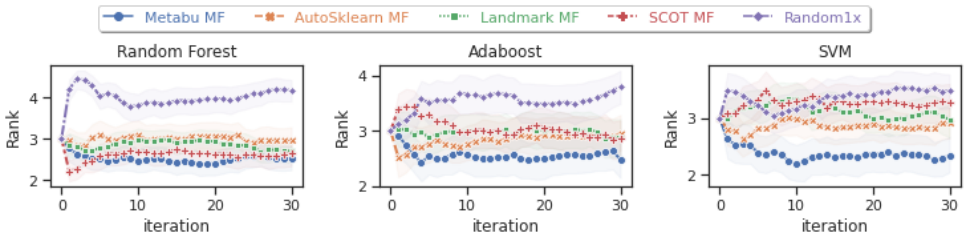
Dataset benchmark. The considered AutoML benchmark is the OpenML Curated Classification suite 2018 [Bischl et al. 2019], including 72 binary or multi-class datasets out of which 64 have enough learning performance data to give a good approximation of their target representation. The performance indicators are measured using Leave-One-Out (details in Appendix C.1). The basic meta-features are computed for each dataset using the open source library PyMFE [Alcobaca et al. 2020].

Hyper-parameter configuration spaces. Metabu is validated in the context of three ML algorithms: Adaboost [Freund and Schapire 1997], RandomForest [Breiman 2001] and SVM [Boser et al. 1992], using their Scikit-learn implementation [Pedregosa et al. 2011]; and two AutoML pipelines, Auto-Sklearn

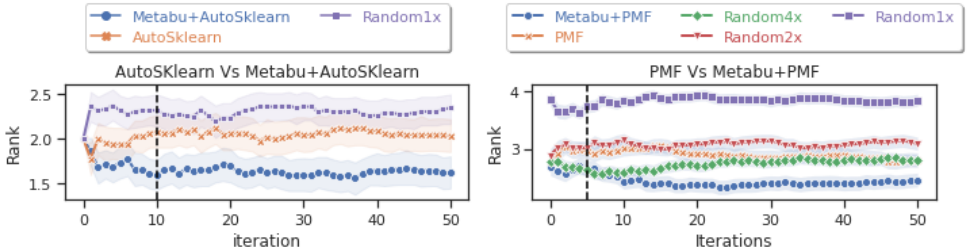
[6] Over the iteration, test performance is only observed when the validation performance is improved.



(a) Task 1: Capturing the target topology; the higher NDCG@k, the better.



(b) Task 2: Sampling the hyper-parameter configuration space; the lower the rank, the better.



(c) Task 3: Initializing a performance model to sample the hyper-parameter configuration space.

Figure 6.4: Empirical assessment of Metabu meta-features comparatively to the baselines meta-feature sets and uniform hyper-parameter sampling (better seen in color).

[Feurer et al. 2015a] and PMF [Fusi et al. 2018]. The associated hyper-parameter configuration spaces are detailed in Appendix C.2.

For Adaboost, RandomForest and SVM, the target representation of each training dataset is based on the top-20 configurations in OpenML (out of 37,289 for Adaboost, 81,336 for RandomForest and 37,075 for SVM), initially generated by van Rijn and Hutter [2018]. For Auto-Sklearn, the target representation is generated from scratch, running 500 configurations per training dataset and retaining the top-20. For PMF, the top-20 configurations are extracted from the collaborative filtering matrix for each training dataset [Fusi et al. 2018].

6.4.2 . Comparative empirical validation of Metabu

The performances of Metabu and the baselines on the three tasks are displayed on Figure 6.4. The overall CPU cost on Task 2 (resp. Task 3) is circa 1,900 (resp. 2,300) hours (full runtimes in Figure C.1). Appendix C.6 reports the detailed results in Tables C.6, C.7 and C.8, indicating the confidence level of the results after a Wilcoxon rank-sum test for performances and Mann Whitney Wilcoxon test for ranks; both with p-value set to 0.05.

$\Rightarrow$ **Task 1**: *Capturing the target topology*.

Figure 6.4a. The results show that the metric based on the Metabu meta-features better matches the target topology than the metric based on the baseline meta-feature sets, all the more so as the number k of nearest neighbors increases. The higher variance of NDCG@k for Metabu is explained as the metric depends on the meta-feature training, while the metrics based on the baselines are deterministic. As could be expected, this variance decreases with k. Despite this variance, Metabu significantly outperforms all baselines for all k and all hyper-parameter configuration spaces.

$\Rightarrow$ **Task 2**: *AutoML with no performance model (Initialization)*.

Figure 6.4b. All rank curves start at 3, as five hyper-parameter configuration samplers are considered. For RandomForest, the sampler based on the SCOT meta-feature set dominates in the first 5 iterations, and remains good at all time; Metabu dominates after the beginning; all other approaches but the uniform sampler yield similar performances. For Adaboost, the sampler based on the Auto-Sklearn meta-feature set dominates in the first 3 iterations, and Metabu is statistically significantly better than all other approaches thereafter. For SVM, Metabu very significantly dominates all other approaches.

$\Rightarrow$ **Task 3**: *AutoML with performance model (Optimization)*.

Figure 6.4c. In first time steps (left of the dashed bars), the performance models of Auto-Sklearn or PMF are initialized using the performances of the hyper-parameter configurations sampled as in Task 2; in the following time steps, the hyper-parameter configurations are sampled using the performance model. The most striking result is that the Metabu+Auto-Sklearn rank improves on that of Auto-Sklearn (Figure 6.4c, left) although they only differ in the initialization of the performance model, and the Auto-Sklearn meta-feature set is optimized to Task 3. Likewise, the rank of Metabu+PMF improves on that of PMF (Figure 6.4c, right). The comparison also involves Random2$\times$ and Random4$\times$ uniform samplers, respectively returning the best performance out of 2 or 4 uniformly sampled configurations [Fusi et al. 2018]; Metabu+PMF significantly improves on Random4$\times$ after the 10th iteration. This suggests that on the OpenML benchmark, the Metabu meta-features efficiently enable both to passively sample the hyper-parameter configuration space, and to retrieve the configurations best appropriate to update the performance model and explore good regions of the space.

6.4.3 . Sensitivity analysis

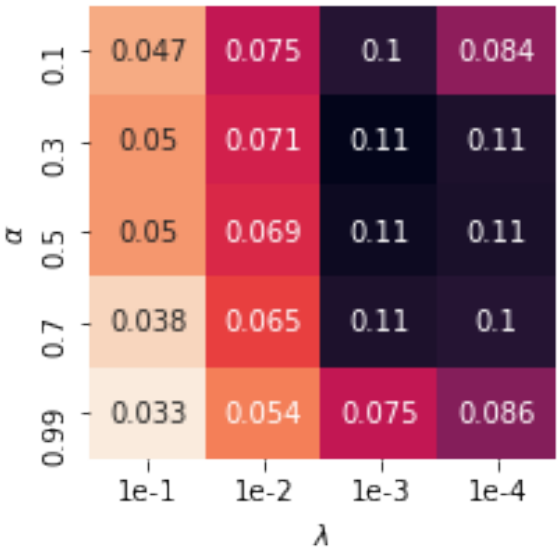


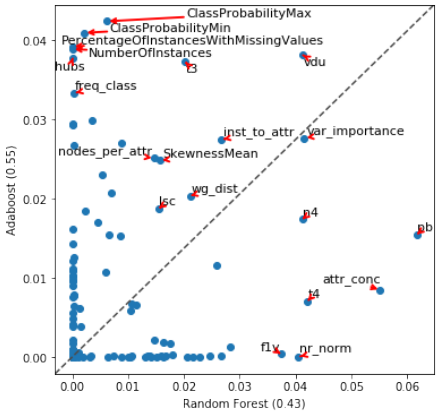
Figure 6.5: Metabu: Sensitivity of NDCG@10 w.r.t. α and λ, comparatively to Auto-Sklearn (darker is better).

The two hyper-parameters of Metabu are the α trade-off parameter between Wasserstein and Gromov-Wasserstein distance (Equation 6.1) and the regularization weight λ (Equation 6.2). The sensitivity of Metabu w.r.t. both parameters is investigated on Task 1, by inspecting the difference NDCG@10(Metabu) - NDCG@10(Auto-Sklearn) for α ranging in $\{0.1, 0.3, 0.5, 0.7, 0.99\}$ and λ in $\{10^{-1}, \ldots, 10^{-4}\}$. The result, displayed in Figure 6.5, shows that the difference is positive in the whole considered domain, with NDCG@10(Metabu) statistically significantly better than NDCG@10(Auto-Sklearn) according to Student t-test with p-value 0.05.

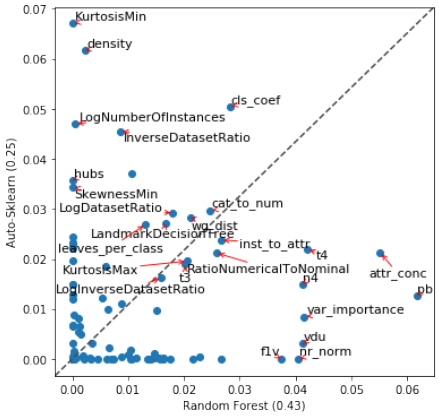
Interestingly, a low sensitivity of Metabu is observed w.r.t. the regularisation weight λ, provided that it is small enough ($\lambda \leq 10^{-3}$). For such small λ values, a low sensitivity is also observed w.r.t. α in a large range ($.3 \leq \alpha \leq .7$). This result confirms the importance of taking into account both the Wasserstein and Gromov-Wasserstein distances on the target representation space: discarding the former ($\alpha \leq .1$) or the latter ($\alpha \geq .99$) significantly degrades the performance, and the performance is stable in the $[.3, .7]$ region.

6.4.4 . Toward understanding the dataset landscape

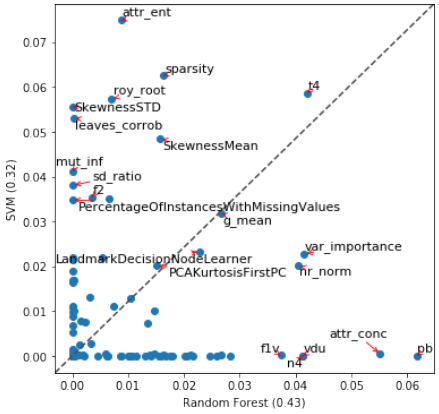
Insight on intrinsic dimension A most interesting result, that is original to our best knowledge, is to provide a principled estimate of the intrinsic dimension of the dataset space w.r.t. the considered ML algorithms. As detailed in Appendix C.5 with a stability analysis, the intrinsic dimension d of the OpenML benchmark is



(a) RandomForest *vs* Adaboost



(b) RandomForest *vs* Auto-Sklearn



(c) RandomForest *vs* Support Vector Machines

Figure 6.6: Comparative importance of meta-features for RandomForest Vs Adaboost (a), Auto-Sklearn (b) and Support Vector Machines (c). The specific Auto-Sklearn meta-features are recognized as their name begins with a capital letter.

circa 6 for Auto-Sklearn, 8 for Adaboost, 9 for RandomForest and 14 for Support Vector Machines. We are surprised indeed to see that the intrinsic dimension corresponding to Support Vector Machines is higher than for Auto-Sklearn, although the Auto-Sklearn portfolio includes the Support Vector Machines algorithm. This fact seems to suggest that the AutoML problem is harder for Auto-Sklearn than for Support Vector Machines, which is inconsistent as the Auto-Sklearn portfolio includes the Support Vector Machines algorithm.

A tentative explanation [7] goes as follows. We distinguish, as factors of the AutoML difficulty, the difficulty of hyper-parameter optimization, and the regularity of the landscape, that is, the fact that configurations good for a dataset are also generally good for a near dataset. The computation of intrinsic dimension only depends on the regularity of the landscape (each dataset being represented with its top hyper-parameters). In other words, the intrinsic dimension essentially reflects the diversity of the set of top hyper-parameters across the benchmark (OpenML CC-18 in our case). Overall, the low intrinsic dimension of Auto-Sklearn is interpreted as: many datasets have similar sets of top hyper-parameters; inversely, the high intrinsic dimension of Support Vector Machines is interpreted as the top hyper-parameters highly vary with the dataset. But this dimensionality does not measure the difficulty of the optimization part (reaching these good configurations).

Importance of hand-crafted meta-features Metabu also delivers some insights into what matters in the dataset landscape, and why a given algorithm should behave better than another on a particular dataset, as follows. Since Metabu meta-features are built from the initial hand-crafted meta-features using the trained linear mapping ψ and depending on the current learning algorithm $\mathcal{A}$, therefore the importance of these initial meta-features can be recovered by inspecting the newly learned Metabu meta-features.

Concretely, the importance of a meta-feature w.r.t. $\mathcal{A}$ is estimated as follows. Let $U = \{u_{i,j}\}$ denote the matrix made of the Multi-Dimensional Scaling representation of the secondary representation over all datasets. The matrix U are then processed using PCA, and let j^* be the index of the column contributing to the first PCA axis. Finally, the importance of a hand-crafted meta-feature $i_{\mathcal{A}}(mf)$ is measured from the norm of its projection on j^*-th column i.e, $i_{\mathcal{A}}(k) = |\psi_{j^*,k}|$.

Two ML algorithms or pipelines $\mathcal{A}$ and $\mathcal{B}$ can thus be visually compared, by plotting each meta-feature as a 2D point with coordinates $(i_{\mathcal{A}}(mf), i_{\mathcal{B}}(mf))$ as shown on Figure 6.6. For instance, in Figure 6.6a, with respectively $\mathcal{A}$ set to RandomForest and $\mathcal{B}$ to Adaboost, one sees that actually few features matter for both RandomForest and Adaboost (the features nearest to the upper right corner), mostly the Dunn index [Dunn 1973] and the features importance. Some findings reassuringly confirm the practitioner's expertise: the percentage of instances with

[7] We are grateful to the anonymous ICLR reviewer, who challenged us to explain this surprising result.

missing values matters much more for Adaboost than for RandomForest; the class imbalance (ClassProbabilityMax and ClassProbabilityMin) matters for Adaboost.

According to Figure 6.6b, meta-features such as KurtosisMin, LogNumberOfInstances, InverseDatasetRatio – all retained as Auto-Sklearn meta-features – are critical for Auto-Sklearn whereas they have no impact for RandomForest. Inversely, some features like "pb" (average Pearson correlation between class and features) matter significantly more for RandomForest than for Auto-Sklearn.

Likewise, the meta-feature importance w.r.t. Support Vector Machines and Random Forest is displayed in Figure 6.6c. The skewness (mean and std over all attributes) matter significantly more for Support Vector Machines than for RandomForest.

Overall, the impact of some meta-features for some learning algorithms is rather intuitive, confirming the practitioner expertise.

6.5 . Partial Conclusion

Metabu provides a partial but promising answer to the AutoML problem. On one hand, it yields new meta-features, preserving (in the sense of Fused Gromov-Wasserstein distance) the topology of the best configurations associated to each dataset. On the other hand, it also provides insights on the importance of hand-crafted meta-features, as well as the intrinsic dimension of an AutoML benchmark.

Metabu successfully addresses Dida limitations (Chapter 5), chiefly, the ability to deal with ordinary datasets and to handle the poor representativity of the AutoML benchmarks. These achievements are made possible as the learned meta-features are but linear combinations of the manually designed meta-features of the literature.

The efficiency of the approach is empirically demonstrated as the Metabu meta-features contribute to outperform strong baselines meta-features, improving state-of-the-art AutoML systems such as Auto-Sklearn [Feurer et al. 2015b] and PMF [Fusi et al. 2018].

Limitations. Although Metabu yields strong empirical performances, results from Task 3 show that the exploration of the configuration space (HPO algorithms) still yields a better performance than the initial configurations provided by the Metabu meta-features. This limitation is interpreted as the fact that the hand-crafted meta-features are insufficiently diverse to represent the true regions of interest with sufficiently fine granularity. Another interpretation is that the training data (the top configurations of the benchmark datasets) is too noisy, e.g. due to overfitting the CV score.

Another practical limitation of Metabu is that the learned meta-features are specific to an ML algorithm, i.e. its configuration space. Naturally, one might think of concatenating the Metabu meta-features related to the main algorithms, though

a naive concatenation does not preserve the topology of any such configuration space.

Perspectives. Along this line, a main perspective for further research is to propose a truly unified meta-feature space, merging the meta-features built from the various ML algorithms. A primary step is to investigate how the topology of the datasets differs depending on the learning algorithm: showing that two datasets are close neighbors in the landscape associated to an ML algorithm, and quite far apart for another ML algorithm might give some (expert- or learning-driven) insights into new meta-features. Comparing these landscapes is, to our best knowledge, an under-explored research area.

Another perspective is to leverage the same Metabu approach, aimed to find a representation aligned with a target topology, but to consider another representation and another target topology. A promising approach to improve the target representation is to take into account the distribution of bad hyper-parameters, or to consider surrogate performance model.

Discussion and Conclusion

As AI and Machine Learning are acknowledged a key technology for the digital age, the issue of delivering peak performances from the great many algorithms on the shelf emerged as a main bottleneck, as early as the end 1980s. Part I has proposed an overview of the research in this area, referred to as AutoML. It is important to note that the development of this field mostly relies on experimental studies: to our best knowledge, there does not exist such things as formal proofs that a given algorithm, with a given configuration, be the best one (w.r.t. other algorithms) on a given dataset.

In this context, the state of the art in AutoML pursues two main research directions: optimization, that is, searching for the best algorithms and configurations w.r.t. a dataset or a distribution of datasets (Part II); and meta-learning, that is, searching for a good representation of datasets, conducive to find, e.g. a good initialization for an optimization algorithm (Part III).

In this thesis, the two directions have been considered, yielding three contributions, respectively described in Chapters 4-6.

In Chapter 4, the proposed Mosaic tackles the fact that the optimization problem is defined on a mixed search space, involving binary and continuous coordinates, where the binary (structure, also referred to as pipeline) part commands the structure of the continuous (referred to as hyper-parameters) part, and the optimization objective strongly depends on the interactions among both parts. While most current state-of-the-art algorithms combine the pipeline selection and hyper-parameter optimization within the same optimization problem (referred to as CASH, Section 2.2), Mosaic tackles the structural and continuous optimization problems by using dedicated approaches for each problem, and enforcing their efficient combination. Specifically, Mosaic handles the structural (pipeline) optimization part using a Monte-Carlo Tree Search algorithm, and it handles the hyper-parameter (continuous) optimization part with Bayesian Optimization. The tight coupling of both optimization modules is enforced through a shared surrogate performance model, exploited and maintained by both modules.

The experiments on OpenML benchmark, containing 100 classification tasks, comparing Mosaic with Auto-Sklearn and TPOT AutoML systems, suggest that the proposed method outperforms its competitors Rakotoarison et al. [2019]. The detailed inspection of the results suggest that this better performance is due to the efficient exploration/exploitation strategy, early discarding unpromising regions and refining the search in the promising ones.

The main limitation of the approach, showed in the experiments, is that Mosaic performances significantly depend on the initialization of the search. This finding motivates the two further contributions, devoted to meta-learning. Only tabular data (as opposed to e.g., images) have been considered in the presented work.

In Chapter 5, our first meta-learning contribution called Dida is presented, leveraging distributional neural networks in order to learn meta-features. This research direction involves two steps: i) defining a meta-learning problem, on the space of datasets (e.g., recognizing whether two datasets are extracted at least partially from the same joint distribution); ii) solving this meta-learning problem using distributional NNs. The sought meta-features then consist of the functions defined as the nodes on the last layers of the distributional NN; they are computed from the dataset itself, viewed as a discrete distribution.
This line of research required some advances, to account for the specific structure of datasets (e.g., the distinction among features and labels; the invariance w.r.t. the permutation of the examples, and the features). Some proofs of concept have been obtained along this line De Bie et al. [2020], showing that on two meta-learning problems (patch classification and performance prediction tasks), Dida yields superior empirical performances compared to Dataset2Vec and DSS. However, Dida faces a main difficulty, which is the shortage of dataset benchmark. Dida basically is a neural net, and thus requires a significant number of examples (here, datasets) to be trained. Furthermore, it hardly takes into account pathological datasets (e.g., with missing values), hindering its application in a real-world AutoML setting.

In Chapter 6, a second contribution to meta-learning called Metabu is presented, aimed to address Dida limitations. Basically, Metabu relies on the great many hand-crafted meta-features, and it learns combinations thereof accounting for the "oracle" topology among datasets, relatively to a particular ML algorithm. This oracle topology is defined by considering that two datasets are similar iff the hyper-parameter configurations delivering peak performance for these datasets are similar. More precisely, optimal transport (OT) is leveraged to define the new meta-features enforcing the oracle topology, using a Fused Gromov Wasserstein OT approach [Vayer et al. 2019]. As Metabu operates on the top of the existing hand-crafted meta-features, it requires less datasets than Dida. Nevertheless, we had to define a (meta) data augmentation strategy and consider the new datasets defined by boostrap from the original OpenML datasets. Another important aspect in Metabu is that it automatically defines the number of meta-features to learn, by estimating the intrinsic dimension of the dataset space [Facco et al. 2017]. This estimation, the first of its kind to our best knowledge, is a first step toward characterizing the AutoML landscape. Another facet of Metabu is that it sheds some light onto what matters when comparing two ML algorithms, in terms of the relative importance of the hand-crafted meta-features. Though the findings only confirm at the moment the long known tricks of the trade (e.g. rather use decision trees or random forests in case of a high fraction of missing values), they might deliver more hints into the comparative strengths of the considered algorithms.

Quite a few perspectives for further research, focusing on the extensions of the presented approaches have been described in the partial conclusion of Chapters 4-6. Stepping back and looking at the overall picture, it is suggested that the ultimate goal for AutoML is to be able to understand when and why to recommend a given ML algorithm. At the moment, AutoML systems mostly proceed by conducting an optimization process, where only the initialization step relies on learned models. Still, a general trend in ML is toward learning explainable models, or at least, explaining the model decisions. While AutoML seems still far from building explainable models, it is suggested that learning explainable meta-features constitutes one significant step toward this aim. Along this line, these meta-features support a visualization of the benchmark datasets, that is amenable to assess the coverage of a benchmark and/or the quality of the experimental validation for a new algorithms.

Lastly, another perspective is to reconsider and extend the configuration space itself. On one hand, Mosaic can be easily adapted to handle variable size pipelines [Wever et al. 2018a], describing the search space in terms of a grammar taking inspiration from [Marinescu et al. 2021].

Bibliography

Salisu Mamman Abdulrahman, Pavel Brazdil, Jan N. van Rijn, and Joaquin Vanschoren. Speeding up algorithm selection using average ranking and active testing by introducing runtime. *Machine Learning*, 107(1):79–108, January 2018. ISSN 1573-0565. doi: 10.1007/s10994-017-5687-8. URL https://doi.org/10.1007/s10994-017-5687-8.

Alan Agresti. *Categorical data analysis*. Wiley series in probability and statistics. Wiley-Interscience, New York, 2nd ed edition, 2002. ISBN 9780471360933.

Ahmed Alaa and Mihaela Schaar. AutoPrognosis: Automated Clinical Prognostic Modeling via Bayesian Optimization with Structured Kernel Learning. In *International Conference on Machine Learning*, pages 139–148. PMLR, July 2018. URL https://proceedings.mlr.press/v80/alaa18b.html.

Edesio Alcobaca, Felipe Siqueira, Adriano Rivolli, Luis P. F. Garcia, Jefferson T. Oliva, and Andre C. P. L. F. de Carvalho. MFE: Towards reproducible meta-feature extraction. *JOURNAL OF MACHINE LEARNING RESEARCH*, 21, 2020. ISSN 1532-4435. URL https://bv.fapesp.br/en/publicacao/182490/mfe-towards-reproducible-meta-feature-extraction.

Shawkat Ali and Kate A. Smith-Miles. A meta-learning approach to automatic kernel selection for support vector machines. *Neurocomputing*, 70(1): 173–186, December 2006. ISSN 0925-2312. doi: 10.1016/j.neucom.2006.03.004. URL https://www.sciencedirect.com/science/article/pii/S0925231206001056.

David M. Allen. The Relationship between Variable Selection and Data Agumentation and a Method for Prediction. *Technometrics*, 16(1):125–127, 1974. ISSN 0040-1706. doi: 10.2307/1267500. URL https://www.jstor.org/stable/1267500.

David Alvarez-Melis and Nicolò Fusi. Geometric Dataset Distances via Optimal Transport. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020. URL https://proceedings.neurips.cc/paper/2020/hash/f52a7b2610fb4d3f74b4106fb80b233d-Abstract.html.

David Alvarez-Melis and Nicolò Fusi. Dataset Dynamics via Gradient Flows in Probability Space. In *Proceedings of the 38th International Conference on Machine*

Learning, pages 219–230. PMLR, July 2021. URL https://proceedings.mlr.press/v139/alvarez-melis21a.html.

Peter Auer, Nicolò Cesa-Bianchi, and Paul Fischer. Finite-time Analysis of the Multiarmed Bandit Problem. *Machine Learning*, 47(2):235–256, May 2002a. ISSN 1573-0565. doi: 10.1023/A:1013689704352. URL https://doi.org/10.1023/A:1013689704352.

Peter Auer, Nicolò Cesa-Bianchi, Yoav Freund, and Robert E. Schapire. The Nonstochastic Multiarmed Bandit Problem. *SIAM Journal on Computing*, 32 (1):48–77, January 2002b. ISSN 0097-5397. doi: 10.1137/S0097539701398375. URL https://epubs.siam.org/doi/10.1137/S0097539701398375.

Noor Awad, Neeratyoy Mallik, and Frank Hutter. DEHB: Evolutionary Hyberband for Scalable, Robust and Efficient Hyperparameter Optimization. volume 3, pages 2147–2153. IJCAI, August 2021. doi: 10.24963/ijcai.2021/296. URL https://www.ijcai.org/proceedings/2021/296.

Woonhyuk Baek, Ildoo Kim, Sungwoong Kim, and Sungbin Lim. AutoCLINT: The Winning Method in AutoCV Challenge 2019. *arXiv:2005.04373 [cs]*, May 2020. URL http://arxiv.org/abs/2005.04373. arXiv: 2005.04373.

Adithya Balaji and Alexander Allen. Benchmarking Automatic Machine Learning Frameworks. *arXiv:1808.06492 [cs, stat]*, August 2018. URL http://arxiv.org/abs/1808.06492. arXiv: 1808.06492.

Rémi Bardenet, Mátyás Brendel, Balázs Kégl, and Michèle Sebag. Collaborative hyperparameter tuning. In *International Conference on Machine Learning*, pages 199–207. PMLR, May 2013. URL https://proceedings.mlr.press/v28/bardenet13.html.

John Bather and Herman Chernoff. Sequential decisions in the control of a spaceship. *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability, Volume 3: Physical Sciences*, 5.3: 181–208, January 1967. URL https://projecteuclid.org/ebooks/berkeley-symposium-on-mathematical-statistics-and-probability/Proceedings-of-the-Fifth-Berkeley-Symposium-on-Mathematical-Statistics-and/chapter/Sequential-decisions-in-the-control-of-a-spaceship/bsmsp/1200513627.

Adrian El Baz, Isabelle Guyon, Zhengying Liu, Jan N. van Rijn, Sebastien Treguer, and Joaquin Vanschoren. MetaDL challenge design and baseline results. In *AAAI Workshop on Meta-Learning and MetaDL Challenge*, pages 1–16. PMLR, August 2021. URL https://proceedings.mlr.press/v140/el-baz21a.html.

Shai Ben-David, John Blitzer, Koby Crammer, Alex Kulesza, Fernando Pereira, and Jennifer Wortman Vaughan. A theory of learning from different domains. *Machine Learning*, 79(1):151–175, May 2010. ISSN 1573-0565. doi: 10.1007/s10994-009-5152-4. URL https://doi.org/10.1007/s10994-009-5152-4.

Hilan Bensusan and Christophe G. Giraud-Carrier. Discovering Task Neighbourhoods Through Landmark Learning Performances. In *Proceedings of the 4th European Conference on Principles of Data Mining and Knowledge Discovery*, PKDD '00, pages 325–330, Berlin, Heidelberg, September 2000. Springer-Verlag. ISBN 9783540410669.

Hilan Bensusan, Christophe Giraud-Carrier, and Claire Kennedy. A higher-order approach to meta-learning. Technical Report, University of Bristol, GBR, 2000.

James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for Hyper-Parameter Optimization. In *Advances in Neural Information Processing Systems*, volume 24. Curran Associates, Inc., 2011. URL https://papers.nips.cc/paper/2011/hash/86e8f7ab32cfd12577bc2619bc635690-Abstract.html.

James Bergstra, Daniel Yamins, and David Cox. Making a Science of Model Search: Hyperparameter Optimization in Hundreds of Dimensions for Vision Architectures. In *International Conference on Machine Learning*, pages 115–123. PMLR, February 2013. URL https://proceedings.mlr.press/v28/bergstra13.html.

Hans-Georg Beyer and Hans-Paul Schwefel. Evolution strategies – A comprehensive introduction. *Natural Computing*, 1(1):3–52, March 2002. ISSN 1572-9796. doi: 10.1023/A:1015059928466. URL https://doi.org/10.1023/A:1015059928466.

Rohan Bhardwaj, Ankita R. Nambiar, and Debojyoti Dutta. A Study of Machine Learning in Healthcare. In *2017 IEEE 41st Annual Computer Software and Applications Conference (COMPSAC)*, volume 2, pages 236–241, July 2017. doi: 10.1109/COMPSAC.2017.164. ISSN: 0730-3157.

Bernd Bischl, Giuseppe Casalicchio, Matthias Feurer, Frank Hutter, Michel Lang, Rafael G. Mantovani, Jan N. van Rijn, and Joaquin Vanschoren. OpenML Benchmarking Suites and the OpenML100. *arXiv:1708.03731 [cs, stat]*, August 2017. URL http://arxiv.org/abs/1708.03731. arXiv: 1708.03731 version: 1.

Bernd Bischl, Giuseppe Casalicchio, Matthias Feurer, Frank Hutter, Michel Lang, Rafael G. Mantovani, Jan N. van Rijn, and Joaquin Vanschoren. OpenML Benchmarking Suites. *arXiv:1708.03731 [cs, stat]*, September 2019. URL http://arxiv.org/abs/1708.03731. arXiv: 1708.03731.

Koen Van der Blom, Alex Serban, Holger Hoos, and Joost Visser. AutoML Adoption in ML Software. May 2021. URL https://openreview.net/forum?id=D5H5LjwvIqt.

Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D. Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, Xin Zhang, Jake Zhao, and Karol Zieba. End to End Learning for Self-Driving Cars. *arXiv:1604.07316 [cs]*, April 2016. URL http://arxiv.org/abs/1604.07316. arXiv: 1604.07316.

Bernhard E. Boser, Isabelle M. Guyon, and Vladimir N. Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the fifth annual workshop on Computational learning theory*, COLT '92, pages 144–152, Pittsburgh, Pennsylvania, USA, July 1992. Association for Computing Machinery. ISBN 9780897914970. doi: 10.1145/130385.130401. URL https://doi.org/10.1145/130385.130401.

Olivier Bousquet, Sylvain Gelly, Karol Kurach, Olivier Teytaud, and Damien Vincent. Critical Hyper-Parameters: No Random, No Cry. June 2017. URL https://arxiv.org/abs/1706.03200v1.

Pavel Brazdil and Christophe Giraud-Carrier. Metalearning and Algorithm Selection: progress, state of the art and introduction to the 2018 Special Issue. *Machine Learning*, 107(1):1–14, January 2018. ISSN 1573-0565. doi: 10.1007/s10994-017-5692-y. URL https://doi.org/10.1007/s10994-017-5692-y.

Vinicius Andrade Brei. Machine Learning in Marketing: Overview, Learning Strategies, Applications, and Future Developments. *Foundations and Trends® in Marketing*, 14(3):173–236, August 2020. ISSN 1555-0753, 1555-0761. doi: 10.1561/1700000065. URL https://www.nowpublishers.com/article/Details/MKT-065.

Leo Breiman. Random Forests. *Machine Learning*, 45(1):5–32, October 2001. ISSN 1573-0565. doi: 10.1023/A:1010933404324. URL https://doi.org/10.1023/A:1010933404324.

Eric Brochu, Vlad M. Cora, and Nando de Freitas. A Tutorial on Bayesian Optimization of Expensive Cost Functions, with Application to Active User Modeling and Hierarchical Reinforcement Learning. *arXiv:1012.2599 [cs]*, December 2010. URL http://arxiv.org/abs/1012.2599. arXiv: 1012.2599.

Sébastien Bubeck, Rémi Munos, Gilles Stoltz, and Csaba Szepesvári. *X*-Armed Bandits. *The Journal of Machine Learning Research*, 12(null):1655–1695, July 2011. ISSN 1532-4435.

Chris Burges, Tal Shaked, Erin Renshaw, Ari Lazier, Matt Deeds, Nicole Hamilton, and Greg Hullender. Learning to rank using gradient descent. In *Proceedings of the 22nd international conference on Machine learning*, ICML '05, pages 89–96, New York, NY, USA, August 2005. Association for Computing Machinery. ISBN 9781595931801. doi: 10.1145/1102351.1102363. URL https://doi.org/10.1145/1102351.1102363.

T. Caliński and J Harabasz. A dendrite method for cluster analysis. *Communications in Statistics*, 3(1):1–27, January 1974. ISSN 0090-3272. doi: 10.1080/03610927408827101. URL https://www.tandfonline.com/doi/abs/10.1080/03610927408827101.

Rich Caruana, Alexandru Niculescu-Mizil, Geoff Crew, and Alex Ksikes. Ensemble selection from libraries of models. In *Proceedings of the twenty-first international conference on Machine learning*, ICML '04, page 18, Banff, Alberta, Canada, July 2004. Association for Computing Machinery. ISBN 9781581138382. doi: 10.1145/1015330.1015432. URL https://doi.org/10.1145/1015330.1015432.

Ciro Castiello, Giovanna Castellano, and Anna Maria Fanelli. Meta-data: Characterization of Input Features for Meta-learning. In Vicenç Torra, Yasuo Narukawa, and Sadaaki Miyamoto, editors, *Modeling Decisions for Artificial Intelligence*, Lecture Notes in Computer Science, pages 457–468, Berlin, Heidelberg, 2005. Springer. ISBN 9783540318835. doi: 10.1007/11526018_45.

Marie-Liesse Cauwet, Camille Couprie, Julien Dehos, Pauline Luc, Jérémy Rapin, Morgane Rivière, Fabien Teytaud, Olivier Teytaud, and Nicolas Usunier. Fully Parallel Hyperparameter Search: Reshaped Space-Filling. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, volume 119 of *Proceedings of Machine Learning Research*, pages 1338–1348. PMLR, 2020. URL http://proceedings.mlr.press/v119/cauwet20a.html.

Guillaume Chaslot, Sander Bakkes, Istvan Szita, and Pieter Spronck. Monte-Carlo Tree Search: A New Framework for Game AI. In Christian Darken and Michael Mateas, editors, *Proceedings of the Fourth Artificial Intelligence and Interactive Digital Entertainment Conference, October 22-24, 2008, Stanford, California, USA*. The AAAI Press, 2008. URL http://www.aaai.org/Library/AIIDE/2008/aiide08-036.php.

Boyuan Chen, Harvey Wu, Warren Mo, Ishanu Chattopadhyay, and Hod Lipson. Autostacker: a compositional evolutionary learning system. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '18, pages 402–409, Kyoto, Japan, July 2018. Association for Computing Machinery. ISBN

9781450356183. doi: 10.1145/3205455.3205586. URL https://doi.org/10.1145/3205455.3205586.

Noam Chomsky. On Formalization and Formal Linguistics. *Natural Language & Linguistic Theory*, 8(1):143–147, 1990. ISSN 0167-806X. URL https://www.jstor.org/stable/4047755.

Edoardo Conti, Vashisht Madhavan, Felipe Petroski Such, Joel Lehman, Kenneth O. Stanley, and Jeff Clune. Improving exploration in evolution strategies for deep reinforcement learning via a population of novelty-seeking agents. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS'18, pages 5032–5043, Montréal, Canada, December 2018. Curran Associates Inc.

Pierre-Arnaud Coquelin and Rémi Munos. Bandit algorithms for tree search. In *Proceedings of the Twenty-Third Conference on Uncertainty in Artificial Intelligence*, UAI'07, pages 67–74, Vancouver, BC, Canada, July 2007. AUAI Press. ISBN 9780974903934.

Nicolas Courty, Rémi Flamary, Devis Tuia, and Alain Rakotomamonjy. Optimal Transport for Domain Adaptation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(9):1853–1865, September 2017. ISSN 1939-3539. doi: 10.1109/TPAMI.2016.2615921.

Adrien Couëtoux, Mario Milone, Mátyás Brendel, Hassan Doghmen, Michèle Sebag, and Olivier Teytaud. Continuous Rapid Action Value Estimates. In *Asian Conference on Machine Learning*, pages 19–31. PMLR, November 2011. URL https://proceedings.mlr.press/v20/couetoux11.html.

David A Cox, John N Little, and Donal O'Shea. *Ideals, varieties, and algorithms: an introduction to computational algebraic geometry and commutative algebra*. 2018. ISBN 9783319374277. OCLC: 1241676194.

Marco Cuturi. Sinkhorn Distances: Lightspeed Computation of Optimal Transport. In *Advances in Neural Information Processing Systems*, volume 26. Curran Associates, Inc., 2013. URL https://papers.nips.cc/paper/2013/hash/af21d0c97db2e27e13572cbf59eb343d-Abstract.html.

G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, 2(4):303–314, December 1989. ISSN 1435-568X. doi: 10.1007/BF02551274. URL https://doi.org/10.1007/BF02551274.

Gwendoline De Bie, Gabriel Peyré, and Marco Cuturi. Stochastic Deep Networks. In *Proceedings of the 36th International Conference on Machine Learning*, pages

1556–1565. PMLR, May 2019. URL https://proceedings.mlr.press/v97/de-bie19a.html.

Gwendoline De Bie, Herilalaina Rakotoarison, Gabriel Peyré, and Michèle Sebag. Distribution-Based Invariant Deep Networks for Learning Meta-Features. *arXiv:2006.13708 [cs, stat]*, October 2020. URL http://arxiv.org/abs/2006.13708. arXiv: 2006.13708.

Alex G. C. de Sá, Walter José G. S. Pinto, Luiz Otavio V. B. Oliveira, and Gisele L. Pappa. RECIPE: A Grammar-Based Framework for Automatically Evolving Classification Pipelines. In James McDermott, Mauro Castelli, Lukas Sekanina, Evert Haasdijk, and Pablo García-Sánchez, editors, *Genetic Programming*, volume 10196, pages 246–261. Springer International Publishing, Cham, 2017. ISBN 9783319556956 9783319556963. doi: 10.1007/978-3-319-55696-3_16. URL http://link.springer.com/10.1007/978-3-319-55696-3_16.

Carl Doersch, Ankush Gupta, and Andrew Zisserman. CrossTransformers: spatially-aware few-shot transfer. *arXiv:2007.11498 [cs]*, December 2020. URL http://arxiv.org/abs/2007.11498. arXiv: 2007.11498.

Tobias Domhan, Jost Tobias Springenberg, and Frank Hutter. Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves. In *Proceedings of the 24th International Conference on Artificial Intelligence*, IJCAI'15, pages 3460–3468, Buenos Aires, Argentina, July 2015. AAAI Press. ISBN 9781577357384.

Iddo Drori, Yamuna Krishnamurthy, Raoni Lourenco, Remi Rampin, Kyunghyun Cho, Claudio Silva, and Juliana Freire. Automatic Machine Learning by Pipeline Synthesis using Model-Based Reinforcement Learning and a Grammar. *ICML Workshop on Automated Machine Learning*, May 2019. URL http://arxiv.org/abs/1905.10345. arXiv: 1905.10345.

Dheeru Dua and Casey Graff. {UCI} Machine Learning Repository, 2017. URL http://archive.ics.uci.edu/ml. University of California, Irvine, School of Information and Computer Sciences.

J. C. Dunn. A Fuzzy Relative of the ISODATA Process and Its Use in Detecting Compact Well-Separated Clusters. *Journal of Cybernetics*, 3(3):32–57, January 1973. ISSN 0022-0280. doi: 10.1080/01969727308546046. URL https://doi.org/10.1080/01969727308546046.

Daria Dzyabura and Hema Yoganarasimhan. Machine learning and marketing. *Handbook of Marketing Analytics*, March 2018. URL https://www.elgaronline.com/view/edcoll/9781784716745/9781784716745.00023.xml.

The Economist. The world's most valuable resource is no longer oil, but data. *The Economist*, May 2017. ISSN 0013-0613. URL https://www.economist.com/leaders/2017/05/06/the-worlds-most-valuable-resource-is-no-longer-oil-but-data.

B. Efron. Bootstrap Methods: Another Look at the Jackknife. *The Annals of Statistics*, 7(1), January 1979. ISSN 0090-5364. doi: 10.1214/aos/1176344552. URL https://projecteuclid.org/journals/annals-of-statistics/volume-7/issue-1/Bootstrap-Methods-Another-Look-at-the-Jackknife/10.1214/aos/1176344552.full.

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

Katharina Eggensperger, Philipp Müller, Neeratyoy Mallik, Matthias Feurer, Rene Sass, Aaron Klein, Noor Awad, Marius Lindauer, and Frank Hutter. HPOBench: A Collection of Reproducible Multi-Fidelity Benchmark Problems for HPO. August 2021. URL https://openreview.net/forum?id=1k4rJYEwda-.

Robert Engels and Christiane Theusinger. Using a Data Metric for Preprocessing Advice for Data Mining Applications. In *In Proceedings of the European Conference on Artificial Intelligence (ECAI-98*, pages 430–434. John Wiley & Sons, 1998.

Hugo Escalante, Wei-Wei Tu, Isabelle Guyon, Daniel Silver, Evelyne Viegas, Yuqiang Chen, Wenyuan Dai, and Qiang Yang. *AutoML @ NeurIPS 2018 challenge: Design and Results*. Springer Verlag, March 2020. ISBN 9783030291358. URL https://hal.inria.fr/hal-03159764.

Suilan Estevez-Velarde, Yoan Gutiérrez, Andrés Montoyo, and Yudivián Almeida-Cruz. AutoML Strategy Based on Grammatical Evolution: A Case Study about Knowledge Discovery from Text. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4356–4365, Florence, Italy, July 2019. Association for Computational Linguistics. doi: 10.18653/v1/P19-1428. URL https://aclanthology.org/P19-1428.

Elena Facco, Maria d'Errico, Alex Rodriguez, and Alessandro Laio. Estimating the intrinsic dimension of datasets by a minimal neighborhood information. *Scientific Reports*, 7(1):12140, September 2017. ISSN 2045-2322. doi: 10.1038/s41598-017-11873-y. URL https://www.nature.com/articles/s41598-017-11873-y.

Stefan Falkner, Aaron Klein, and Frank Hutter. BOHB: Robust and Efficient Hyperparameter Optimization at Scale. In *International Conference on Machine Learning*, pages 1437–1446. PMLR, July 2018. URL https://proceedings.mlr.press/v80/falkner18a.html.

Matthias Feurer and Frank Hutter. Hyperparameter Optimization. In Frank Hutter, Lars Kotthoff, and Joaquin Vanschoren, editors, *Automated Machine Learning: Methods, Systems, Challenges*, The Springer Series on Challenges in Machine Learning, pages 3–33. Springer International Publishing, Cham, 2019. ISBN 9783030053185. doi: 10.1007/978-3-030-05318-5_1. URL https://doi.org/10.1007/978-3-030-05318-5_1.

Matthias Feurer, Aaron Klein, Katharina Eggensperger, Jost Springenberg, Manuel Blum, and Frank Hutter. Efficient and Robust Automated Machine Learning. In *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc., 2015a. URL https://papers.nips.cc/paper/2015/hash/11d0e6287202fced83f79975ec59a3a6-Abstract.html.

Matthias Feurer, Jost Tobias Springenberg, and Frank Hutter. Initializing bayesian hyperparameter optimization via meta-learning. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, AAAI'15, pages 1128–1135, Austin, Texas, January 2015b. AAAI Press. ISBN 9780262511292.

Matthias Feurer, Katharina Eggensperger, S. Falkner, M. Lindauer, and F. Hutter. Practical Automated Machine Learning for the AutoML Challenge 2018. In *AutoML Workshop*, 2018. URL https://www.semanticscholar.org/paper/Practical-Automated-Machine-Learning-for-the-AutoML-Feurer-Eggensperger/12b48ce5a5cb66fc92e8c5b7aa4a651bb4e98a55.

Matthias Feurer, Katharina Eggensperger, Stefan Falkner, Marius Lindauer, and Frank Hutter. Auto-Sklearn 2.0: Hands-free AutoML via Meta-Learning. *arXiv:2007.04074 [cs, stat]*, September 2021a. URL http://arxiv.org/abs/2007.04074. arXiv: 2007.04074.

Matthias Feurer, Benjamin Letham, Frank Hutter, and Eytan Bakshy. Practical Transfer Learning for Bayesian Optimization. *arXiv:1802.02219 [cs, stat]*, April 2021b. URL http://arxiv.org/abs/1802.02219. arXiv: 1802.02219.

Álvaro Fialho, Luís Da Costa, Marc Schoenauer, and Michèle Sebag. Extreme Value Based Adaptive Operator Selection. In Günter Rudolph, Thomas Jansen, Nicola Beume, Simon Lucas, and Carlo Poloni, editors, *Parallel Problem Solving from Nature – PPSN X*, Lecture Notes in Computer Science, pages 175–184, Berlin, Heidelberg, 2008. Springer. ISBN 9783540877004. doi: 10.1007/978-3-540-87700-4_18.

Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks. In *ICML*, 2017.

Forbes. Council Post: Data Is The New Oil – And That's A Good Thing, 2018. URL https://www.forbes.com/sites/forbestechcouncil/2019/11/15/data-is-the-new-oil-and-thats-a-good-thing/.

Luca Franceschi, Paolo Frasconi, Saverio Salzo, Riccardo Grazzi, and Massimiliano Pontil. Bilevel Programming for Hyperparameter Optimization and Meta-Learning. In *International Conference on Machine Learning*, pages 1568–1577. PMLR, July 2018. URL https://proceedings.mlr.press/v80/franceschi18a.html.

Peter I. Frazier. A Tutorial on Bayesian Optimization. *arXiv:1807.02811 [cs, math, stat]*, July 2018. URL http://arxiv.org/abs/1807.02811. arXiv: 1807.02811.

Yoav Freund and Robert E Schapire. A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting. *Journal of Computer and System Sciences*, 55(1):119–139, August 1997. ISSN 0022-0000. doi: 10.1006/jcss.1997.1504. URL https://www.sciencedirect.com/science/article/pii/S002200009791504X.

Nicolo Fusi, Rishit Sheth, and Melih Elibol. Probabilistic Matrix Factorization for Automated Machine Learning. In *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018. URL https://proceedings.neurips.cc/paper/2018/hash/b59a51a3c0bf9c5228fde841714f523a-Abstract.html.

Matteo Gagliolo and Jürgen Schmidhuber. Algorithm Selection as a Bandit Problem with Unbounded Losses. In Christian Blum and Roberto Battiti, editors, *Learning and Intelligent Optimization*, Lecture Notes in Computer Science, pages 82–96, Berlin, Heidelberg, 2010. Springer. ISBN 9783642138003. doi: 10.1007/978-3-642-13800-3_7.

Adam Gaier and David Ha. Weight Agnostic Neural Networks. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://papers.nips.cc/paper/2019/hash/e98741479a7b998f88b8f8c9f0b6b6f1-Abstract.html.

Yaroslav Ganin, Evgeniya Ustinova, Hana Ajakan, Pascal Germain, Hugo Larochelle, François Laviolette, Mario Marchand, and Victor Lempitsky. Domain-adversarial training of neural networks. *The Journal of Machine Learning Research*, 17(1):2096–2030, January 2016. ISSN 1532-4435.

Seymour Geisser. The Predictive Sample Reuse Method with Applications. *Journal of the American Statistical Association*, 70(350):320–328, 1975. ISSN 0162-1459. doi: 10.2307/2285815. URL https://www.jstor.org/stable/2285815.

Sylvain Gelly and David Silver. Combining online and offline knowledge in UCT. In *Proceedings of the 24th international conference on Machine learning*, ICML '07, pages 273–280, Corvalis, Oregon, USA, June 2007. Association for Computing Machinery. ISBN 9781595937933. doi: 10.1145/1273496.1273531. URL https://doi.org/10.1145/1273496.1273531.

Sylvain Gelly and David Silver. Monte-Carlo tree search and rapid action value estimation in computer Go. *Artificial Intelligence*, 175(11):1856–1875, July 2011. ISSN 0004-3702. doi: 10.1016/j.artint.2011.03.007. URL https://www.sciencedirect.com/science/article/pii/S000437021100052X.

Pieter Gijsbers and Joaquin Vanschoren. GAMA: Genetic Automated Machine learning Assistant. *Journal of Open Source Software*, 4(33):1132, January 2019. ISSN 2475-9066. doi: 10.21105/joss.01132. URL https://joss.theoj.org/papers/10.21105/joss.01132.

Pieter Gijsbers, Erin LeDell, Janek Thomas, Sébastien Poirier, Bernd Bischl, and Joaquin Vanschoren. An Open Source AutoML Benchmark. *arXiv:1907.00909 [cs, stat]*, July 2019. URL http://arxiv.org/abs/1907.00909. arXiv: 1907.00909.

Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pages 249–256. JMLR Workshop and Conference Proceedings, March 2010. URL https://proceedings.mlr.press/v9/glorot10a.html.

Jean-Bastien Grill, Florian Strub, Florent Altché, Corentin Tallec, Pierre H. Richemond, Elena Buchatskaya, Carl Doersch, Bernardo Ávila Pires, Zhaohan Guo, Mohammad Gheshlaghi Azar, Bilal Piot, Koray Kavukcuoglu, Rémi Munos, and Michal Valko. Bootstrap Your Own Latent - A New Approach to Self-Supervised Learning. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020. URL https://proceedings.neurips.cc/paper/2020/hash/f3ada80d5c4ee70142b17b8192b2958e-Abstract.html.

Isabelle Guyon, Kristin Bennett, Gavin Cawley, Hugo Jair Escalante, Sergio Escalera, Tin Kam Ho, Núria Macià, Bisakha Ray, Mehreen Saeed, Alexander

Statnikov, and Evelyne Viegas. Design of the 2015 ChaLearn AutoML challenge. In *2015 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, July 2015. doi: 10.1109/IJCNN.2015.7280767. ISSN: 2161-4407.

Nikolaus Hansen. The CMA Evolution Strategy: A Tutorial. 2005. URL https://hal.inria.fr/hal-01297037.

Jason Hartford, Devon Graham, Kevin Leyton-Brown, and Siamak Ravanbakhsh. Deep Models of Interactions Across Sets. In *Proceedings of the 35th International Conference on Machine Learning*, pages 1909–1918. PMLR, July 2018. URL https://proceedings.mlr.press/v80/hartford18a.html.

Tin Kam Ho and M. Basu. Complexity measures of supervised classification problems. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(3): 289–300, March 2002. ISSN 1939-3539. doi: 10.1109/34.990132.

Sepp Hochreiter and Jürgen Schmidhuber. Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780, November 1997. ISSN 0899-7667. doi: 10.1162/neco.1997.9.8.1735. URL https://doi.org/10.1162/neco.1997.9.8.1735.

Holger Hoos, Roland Kaminski, Marius Lindauer, and Torsten Schaub. aspeed: Solver scheduling via answer set programming 1. *Theory and Practice of Logic Programming*, 15(1):117–142, January 2015. ISSN 1471-0684, 1475-3081. doi: 10.1017/S1471068414000015. URL https://www.cambridge.org/core/journals/theory-and-practice-of-logic-programming/article/abs/aspeed-solver-scheduling-via-answer-set-programming-1/FBFF4331304A6841F966C2AFA9E66D55.

Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. Automated Configuration of Mixed Integer Programming Solvers. In Andrea Lodi, Michela Milano, and Paolo Toth, editors, *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, Lecture Notes in Computer Science, pages 186–202, Berlin, Heidelberg, 2010. Springer. ISBN 9783642135200. doi: 10.1007/978-3-642-13520-0_23.

Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. Sequential Model-Based Optimization for General Algorithm Configuration. In Carlos A. Coello Coello, editor, *Learning and Intelligent Optimization*, Lecture Notes in Computer Science, pages 507–523, Berlin, Heidelberg, 2011. Springer. ISBN 9783642255663. doi: 10.1007/978-3-642-25566-3_40.

Frank Hutter, Holger Hoos, and Kevin Leyton-Brown. An Efficient Approach for Assessing Hyperparameter Importance. In *Proceedings of the 31st International Conference on Machine Learning*, pages 754–762. PMLR, January 2014. URL https://proceedings.mlr.press/v32/hutter14.html.

Frank Hutter, Lars Kotthoff, and Joaquin Vanschoren, editors. *Automated Machine Learning: Methods, Systems, Challenges*. The Springer Series on Challenges in Machine Learning. Springer International Publishing, Cham, 2019. ISBN 9783030053178 9783030053185. doi: 10.1007/978-3-030-05318-5. URL http://link.springer.com/10.1007/978-3-030-05318-5.

Max Jaderberg, Valentin Dalibard, Simon Osindero, Wojciech M. Czarnecki, Jeff Donahue, Ali Razavi, Oriol Vinyals, Tim Green, Iain Dunning, Karen Simonyan, Chrisantha Fernando, and Koray Kavukcuoglu. Population Based Training of Neural Networks. *arXiv:1711.09846 [cs]*, November 2017. URL http://arxiv.org/abs/1711.09846. arXiv: 1711.09846.

Hadi S. Jomaa, Lars Schmidt-Thieme, and Josif Grabocka. Dataset2Vec: learning dataset meta-features. *Data Mining and Knowledge Discovery*, 35(3):964–985, May 2021. ISSN 1573-756X. doi: 10.1007/s10618-021-00737-9. URL https://doi.org/10.1007/s10618-021-00737-9.

Donald R. Jones, Matthias Schonlau, and William J. Welch. Efficient Global Optimization of Expensive Black-Box Functions. *Journal of Global Optimization*, 13(4):455–492, December 1998. ISSN 1573-2916. doi: 10.1023/A:1008306431147. URL https://doi.org/10.1023/A:1008306431147.

A. Kalousis and M. Hilario. Model selection via meta-learning: a comparative study. In *Proceedings 12th IEEE Internationals Conference on Tools with Artificial Intelligence. ICTAI 2000*, pages 406–413, November 2000. doi: 10.1109/TAI.2000.889901. ISSN: 1082-3409.

Borhan Kazimipour, Xiaodong Li, and A. K. Qin. Initialization methods for large scale global optimization. In *2013 IEEE Congress on Evolutionary Computation*, pages 2750–2757, June 2013. doi: 10.1109/CEC.2013.6557902. ISSN: 1941-0026.

Nicolas Keriven and Gabriel Peyré. Universal Invariant and Equivariant Graph Neural Networks. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://papers.nips.cc/paper/2019/hash/ea9268cb43f55d1d12380fb6ea5bf572-Abstract.html.

Pascal Kerschke, Holger H. Hoos, Frank Neumann, and Heike Trautmann. Automated Algorithm Selection: Survey and Perspectives. *Evolutionary Computation*, 27(1):3–45, March 2019. ISSN 1063-6560. doi: 10.1162/evco_a_00242. URL https://doi.org/10.1162/evco_a_00242.

Jörg-Uwe Kietz, Floarea Serban, Abraham Bernstein, and Simon Fischer. Designing KDD-workflows via HTN-planning. In *Proceedings of the 20th European Conference on Artificial Intelligence*, ECAI'12, pages 1011–1012, Montpellier, France, August 2012. IOS Press. ISBN 9781614990970.

Jungtaek Kim, Saehoon Kim, and Seungjin Choi. Learning to Warm-Start Bayesian Hyperparameter Optimization. *arXiv:1710.06219 [cs, stat]*, October 2018a. URL http://arxiv.org/abs/1710.06219. arXiv: 1710.06219.

Jungtaek Kim, Saehoon Kim, and Seungjin Choi. Learning to Warm-Start Bayesian Hyperparameter Optimization. *arXiv:1710.06219 [cs, stat]*, October 2018b. URL http://arxiv.org/abs/1710.06219. arXiv: 1710.06219.

Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. URL http://arxiv.org/abs/1412.6980.

Aaron Klein, Stefan Falkner, Simon Bartels, Philipp Hennig, and Frank Hutter. Fast Bayesian Optimization of Machine Learning Hyperparameters on Large Datasets. In *Artificial Intelligence and Statistics*, pages 528–536. PMLR, April 2017. URL https://proceedings.mlr.press/v54/klein17a.html.

Levente Kocsis and Csaba Szepesvári. Bandit Based Monte-Carlo Planning. In Johannes Fürnkranz, Tobias Scheffer, and Myra Spiliopoulou, editors, *Machine Learning: ECML 2006*, Lecture Notes in Computer Science, pages 282–293, Berlin, Heidelberg, 2006. Springer. ISBN 9783540460565. doi: 10.1007/11871842_29.

Brent Komer, James Bergstra, and Chris Eliasmith. Hyperopt-Sklearn: Automatic Hyperparameter Configuration for Scikit-Learn. *Proceedings of the 13th Python in Science Conference*, pages 32–37, 2014. doi: 10.25080/Majora-14bd3278-006. URL https://conference.scipy.org/proceedings/scipy2014/komer.html.

Risi Kondor and Shubhendu Trivedi. On the Generalization of Equivariance and Convolution in Neural Networks to the Action of Compact Groups. In Jennifer G. Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pages 2752–2760. PMLR, 2018. URL http://proceedings.mlr.press/v80/kondor18a.html.

Lars Kotthoff. Algorithm Selection for Combinatorial Search Problems: A Survey. In Christian Bessiere, Luc De Raedt, Lars Kotthoff, Siegfried Nijssen, Barry O'Sullivan, and Dino Pedreschi, editors, *Data Mining and Constraint Programming: Foundations of a Cross-Disciplinary Approach*, Lecture Notes in Computer Science, pages 149–190. Springer International Publishing, Cham,

2016. ISBN 9783319501376. doi: 10.1007/978-3-319-50137-6_7. URL https://doi.org/10.1007/978-3-319-50137-6_7.

Lars Kotthoff, Pascal Kerschke, Holger Hoos, and Heike Trautmann. Improving the State of the Art in Inexact TSP Solving Using Per-Instance Algorithm Selection. In Clarisse Dhaenens, Laetitia Jourdan, and Marie-Eléonore Marmion, editors, *Learning and Intelligent Optimization*, Lecture Notes in Computer Science, pages 202–217, Cham, 2015. Springer International Publishing. ISBN 9783319190846. doi: 10.1007/978-3-319-19084-6_18.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. ImageNet classification with deep convolutional neural networks. In *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'12, pages 1097–1105, Lake Tahoe, Nevada, December 2012. Curran Associates Inc.

J. B. Kruskal. Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis. *Psychometrika*, 29(1):1–27, March 1964. ISSN 1860-0980. doi: 10.1007/BF02289565. URL https://doi.org/10.1007/BF02289565.

John Langford and Tong Zhang. The Epoch-Greedy Algorithm for Multi-armed Bandits with Side Information. In *Advances in Neural Information Processing Systems*, volume 20. Curran Associates, Inc., 2008. URL https://papers.nips.cc/paper/2007/hash/4b04a686b0ad13dce35fa99fa4161c65-Abstract.html.

Tor Lattimore and Csaba Szepesvári. *Bandit Algorithms*. Cambridge University Press, Cambridge, 2020. ISBN 9781108486828. doi: 10.1017/9781108571401. URL https://www.cambridge.org/core/books/bandit-algorithms/8E39FD004E6CE036680F90DD0C6F09FC.

Moshe Leshno, Vladimir Ya. Lin, Allan Pinkus, and Shimon Schocken. Multilayer feedforward networks with a nonpolynomial activation function can approximate any function. *Neural Networks*, 6(6):861–867, January 1993. ISSN 08936080. doi: 10.1016/S0893-6080(05)80131-5. URL https://linkinghub.elsevier.com/retrieve/pii/S0893608005801315.

Elizaveta Levina and Peter Bickel. Maximum Likelihood Estimation of Intrinsic Dimension. In *Advances in Neural Information Processing Systems*, volume 17. MIT Press, 2005. URL https://papers.nips.cc/paper/2004/hash/74934548253bcab8490ebd74afed7031-Abstract.html.

Liam Li, Kevin Jamieson, Afshin Rostamizadeh, Ekaterina Gonina, Jonathan Bentzur, Moritz Hardt, Benjamin Recht, and Ameet Talwalkar. A System for Massively Parallel Hyperparameter Tuning. *Proceedings of Machine Learning and*

Systems, 2:230–246, March 2020. URL https://proceedings.mlsys.org/paper/2020/hash/f4b9ec30ad9f68f89b29639786cb62ef-Abstract.html.

Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. Hyperband: a novel bandit-based approach to hyperparameter optimization. *The Journal of Machine Learning Research*, 18(1):6765–6816, January 2017. ISSN 1532-4435.

Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. Hyperband: A Novel Bandit-Based Approach to Hyperparameter Optimization. *Journal of Machine Learning Research*, 18(185):1–52, 2018. ISSN 1533-7928. URL http://jmlr.org/papers/v18/16-558.html.

Sungbin Lim, Ildoo Kim, Taesup Kim, Chiheon Kim, and Sungwoong Kim. Fast AutoAugment. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://papers.nips.cc/paper/2019/hash/6add07cf50424b14fdf649da87843d01-Abstract.html.

Marius Lindauer, Katharina Eggensperger, Matthias Feurer, André Biedenkapp, Joshua Marben, Philipp Müller, and Frank Hutter. BOAH: A Tool Suite for Multi-Fidelity Bayesian Optimization & Analysis of Hyperparameters. *arXiv:1908.06756 [cs, stat]*, August 2019. URL http://arxiv.org/abs/1908.06756. arXiv: 1908.06756.

Marius Lindauer, Katharina Eggensperger, Matthias Feurer, André Biedenkapp, Difan Deng, Carolin Benjamins, Tim Ruhkopf, René Sass, and Frank Hutter. SMAC3: A Versatile Bayesian Optimization Package for Hyperparameter Optimization. *Journal of Machine Learning Research*, 23(54):1–9, 2022. ISSN 1533-7928. URL http://jmlr.org/papers/v23/21-0888.html.

Hanxiao Liu, Karen Simonyan, and Yiming Yang. DARTS: Differentiable Architecture Search. September 2018a. URL https://openreview.net/forum?id=S1eYHoC5FX.

Jialin Liu, Antoine Moreau, Mike Preuss, Jeremy Rapin, Baptiste Roziere, Fabien Teytaud, and Olivier Teytaud. Versatile black-box optimization. In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*, GECCO '20, pages 620–628, Cancún, Mexico, June 2020a. Association for Computing Machinery. ISBN 9781450371285. doi: 10.1145/3377930.3389838. URL https://doi.org/10.1145/3377930.3389838.

Sijia Liu, Parikshit Ram, Deepak Vijaykeerthy, Djallel Bouneffouf, Gregory Bramble, Horst Samulowitz, Dakuo Wang, Andrew Conn, and Alexander Gray. An ADMM Based Framework for AutoML Pipeline Configuration. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(04):4892–4899, April 2020b.

ISSN 2374-3468. doi: 10.1609/aaai.v34i04.5926. URL https://ojs.aaai.org/index.php/AAAI/article/view/5926.

Zhengying Liu. *Automated Deep Learning : Principles and Practice*. phdthesis, Université Paris-Saclay, November 2021. URL https://tel.archives-ouvertes.fr/tel-03464519.

Zhengying Liu, Olivier Bousquet, André Elisseeff, Sergio Escalera, Isabelle Guyon, Julio Jacques, Adrien Pavao, Danny Silver, Lisheng Sun-Hosoya, Sebastien Treguer, Wei-Wei Tu, Jingsong Wang, and Quanming Yao. AutoDL Challenge Design and Beta Tests-Towards automatic deep learning. In *MetaLearn workshop @ NeurIPS2018*, Montreal, Canada, December 2018b. URL https://hal.archives-ouvertes.fr/hal-01906226.

Zhengying Liu, Isabelle Guyon, Julio Jacques Junior, Meysam Madadi, Sergio Escalera, Adrien Pavao, Hugo Jair Escalante, Wei-Wei Tu, Zhen Xu, and Sebastien Treguer. AutoCV Challenge Design and Baseline Results. In *CAp 2019 - Conférence sur l'Apprentissage Automatique*, Toulouse, France, July 2019. URL https://hal.archives-ouvertes.fr/hal-02265053.

Ana C. Lorena, Luís P. F. Garcia, Jens Lehmann, Marcilio C. P. Souto, and Tin Kam Ho. How Complex Is Your Classification Problem? A Survey on Measuring Classification Complexity. *ACM Computing Surveys*, 52(5):107:1–107:34, September 2019. ISSN 0360-0300. doi: 10.1145/3347711. URL https://doi.org/10.1145/3347711.

Ilya Loshchilov and Frank Hutter. CMA-ES for Hyperparameter Optimization of Deep Neural Networks. *CoRR*, abs/1604.07269, 2016. URL http://arxiv.org/abs/1604.07269. arXiv: 1604.07269.

Manuel Loth, Michèle Sebag, Youssef Hamadi, and Marc Schoenauer. Bandit-Based Search for Constraint Programming. In Christian Schulte, editor, *Principles and Practice of Constraint Programming*, Lecture Notes in Computer Science, pages 464–480, Berlin, Heidelberg, 2013. Springer. ISBN 9783642406270. doi: 10.1007/978-3-642-40627-0_36.

Laurens van der Maaten and Geoffrey Hinton. Visualizing Data using t-SNE. *Journal of Machine Learning Research*, 9(86):2579–2605, 2008. ISSN 1533-7928. URL http://jmlr.org/papers/v9/vandermaaten08a.html.

Radu Marinescu, Akihiro Kishimoto, Parikshit Ram, Ambrish Rawat, Martin Wistuba, Paulito P. Palmes, and Adi Botea. Searching for Machine Learning Pipelines Using a Context-Free Grammar. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35(10):8902–8911, May 2021. ISSN 2374-3468. URL https://ojs.aaai.org/index.php/AAAI/article/view/17077.

Haggai Maron, Ethan Fetaya, Nimrod Segol, and Yaron Lipman. On the Universality of Invariant Networks. In *Proceedings of the 36th International Conference on Machine Learning*, pages 4363–4371. PMLR, May 2019. URL https://proceedings.mlr.press/v97/maron19a.html.

Haggai Maron, Or Litany, Gal Chechik, and Ethan Fetaya. On Learning Sets of Symmetric Elements. In *Proceedings of the 37th International Conference on Machine Learning*, pages 6734–6744. PMLR, November 2020. URL https://proceedings.mlr.press/v119/maron20a.html.

Laurent Meunier, Herilalaina Rakotoarison, Pak Kan Wong, Baptiste Roziere, Jeremy Rapin, Olivier Teytaud, Antoine Moreau, and Carola Doerr. Black-Box Optimization Revisited: Improving Algorithm Selection Wizards through Massive Benchmarking. *IEEE Transactions on Evolutionary Computation*, 2021. URL https://hal.inria.fr/hal-03154019.

Donald Michie, D. J. Spiegelhalter, and C. C. Taylor, editors. *Machine learning, neural and statistical classification*. Ellis Horwood series in artificial intelligence. Ellis Horwood, New York, 1994. ISBN 9780131063600.

Risto Miikkulainen, Jason Liang, Elliot Meyerson, Aditya Rawal, Daniel Fink, Olivier Francon, Bala Raju, Hormoz Shahrzad, Arshak Navruzyan, Nigel Duffy, and Babak Hodjat. Evolving Deep Neural Networks. In *Artificial Intelligence in the Age of Neural Networks and Brain Computing*, pages 293–312. Elsevier, 2019. ISBN 9780128154809. doi: 10.1016/B978-0-12-815480-9.00015-3. URL https://linkinghub.elsevier.com/retrieve/pii/B9780128154809000153.

Mustafa Misir and Michèle Sebag. Alors: An algorithm recommender system. *Artificial Intelligence*, 244:291–314, March 2017. ISSN 0004-3702. doi: 10.1016/j.artint.2016.12.001. URL https://www.sciencedirect.com/science/article/pii/S0004370216301436.

Melanie Mitchell. *An introduction to genetic algorithms*. Complex adaptive systems. MIT Press, Cambridge, Mass, 1996. ISBN 9780262133166.

Jonas Mockus. *Bayesian Approach to Global Optimization: Theory and Applications*. Mathematics and its Applications. Springer Netherlands, 1989. ISBN 9789401068987. doi: 10.1007/978-94-009-0909-0. URL https://www.springer.com/gp/book/9789401068987.

Felix Mohr, Marcel Wever, and Eyke Hüllermeier. ML-Plan: Automated machine learning via hierarchical planning. *Machine Learning*, 107(8):1495–1515, September 2018. ISSN 1573-0565. doi: 10.1007/s10994-018-5735-z. URL https://doi.org/10.1007/s10994-018-5735-z.

Mario A. Muñoz, Laura Villanova, Davaatseren Baatar, and Kate Smith-Miles. Instance spaces for machine learning classification. *Machine Learning*, 107(1): 109–147, January 2018. ISSN 1573-0565. doi: 10.1007/s10994-017-5629-5. URL https://doi.org/10.1007/s10994-017-5629-5.

Facundo Mémoli. Gromov–Wasserstein Distances and the Metric Approach to Object Matching. *Foundations of Computational Mathematics*, 11(4):417–487, August 2011. ISSN 1615-3383. doi: 10.1007/s10208-011-9093-5. URL https://doi.org/10.1007/s10208-011-9093-5.

Eugene Ndiaye, Tam Le, Olivier Fercoq, Joseph Salmon, and Ichiro Takeuchi. Safe Grid Search with Optimal Complexity. In *International Conference on Machine Learning*, pages 4771–4780. PMLR, May 2019. URL https://proceedings.mlr.press/v97/ndiaye19a.html.

Khai Nguyen, Son Nguyen, Nhat Ho, Tung Pham, and Hung Bui. Improving Relational Regularized Autoencoders with Spherical Sliced Fused Gromov Wasserstein. September 2020. URL https://openreview.net/forum?id=DiQD7FWL233.

P. Nguyen, M. Hilario, and A. Kalousis. Using Meta-mining to Support Data Mining Workflow Planning and Optimization. *Journal of Artificial Intelligence Research*, 51:605–644, November 2014. ISSN 1076-9757. doi: 10.1613/jair.4377. URL https://www.jair.org/index.php/jair/article/view/10918.

Randal S. Olson, Nathan Bartley, Ryan J. Urbanowicz, and Jason H. Moore. Evaluation of a Tree-based Pipeline Optimization Tool for Automating Data Science. In *Proceedings of the Genetic and Evolutionary Computation Conference 2016*, GECCO '16, pages 485–492, Denver, Colorado, USA, July 2016. Association for Computing Machinery. ISBN 9781450342063. doi: 10.1145/2908812.2908918. URL https://doi.org/10.1145/2908812.2908918.

Sinno Jialin Pan and Qiang Yang. A Survey on Transfer Learning. *IEEE Transactions on Knowledge and Data Engineering*, 22(10):1345–1359, October 2010. ISSN 1558-2191. doi: 10.1109/TKDE.2009.191.

Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Édouard Duchesnay. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12(85):2825–2830, 2011. ISSN 1533-7928. URL http://jmlr.org/papers/v12/pedregosa11a.html.

Yonghong Peng, Peter A. Flach, Carlos Soares, and Pavel Brazdil. Improved Dataset Characterisation for Meta-learning. In Steffen Lange, Ken Satoh, and

Carl H. Smith, editors, *Discovery Science*, Lecture Notes in Computer Science, pages 141–152, Berlin, Heidelberg, 2002. Springer. ISBN 9783540361824. doi: 10.1007/3-540-36182-0_14.

Gabriel Peyré and Marco Cuturi. Computational Optimal Transport: With Applications to Data Science. *Foundations and Trends® in Machine Learning*, 11 (5-6):355–607, February 2019. ISSN 1935-8237, 1935-8245. doi: 10.1561/2200000073. URL https://www.nowpublishers.com/article/Details/MAL-073.

Bernhard Pfahringer, Hilan Bensusan, and Christophe G. Giraud-Carrier. Meta-Learning by Landmarking Various Learning Algorithms. In Pat Langley, editor, *Proceedings of the Seventeenth International Conference on Machine Learning (ICML 2000), Stanford University, Stanford, CA, USA, June 29 - July 2, 2000*, pages 743–750. Morgan Kaufmann, 2000.

Florian Pfisterer, Jan N. van Rijn, Philipp Probst, Andreas Müller, and Bernd Bischl. Learning Multiple Defaults for Machine Learning Algorithms. *arXiv:1811.09409 [cs, stat]*, April 2021. URL http://arxiv.org/abs/1811.09409. arXiv: 1811.09409.

Bruno Almeida Pimentel and André C. P. L. F. de Carvalho. A new data characterization for selecting clustering algorithms using meta-learning. *Inf. Sci.*, 477: 203–219, 2019. doi: 10.1016/j.ins.2018.10.043.

Charles R. Qi, Hao Su, Kaichun Mo, and Leonidas J. Guibas. PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation. pages 652–660, 2017. URL https://openaccess.thecvf.com/content_cvpr_2017/html/Qi_PointNet_Deep_Learning_CVPR_2017_paper.html.

Qazi Ibadur Rahman and Gerhard Schmeisser. *Analytic theory of polynomials*. Number new ser., 26 in London Mathematical Society monographs. Clarendon Press ; Oxford University Press, Oxford : New York, 2002. ISBN 9780198534938.

Herilalaina Rakotoarison, Marc Schoenauer, and Michèle Sebag. Automated Machine Learning with Monte-Carlo Tree Search. pages 3296–3303, 2019. URL https://www.ijcai.org/proceedings/2019/457.

Herilalaina Rakotoarison, Louisot Milijaona, Andry Rasoanaivo, Michele Sebag, and Marc Schoenauer. Learning meta-features for AutoML. September 2021. URL https://openreview.net/forum?id=DTkEfj0Ygb8.

Jeremy Rapin and Olivier Teytaud. Nevergrad - A gradient-free optimization platform, 2018. URL https://GitHub.com/FacebookResearch/Nevergrad.

Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian Processes for Machine Learning*. Adaptive Computation and Machine Learning series. MIT Press, Cambridge, MA, USA, November 2005. ISBN 9780262182539.

Siamak Ravanbakhsh, Jeff Schneider, and Barnabás Póczos. Equivariance Through Parameter-Sharing. In *Proceedings of the 34th International Conference on Machine Learning*, pages 2892–2901. PMLR, July 2017. URL https://proceedings.mlr.press/v70/ravanbakhsh17a.html.

Esteban Real, Chen Liang, David So, and Quoc Le. AutoML-Zero: Evolving Machine Learning Algorithms From Scratch. In *International Conference on Machine Learning*, pages 8007–8019. PMLR, November 2020. URL https://proceedings.mlr.press/v119/real20a.html.

John R. Rice. The Algorithm Selection Problem. In Morris Rubinoff and Marshall C. Yovits, editors, *Advances in Computers*, volume 15, pages 65–118. Elsevier, January 1976. doi: 10.1016/S0065-2458(08)60520-3. URL https://www.sciencedirect.com/science/article/pii/S0065245808605203.

Jan N. van Rijn, Florian Pfisterer, Janek Thomas, Andreas Muller, Bernd Bischl, and J. Vanschoren. Meta learning for defaults: symbolic defaults. December 2018. URL https://research.tue.nl/en/publications/meta-learning-for-defaults-symbolic-defaults.

Sebastian Risi and Kenneth O. Stanley. An enhanced hypercube-based encoding for evolving the placement, density, and connectivity of neurons. *Artificial Life*, 18 (4):331–363, October 2012. ISSN 1064-5462. doi: 10.1162/ARTL_a_00071. URL https://doi.org/10.1162/ARTL_a_00071.

Adriano Rivolli, Luís P. F. Garcia, Carlos Soares, Joaquin Vanschoren, and André C. P. L. F. de Carvalho. Characterizing classification datasets: a study of meta-features for meta-learning. *arXiv:1808.10406 [cs, stat]*, August 2019. URL http://arxiv.org/abs/1808.10406. arXiv: 1808.10406.

Adriano Rivolli, Luís P. F. Garcia, Carlos Soares, Joaquin Vanschoren, and André C. P. L. F. de Carvalho. Meta-features for meta-learning. *Knowledge-Based Systems*, 240:108101, March 2022. ISSN 0950-7051. doi: 10.1016/j.knosys.2021.108101. URL https://www.sciencedirect.com/science/article/pii/S0950705121011631.

Herbert Robbins. Some aspects of the sequential design of experiments. *Bulletin of the American Mathematical Society*, 58(5):527–535, 1952. ISSN 0002-9904, 1936-881X. doi: 10.1090/S0002-9904-1952-09620-8. URL https://www.ams.org/bull/1952-58-05/S0002-9904-1952-09620-8/.

Mostafa A. Salama, Aboul Ella Hassanien, and Kenneth Revett. Employment of neural network and rough set in meta-learning. *Memetic Computing*, 5(3):165–177, September 2013. ISSN 1865-9292. doi: 10.1007/s12293-013-0114-6. URL https://doi.org/10.1007/s12293-013-0114-6.

Tim Salimans, Jonathan Ho, Xi Chen, Szymon Sidor, and Ilya Sutskever. Evolution Strategies as a Scalable Alternative to Reinforcement Learning. *arXiv:1703.03864 [cs, stat]*, September 2017. URL http://arxiv.org/abs/1703.03864. arXiv: 1703.03864.

Filippo Santambrogio. *Optimal Transport for Applied Mathematicians: Calculus of Variations, PDEs, and Modeling*. Number 87 in Progress in Nonlinear Differential Equations and Their Applications. Springer International Publishing, Cham, 1st ed. 2015 edition, 2015. ISBN 9783319208282.

Andrew W. Senior, Richard Evans, John Jumper, James Kirkpatrick, Laurent Sifre, Tim Green, Chongli Qin, Augustin **v**Zídek, Alexander W. R. Nelson, Alex Bridgland, Hugo Penedones, Stig Petersen, Karen Simonyan, Steve Crossan, Pushmeet Kohli, David T. Jones, David Silver, Koray Kavukcuoglu, and Demis Hassabis. Improved protein structure prediction using potentials from deep learning. *Nature*, 577(7792):706–710, January 2020. ISSN 1476-4687. doi: 10.1038/s41586-019-1923-7. URL https://www.nature.com/articles/s41586-019-1923-7.

Xuedong Shang, Emilie Kaufmann, and Michal Valko. A simple dynamic bandit algorithm for hyper-parameter tuning. June 2019. URL https://hal.inria.fr/hal-02145200.

J. Shawe-Taylor. Symmetries and discriminability in feedforward network architectures. *IEEE Transactions on Neural Networks*, 4(5):816–826, September 1993. ISSN 1941-0093. doi: 10.1109/72.248459.

David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, January 2016. ISSN 1476-4687. doi: 10.1038/nature16961. URL https://www.nature.com/articles/nature16961.

David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy Lillicrap, Fan Hui, Laurent Sifre, George van den

Driessche, Thore Graepel, and Demis Hassabis. Mastering the game of Go without human knowledge. *Nature*, 550(7676):354–359, October 2017. ISSN 1476-4687. doi: 10.1038/nature24270. URL https://www.nature.com/articles/nature24270.

Kate Smith-Miles and Leo Lopes. Measuring instance difficulty for combinatorial optimization problems. *Computers & Operations Research*, 39(5):875–889, May 2012. ISSN 0305-0548. doi: 10.1016/j.cor.2011.07.006. URL https://www.sciencedirect.com/science/article/pii/S0305054811001997.

Jake Snell, Kevin Swersky, and Richard S. Zemel. Prototypical Networks for Few-shot Learning. *arXiv:1703.05175 [cs, stat]*, June 2017. URL http://arxiv.org/abs/1703.05175. arXiv: 1703.05175.

Jasper Snoek, Hugo Larochelle, and Ryan P. Adams. Practical Bayesian optimization of machine learning algorithms. In *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'12, pages 2951–2959, Lake Tahoe, Nevada, December 2012. Curran Associates Inc.

Qinbao Song, Guangtao Wang, and Chao Wang. Automatic recommendation of classification algorithms based on data set characteristics. *Pattern Recognition*, 45(7):2672–2689, July 2012. ISSN 0031-3203. doi: 10.1016/j.patcog.2011.12.025. URL https://www.sciencedirect.com/science/article/pii/S0031320312000192.

Dimitris Souravlias, Konstantinos E. Parsopoulos, Ilias S. Kotsireas, and Panos M. Pardalos. *Algorithm Portfolios: Advances, Applications, and Challenges*. SpringerBriefs in Optimization. Springer International Publishing, 2021. ISBN 9783030685133. doi: 10.1007/978-3-030-68514-0. URL https://www.springer.com/gp/book/9783030685133.

Jost Tobias Springenberg, Aaron Klein, Stefan Falkner, and Frank Hutter. Bayesian Optimization with Robust Bayesian Neural Networks. In *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc., 2016. URL https://proceedings.neurips.cc/paper/2016/hash/a96d3afec184766bfeca7a9f989fc7e7-Abstract.html.

Kenneth O. Stanley and Risto Miikkulainen. Evolving Neural Networks through Augmenting Topologies. *Evolutionary Computation*, 10(2):99–127, June 2002. ISSN 1063-6560. doi: 10.1162/106365602320169811.

Kenneth O. Stanley, David B. D'Ambrosio, and Jason Gauci. A Hypercube-Based Encoding for Evolving Large-Scale Neural Networks. *Artificial Life*, 15(2):185–212, April 2009. ISSN 1064-5462. doi: 10.1162/artl.2009.15.2.15202.

Kenneth O. Stanley, Jeff Clune, Joel Lehman, and Risto Miikkulainen. Designing neural networks through neuroevolution. *Nature Machine Intelligence*, 1(1):24–35, January 2019. ISSN 2522-5839. doi: 10.1038/s42256-018-0006-z. URL https://www.nature.com/articles/s42256-018-0006-z.

Quan Sun and Bernhard Pfahringer. Pairwise meta-rules for better meta-learning-based algorithm ranking. *Machine Learning*, 93(1):141–161, October 2013. ISSN 1573-0565. doi: 10.1007/s10994-013-5387-y. URL https://doi.org/10.1007/s10994-013-5387-y.

Richard S. Sutton and Andrew G. Barto. *Introduction to Reinforcement Learning*. MIT Press, Cambridge, MA, USA, 1st edition, 1998. ISBN 9780262193986.

Kevin Swersky, Jasper Snoek, and Ryan P Adams. Multi-Task Bayesian Optimization. In *Advances in Neural Information Processing Systems*, volume 26. Curran Associates, Inc., 2013. URL https://papers.nips.cc/paper/2013/hash/f33ba15effa5c10e873bf3842afb46a6-Abstract.html.

Kevin Swersky, Jasper Snoek, and Ryan Prescott Adams. Freeze-Thaw Bayesian Optimization. *arXiv:1406.3896 [cs, stat]*, June 2014. URL http://arxiv.org/abs/1406.3896. arXiv: 1406.3896 version: 1.

William R. Thompson. On the Likelihood that One Unknown Probability Exceeds Another in View of the Evidence of Two Samples. *Biometrika*, 25(3/4):285–294, 1933. ISSN 0006-3444. doi: 10.2307/2332286. URL https://www.jstor.org/stable/2332286.

Chris Thornton, Frank Hutter, Holger H. Hoos, and Kevin 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*, KDD '13, pages 847–855, Chicago, Illinois, USA, August 2013. Association for Computing Machinery. ISBN 9781450321747. doi: 10.1145/2487575.2487629. URL https://doi.org/10.1145/2487575.2487629.

Ali Tizghadam, Hamzeh Khazaei, Mohammad H. Y. Moghaddam, and Yasser Hassan. Machine Learning in Transportation. *Journal of Advanced Transportation*, 2019:e4359785, June 2019. ISSN 0197-6729. doi: 10.1155/2019/4359785. URL https://www.hindawi.com/journals/jat/2019/4359785/.

Jan N. van Rijn and Frank Hutter. Hyperparameter Importance Across Datasets. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '18, pages 2367–2376, London, United Kingdom, July 2018. Association for Computing Machinery. ISBN 9781450355520. doi: 10.1145/3219819.3220058. URL https://doi.org/10.1145/3219819.3220058.

Joaquin Vanschoren. Meta-Learning. In Frank Hutter, Lars Kotthoff, and Joaquin Vanschoren, editors, *Automated Machine Learning: Methods, Systems, Challenges*, The Springer Series on Challenges in Machine Learning, pages 35–61. Springer International Publishing, Cham, 2019. ISBN 9783030053185. doi: 10.1007/978-3-030-05318-5_2. URL https://doi.org/10.1007/978-3-030-05318-5_2.

Joaquin Vanschoren, Jan N. van Rijn, Bernd Bischl, and Luis Torgo. OpenML: networked science in machine learning. *ACM SIGKDD Explorations Newsletter*, 15(2):49–60, June 2014. ISSN 1931-0145. doi: 10.1145/2641190.2641198. URL https://doi.org/10.1145/2641190.2641198.

Vladimir Vapnik. *The Nature of Statistical Learning Theory*. Information Science and Statistics. Springer-Verlag, New York, 2 edition, 2000. ISBN 9780387987804. doi: 10.1007/978-1-4757-3264-1. URL https://www.springer.com/gp/book/9780387987804.

Titouan Vayer, Nicolas Courty, Romain Tavenard, Chapel Laetitia, and Rémi Flamary. Optimal Transport for structured data with application on graphs. In *Proceedings of the 36th International Conference on Machine Learning*, pages 6275–6284. PMLR, May 2019. URL https://proceedings.mlr.press/v97/titouan19a.html.

Ricardo Vilalta. Understanding Accuracy Performance Through Concept Characterization and Algorithm Analysis. In *Workshop on Recent Advances in Meta-Learning and Future Work, 16th International Conference on Machine Learning*, pages 3–9, 1999.

Jingsong Wang, Tom Ko, Zhen Xu, Xiawei Guo, Souxiang Liu, Wei-Wei Tu, and Lei Xie. AutoSpeech 2020: The Second Automated Machine Learning Challenge for Speech Classification. *arXiv:2010.13130 [cs]*, October 2020. URL http://arxiv.org/abs/2010.13130. arXiv: 2010.13130.

Simon Wessing and Mike Preuss. The true destination of EGO is multi-local optimization. In *2017 IEEE Latin American Conference on Computational Intelligence (LA-CCI)*, pages 1–6, November 2017. doi: 10.1109/LA-CCI.2017.8285677.

Marcel Wever, Felix Mohr, and Eyke Hüllermeier. Automated Multi-Label Classification based on ML-Plan. *arXiv:1811.04060 [cs, stat]*, November 2018a. URL http://arxiv.org/abs/1811.04060. arXiv: 1811.04060.

Marcel Wever, Felix Mohr, and Eyke Hüllermeier. ML-Plan for Unlimited-Length Machine Learning Pipelines. 2018b.

Jenna Wiens and Erica S. Shenoy. Machine Learning for Healthcare: On the Verge of a Major Shift in Healthcare Epidemiology. *Clinical Infectious Diseases: An Official Publication of the Infectious Diseases Society of America*, 66(1):149–153, January 2018. ISSN 1537-6591. doi: 10.1093/cid/cix731.

Martin Wistuba, Nicolas Schilling, and Lars Schmidt-Thieme. Scalable Gaussian process-based transfer surrogates for hyperparameter optimization. *Machine Learning*, 107(1):43–78, January 2018. ISSN 1573-0565. doi: 10.1007/s10994-017-5684-y. URL https://doi.org/10.1007/s10994-017-5684-y.

D. Wolpert and W. Macready. No Free Lunch Theorems for Search. *undefined*, 1995. URL https://www.semanticscholar.org/paper/No-Free-Lunch-Theorems-for-Search-Wolpert-Macready/93a7f5b7f51622430a4a4d4002152b277e4be470.

David H. Wolpert. The Lack of A Priori Distinctions Between Learning Algorithms. *Neural Computation*, 8(7):1341–1390, October 1996. ISSN 0899-7667. doi: 10.1162/neco.1996.8.7.1341.

D.H. Wolpert and W.G. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, April 1997. ISSN 1941-0026. doi: 10.1109/4235.585893.

Michael Woodroofe. A One-Armed Bandit Problem with a Concomitant Variable. *Journal of the American Statistical Association*, 74(368):799–806, 1979. ISSN 0162-1459. doi: 10.2307/2286402. URL https://www.jstor.org/stable/2286402.

Hongteng Xu, Dixin Luo, and Lawrence Carin. Scalable Gromov-Wasserstein Learning for Graph Partitioning and Matching. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019a. URL https://proceedings.neurips.cc/paper/2019/hash/6e62a992c676f611616097dbea8ea030-Abstract.html.

Hongteng Xu, Dixin Luo, Hongyuan Zha, and Lawrence Carin Duke. Gromov-Wasserstein Learning for Graph Matching and Node Embedding. In *Proceedings of the 36th International Conference on Machine Learning*, pages 6932–6941. PMLR, May 2019b. URL https://proceedings.mlr.press/v97/xu19b.html.

Hongteng Xu, Dixin Luo, Ricardo Henao, Svati Shah, and Lawrence Carin. Learning Autoencoders with Relational Regularization. In *Proceedings of the 37th International Conference on Machine Learning*, pages 10576–10586. PMLR, November 2020. URL https://proceedings.mlr.press/v119/xu20e.html.

L. Xu, F. Hutter, H. Hoos, and Kevin Leyton-Brown. Hydra-MIP : Automated Algorithm Configuration and Selection for Mixed Integer Programming. 2011. URL https://www.semanticscholar.org/paper/Hydra-MIP-%3A-Automated-Algorithm-Configuration-and-Xu-Hutter/302f936148bb19527646b523456d36ff7afbd505.

Lin Xu, Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. SATzilla: Portfolio-based Algorithm Selection for SAT. *J. Artif. Intell. Res.*, 32:565–606, 2008. doi: 10.1613/jair.2490.

Chengrun Yang, Yuji Akimoto, Dae Won Kim, and Madeleine Udell. OBOE: Collaborative Filtering for AutoML Model Selection. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '19, pages 1173–1183, Anchorage, AK, USA, July 2019. Association for Computing Machinery. ISBN 9781450362016. doi: 10.1145/3292500.3330909. URL https://doi.org/10.1145/3292500.3330909.

Estefania Yap, Mario Muñoz, Kate Smith-Miles, and Arnaud Liefooghe. Instance space analysis of combinatorial multi-objective optimization problems. volume 2020 IEEE Congress on Evolutionary Computation (CEC), 2020. doi: 10.1109/CEC48606.2020.9185664. URL https://hal.archives-ouvertes.fr/hal-02920051.

Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Russ R Salakhutdinov, and Alexander J Smola. Deep Sets. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://papers.nips.cc/paper/2017/hash/f22e4747da1aa27e363d86d40ff442fe-Abstract.html.

Fotios Zantalis, Grigorios Koulouras, Sotiris Karabetsos, and Dionisis Kandris. A Review of Machine Learning and IoT in Smart Transportation. *Future Internet*, 11(4):94, April 2019. doi: 10.3390/fi11040094. URL https://www.mdpi.com/1999-5903/11/4/94.

Barret Zoph and Quoc V. Le. Neural Architecture Search with Reinforcement Learning. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017. URL https://openreview.net/forum?id=r1Ue8Hcxg.

Marc-André Zöller and Marco F. Huber. Benchmark and Survey of Automated Machine Learning Frameworks. *Journal of Artificial Intelligence Research*, 70: 409–472, January 2021. ISSN 1076-9757. doi: 10.1613/jair.1.11854. URL https://www.jair.org/index.php/jair/article/view/11854.

A - Supplementary Material - Mosaic

A.1 . Mosaic Search Space

Mosaic shares the same search space as Auto-Sklearn. Table A.1 resume all components of one complete pipeline $\mathbf{x}$. Table A.2 illustrates an overview of hyper-parameter for each data preprocessing methode. Table A.3, A.4 presents an exhaustive list of hyper-parameter of each learning algorithm considered in Mosaic. All names (of algorithms, their parameters, and possibly the names of the options) are those used in Scikit-Learn library.

A.2 . Detailed results (Vanilla setting)

This section presents detailed results for the Vanilla variants of Mosaic and Auto-Sklearn out of 10 independent runs (the results at the stopping time are graphically presented in Figure 4 of the paper). A general remark is that Mosaic has a higher variance w.r.t. the median results, as summarized on Table 4.1, Mosaic is significantly better on 21 datasets (using a MWW test at 95% confidence), and non-significantly better on 35 datasets. It is significantly worse on 6 datasets and non-significantly worse on 18 datasets. Results on dataset with ID in $3021, 3946, 3948, 3950, 34536, 34539$ are not available because of technical problems (memory and dataset quality issues).

Pipeline	Available choice
Balancing	none, weighting
Categorical_encoding	no_encoding, one_hot_encoding
Imputation	mean, median, most_frequent
Rescaling	minmax, none, normalize, quantile_transformer, robust_scaler, standardize
Classifier	adaboost, bernoulli_nb, decision_tree, extra_trees, gaussian_nb, gradient_boosting, k_nearest_neighbors, lda, liblinear_svc, libsvm_svc, multinomial_nb, passive_aggressive, qda, random_forest, sgd, xgradient_boosting
Preprocessor	extra_trees_preproc_for_classification, fast_ica, feature_agglomeration, kernel_pca, kitchen_sinks, liblinear_svc_preprocessor, no_preprocessing, nystroem_sampler, pca, polynomial, random_trees_embedding, select_percentile_classification, select_rates

Table A.1: Pipeline components for each complete configuration

Algorithm	Parameter	Type	Domain	Default
extra_trees_preproc_for_classification	bootstrap	categorical	True,False	[False]
extra_trees_preproc_for_classification	criterion	categorical	gini,entropy	[gini]
extra_trees_preproc_for_classification	max_depth	categorical	None	[None]
extra_trees_preproc_for_classification	max_features	real	[0.0,1.0]	[0.5]
extra_trees_preproc_for_classification	max_leaf_nodes	categorical	None	[None]
extra_trees_preproc_for_classification	min_impurity_decrease	categorical	0.0	[0.0]
extra_trees_preproc_for_classification	min_samples_leaf	integer	[1,20]	[1]
extra_trees_preproc_for_classification	min_samples_split	integer	[2,20]	[2]
extra_trees_preproc_for_classification	min_weight_fraction_leaf	categorical	0.0	[0.0]
extra_trees_preproc_for_classification	n_estimators	categorical	100	[100]
fast_ica	algorithm	categorical	parallel,deflation	[parallel]
fast_ica	fun	categorical	logcosh,exp,cube	[logcosh]
fast_ica	whiten	categorical	False,True	[False]
feature_agglomeration	affinity	categorical	euclidean,manhattan,cosine	[euclidean]
feature_agglomeration	linkage	categorical	ward,complete,average	[ward]
feature_agglomeration	n_clusters	integer	[2,400]	[25]
feature_agglomeration	pooling_func	categorical	mean,median,max	[mean]
kernel_pca	kernel	categorical	poly,rbf,sigmoid,cosine	[rbf]
kernel_pca	n_components	integer	[10,2000]	[100]
kitchen_sinks	gamma	real (log)	[3.0517578125e-05,8.0]	[1.0]
kitchen_sinks	n_components (log)	integer	[50,10000]	[100]
liblinear_svc_preprocessor	C	real (log)	[0.03125,32768.0]	[1.0]
liblinear_svc_preprocessor	dual	categorical	False	[False]
liblinear_svc_preprocessor	fit_intercept	categorical	True	[True]
liblinear_svc_preprocessor	intercept_scaling	categorical	1	[1]
liblinear_svc_preprocessor	loss	categorical	hinge,squared_hinge	[squared_hinge]
liblinear_svc_preprocessor	multi_class	categorical	ovr	[ovr]
liblinear_svc_preprocessor	penalty	categorical	l1	[l1]
liblinear_svc_preprocessor	tol	real (log)	[1e-05,0.1]	[0.0001]
nystroem_sampler	kernel	categorical	poly,rbf,sigmoid,cosine	[rbf]
nystroem_sampler	n_components	integer (log)	[50,10000]	[100]
pca	keep_variance	real	[0.5,0.9999]	[0.9999]
pca	whiten	categorical	False,True	[False]
polynomial	degree	integer	[2,3]	[2]
polynomial	include_bias	categorical	True,False	[True]
polynomial	interaction_only	categorical	False,True	[False]
random_trees_embedding	bootstrap	categorical	True,False	[True]
random_trees_embedding	max_depth	integer	[2,10]	[5]
random_trees_embedding	max_leaf_nodes	categorical	None	[None]
random_trees_embedding	min_samples_leaf	integer	[1,20]	[1]
random_trees_embedding	min_samples_split	integer	[2,20]	[2]
random_trees_embedding	min_weight_fraction_leaf	categorical	1.0	[1.0]
random_trees_embedding	n_estimators	integer	[10,100]	[10]
select_percentile_classification	percentile	real	[1.0,99.0]	[50.0]
select_percentile_classification	score_func	categorical	chi2,f_classif,mutual_info	[chi2]
select_rates	alpha	real	[0.01,0.5]	[0.1]
select_rates	mode	categorical	fpr,fdr,fwe	[fpr]
select_rates	score_func	categorical	chi2,f_classif	[chi2]
fast_ica	n_components	integer	[10,2000]	[100]
kernel_pca	coef0	real	[-1.0,1.0]	[0.0]
kernel_pca	degree	integer	[2,5]	[3]
kernel_pca	gamma	real (log)	[3.0517578125e-05, 8.0]	[1.0]
nystroem_sampler	coef0	real	[-1.0,1.0]	[0.0]
nystroem_sampler	degree	integer	[2,5]	[3]
nystroem_sampler	gamma	real (log)	[3.0517578125e-05,8.0]	[0.1]

Table A.2: Configuration space of data-preprocessing method

Algorithm	Parameter	Type	Domain	Default
adaboost	algorithm	categorical	SAMME.R,SAMME	[SAMME.R]
adaboost	learning_rate	real (log)	[0.01,2.0]	[0.1]
adaboost	max_depth	integer	[1,10]	[1]
adaboost	n_estimators	integer	[50,500]	[50]
bernoulli_nb	alpha	real (log)	[0.01,100.0]	[1.0]
bernoulli_nb	fit_prior	categorical	True,False	[True]
decision_tree	criterion	categorical	gini,entropy	[gini]
decision_tree	max_depth	real	[0.0,2.0]	[0.5]
decision_tree	max_features	categorical	1.0	[1.0]
decision_tree	max_leaf_nodes	categorical	None	[None]
decision_tree	min_impurity_decrease	categorical	0.0	[0.0]
decision_tree	min_samples_leaf	integer	[1,20]	[1]
decision_tree	min_samples_split	integer	[2,20]	[2]
decision_tree	min_weight_fraction_leaf	categorical	0.0	[0.0]
extra_trees	bootstrap	categorical	True,False	[False]
extra_trees	criterion	categorical	gini,entropy	[gini]
extra_trees	max_depth	categorical	None	[None]
extra_trees	max_features	real	[0.0,1.0]	[0.5]
extra_trees	max_leaf_nodes	categorical	None	[None]
extra_trees	min_impurity_decrease	categorical	0.0	[0.0]
extra_trees	min_samples_leaf	integer	[1,20]	[1]
extra_trees	min_samples_split	integer	[2,20]	[2]
extra_trees	min_weight_fraction_leaf	categorical	0.0	[0.0]
extra_trees	n_estimators	categorical	100	[100]
gradient_boosting	criterion	categorical	friedman_mse,mse,mae	[mse]
gradient_boosting	learning_rate	real (log)	[0.01,1.0]	[0.1]
gradient_boosting	loss	categorical	deviance	[deviance]
gradient_boosting	max_depth	integer	[1,10]	[3]
gradient_boosting	max_features	real	[0.1,1.0]	[1.0]
gradient_boosting	max_leaf_nodes	categorical	None	[None]
gradient_boosting	min_impurity_decrease	categorical	0.0	[0.0]
gradient_boosting	min_samples_leaf	integer	[1,20]	[1]
gradient_boosting	min_samples_split	integer	[2,20]	[2]
gradient_boosting	min_weight_fraction_leaf	categorical	0.0	[0.0]
gradient_boosting	n_estimators	integer	[50,500]	[100]
gradient_boosting	subsample	real	[0.01,1.0]	[1.0]
k_nearest_neighbors	n_neighbors	integer (log)	[1,100]	[1]
k_nearest_neighbors	p	categorical	1,2	[2]
k_nearest_neighbors	weights	categorical	uniform,distance	[uniform]
lda	n_components	integer	[1,250]	[10]
lda	shrinkage	categorical	None,auto,manual	[None]
lda	tol	real (log)	[1e-05,0.1]	[0.0001]
liblinear_svc	C	real (log)	[0.03125,32768.0]	[1.0]
liblinear_svc	dual	categorical	False	[False]
liblinear_svc	fit_intercept	categorical	True	[True]
liblinear_svc	intercept_scaling	categorical	1	[1]
liblinear_svc	loss	categorical	hinge,squared_hinge	[squared_hinge]
liblinear_svc	multi_class	categorical	ovr	[ovr]
liblinear_svc	penalty	categorical	l1,l2	[l2]
liblinear_svc	tol	real (log)	[1e-05,0.1]	[0.0001]
libsvm_svc	C	real	[0.03125,32768.0]	[1.0]log
libsvm_svc	gamma	real (log)	[3.0517578125e-05,8.0]	[0.1]
libsvm_svc	kernel	categorical	rbf,poly,sigmoid	[rbf]
libsvm_svc	max_iter	categorical	-1	[-1]
libsvm_svc	shrinking	categorical	True,False	[True]
libsvm_svc	tol	real (log)	[1e-05,0.1]	[0.001]
multinomial_nb	alpha	real (log)	[0.01,100.0]	[1.0]
multinomial_nb	fit_prior	categorical	True,False	[True]

Table A.3: Configuration space of learning algorithm (1/2)

Algorithm	Parameter	Type	Domain	Default
passive_aggressive	C	real (log)	[1e-05,10.0]	[1.0]
passive_aggressive	average	categorical	False,True	[False]
passive_aggressive	fit_intercept	categorical	True	[True]
passive_aggressive	loss	categorical	hinge,squared_hinge	[hinge]
passive_aggressive	tol	real (log)	[1e-05,0.1]	[0.0001]
qda	reg_param	real	[0.0,1.0]	[0.0]
random_forest	bootstrap	categorical	True,False	[True]
random_forest	criterion	categorical	gini,entropy	[gini]
random_forest	max_depth	categorical	None	[None]
random_forest	max_features	real	[0.0,1.0]	[0.5]
random_forest	max_leaf_nodes	categorical	None	[None]
random_forest	min_impurity_decrease	categorical	0.0	[0.0]
random_forest	min_samples_leaf	integer	[1,20]	[1]
random_forest	min_samples_split	integer	[2,20]	[2]
random_forest	min_weight_fraction_leaf	categorical	0.0	[0.0]
random_forest	n_estimators	categorical	100	[100]
sgd	alpha	real (log)	[1e-07,0.1]	[0.0001]
sgd	average	categorical	False,True	[False]
sgd	fit_intercept	categorical	True	[True]
sgd	learning_rate	categorical	optimal,invscaling,constant	[invscaling]
sgd	loss	categorical	hinge, log ,modified_huber, squared_hinge, perceptron	[log]
sgd	penalty	categorical	l1,l2,elasticnet	[l2]
sgd	tol	real (log)	[1e-05,0.1]	[0.0001]
sgd	epsilon	real (log)	[1e-05,0.1]	[0.0001]
sgd	eta0	real (log)	[1e-07,0.1]	[0.01]
sgd	l1_ratio	real (log)	[1e-09,1.0]	[0.15]
sgd	power_t	real	[1e-05,1.0]	[0.5]
xgradient_boosting	base_score	categorical	0.5	[0.5]
xgradient_boosting	booster	categorical	gbtree,dart	[gbtree]
xgradient_boosting	colsample_bylevel	real	[0.1,1.0]	[1.0]
xgradient_boosting	colsample_bytree	real	[0.1,1.0]	[1.0]
xgradient_boosting	gamma	categorical	0	[0]
xgradient_boosting	learning_rate (log)	real	[0.001,1.0]	[0.1]
xgradient_boosting	max_delta_step	categorical	0	[0]
xgradient_boosting	max_depth	integer	[1,20]	[3]
xgradient_boosting	min_child_weight	integer	[0,20]	[1]
xgradient_boosting	n_estimators	categorical	512	[512]
xgradient_boosting	reg_alpha (log)	real	[1e-10,0.1]	[1e-10]
xgradient_boosting	reg_lambda (log)	real	[1e-10,0.1]	[1e-10]
xgradient_boosting	scale_pos_weight	categorical	1	[1]
xgradient_boosting	subsample	real	[0.01,1.0]	[1.0]
lda	shrinkage_factor	real	[0.0,1.0]	[0.5]
libsvm_svc	coef0	real	[-1.0,1.0]	[0.0]
libsvm_svc	degree	integer	[2,5]	[3]
xgradient_boosting	normalize_type	categorical	tree,forest	[tree]
xgradient_boosting	rate_drop	real	[1e-10,0.9999999999]	[0.5]
xgradient_boosting	sample_type	categorical	uniform,weighted	[uniform]

Table A.4: Configuration space of learning algorithm (2/2)

	Auto-Sklearn				Mosaic			
Dataset id	Min	Max	Median	std	Min	Max	Median	std
3	0.984	0.997	0.99	0.004	0.987	0.997	**0.993**	0.004
6	0.964	0.964	0.964	0.0	0.967	0.973	**0.971**	0.002
11	1.0	1.0	1.0	0.0	0.933	1.0	1.0	0.021
12	0.97	0.985	0.975	0.006	0.965	0.985	**0.978**	0.006
14	0.775	0.86	0.805	0.032	0.81	0.88	**0.825**	0.022
15	0.967	0.989	**0.978**	0.01	0.947	0.989	0.973	0.012
16	0.955	0.98	0.968	0.008	0.96	0.98	**0.97**	0.007
18	0.685	0.76	**0.73**	0.023	0.685	0.76	0.698	0.027
20	0.97	0.985	0.972	0.006	0.965	0.985	**0.98**	0.008
21	0.962	0.998	0.992	0.01	0.945	1.0	**1.0**	0.017
22	0.81	0.865	**0.825**	0.017	0.78	0.83	0.812	0.014
23	0.552	0.615	**0.601**	0.022	0.538	0.598	0.571	0.019
24	1.0	1.0	1.0	0.0	1.0	1.0	1.0	0.0
28	0.982	0.995	0.988	0.005	0.986	0.993	**0.991**	0.003
29	0.819	0.899	0.884	0.027	0.858	0.891	0.884	0.011
31	0.698	0.805	0.758	0.036	0.7	0.788	**0.762**	0.028
32	0.992	0.996	0.992	0.002	0.993	0.997	**0.996**	0.001
36	0.97	0.991	0.983	0.007	0.961	0.991	**0.985**	0.01
37	0.797	0.873	**0.817**	0.025	0.77	0.881	0.814	0.032
41	0.86	0.971	0.934	0.034	0.924	0.977	**0.935**	0.021
43	0.919	0.933	0.923	0.005	0.911	0.939	**0.931**	0.009
45	0.939	0.959	**0.954**	0.005	0.948	0.965	0.953	0.005
49	0.924	1.0	1.0	0.034	0.955	1.0	1.0	0.019
53	0.763	0.894	0.848	0.04	0.848	0.907	**0.872**	0.018
58	0.857	0.873	0.86	0.005	0.848	0.872	**0.867**	0.007
219	0.897	0.922	0.897	0.009	0.895	0.934	**0.899**	0.013
2074	0.872	0.899	0.882	0.009	0.891	0.91	**0.897**	0.006
2079	0.578	0.694	0.621	0.038	0.515	0.667	**0.638**	0.06
3022	0.909	1.0	0.96	0.04	0.939	1.0	**0.975**	0.023
3481	0.915	0.962	0.915	0.016	0.926	0.954	**0.938**	0.008
3485	0.924	0.977	0.977	0.017	0.977	0.988	0.977	0.004
3492	1.0	1.0	1.0	0.0	0.982	1.0	1.0	0.008
3493	1.0	1.0	1.0	0.0	1.0	1.0	1.0	0.0
3494	0.964	0.964	0.964	0.0	0.964	0.964	0.964	0.0
3510	0.973	0.995	**0.989**	0.009	0.985	0.992	0.988	0.003
3512	1.0	1.0	1.0	0.0	1.0	1.0	1.0	0.0
3543	1.0	1.0	1.0	0.0	1.0	1.0	1.0	0.0
3549	0.96	0.992	**0.984**	0.009	0.975	0.992	0.983	0.006
3560	0.181	0.286	0.213	0.036	0.183	0.255	**0.231**	0.024
3561	0.614	0.67	0.658	0.019	0.605	0.747	**0.682**	0.048
3567	1.0	1.0	1.0	0.0	1.0	1.0	1.0	0.0
3889	0.977	0.996	0.987	0.006	0.987	0.997	**0.994**	0.003
3891	0.951	0.971	0.955	0.008	0.957	0.974	**0.963**	0.006

Table A.5: Per dataset comparison statistics between Mosaic$\dot{V}$anilla and Auto-Sklearn$\dot{V}$anilla (Part I)

	Auto-Sklearn				Mosaic			
Dataset id	Min	Max	Median	Std	Min	Max	Median	std
3896	0.769	0.8	0.786	0.008	0.778	0.803	**0.791**	0.008
3899	0.927	0.946	0.934	0.007	0.938	0.95	**0.942**	0.004
3902	0.807	0.906	**0.867**	0.03	0.843	0.898	0.862	0.015
3903	0.588	0.747	**0.697**	0.049	0.592	0.778	0.693	0.062
3904	0.637	0.665	0.655	0.009	0.639	0.681	**0.661**	0.014
3913	0.756	0.857	**0.824**	0.028	0.634	0.847	0.747	0.061
3917	0.702	0.789	0.72	0.024	0.686	0.772	**0.726**	0.026
3918	0.599	0.866	**0.778**	0.082	0.656	0.88	0.749	0.07
3954	0.852	0.867	0.852	0.006	0.851	0.87	**0.862**	0.005
7592	0.774	0.841	0.824	0.021	0.818	0.844	**0.838**	0.008
9914	0.906	0.984	**0.952**	0.025	0.902	0.967	0.951	0.021
9946	0.938	0.986	0.972	0.017	0.958	1.0	**0.979**	0.013
9950	0.857	0.967	0.924	0.036	0.925	0.958	**0.942**	0.017
9952	0.851	0.881	0.862	0.011	0.857	0.91	**0.879**	0.015
9954	0.825	0.835	**0.825**	0.003	0.77	0.84	0.812	0.021
9955	0.585	0.635	0.62	0.014	0.595	0.675	**0.643**	0.024
9956	0.785	0.855	0.835	0.018	0.805	0.9	**0.85**	0.028
9957	0.804	0.887	**0.863**	0.024	0.776	0.887	0.846	0.036
9960	0.983	1.0	0.991	0.006	0.987	0.998	**0.994**	0.004
9964	0.906	0.944	0.919	0.013	0.906	0.95	**0.928**	0.015
9967	0.996	0.996	0.996	0.0	0.993	1.0	**1.0**	0.004
9968	1.0	1.0	1.0	0.0	1.0	1.0	1.0	0.0
9970	0.926	1.0	**0.992**	0.031	0.951	1.0	0.971	0.016
9971	0.627	0.739	**0.703**	0.04	0.539	0.715	0.656	0.05
9976	0.762	0.881	0.862	0.04	0.8	0.888	**0.863**	0.03
9977	0.96	0.97	0.96	0.003	0.96	0.97	**0.964**	0.003
9978	0.758	0.883	0.859	0.038	0.813	0.895	**0.86**	0.025
9979	1.0	1.0	1.0	0.0	1.0	1.0	1.0	0.0
9980	0.655	0.745	**0.705**	0.027	0.515	0.715	0.668	0.056
9981	0.935	0.972	0.954	0.011	0.944	0.972	**0.963**	0.009
9983	0.916	0.965	0.944	0.015	0.941	0.972	**0.964**	0.009
9985	0.438	0.445	0.445	0.002	0.44	0.507	**0.471**	0.017
9986	0.993	0.993	0.993	0.0	0.993	0.996	**0.996**	0.001
10093	0.994	1.0	**1.0**	0.003	0.987	1.0	0.994	0.004
10101	0.617	0.706	0.673	0.034	0.598	0.766	**0.683**	0.055
14964	0.836	0.839	0.839	0.001	0.84	0.908	**0.857**	0.019
14965	0.807	0.853	**0.836**	0.016	0.799	0.862	0.828	0.019
14966	0.763	0.799	0.772	0.011	0.76	0.805	**0.789**	0.014
14967	0.989	1.0	**1.0**	0.004	0.995	1.0	0.999	0.002
14968	0.762	0.881	0.827	0.037	0.767	0.908	**0.843**	0.049
14969	0.571	0.635	0.579	0.02	0.573	0.622	**0.606**	0.015
14970	0.975	0.985	0.979	0.003	0.981	0.991	**0.988**	0.004
34537	0.961	0.961	0.961	0.0	0.951	0.965	0.961	0.004
34538	1.0	1.0	1.0	0.0	1.0	1.0	1.0	0.0
125920	0.445	0.557	**0.54**	0.034	0.464	0.654	0.527	0.057
125921	0.715	0.805	**0.763**	0.027	0.69	0.808	0.755	0.032
125922	0.984	0.998	0.992	0.005	0.989	0.996	**0.993**	0.002
125923	0.837	0.886	0.871	0.016	0.846	0.912	**0.881**	0.019
146195	0.583	0.692	0.649	0.038	0.617	0.718	**0.662**	0.034
146606	0.717	0.717	0.717	0.0	0.718	0.729	**0.722**	0.003
146607	0.726	0.774	0.746	0.017	0.756	0.788	**0.763**	0.012

Table A.6: Per dataset comparison statistics between Mosaic$\dot{V}$anilla and Auto-Sklearn$\dot{V}$anilla. Part(II)

B - Supplementary Material - Dida

B.1 . Extension to arbitrary distributions

General notations. Let $X \in \mathcal{R}(\mathbb{R}^d)$ denote a random vector on $\mathbb{R}^d$ with $\alpha_X \in \mathcal{P}(\mathbb{R}^d)$ its law (a positive Radon measure with unit mass). By definition, its expectation denoted $\mathbb{E}(X)$ reads $\mathbb{E}(X) = \int_{\mathbf{R}^d} x \mathrm{d}\alpha_X(x) \in \mathbb{R}^d$, and for any continuous function $f : \mathbb{R}^d \to \mathbb{R}^r$, $\mathbb{E}(f(X)) = \int_{\mathbf{R}^d} f(x)\mathrm{d}\alpha_X(x)$. In the following, two random vectors X and X' with same law α_X are considered indistinguishable, noted $X' \sim X$. Letting $f : \mathbb{R}^d \mapsto \mathbb{R}^r$ denote a function on $\mathbb{R}^d$, the push-forward operator by f, noted $f_\sharp : \mathcal{P}(\mathbb{R}^d) \mapsto \mathcal{P}(\mathbb{R}^r)$ is defined as follows, for any g continuous function from $\mathbb{R}^d$ to $\mathbb{R}^r$ (g in $\mathcal{C}(\mathbb{R}^d; \mathbb{R}^r)$):

$$\forall g \in \mathcal{C}(\mathbb{R}^d; \mathbb{R}^r) \quad \int_{\mathbf{R}^r} g \mathrm{d}(f_\sharp \alpha) \stackrel{\text{def.}}{=} \int_{\mathbf{R}^d} g(f(x))\mathrm{d}\alpha(x)$$

Letting $\{x_i\}$ be a set of points in $\mathbb{R}^d$ with $w_i \geq 0$ such that $\sum_i w_i = 1$, the discrete measure $\alpha_X = \sum_i w_i \delta_{x_i}$ is the sum of the Dirac measures δ_{x_i} weighted by w_i.

Invariances. In this paper, we consider functions on probability measures that are *invariant with respect to permutations of coordinates*. Therefore, denoting S_d the d-sized permutation group, we consider measures over a symmetrized compact $\Omega \subset \mathbb{R}^d$ equipped with the following equivalence relation: for α, $\beta \in \mathcal{P}(\Omega), \alpha \sim \beta \iff \exists \sigma \in S_d, \beta = \sigma_\sharp \alpha$, such that a measure and its permuted counterpart are indistinguishable in the corresponding quotient space, denoted alternatively $\mathcal{P}(\Omega)_{/\sim}$ or $\mathcal{R}(\Omega)_{/\sim}$. A function $\varphi : \Omega^n \to \mathbb{R}$ is said to be invariant (by permutations of coordinates) iff $\forall \sigma \in S_d, \varphi(x_1, \ldots, x_n) = \varphi(\sigma(x_1), \ldots, \sigma(x_n))$ (Definition 5.1).

Tensorization. Letting X and Y respectively denote two random vectors on $\mathcal{R}(\mathbb{R}^d)$ and $\mathcal{R}(\mathbb{R}^p)$, the tensor product vector $X \otimes Y$ is defined as: $X \otimes Y \stackrel{\text{def.}}{=} (X', Y') \in \mathcal{R}(\mathbb{R}^d \times \mathbb{R}^p)$, where X' and Y' are independent and have the same law as X and Y, i.e. $\mathrm{d}(\alpha_{X\otimes Y})(x, y) = \mathrm{d}\alpha_X(x)\mathrm{d}\alpha_Y(y)$. In the finite case, for $\alpha_X = \frac{1}{n}\sum_i \delta_{x_i}$ and $\alpha_Y = \frac{1}{m}\sum_j \delta_{y_j}$, then $\alpha_{X\otimes Y} = \frac{1}{nm}\sum_{i,j} \delta_{x_i, y_j}$, weighted sum of Dirac measures on all pairs (x_i, y_j). The $k-$fold tensorization of a random vector $X \sim \alpha_X$, with law $\alpha_X^{\otimes k}$, generalizes the above construction to the case of k independent random variables with law α_X. Tensorization will be used to define the law of datasets, and design universal architectures (Appendix B.3).

Invariant layers. In the general case, a G-invariant layer f_φ with invariant map $\varphi : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}^r$ such that φ satisfies

$$\forall (x_1, x_2) \in (\mathbb{R}^d)^2, \forall \sigma \in G, \varphi(\sigma(x_1), \sigma(x_2)) = \varphi(x_1, x_2)$$

is defined as

$$f_\varphi : X \in \mathcal{R}(\mathbb{R}^d)_{/\sim} \mapsto \mathbb{E}_{X' \sim X}[\varphi(X, X')] \in \mathcal{R}(\mathbb{R}^r)_{/\sim}$$

where the expectation is taken over $X' \sim X$. Note that considering the couple (X, X') of independent random vectors $X' \sim X$ amounts to consider the tensorized law $\alpha_X \otimes \alpha_X$.

Remark 3. *Taking as input a discrete distribution $\alpha_X = \sum_{i=1}^n w_i \delta_{x_i}$, the invariant layer outputs another discrete distribution $\alpha_Y = \sum_{i=1}^n w_i \delta_{y_i}$ with $y_i = \sum_{j=1}^n w_j \varphi(x_i, x_j)$; each input point x_i is mapped onto y_i summarizing the pairwise interactions with x_i after φ.*

Remark 4. *(Generalization to arbitrary invariance groups) The definition of invariant φ can be generalized to arbitrary invariance groups operating on $\mathbb{R}^d$, in particular sub-groups of the permutation group S_d. After Maron et al. [2020] (Theorem 5), a simple and only way to design an invariant linear function is to consider $\varphi(z, z') = \psi(z + z')$ with ψ being G-invariant. How to design invariant functions in the general non-linear case is left for further work.*

Remark 5. *Invariant layers can also be generalized to handle higher order interactions functionals, namely $f_\varphi(X) \stackrel{\text{def.}}{=} \mathbb{E}_{X_2,\ldots,X_N \sim X}[\varphi(X, X_2, \ldots, X_N)]$, which amounts to consider, in the discrete case, N-uple of inputs points $(x_{j_1}, \ldots, x_{j_N})$.*

B.2 . Proofs on Regularity

Wasserstein distance. The regularity of the involved functionals is measured w.r.t. the 1-Wasserstein distance between two probability distributions $(\alpha, \beta) \in \mathcal{P}(\mathbb{R}^d)$

$$\begin{aligned} \mathrm{W}_1(\alpha, \beta) &\stackrel{\text{def.}}{=} \min_{\pi_1 = \alpha, \pi_2 = \beta} \int_{\mathbf{R}^d \times \mathbf{R}^d} \|x - y\| \mathrm{d}\pi(x, y) \\ &\stackrel{\text{def.}}{=} \min_{X \sim \alpha, Y \sim \beta} \mathbb{E}(\|X - Y\|) \end{aligned}$$

where the minimum is taken over measures on $\mathbb{R}^d \times \mathbb{R}^d$ with marginals $\alpha, \beta \in \mathcal{P}(\mathbb{R}^d)$. W_1 is known to be a norm Santambrogio [2015], that can be conveniently computed using

$$\mathrm{W}_1(\alpha, \beta) = \mathrm{W}_1(\alpha - \beta) = \max_{\mathrm{Lip}(g) \leq 1} \int_{\mathbf{R}^d} g \mathrm{d}(\alpha - \beta),$$

where $\mathrm{Lip}(g)$ is the Lipschitz constant of $g : \mathbb{R}^d \to \mathbb{R}$ with respect to the Euclidean norm (unless otherwise stated). For simplicity and by abuse of notations, $\mathrm{W}_1(X, Y)$ is used instead of $\mathrm{W}_1(\alpha, \beta)$ when $X \sim \alpha$ and $Y \sim \beta$. The convergence in law denoted $\rightharpoonup$ is equivalent to the convergence in Wasserstein distance in the sense that $X_k \rightharpoonup X$ is equivalent to $\mathrm{W}_1(X_k, X) \to 0$.

Permutation-invariant Wasserstein distance. The Wasserstein distance is quotiented according to the permutation invariance equivalence classes: for $\alpha, \beta \in \mathcal{P}(\mathbb{R}^d)$

$$\begin{aligned}\overline{\mathrm{W}}_1(\alpha, \beta) &\overset{\text{def.}}{=} \min_{\sigma \in S_d} \mathrm{W}_1(\sigma_\sharp \alpha, \beta) \\ &= \min_{\sigma \in S_d} \max_{\mathrm{Lip}(g) \leq 1} \int_{\mathbf{R}^d} g \circ \sigma \mathrm{d}\alpha - \int_{\mathbf{R}^d} g \mathrm{d}\beta\end{aligned}$$

such that $\overline{\mathrm{W}}_1(\alpha, \beta) = 0 \iff \alpha \sim \beta$. $\overline{\mathrm{W}}_1$ defines a norm on $\mathcal{P}(\mathbb{R}^d)_{/\sim}$.

Lipschitz property. A map $f : \mathcal{R}(\mathbb{R}^d) \to \mathcal{R}(\mathbb{R}^r)$ is continuous for the convergence in law (aka the weak* of measures) if for any sequence $X_k \rightharpoonup X$, then $f(X_k) \rightharpoonup f(X)$. Such a map is furthermore said to be C-Lipschitz for the permutation invariant 1-Wasserstein distance if

$$\forall (X, Y) \in (\mathcal{R}(\mathbb{R}^d)_{/\sim})^2, \ \overline{\mathrm{W}}_1(f(X), f(Y)) \leq C \overline{\mathrm{W}}_1(X, Y). \tag{B.1}$$

Lipschitz properties enable us to analyze robustness to input perturbations, since it ensures that if the input distributions of random vectors are close in the permutation invariant Wasserstein sense, the corresponding output laws are close, too.

Proofs of Section 5.4.2.

Proof. (Proposition 1). For $\alpha, \beta \in \mathcal{P}(\mathbb{R}^d)$, Proposition 1 from De Bie et al. [2019] yields $\mathrm{W}_1(f_\varphi(\alpha), f_\varphi(\beta)) \leq 2r \, \mathrm{Lip}(\varphi) \, \mathrm{W}_1(\alpha, \beta)$, hence, for $\sigma \in G$,

$$\begin{aligned}\mathrm{W}_1(\sigma_\sharp f_\varphi(\alpha), f_\varphi(\beta)) &\leq \mathrm{W}_1(\sigma_\sharp f_\varphi(\alpha), f_\varphi(\alpha)) \\ &\quad + \mathrm{W}_1(f_\varphi(\alpha), f_\varphi(\beta)) \\ &\leq \mathrm{W}_1(\sigma_\sharp f_\varphi(\alpha), f_\varphi(\alpha)) \\ &\quad + 2r \, \mathrm{Lip}(\varphi) \, \mathrm{W}_1(\alpha, \beta)\end{aligned}$$

hence, taking the infimum over σ yields

$$\begin{aligned}\overline{\mathrm{W}}_1(f_\varphi(\alpha), f_\varphi(\beta)) &\leq \overline{\mathrm{W}}_1(\quad f_\varphi(\alpha), f_\varphi(\alpha)) \\ &\quad + 2r \, \mathrm{Lip}(\varphi) \, \mathrm{W}_1(\alpha, \beta) \\ &\leq 2r \, \mathrm{Lip}(\varphi) \, \mathrm{W}_1(\alpha, \beta)\end{aligned}$$

Since f_φ is invariant, for $\sigma \in G$, $f_\varphi(\mathbf{z}) = f_\varphi(\sigma_\sharp \mathbf{z})$,

$$\overline{\mathrm{W}}_1(f_\varphi(\alpha), f_\varphi(\beta)) \leq 2r \operatorname{Lip}(\varphi) \mathrm{W}_1(\sigma_\sharp \alpha, \beta)$$

Taking the infimum over σ yields the result. □

Proof. (Proposition 2). To upper bound $\overline{\mathrm{W}}_1(\xi_\sharp f_\varphi(\tau_\sharp \alpha), f_\varphi(\alpha))$ for $\alpha \in \mathcal{P}(\mathbb{R}^d)$, we proceed as follows, using proposition 3 from De Bie et al. [2019] and proposition 1:

$$\begin{aligned}
\mathrm{W}_1(\xi_\sharp f_\varphi(\tau_\sharp \alpha_\varphi(\alpha)), f_\varphi(\alpha)) \leq & \mathrm{W}_1(\xi_\sharp f_\varphi(\tau_\sharp \alpha), f_\varphi(\tau_\sharp \alpha)) \\
& + \mathrm{W}_1(f_\varphi(\tau_\sharp \alpha), f_\varphi(\alpha)) \\
\leq & \|\xi - id\|_{L^1(f_\varphi(\tau_\sharp \alpha))} \\
& + \operatorname{Lip}(f_\varphi) \mathrm{W}_1(\tau_\sharp \alpha, \alpha) \\
\leq & \sup_{y \in f_\varphi(\tau(\Omega))} \|\xi(y) - y\|_2 \\
& + 2r \operatorname{Lip}(\varphi) \sup_{x \in \Omega} \|\tau(x) - x\|_2
\end{aligned}$$

For $\sigma \in G$, we get

$$\begin{aligned}
\mathrm{W}_1(\sigma_\sharp \xi_\sharp f_\varphi(\tau_\sharp \alpha), f_\varphi(\alpha)) \leq & \mathrm{W}_1(\sigma_\sharp \xi_\sharp f_\varphi(\tau_\sharp \alpha), \xi_\sharp f_\varphi(\tau_\sharp \alpha)) \\
& + \mathrm{W}_1(\xi_\sharp f_\varphi(\tau_\sharp \alpha), f_\varphi(\alpha))
\end{aligned}$$

Taking the infimum over σ yields

$$\begin{aligned}
\overline{\mathrm{W}}_1(\xi_\sharp f_\varphi(\tau_\sharp \alpha), f_\varphi(\alpha)) \leq & \mathrm{W}_1(\xi_\sharp f_\varphi(\tau_\sharp \alpha), f_\varphi(\alpha)) \\
\leq & \sup_{y \in f_\varphi(\tau(\Omega))} \|\xi(y) - y\|_2 \\
& + 2rC(\varphi) \sup_{x \in \Omega} \|\tau(x) - x\|_2
\end{aligned}$$

Similarly, for $\alpha, \beta \in (\mathcal{P}(\mathbb{R}^d))^2$,

$$\begin{aligned}
\mathrm{W}_1(\xi_\sharp f_\varphi(\tau_\sharp \alpha), \xi_\sharp f_\varphi(\tau_\sharp \beta)) \leq & \operatorname{Lip}(\xi) \mathrm{W}_1(f_\varphi(\tau_\sharp \alpha), f_\varphi(\tau_\sharp \beta)) \\
\leq & \operatorname{Lip}(\xi) \quad \operatorname{Lip}(f_\varphi) \mathrm{W}_1(\tau_\sharp \alpha, \tau_\sharp \beta) \\
\leq & 2r \operatorname{Lip}(\varphi) \operatorname{Lip}(\xi) \operatorname{Lip}(\tau) \mathrm{W}_1(\alpha, \beta)
\end{aligned}$$

hence, for $\sigma \in G$,

$$\begin{aligned}
\mathrm{W}_1(\sigma_\sharp \xi_\sharp f_\varphi(\tau_\sharp \alpha), \xi_\sharp f_\varphi(\tau_\sharp \beta)) \leq & \mathrm{W}_1(\sigma_\sharp \xi_\sharp f_\varphi(\tau_\sharp \alpha), \xi_\sharp f_\varphi(\tau_\sharp \alpha)) \\
& + \mathrm{W}_1(\xi_\sharp f_\varphi(\tau_\sharp \alpha), \xi_\sharp f_\varphi(\tau_\sharp \beta))
\end{aligned}$$

and taking the infimum over σ yields

$$\begin{aligned}
\overline{\mathrm{W}}_1(\xi_\sharp f_\varphi(\tau_\sharp \alpha), \xi_\sharp f_\varphi(\tau_\sharp \beta)) \leq & \mathrm{W}_1(\xi_\sharp f_\varphi(\tau_\sharp \alpha), \xi_\sharp f_\varphi(\tau_\sharp \beta)) \\
\leq & 2r \operatorname{Lip}(\varphi) \operatorname{Lip}(\xi) \operatorname{Lip}(\tau) \mathrm{W}_1(\alpha, \beta)
\end{aligned}$$

Since τ is equivariant: namely, for $\alpha \in \mathcal{P}(\mathbb{R}^d)$, $\sigma \in G, \tau_\sharp(\sigma_\sharp\alpha) = \sigma_\sharp(\tau_\sharp\alpha)$, hence, since f_φ is invariant, $f_\varphi(\tau_\sharp(\sigma_\sharp\alpha)) = f_\varphi(\sigma_\sharp(\tau_\sharp\alpha)) = f_\varphi(\tau_\sharp\alpha)$, hence for $\sigma \in G$,

$$\overline{\mathrm{W}}_1(\xi_\sharp f_\varphi(\tau_\sharp\alpha),\xi_\sharp f_\varphi(\tau_\sharp\beta)) \\ \leq 2r\,\mathrm{Lip}(\varphi)\,\mathrm{Lip}(\xi)\,\mathrm{Lip}(\tau)\,\mathrm{W}_1(\sigma_\sharp\alpha,\beta)$$

Taking the infimum over σ yields the result. □

B.3 . Proofs on Universality

Detailed proof of Theorem 1. This paragraph details the result in the case of S_d-invariance, while the next one focuses on invariances w.r.t. products of permutations. Before providing a proof of Theorem 1 we first state two useful lemmas. Lemma 1 is mentioned for completeness, referring the reader to De Bie et al. [2019], Lemma 1 for a proof.

Lemma 1. *Let $(S_j)_{j=1}^N$ be a partition of a domain including Ω ($S_j \subset \mathbb{R}^d$) and let $x_j \in S_j$. Let $(\varphi_j)_{j=1}^N$ a set of bounded functions $\varphi_j : \Omega \to \mathbb{R}$ supported on S_j, such that $\sum_j \varphi_j = 1$ on Ω. For $\alpha \in \mathcal{P}(\Omega)$, we denote $\hat{\alpha}_N \overset{\text{def.}}{=} \sum_{j=1}^N \alpha_j \delta_{x_j}$ with $\alpha_j \overset{\text{def.}}{=} \int_{S_j} \varphi_j \mathrm{d}\alpha$. One has, denoting $\Delta_j \overset{\text{def.}}{=} \max_{x\in S_j} \|x_j - x\|$,*

$$\mathrm{W}_1(\hat{\alpha}_N, \alpha) \leq \max_{1\leq j\leq N} \Delta_j.$$

Lemma 2. *Let $f : \mathbb{R}^d \to \mathbb{R}^q$ a $1/p$-Hölder continuous function ($p \geq 1$), then there exists a constant $C > 0$ such that for all $\alpha, \beta \in \mathcal{P}(\mathbb{R}^d)$, $\mathrm{W}_1(f_\sharp\alpha, f_\sharp\beta) \leq C\,\mathrm{W}_1(\alpha,\beta)^{1/p}$.*

Proof. For any transport map π with marginals α and β, $1/p$-Hölderness of f with constant C yields $\int ||f(x) - f(y)||_2 \mathrm{d}\pi(x,y) \leq C \int ||x-y||_2^{1/p} \mathrm{d}\pi(x,y) \leq C \left(\int ||x-y||_2 \mathrm{d}\pi(x,y)\right)^{1/p}$ using Jensen's inequality ($p \leq 1$). Taking the infimum over π yields $\mathrm{W}_1(f_\sharp\alpha, f_\sharp\beta) \leq C\,\mathrm{W}_1(\alpha,\beta)^{1/p}$. □

Now we are ready to dive into the proof. Let $\alpha \in \mathcal{P}(\mathbb{R}^d)$. We consider:

- $h : x = (x_1, \ldots, x_d) \in \mathbb{R}^d \mapsto \left(\sum_{1\leq j_1<\ldots<j_i\leq d} x_{j_1} \cdot \ldots \cdot x_{j_i}\right)_{i=1\ldots d} \in \mathbb{R}^d$ the collection of d elementary symmetric polynomials; h does not lead to a loss in information, in the sense that it generates the ring of S_d-invariant polynomials (see for instance Cox et al. [2018], chapter 7, theorem 3) while preserving the classes (see the proof of Lemma 2, appendix D from Maron et al. [2020]);
- h is obviously not injective, so we consider $\pi : \mathbb{R}^d \to \mathbb{R}^d/S_d$ the projection onto $\mathbb{R}^d/S_d$: $h = \tilde{h} \circ \pi$ such that $\tilde{h}$ is bijective from $\pi(\Omega)$ to its image Ω', compact of $\mathbb{R}^d$; $\tilde{h}$ and $\tilde{h}^{-1}$ are continuous;

- Let $(\varphi_i)_{i=1\dots N}$ the piecewise affine P1 finite element basis, which are hat functions on a discretization $(S_i)_{i=1\dots N}$ of $\Omega' \subset \mathbb{R}^d$, with centers of cells $(y_i)_{i=1\dots N}$. We then define $g : x \in \mathbb{R}^d \mapsto (\varphi_1(x), \dots, \varphi_N(x)) \in \mathbb{R}^N$;
- $f : (\alpha_1, \dots, \alpha_N) \in \mathbb{R}^N \mapsto \mathcal{F}\left(\sum_{i=1}^N \alpha_i \delta_{\tilde{h}^{-1}(y_i)}\right) \in \mathbb{R}$.

We approximate $\mathcal{F}$ using the following steps:

- Lemma 1 (see Lemma 1 from De Bie et al. [2019]) yields that $h_\sharp\alpha$ and $\widehat{h_\sharp\alpha} = \sum_{i=1}^N \alpha_i \delta_{y_i}$ are close: $\mathrm{W}_1(h_\sharp\alpha, \widehat{h_\sharp\alpha}) \leq \sqrt{d}/N^{1/d}$;
- The map $\tilde{h}^{-1}$ is regular enough ($1/d$-Hölder) such that according to Lemma 2, there exists a constant $C > 0$ such that

$$\begin{aligned}\mathrm{W}_1(\tilde{h}_\sharp^{-1}(h_\sharp\alpha), \tilde{h}_\sharp^{-1}\widehat{h_\sharp\alpha}) &\leq C\,\mathrm{W}_1(h_\sharp\alpha, \widehat{h_\sharp\alpha})^{1/d} \\ &\leq Cd^{1/2d}/N^{1/d^2}\end{aligned}$$

  Hence

$$\begin{aligned}\overline{\mathrm{W}}_1(\alpha, \tilde{h}_\sharp^{-1}\widehat{h_\sharp\alpha}) &\overset{\text{def.}}{=} \inf_{\sigma\in S_d} \mathrm{W}_1(\sigma_\sharp\alpha, \tilde{h}_\sharp^{-1}\widehat{h_\sharp\alpha}) \\ &\leq Cd^{1/2d}/N^{1/d^2}.\end{aligned}$$

  Note that h maps the roots of polynomial $\prod_{i=1}^d (X - x^{(i)})$ to its coefficients (up to signs). Theorem 1.3.1 from Rahman and Schmeisser [2002] yields continuity and $1/d$-Hölderness of the reverse map. Hence $\tilde{h}^{-1}$ is $1/d$-Hölder.
- Since Ω is compact, by Banach-Alaoglu theorem, we obtain that $\mathcal{P}(\Omega)$ is weakly-* compact, hence $\mathcal{P}(\Omega)_{/\sim}$ also is. Since $\mathcal{F}$ is continuous, it is thus uniformly weak-* continuous: for any $\epsilon > 0$, there exists $\delta > 0$ such that $\overline{\mathrm{W}}_1(\alpha, \tilde{h}_\sharp^{-1}\widehat{h_\sharp\alpha}) \leq \delta$ implies $|\mathcal{F}(\alpha) - \mathcal{F}(\tilde{h}_\sharp^{-1}\widehat{h_\sharp\alpha})| < \epsilon$. Choosing N large enough such that $Cd^{1/2d}/N^{1/d^2} \leq \delta$ therefore ensures that $|\mathcal{F}(\alpha) - \mathcal{F}(\tilde{h}_\sharp^{-1}\widehat{h_\sharp\alpha})| < \epsilon$.

Extension of Theorem 1 to products of permutation groups.

Corollary 1. *Let $\mathcal{F} : \mathcal{P}(\Omega)_{/\sim} \to \mathbb{R}$ a continuous $S_{d_1} \times \dots \times S_{d_n}$-invariant map ($\sum_i d_i = d$), where Ω is a symmetrized compact over $\mathbb{R}^d$. Then $\forall \epsilon > 0$, there exists three continuous maps f, g, h such that*

$$\forall \alpha \in \mathcal{M}_+^1(\Omega)_{/\sim}, |\mathcal{F}(\alpha) - f \circ \mathbb{E} \circ g(h_\sharp\alpha)| < \epsilon$$

where h is invariant; g, h are independent of $\mathcal{F}$.

Proof. We provide a proof in the case $G = S_d \times S_p$, which naturally extends to any product group $G = S_{d_1} \times \ldots \times S_{d_n}$. We trade h for the collection of elementary symmetric polynomials in the first d variables; and in the last p variables:

$$\begin{aligned} h : (x_1, \ldots, x_d, y_1, \ldots, y_p) &\in \mathbb{R}^{d+p} \\ \mapsto ([\sum_{j_1<\ldots<j_i} x_{j_1} \ldots x_{j_i}]_{i=1}^{d}; [\sum_{j_1<\ldots<j_i} y_{j_1} \ldots y_{j_i}]_{i=1}^{p}) & \\ \in \mathbb{R}^{d+p} & \end{aligned}$$

up to normalizing constants (see Lemma 4). Step 1 (in Lemma 3) consists in showing that h does not lead to a loss of information, in the sense that it generates the ring of $S_d \times S_p-$invariant polynomials. In step 2 (in Lemma 4), we show that $\tilde{h}^{-1}$ is $1/\max(d,p)-$Hölder. Combined with the proof of Theorem 1, this amounts to showing that the concatenation of Hölder functions (up to normalizing constants) is Hölder. With these ingredients, the sketch of the previous proof yields the result. □

Lemma 3. *Let the collection of symmetric invariant polynomials* $[P_i(X_1, \ldots, X_d)]_{i=1}^{d} \overset{\text{def.}}{=} [\sum_{j_1<\ldots<j_i} X_{j_1} \ldots X_{j_i}]_{i=1}^{d}$ *and* $[Q_i(Y_1, \ldots, Y_p)]_{i=1}^{p} = [\sum_{j_1<\ldots<j_i} Y_{j_1} \ldots Y_{j_i}]_{i=1}^{p}$. *The* $d + p-$*sized family* $(P_1, \ldots, P_d, Q_1, \ldots, Q_p)$ *generates the ring of* $S_d \times S_p-$*invariant polynomials.*

Proof. The result comes from the fact the fundamental theorem of symmetric polynomials (see Cox et al. [2018] chapter 7, theorem 3) does not depend on the base field. Every $S_d \times S_p-$invariant polynomial $P(X_1, \ldots, X_d, Y_1, \ldots, Y_p)$ is also $S_d \times I_p-$invariant with coefficients in $\mathbb{R}[Y_1, \ldots, Y_p]$, hence it can be written $P = R(Y_1, \ldots, Y_p)(P_1, \ldots, P_d)$. It is then also S_p-invariant with coefficients in $\mathbb{R}[P_1, \ldots, P_d]$, hence it can be written $P = S(Q_1, \ldots, Q_p)(P_1, \ldots, P_d) \in \mathbb{R}[P_1, \ldots, P_d, Q_1, \ldots, Q_p]$. □

Lemma 4. *Let* $h : (x, y) \in \Omega \subset \mathbb{R}^{d+p} \mapsto (f(x)/C_1, g(y)/C_2) \in \mathbb{R}^{d+p}$ *where* Ω *is compact,* $f : \mathbb{R}^d \to \mathbb{R}^d$ *is* $1/d-$*Hölder with constant* C_1 *and* $g : \mathbb{R}^p \to \mathbb{R}^p$ *is* $1/p-$*Hölder with constant* C_2*. Then* h *is* $1/\max(d,p)-$*Hölder.*

Proof. Without loss of generality, we consider $d > p$ so that $\max(d,p) = d$, and f, g normalized (f.i. $\forall x, x_0 \in (\mathbb{R}^d)^2, \|f(x) - f(x_0)\|_1 \leq \|x - x_0\|_1^{1/d}$). For $(x,y), (x_0,y_0) \in \Omega^2$, $\|h(x,y) - h(x_0,y_0)\|_1 \leq \|f(x) - f(x_0)\|_1 + \|g(y) - g(y_0)\|_1 \leq \|x - x_0\|_1^{1/d} + \|y - y_0\|_1^{1/p}$ since both f, g are Hölder. We denote D the diameter of Ω, such that both $\|x - x_0\|_1/D \leq 1$ and $\|y - y_0\|_1/D \leq 1$ hold. Therefore $\|h(x,y) - h(x_0,y_0)\|_1 \leq D^{1/d} \left(\frac{\|x-x_0\|_1}{D}\right)^{1/d} + D^{1/p} \left(\frac{\|y-y_0\|_1}{D}\right)^{1/p} \leq 2^{1-1/d} D^{1/p-1/d} \|(x,y) - (x_0,y_0)\|_1^{1/d}$ using Jensen's inequality, hence the result. □

In the next two paragraphs, we focus the case of S_d-invariant functions for the sake of clarity, without loss of generality. Indeed, the same technique applies to $G-$invariant functions as h in that case has the same structure: its first d_X components are $S_{d_X}-$invariant functions of the first d_X variables and its last d_Y components are $S_{d_Y}-$invariant functions of the last variables.

Extension of Theorem 1 to distributions on spaces of varying dimension.

Corollary 2. *Let $I = [0;1]$ and, for $k \in [1; d_m], \mathcal{F}_k : \mathcal{P}(I^k) \to \mathbb{R}$ continuous and S_k-invariant. Suppose $(\mathcal{F}_k)_{k=1\dots d_m-1}$ are restrictions of $\mathcal{F}_{d_m}$, namely, $\forall \alpha_k \in \mathcal{P}(I^k), \mathcal{F}_k(\alpha_k) = \mathcal{F}_{d_m}(\alpha_k \otimes \delta_0^{\otimes d_m - k})$. Then functions f and g from Theorem 1 are uniform: there exists f, g continuous, $h_1, \dots, h_{d_m}$ continuous invariant such that*

$$\forall k = 1 \dots d_m, \forall \alpha_k \in \mathcal{P}(I^k), |\mathcal{F}_k(\alpha_k) - f \circ \mathbb{E} \circ g(h_{k\sharp}\alpha_k)| < \epsilon.$$

Proof. Theorem 1 yields continuous f, g and a continuous invariant h_{d_m} such that $\forall \alpha \in \mathcal{P}(I^{d_m}), |\mathcal{F}_{d_m} - f \circ \mathbb{E} \circ g(h_{d_m\sharp}\alpha)| < \epsilon$. For $k = 1 \dots d_m - 1$, we denote $h_k : (x_1, \dots, x_k) \in \mathbb{R}^k \mapsto ((\sum_{1 \leq j_1 < \dots < j_i \leq k} x^{(j_1)} \cdot \dots \cdot x^{(j_i)})_{i=1\dots k}, 0 \dots, 0) \in \mathbb{R}^{d_m}$. With the hypothesis, for $k = 1 \dots d_m - 1$, $\alpha_k \in \mathcal{P}(I^k)$, the fact that $h_{k\sharp}(\alpha_k) = h_{d_m\sharp}(\alpha_k \otimes \delta_0^{\otimes d_m - k})$ yields the result. □

Approximation by invariant neural networks. Based on theorem 1, $\mathcal{F}$ is uniformly close to $f \circ \mathbb{E} \circ g \circ h$:

- We approximate f by a neural network $f_\theta : x \in \mathbb{R}^N \mapsto C_1\lambda(A_1 x + b_1) \in \mathbb{R}$, where p_1 is an integer, $A_1 \in \mathbb{R}^{p_1 \times N}, C_1 \in \mathbb{R}^{1 \times p_1}$ are weights, $b_1 \in \mathbb{R}^{p_1}$ is a bias and λ is a non-linearity.
- Since each component φ_j of $\varphi = g \circ h$ is permutation-invariant, it has the representation $\varphi_j : x = (x_1, \dots, x_d) \in \mathbb{R}^d \mapsto \rho_j\left(\sum_{i=1}^d u(x_i)\right)$ Zaheer et al. [2017] (which is a special case of our layers with a base function only depending on its first argument, see Section 5.3.1), $\rho_j : \mathbb{R}^{d+1} \to \mathbb{R}$, and $u : \mathbb{R} \to \mathbb{R}^{d+1}$ independent of j (see Zaheer et al. [2017], theorem 7).
- We can approximate ρ_j and u by neural networks $\rho_{j,\theta} : x \in \mathbb{R}^{d+1} \mapsto C_{2,j}\lambda(A_{2,j}x + b_{2,j}) \in \mathbb{R}$ and $u_\theta : x \in \mathbb{R}^d \mapsto C_3\lambda(A_3 x + b_3) \in \mathbb{R}^{d+1}$, where $p_{2,j}, p_3$ are integers, $A_{2,j} \in \mathbb{R}^{p_{2,j} \times (d+1)}, C_{2,j} \in \mathbb{R}^{1 \times p_{2,j}}, A_3 \in \mathbb{R}^{p_3 \times 1}, C_3 \in \mathbb{R}^{(d+1) \times p_3}$ are weights and $b_{2,j} \in \mathbb{R}^{p_{2,j}}, b_3 \in \mathbb{R}^{p_3}$ are biases, and denote $\varphi_\theta(x) = (\varphi_{j,\theta}(x))_j \overset{\text{def.}}{=} (\rho_{j,\theta}(\sum_{i=1}^d u_\theta(x_i)))_j$.

Indeed, we upper-bound the difference of interest $|\mathcal{F}(\alpha) - f_\theta\left(\mathbb{E}_{X\sim\alpha}\left(\varphi_\theta(X)\right)\right)|$ by triangular inequality by the sum of three terms:

- $|\mathcal{F}(\alpha) - f\left(\mathbb{E}_{X\sim\alpha}\left(\varphi(X)\right)\right)|$

- $|f\left(\mathbb{E}_{X\sim\alpha}\left(\varphi(X)\right)\right) - f_\theta\left(\mathbb{E}_{X\sim\alpha}\left(\varphi(X)\right)\right)|$
- $|f_\theta\left(\mathbb{E}_{X\sim\alpha}\left(\varphi(X)\right)\right) - f_\theta\left(\mathbb{E}_{X\sim\alpha}\left(\varphi_\theta(X)\right)\right)|$

and bound each term by $\frac{\epsilon}{3}$, which yields the result. The bound on the first term directly comes from theorem 1 and yields a constant N which depends on ϵ. The bound on the second term is a direct application of the universal approximation theorem (UAT) [Cybenko 1989, Leshno et al. 1993]. Indeed, since α is a probability measure, input values of f lie in a compact subset of $\mathbb{R}^N$: $\| \int_\Omega g \circ h(x) \mathrm{d}\alpha \|_\infty \leq \max_{x\in\Omega} \max_i |g_i \circ h(x)|$, hence the theorem is applicable as long as λ is a nonconstant, bounded and continuous activation function. Let us focus on the third term. Uniform continuity of f_θ yields the existence of $\delta > 0$ s.t. $||u-v||_1 < \delta$ implies $|f_\theta(u) - f_\theta(v)| < \frac{\epsilon}{3}$. Let us apply the UAT: each component φ_j of h can be approximated by a neural network $\varphi_{j,\theta}$. Therefore:

$$
\begin{aligned}
\|\mathbb{E}_{X\sim\alpha}\left(\varphi(X) - \varphi_\theta(X)\right)\|_1 &\leq \mathbb{E}_{X\sim\alpha}\|\varphi(X) - \varphi_\theta(X)\|_1 \\
&\leq \sum_{j=1}^{N} \int_\Omega |\varphi_j(x) - \varphi_{j,\theta}(x)| \mathrm{d}\alpha(x) \\
&\leq \sum_{j=1}^{N} \int_\Omega |\varphi_j(x) - \rho_{j,\theta}(\sum_{i=1}^{d} u(x_i))| \mathrm{d}\alpha(x) \\
&+ \sum_{j=1}^{N} \int_\Omega |\rho_{j,\theta}(\sum_{i=1}^{d} u(x_i)) - \rho_{j,\theta}(\sum_{i=1}^{d} u_\theta(x_i))| \mathrm{d}\alpha(x) \\
&\leq N\frac{\delta}{2N} + N\frac{\delta}{2N} = \delta
\end{aligned}
$$

using the triangular inequality and the fact that α is a probability measure. The first term is small by UAT on ρ_j while the second also is, by UAT on u and uniform continuity of $\rho_{j,\theta}$. Therefore, by uniform continuity of f_θ, we can conclude.

Universality of tensorization. This complementary theorem provides insight into the benefits of tensorization for approximating invariant regression functionals, as long as the test function is invariant.

Theorem 2. *The algebra*

$$
\mathcal{A}_\Omega \overset{\text{def.}}{=} \left\{ \mathcal{F} : \mathcal{P}(\Omega)_{/\sim} \to \mathbb{R}, \exists n \in \mathbb{NN}, \exists \varphi : \Omega^n \to \mathbb{R} \textit{invariant}, \forall \alpha, \mathcal{F}(\alpha) = \int_{\Omega^n} \varphi \mathrm{d}\alpha^{\otimes n} \right\}
$$

where $\otimes n$ denotes the n-fold tensor product, is dense in $\mathcal{C}(\mathcal{M}_+^1(\Omega)_{/\sim})$.

Proof. This result follows from the Stone-Weierstrass theorem. Since Ω is compact, by Banach-Alaoglu theorem, we obtain that $\mathcal{P}(\Omega)$ is weakly-* compact, hence $\mathcal{P}(\Omega)_{/\sim}$ also is. In order to apply Stone-Weierstrass, we show

that $\mathcal{A}_\Omega$ contains a non-zero constant function and is an algebra that separates points. A (non-zero, constant) 1-valued function is obtained with $n = 1$ and $\varphi = 1$. Stability by scalar is straightforward. For stability by sum: given $(\mathcal{F}_1, \mathcal{F}_2) \in \mathcal{A}_\Omega^2$ (with associated functions (φ_1, φ_2) of tensorization degrees (n_2, n_2)), we denote $n \overset{\text{def.}}{=} \max(n_1, n_2)$ and $\varphi(x_1, \ldots, x_n) \overset{\text{def.}}{=} \varphi_1(x_1, \ldots, x_{n_1}) + \varphi_2(x_1, \ldots, x_{n_2})$ which is indeed invariant, hence $\mathcal{F}_1 + \mathcal{F}_2 = \int_{\Omega^n} \varphi \mathrm{d}\alpha^{\otimes n} \in \mathcal{A}_\Omega$. Similarly, for stability by product: denoting this time $n = n_1 + n_2$, we introduce the invariant $\varphi(x_1, \ldots, x_n) = \varphi_1(x_1, \ldots, x_{n_1}) \times \varphi_2(x_{n_1+1}, \ldots, x_n)$, which shows that $\mathcal{F} = \mathcal{F}_1 \times \mathcal{F}_2 \in \mathcal{A}_\Omega$ using Fubini's theorem. Finally, $\mathcal{A}_\Omega$ separates points: if $\alpha \neq \nu$, then there exists a symmetrized domain S such that $\alpha(S) \neq \nu(S)$: indeed, if for all symmetrized domains S, $\alpha(S) = \nu(S)$, then $\alpha(\Omega) = \nu(\Omega)$ which is absurd. Taking $n = 1$ and $\varphi = 1_S$ (invariant since S is symmetrized) yields an $\mathcal{F}$ such that $\mathcal{F}(\alpha) \neq \mathcal{F}(\nu)$. □

B.4 . Experimental validation, supplementary material

Both Dida and baselines source code are provided in the last file of the supplementary material.

B.4.1 . Benchmark Details

Three benchmarks are used (Table 5.1): TOY and UCI, taken from Jomaa et al. [2021], and OpenML CC-18. TOY includes 10,000 datasets, where instances are distributed along mixtures of Gaussian, intertwinning moons and rings in $\mathbb{R}^2$, with 2 to 7 classes. UCI includes 121 datasets from the UCI Irvine repository Dua and Graff [2017]. Datasets UCI and OpenML are normalized as follows: categorical features are one-hot encoded; numerical features are normalized; missing values are imputed with the feature mean (continuous features) or median (for categorical features). Patches are defined as follows. Given an initial dataset, a number d_X of features and a number n of examples are uniformly selected in the considered ranges (depending on the benchmark) described in Table B.1. A patch is defined by (i) retaining n examples uniformly selected with replacement in this initial dataset; (ii) retaining d_X features uniformly selected with replacement among the initial features.

	Patch Identification		Ranking Hyper-parameter
Dataset	TOY	UCI	OpenML
# Features	2	[2, 15]	[3, 11]
# Examples	200	[200, 500]	[700, 900]

Table B.1: Patch characteristics

B.4.2 . Baseline Details

Dataset2Vec details. The publicly available implementation of Dataset2Vec [1] is implemented in TensorFlow, which is incompatible with our evaluation pipeline written in PyTorch. For this reason, we have included as baselines: (i) the reported accuracy from Jomaa et al. [2021]; (ii) the computed accuracy from our own implementation of Dataset2Vec, based on a uniform sampling of the features. As said, this implementation only aims at solely making up for the feature sampling procedure. The architecture is the same as reported in Jomaa et al. [2021], Equation 4, namely

$$D : \mathbf{z} \in \mathbf{Z}_n(\mathbb{R}^d) \mapsto h\left(\frac{1}{d_X d_Y}\sum_{m=1}^{d_X}\sum_{t=1}^{d_Y} g\left(\frac{1}{n}\sum_{i=1}^{n} f(x_i[m], y_i[t])\right)\right) \tag{B.2}$$

where functions f, g, h characterizing the architecture are chosen as depicted in the publicly available file $config.py$[2]. More precisely, f, g are FC(128)-ReLU-ResFC(128, 128, 128)-FC(128) and h is FC(128)-ReLU-FC(128)-ReLU where ResFC is a sequence of fully connected layer with skip connection. We provide our implementation of Dataset2Vec in the supplementary material.

DSS layer details. We built our own implementation of invariant DSS layers, as follows. Linear invariant DSS layers (see Maron et al. [2020], Theorem 5, 3.) are of the form

$$L_{inv} : X \in \mathbb{R}^{n\times d} \mapsto L^H(\sum_{j=1}^{n} x_j) \in \mathbb{R}^K \tag{B.3}$$

where $L^H : \mathbb{R}^d \to \mathbb{R}^K$ is a linear H-invariant function. Our applicative setting requires that the implementation accommodates to varying input dimensions d as well as permutation invariance, hence we consider the Deep Sets representation (see Zaheer et al. [2017], Theorem 7)

$$L^H : x = (x_1, \ldots, x_d) \in \mathbb{R}^d \mapsto \rho\left(\sum_{i=1}^{d} \varphi(x_i)\right) \in \mathbb{R}^K \tag{B.4}$$

where $\varphi : \mathbb{R} \to \mathbb{R}^{d+1}$ and $\rho : \mathbb{R}^{d+1} \to \mathbb{R}^K$ are modelled as (i) purely linear functions; (ii) FC networks, which extends the initial linear setting (B.3). In our case, $H = S_{d_X} \times S_{d_Y}$, hence, two invariant layers of the form (B.3-B.4) are combined to suit both feature- and label-invariance requirements. Both outputs are concatenated and followed by an FC network to form the DSS meta-features.

[1] See https://github.com/hadijomaa/dataset2vec

[2] See https://github.com/hadijomaa/dataset2vec/blob/master/config.py

The last experiments use DSS equivariant layers (see Maron et al. [2020], Theorem 1), which take the form

$$L_{eq} : X \in \mathbb{R}^{n\times d} \mapsto \left(L^1_{eq}(x_i) + L^2_{eq}(\sum_{j\neq i} x_j) \right)_{i\in[n]} \in \mathbb{R}^{n\times d} \tag{B.5}$$

where L^1_{eq} and L^2_{eq} are linear H-equivariant layers. Similarly, both feature- and label-equivariance requirements are handled via the Deep Sets representation of equivariant functions (see Zaheer et al. [2017], Lemma 3) and concatenated to be followed by an invariant layer, forming the DSS meta-features. All methods are allocated the same number of parameters to ensure fair comparison. We provide our implementation of the DSS layers in the supplementary material.

No-FInv-DSS **baseline (no invariance in feature permutation).** This baseline aims at showcasing the empirical relevance of the invariance requirement in feature and label permutations, while retaining invariance in permutation with respect to the datasets. To this end, aggregation with respect to the examples is performed as exemplified in Zaheer et al. [2017], Theorem 2, namely

$$L : \mathbf{z} = (\mathbf{z}_1, \ldots, \mathbf{z}_n) \in Z(\mathbb{R}^d) \mapsto \frac{1}{n}\sum_{i=1}^{n} g(\mathbf{z}_i) \in \mathbb{R}^K \tag{B.6}$$

where $g : \mathbb{R}^d \to \mathbb{R}^K$ is an MLP with FC(128)-ReLU-FC(64)-ReLU-FC(32)-ReLU layers. To ensure label information is captured, the output is concatenated to the mean of labels $\bar{y} \stackrel{\text{def.}}{=} \frac{1}{n}\sum_{i=1}^{n} y_i$ and followed by and MLP with FC(1024)-ReLU-FC(700)-ReLU-FC(512) layers. The so-called No-FInv-DSS baseline defined as such, can be summed up as follows

$$\mathbf{z} \in \mathbf{Z}(\mathbb{R}^d) \mapsto \text{MLP}([L(\mathbf{z}); \bar{y}]) \tag{B.7}$$

Hand-crafted meta-features. For the sake of reproducibility, the list of meta-features used in Section 5.5 is given in Table B.2. Note that meta-features related to missing values and categorical features are omitted, as being irrelevant for the considered benchmarks. Hand-crafted meta-features are extracted using BYU `metalearn` library. In total, we extracted 43 meta-features.

B.4.3 . Hyper-parameter spaces.

In Task 2, the hyper-parameter configuration spaces of each algorithm are summarized in Table B.3.

Meta-features	Mean	Min	Max
Quartile2ClassProbability	0.500	0.75	0.25
MinorityClassSize	487.423	426.000	500.000
Quartile3CardinalityOfNumericFeatures	224.354	0.000	976.000
RatioOfCategoricalFeatures	0.347	0.000	1.000
MeanCardinalityOfCategoricalFeatures	0.907	0.000	2.000
SkewCardinalityOfNumericFeatures	0.148	-2.475	3.684
RatioOfMissingValues	0.001	0.000	0.250
MaxCardinalityOfNumericFeatures	282.461	0.000	977.000
Quartile2CardinalityOfNumericFeatures	185.555	0.000	976.000
KurtosisClassProbability	-2.025	-3.000	-2.000
NumberOfNumericFeatures	3.330	0.000	30.000
NumberOfInstancesWithMissingValues	2.800	0.000	1000.000
MaxCardinalityOfCategoricalFeatures	0.917	0.000	2.000
Quartile1CardinalityOfCategoricalFeatures	0.907	0.000	2.000
MajorityClassSize	512.577	500.000	574.000
MinCardinalityOfCategoricalFeatures	0.879	0.000	2.000
Quartile2CardinalityOfCategoricalFeatures	0.915	0.000	2.000
NumberOfCategoricalFeatures	1.854	0.000	27.000
NumberOfFeatures	5.184	4.000	30.000
Dimensionality	0.005	0.004	0.030
SkewCardinalityOfCategoricalFeatures	-0.050	-4.800	0.707
KurtosisCardinalityOfCategoricalFeatures	-1.244	-3.000	21.040
StdevCardinalityOfNumericFeatures	68.127	0.000	678.823
StdevClassProbability	0.018	0.000	0.105
KurtosisCardinalityOfNumericFeatures	-1.060	-3.000	12.988
NumberOfInstances	1000.000	1000.000	1000.000
Quartile3CardinalityOfCategoricalFeatures	0.916	0.000	2.000
NumberOfMissingValues	2.800	0.000	1000.000
Quartile1ClassProbability	0.494	0.463	0.500
StdevCardinalityOfCategoricalFeatures	0.018	0.000	0.707
MeanClassProbability	0.500	0.500	0.500
NumberOfFeaturesWithMissingValues	0.003	0.000	1.000
MaxClassProbability	0.513	0.500	0.574
NumberOfClasses	2.000	2.000	2.000
MeanCardinalityOfNumericFeatures	197.845	0.000	976.000
SkewClassProbability	0.000	-0.000	0.000
Quartile3ClassProbability	0.506	0.500	0.537
MinCardinalityOfNumericFeatures	138.520	0.000	976.000
MinClassProbability	0.487	0.426	0.500
RatioOfInstancesWithMissingValues	0.003	0.000	1.000
Quartile1CardinalityOfNumericFeatures	160.748	0.000	976.000
RatioOfNumericFeatures	0.653	0.000	1.000
RatioOfFeaturesWithMissingValues	0.001	0.000	0.250

Table B.2: Hand-crafted meta-features

	Parameter	Parameter values	Scale
LR	warm start	True, Fase	
	fit intercept	True, Fase	
	tol	[0.00001, 0.0001]	
	C	[1e-4, 1e4]	log
	solver	newton-cg, lbfgs, liblinear, sag, saga	
	max_iter	[5, 1000]	
SVM	kernel	linear, rbf, poly, sigmoid	
	C	[0.0001, 10000]	log
	shrinking	True, False	
	degree	[1, 5]	
	coef0	[0, 10]	
	gamma	[0.0001, 8]	
	max_iter	[5, 1000]	
KNN	n_neighbors	[1, 100]	log
	p	[1, 2]	
	weights	uniform, distance	
SGD	alpha	[0.1, 0.0001]	log
	average	True, False	
	fit_intercept	True, False	
	learning rate	optimal, invscaling, constant	
	loss	hinge, log, modified_huber, squared_hinge, perceptron	
	penalty	l1, l2, elasticnet	
	tol	[1e-05, 0.1]	log
	eta0	[1e-7, 0.1]	log
	power_t	[1e-05, 0.1]	log
	epsilon	[1e-05, 0.1]	log
	l1_ratio	[1e-05, 0.1]	log

Table B.3: Hyper-parameter configurations

C - Supplementary Material - Metabu

C.1 . Measuring performance indicators

As said, the OpenML benchmark includes 72 datasets, with only 64 of them having a target representation. The other 8 datasets are too heavy (e.g. ImageNet) to launch the many runs required to estimate their target representation.

For Task 1, the performance indicator is measured along a Leave-One-Out procedure, with 64 folds:
in each fold, all datasets but one are used to train the Metabu meta-features; the NDCG@k is measured on the remaining dataset. Eventually, the NDCG@k are averaged over all 64 folds.

For Tasks 2 and 3, the performance indicator is likewise measured using a Leave-One-Out procedure with 64 folds. The difference is that besides the remaining dataset, the 8 datasets with no target representation at all are also used as test datasets.

In Tasks 2 and 3, the performance associated with a hyper-parameter configuration for a dataset is computed after training the model on 1 CPU with time budget of 15 mn, with memory less than 8Gb, using the train/validation/test splits given by OpenML; the validation score is estimated using a 5-CV strategy.

In Task 2, for each test dataset and meta-feature set mf:

- The distribution
$$\hat{\mathbf{z}}_{mf} = \frac{1}{Z} \sum_{\ell=1}^{10} exp(-\ell)\mathbf{z}_\ell$$
is defined, with $\mathbf{z}_\ell$ the target representation of the ℓ-th neighbor of the considered dataset, among the training datasets, according to the Euclidean distance based on the meta-features.
- For $1 \leq t \leq T$, a hyper-parameter configuration is independently drawn from $\hat{\mathbf{z}}_{mf}$, and a model is learned using this configuration;
- The performance of this model is measured on a validation dataset;
- The model with best validation performance up to iteration t is retained for each meta-feature set;
- The rank $r(t, mf)$ is determined by comparing the performance on the test set, of the models retained for each meta-feature set.
- The performance curve reports $r(t, mf)$, averaged over test datasets.

In Task 3, the meta-features are used to initialize the performance model:

Classifier	HP	Range
Adaboost	imputation	mean, median, most frequent
	n_estimator	$[50, 500]$
	algorithm	SAMME, SAMME.R
	max_depth	$[1, 10]$
RF	imputation	mean, median, most frequent
	criterion	gini, entropy
	max_features	$]0, 1]$
	min_samples_split	$[2, 20]$
	min_samples_leaf	$[1, 20]$
	bootstrap	True, False
SVM	imputation	mean, median, most frequent
	C	$[0.03125,\ 32768]$
	kernal	rbf, poly, sigmoid
	degree	$[1, 5]$
	gamma	$[3.0517578125 \times 10^{-5}, 8]$
	coef0	$[-1, 1]$
	shrinking	True, False
	tol	$[10^{-5},\ 10^{-1}]$

Table C.1: Hyper-parameter ranges of Adaboost, Random Forest and SVM

- In Auto-Sklearn, the performance model for Auto-Sklearn is initialized as follows. The best configurations for the top-10 neighbors of the current dataset are retained and run on the current dataset; their performance is used to initialize the Bayesian Optimisation search using the SMAC BO implementation [Hutter et al. 2011]. These top-10 neighbors are computed using the Euclidean distance on the meta-feature set. Note that Auto-Sklearn meta-features were crafted to achieve automatic configuration selection in the context of the Auto-Sklearn pipeline [Pedregosa et al. 2011], thus constituting a most strong baseline on Task 3.
- For PMF, the best configurations for the top-5 neighbors of the current dataset are likewise selected; their performance is computed to fill the row of the collaborative matrix associated to the current dataset, and determine the latent representation of the current dataset. The probabilistic model learned from the matrix is used to select further hyper-parameter configurations; their performances are computed and used to refine the latent representation of the dataset.

C.2 . The hyper-parameter configuration spaces

The hyper-parameters used for Adaboost, Random Forest and SVM and their range are detailed in Tables C.1 and C.2. For Auto-Sklearn, we only included the list of considered hyper-parameters; their ranges are detailed in Auto-Sklearn [Feurer et al. 2015a]. The hyper-parameter space used in PMF is the same as in Auto-Sklearn. The Metabu implementation uses the ConfigSpace library [Lindauer et al. 2019] to manage the hyper-parameters.

C.3 . List of meta-features

Methods	Parameters
balancing	strategy
adaboost	learning_rate, max_depth, n_estimators
bernoulli_nb	fit_prior
decision_tree	max_depth_factor, max_features, max_leaf_nodes, min_impurity_decrease, min_samples_leaf, min_samples_split, min_weight_fraction_leaf
extra_trees	criterion, max_depth, max_features, max_leaf_nodes, min_impurity_decrease, min_samples_leaf, min_samples_split, min_weight_fraction_leaf
gradient_boosting	l2_regularization, learning_rate, loss, max_bins, max_depth, max_leaf_nodes, min_samples_leaf, scoring, tol, n_iter_no_change, validation_fraction
k_nearest_neighbors	p, weights
lda	tol, shrinkage_factor
liblinear_svc	dual, fit_intercept, intercept_scaling, loss, multi_class, penalty, tol
libsvm_svc	gamma, kernel, max_iter, shrinking, tol, coef0, degree
mlp	alpha, batch_size, beta_1, beta_2, early_stopping, epsilon, hidden_layer_depth, learning_rate_init, n_iter_no_change, num_nodes_per_layer, shuffle, solver, tol, validation_fraction
multinomial_nb	fit_prior
passive_aggressive	average, fit_intercept, loss, tol
qda	reg_param
random_forest	criterion, max_depth, max_features, max_leaf_nodes, min_impurity_decrease, min_samples_leaf, min_samples_split, min_weight_fraction_leaf
sgd	average, fit_intercept, learning_rate, loss, penalty, tol, epsilon, eta0, l1_ratio, power_t
extra_trees_preproc_for_classification	criterion, max_depth, max_features, max_leaf_nodes, min_impurity_decrease, min_samples_leaf, min_samples_split, min_weight_fraction_leaf, n_estimators
fast_ica	fun, whiten, n_components
feature_agglomeration	linkage, n_clusters, pooling_func
kernel_pca	n_components, coef0, degree, gamma
kitchen_sinks	n_components
liblinear_svc_preprocessor	dual, fit_intercept, intercept_scaling, loss, multi_class, penalty, tol
nystroem_sampler	n_components, coef0, degree, gamma
pca	whiten
polynomial	include_bias, interaction_only
random_trees_embedding	max_depth, max_leaf_nodes, min_samples_leaf, min_samples_split, min_weight_fraction_leaf, n_estimators
select_percentile_classification	score_func
select_rates_classification	score_func, mode

Table C.2: List of hyper-parameters considered in Auto-Sklearn pipeline.

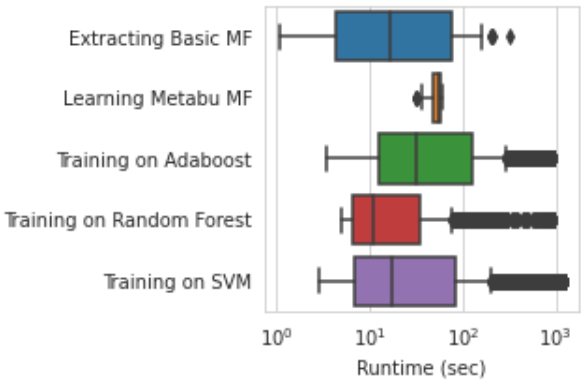


Figure C.1: Metabu computational effort: average runtime of the meta-feature extraction (in blue) and Metabu training (in orange). The average training time of one hyper-parameter on Adaboost (green), Random Forest (red) and SVM (purple) pipelines are added for comparison.

The list of meta-features used in the experiments is detailed in Tables C.3 and C.4. Meta-features are extracted with PyMFE [Alcobaca et al. 2020] except for Auto-Sklearn, SCOT and Landmark meta-features which are computed from the Auto-Sklearn library.

C.4 . Computational effort

Figure C.1 indicates the runtime[1] for pre-processing (extracting the 135 meta-features, top row), and for training Metabu (second row). The average training time for learning one model is indicated for comparison (from row 3 to 5: Adaboost, RandomForest and SVM).

C.5 . The stability of the intrinsic dimension

In Table C.5, we investigate how the intrinsic dimension varies when considering various numbers of datasets in OpenML. It is observed that the intrinsic dimension tends to increase with the number of considered datasets, particularly so for SVM. This suggests that the hyper-parameter configurations investigated in the OpenML benchmark are not sufficiently representative of the (good regions of the) configuration space.

C.6 . Detailed results

[1] On Intel(R) Xeon(R) CPU E5-2660 v2 @ 2.20GHz.

Meta-features	Description	Auto-Sklearn	Landmark	SCOT	Metabu
best_node	Performance of a the best single decision tree node.				+
elite_nn	Performance of Elite Nearest Neighbor.				+
linear_discr	Performance of the Linear Discriminant classifier.				+
naive_bayes	Performance of the Naive Bayes classifier.				+
one_nn	Performance of the 1-Nearest Neighbor classifier.				+
random_node	Performance of the single decision tree node model induced by a random attribute.				+
worst_node	Performance of the single decision tree node model induced by the worst informative attribute.				+
one_itemset	Compute the one itemset meta-feature.				+
two_itemset	Compute the two itemset meta-feature.				+
c1	Compute the entropy of class proportions.				+
c2	Compute the imbalance ratio.				+
cls_coef	Clustering coefficient.				+
density	Average density of the network.				+
f1	Maximum Fisher's discriminant ratio.				+
f1v	Directional-vector maximum Fisher's discriminant ratio.				+
f2	Volume of the overlapping region.				+
f3	Compute feature maximum individual efficiency.				+
f4	Compute the collective feature efficiency.				+
hubs	Hub score.				+
l1	Sum of error distance by linear programming.				+
l2	Compute the OVO subsets error rate of linear classifier.				+
l3	Non-Linearity of a linear classifier.				+
lsc	Local set average cardinality.				+
n1	Compute the fraction of borderline points.				+
n2	Ratio of intra and extra class nearest neighbor distance.				+
n3	Error rate of the nearest neighbor classifier.				+
n4	Compute the non-linearity of the k-NN Classifier.				+
t1	Fraction of hyperspheres covering data.				+
t2	Compute the average number of features per dimension.				+
t3	Compute the average number of PCA dimensions per points.				+
t4	Compute the ratio of the PCA dimension to the original dimension.				+
ch	Compute the Calinski and Harabasz index.				+
int	Compute the INT index.				+
nre	Compute the normalized relative entropy.				+
pb	Compute the pearson correlation between class matching and instance distances.				+
sc	Compute the number of clusters with size smaller than a given size.				+
sil	Compute the mean silhouette value.				+
vdb	Compute the Davies and Bouldin Index.				+
vdu	Compute the Dunn Index.				+
leaves	Compute the number of leaf nodes in the DT model.				+
leaves_branch	Compute the size of branches in the DT model.				+
leaves_corrob	Compute the leaves corroboration of the DT model.				+
leaves_homo	Compute the DT model Homogeneity for every leaf node.				+
leaves_per_class	Compute the proportion of leaves per class in DT model.				+
nodes	Compute the number of non-leaf nodes in DT model.				+
nodes_per_attr	Compute the ratio of nodes per number of attributes in DT model.				+
nodes_per_inst	Compute the ratio of non-leaf nodes per number of instances in DT model.				+
nodes_per_level	Compute the ratio of number of nodes per tree level in DT model.				+
nodes_repeated	Compute the number of repeated nodes in DT model.				+
tree_depth	Compute the depth of every node in the DT model.				+
tree_imbalance	Compute the tree imbalance for each leaf node.				+
tree_shape	Compute the tree shape for every leaf node.				+
var_importance	Compute the features importance of the DT model for each attribute.				+
can_cor	Compute canonical correlations of data.				+
cor	Compute the absolute value of the correlation of distinct dataset column pairs.				+
cov	Compute the absolute value of the covariance of distinct dataset attribute pairs.				+
eigenvalues	Compute the eigenvalues of covariance matrix from dataset.				+
g_mean	Compute the geometric mean of each attribute.				+
gravity	Compute the distance between minority and majority classes center of mass.				+
h_mean	Compute the harmonic mean of each attribute.				+
iq_range	Compute the interquartile range (IQR) of each attribute.				+
kurtosis	Compute the kurtosis of each attribute.				+
lh_trace	Compute the Lawley-Hotelling trace.				+
mad	Compute the Median Absolute Deviation (MAD) adjusted by a factor.				+
max	Compute the maximum value from each attribute.				+
mean	Compute the mean value of each attribute.				+
median	Compute the median value from each attribute.				+
min	Compute the minimum value from each attribute.				+
nr_cor_attr	Compute the number of distinct highly correlated pair of attributes.				+
nr_disc	Compute the number of canonical correlation between each attribute and class.				+
nr_norm	Compute the number of attributes normally distributed based in a given method.				+
nr_outliers	Compute the number of attributes with at least one outlier value.				+
p_trace	Compute the Pillai's trace.				+
range	Compute the range (max - min) of each attribute.				+

Table C.3: List of meta-features, 1/2

Meta-features	Description	Auto-Sklearn	Landmark	SCOT	Metabu
roy_root	Compute the Roy's largest root.				+
sd	Compute the standard deviation of each attribute.				+
sd_ratio	Compute a statistical test for homogeneity of covariances.				+
skewness	Compute the skewness for each attribute.				+
sparsity	Compute (possibly normalized) sparsity metric for each attribute.				+
t_mean	Compute the trimmed mean of each attribute.				+
var	Compute the variance of each attribute.				+
w_lambda	Compute the Wilks' Lambda value.				+
attr_conc	Compute concentration coef. of each pair of distinct attributes.				+
attr_ent	Compute Shannon's entropy for each predictive attribute.				+
class_conc	Compute concentration coefficient between each attribute and class.				+
class_ent	Compute target attribute Shannon's entropy.				+
eq_num_attr	Compute the number of attributes equivalent for a predictive task.				+
joint_ent	Compute the joint entropy between each attribute and class.				+
mut_inf	Compute the mutual information between each attribute and target.				+
ns_ratio	Compute the noisiness of attributes.				+
cohesiveness	Compute the improved version of the weighted distance, that captures how dense or sparse is the example distribution.				+
conceptvar	Compute the concept variation that estimates the variability of class labels among examples.				+
impconceptvar	Compute the improved concept variation that estimates the variability of class labels among examples.				+
wg_dist	Compute the weighted distance, that captures how dense or sparse is the example distribution.				+
attr_to_inst	Compute the ratio between the number of attributes.				+
cat_to_num	Compute the ratio between the number of categoric and numeric features.				+
freq_class	Compute the relative frequency of each distinct class.				+
inst_to_attr	Compute the ratio between the number of instances and attributes.				+
nr_attr	Compute the total number of attributes.				+
nr_bin	Compute the number of binary attributes.				+
nr_cat	Compute the number of categorical attributes.				+
nr_class	Compute the number of distinct classes.				+
nr_inst	Compute the number of instances (rows) in the dataset.				+
nr_num	Compute the number of numeric features.				+
num_to_cat	Compute the number of numerical and categorical features.				+
PCASkewnessFirstPC	Skewness of examples on the first principal component				+
PCAKurtosisFirstPC	Kurtosis of examples on the first principal component				+
PCAFracOfCompFor95Per	Fraction of component of an overall explained variance of 95%			+	+
Landmark1NN	Performance one nearest neighbor classifier				+
LandmarkRandomNodeLearner	Performance of decision when considering only one feature				+
LandmarkDecisionNodeLearner	Performance of decision when considering all features				+
LandmarkDecisionTree	Performance of decision tree classifier		+		+
LandmarkNaiveBayes	Performance of Naive Bayes classifier		+		+
LandmarkLDA	Performance of LDA classifier		+		+
SkewnessSTD	Standard deviation of feature skewness	+			+
SkewnessMean	Mean of feature skewness	+			+
SkewnessMax	Maximum of feature skewness	+			+
SkewnessMin	Minimum of feature skewness	+			+
KurtosisSTD	Standard deviation of feature kurtosis coefficiants	+			+
KurtosisMean	Mean of feature kurtosis coefficiants	+			+
KurtosisMax	Max of feature kurtosis coefficiants	+			+
KurtosisMin	Mean of feature kurtosis coefficiants	+			+
SymbolsSum	Sum of categorical feature symbols	+			+
SymbolsSTD	Standard deviation of categorical feature symbols	+			+
SymbolsMean	Mean of categorical feature symbols	+			+
SymbolsMax	Max of categorical feature symbols	+			+
SymbolsMin	Min of categorical feature symbols	+			+
ClassProbabilitySTD	Standard deviation of class probabilities	+			+
ClassProbabilityMean	Mean of class probabilities	+			+
ClassProbabilityMax	Maximum of class probabilities	+	+		+
ClassProbabilityMin	Minimum of class probabilities	+			+
InverseDatasetRatio	Inverse of dataset ratio	+			+
DatasetRatio	Dataset ratio	+			+
RatioNominalToNumerical	Ratio number of nominal to numerical features	+			+
RatioNumericalToNominal	Ratio numerical to nominal	+			+
NumberOfCategoricalFeatures	Number of categorical features	+	+		+
NumberOfNumericFeatures	Number of numeric features	+	+		+
NumberOfMissingValues	Number of missing values	+			+
NumberOfFeaturesWithMissingValues	Number of features with missing values	+			+
NumberOfInstancesWithMissingValues	Number of instances with missing values	+			+
NumberOfFeatures	Number of features	+	+		+
NumberOfClasses	Number of classes	+	+	+	+
NumberOfInstances	Number of instances	+			+
LogInverseDatasetRatio	log of the inverse dataset ratio	+		+	+
LogDatasetRatio	Log of dataset ratio	+			+
PercentageOfMissingValues	Percentage of missing values	+			+
PercentageOfFeaturesWithMissingValues	Percentage of features with missing values	+			+
PercentageOfInstancesWithMissingValues	Percentage of instances with missing values	+			+
LogNumberOfFeatures	Log number of features	+		+	+
LogNumberOfInstances	Log number of instances	+			+

Table C.4: List of meta-features, 2/2

Dataset Ratio	**0.1**	**0.25**	**0.5**	**0.75**	**1**
Adaboost	5.65 (2.74)	6.81 (1.81)	7.14 (1.44)	6.59 (1.29)	6.98
Random Forest	5.33 (2.17)	7.14 (2.18)	7.44 (1.28)	8.48 (1.56)	8.49
SVM	8.56 (2.54)	11.54 (2.71)	12.83 (3.16)	13.99 (2.40)	14.41
AutoSkLearn	5.17 (2.08)	4.47 (1.26)	4.98 (0.95)	5.34 (1.06)	5.51

Table C.5: Intrinsic dimension of the dataset space w.r.t. ML algorithms Adaboost, RandomForest, SVM and Auto-Sklearn, depending on the fraction of datasets considered in OpenML

Detailed results of Task 2 are presented in Table C.6 for Random Forest, Table C.7 for Adaboost and Table C.8 for SVM. We consider the Mann Whitney Wilcoxon test to assess the significance of the rankings.

OpenML Task ID	Metabu MF	Auto-SklearnMF	Landmark MF	SCOT MF	Random1x
Average Rank	**2.50**	2.97	2.70	2.64	4.17
3	0.993 ±0.000*	0.993 ± 0.001	0.993 ±0.000*	0.993 ± 0.000	0.993 ± 0.001
6	0.965 ±0.001*	0.964 ± 0.001	0.958 ± 0.007	0.964 ±0.002*	0.948 ± 0.007
11	0.655 ±0.002*	0.657 ± 0.002	0.658 ±0.001*	0.656 ± 0.001	0.657 ± 0.002
12	0.964 ±0.002*	0.961 ± 0.001	0.965 ±0.001*	0.963 ± 0.003	0.961 ± 0.004
14	0.820 ±0.005*	0.812 ± 0.004	0.813 ± 0.005	0.814 ±0.003*	0.808 ± 0.005
15	0.983 ±0.004*	0.982 ± 0.005	0.983 ±0.001*	0.981 ± 0.005	0.983 ± 0.004
16	0.955 ±0.007*	0.951 ± 0.010	0.958 ± 0.004	0.959 ±0.003*	0.950 ± 0.005
18	0.675 ±0.002*	0.673 ± 0.005	0.676 ± 0.002	0.678 ± 0.001	0.679 ±0.007*
22	0.765 ±0.001*	0.761 ± 0.004	0.760 ± 0.005	0.769 ±0.003*	0.748 ± 0.014
23	0.535 ± 0.004	0.535 ± 0.004	0.541 ± 0.005	0.542 ± 0.008	**0.552 ± 0.006**
28	0.983 ±0.001*	0.982 ± 0.001	0.983 ± 0.000	0.983 ±0.001*	0.978 ± 0.001
29	0.884 ±0.006*	0.878 ± 0.004	0.882 ±0.007*	0.878 ± 0.005	0.882 ± 0.004
31	0.709 ±0.003*	0.725 ±0.013*	0.716 ± 0.003	0.707 ± 0.010	0.713 ± 0.007
32	0.993 ±0.001*	0.992 ± 0.001	0.992 ± 0.001	0.994 ±0.000*	0.989 ± 0.000
37	0.811 ±0.003*	0.810 ± 0.006	0.812 ±0.005*	0.810 ± 0.011	0.808 ± 0.007
43	0.913 ±0.002*	0.914 ± 0.002	0.915 ± 0.002	0.917 ±0.001*	0.908 ± 0.003
45	0.946 ± 0.001	0.946 ± 0.002	0.947 ± 0.002	**0.953 ± 0.004**	0.944 ± 0.003
49	0.962 ±0.002*	0.965 ±0.002*	0.964 ± 0.000	0.963 ± 0.002	0.957 ± 0.005
53	0.780 ±0.009*	0.777 ± 0.001	0.786 ±0.006*	0.767 ± 0.012	0.768 ± 0.006
219	0.923 ±0.002*	0.919 ± 0.009	0.918 ± 0.004	0.923 ±0.003*	0.913 ± 0.001
2074	0.891 ±0.001*	0.889 ± 0.003	0.889 ±0.002*	0.888 ± 0.003	0.880 ± 0.003
2079	0.638 ±0.007*	0.628 ± 0.010	0.647 ±0.005*	0.639 ± 0.007	0.621 ± 0.009
3021	0.949 ±0.003*	0.943 ± 0.004	0.949 ±0.003*	0.945 ± 0.005	0.933 ± 0.005
3022	0.962 ±0.003*	0.959 ±0.010*	0.945 ± 0.017	0.927 ± 0.014	0.906 ± 0.025
3549	0.951 ± 0.015	**0.984 ± 0.002**	0.967 ± 0.023	0.977 ± 0.007	0.957 ± 0.023
3560	0.253 ±0.006*	0.253 ±0.021*	0.248 ± 0.017	0.250 ± 0.023	0.241 ± 0.013
3902	0.756 ±0.002*	0.754 ±0.009*	0.734 ± 0.009	0.743 ± 0.020	0.753 ± 0.007
3903	0.551 ±0.013*	0.554 ± 0.003	0.555 ± 0.008	0.557 ±0.010*	0.549 ± 0.006
3904	0.602 ± 0.001	0.602 ± 0.003	0.600 ± 0.002	**0.606 ± 0.001**	0.594 ± 0.002
3913	0.619 ± 0.006	0.616 ± 0.016	**0.634 ± 0.013**	0.633 ± 0.001	0.609 ± 0.021
3917	0.667 ±0.015*	0.677 ±0.009*	0.669 ± 0.002	0.672 ± 0.005	0.659 ± 0.004
3918	0.655 ±0.003*	0.661 ±0.008*	0.651 ± 0.008	0.649 ± 0.006	0.652 ± 0.009
7592	0.778 ±0.002*	0.781 ±0.001*	0.777 ± 0.003	0.778 ± 0.002	0.777 ± 0.002
9910	0.807 ±0.002*	0.809 ±0.002*	0.807 ± 0.004	0.805 ± 0.002	0.797 ± 0.007
9946	0.953 ±0.006*	0.951 ± 0.007	0.957 ± 0.008	0.962 ±0.011*	0.941 ± 0.010
9952	0.890 ±0.001*	0.891 ±0.001*	0.882 ± 0.007	0.880 ± 0.006	0.878 ± 0.003
9957	0.866 ±0.007*	0.864 ± 0.008	0.858 ± 0.002	0.872 ±0.002*	0.868 ± 0.005
9960	0.994 ±0.000*	0.993 ± 0.000	0.994 ±0.000*	0.993 ± 0.000	0.993 ± 0.000
9964	0.927 ±0.007*	0.915 ±0.020*	0.912 ± 0.014	0.914 ± 0.005	0.891 ± 0.015
9971	0.563 ± 0.018	**0.587 ± 0.005**	0.584 ± 0.032	0.560 ± 0.020	0.566 ± 0.006
9976	0.845 ±0.007*	0.846 ±0.004*	0.841 ± 0.010	0.840 ± 0.005	0.842 ± 0.006
9977	0.961 ±0.000*	0.960 ± 0.000	0.961 ±0.000*	0.961 ± 0.001	0.960 ± 0.001
9978	0.672 ±0.007*	0.680 ±0.006*	0.671 ± 0.003	0.677 ± 0.009	0.670 ± 0.004
9981	0.926 ± 0.002	0.936 ± 0.011	**0.947 ± 0.019**	0.943 ± 0.030	0.927 ± 0.022
9985	0.475 ±0.007*	0.478 ± 0.004	0.475 ± 0.002	0.479 ±0.004*	0.467 ± 0.007
10093	0.983 ±0.001*	0.984 ± 0.001	0.988 ± 0.004	0.988 ±0.007*	0.987 ± 0.003
10101	0.621 ±0.004*	0.611 ± 0.005	0.621 ±0.005*	0.616 ± 0.005	0.614 ± 0.003
14952	0.965 ±0.000*	0.963 ± 0.002	0.965 ±0.001*	0.963 ± 0.001	0.956 ± 0.004
14954	0.844 ±0.022*	0.835 ± 0.023	0.853 ±0.009*	0.833 ± 0.004	0.799 ± 0.013
14965	0.711 ±0.001*	0.712 ±0.002*	0.710 ± 0.004	0.710 ± 0.003	0.709 ± 0.002
14969	0.597 ±0.005*	0.588 ± 0.007	0.591 ± 0.005	0.595 ±0.002*	0.575 ± 0.008
125920	0.598 ±0.010*	0.597 ± 0.009	0.600 ± 0.021	0.601 ±0.005*	0.589 ± 0.010
125922	0.976 ±0.002*	0.976 ± 0.002	0.976 ±0.001*	0.973 ± 0.005	0.968 ± 0.004
146195	0.642 ±0.002*	0.638 ± 0.004	0.644 ±0.002*	0.642 ± 0.002	0.622 ± 0.005
146800	0.971 ± 0.013	0.980 ± 0.009	0.974 ± 0.006	**0.986 ± 0.002**	0.962 ± 0.010
146817	0.825 ±0.004*	0.817 ± 0.009	0.826 ±0.007*	0.824 ± 0.012	0.812 ± 0.004
146819	0.861 ±0.014*	0.859 ± 0.010	0.861 ± 0.015	0.870 ±0.002*	0.869 ± 0.005
146820	0.863 ±0.004*	0.855 ± 0.014	0.851 ± 0.018	0.831 ± 0.005	0.855 ±0.010*
146821	0.971 ±0.001*	0.972 ±0.001*	0.969 ± 0.002	0.970 ± 0.001	0.971 ± 0.004
146822	0.934 ±0.001*	0.932 ± 0.005	0.930 ± 0.006	0.934 ±0.001*	0.931 ± 0.004
146824	0.968 ±0.003*	0.969 ± 0.001	0.968 ± 0.003	0.969 ±0.002*	0.960 ± 0.007
146825	0.294 ±0.509*	0.582 ± 0.504	0.293 ± 0.507	0.872 ±0.006*	0.869 ± 0.003
167119	0.767 ±0.001*	0.764 ± 0.003	0.762 ± 0.003	0.765 ±0.002*	0.765 ± 0.000
167121	0.275 ±0.476*	0.000 ± 0.000	0.582 ±0.505*	0.000 ± 0.000	0.274 ± 0.475
167125	0.921 ±0.000*	0.922 ± 0.002	0.924 ±0.003*	0.920 ± 0.001	0.899 ± 0.007
167140	0.930 ± 0.004	0.935 ± 0.001	0.933 ± 0.002	**0.938 ± 0.001**	0.924 ± 0.003
167141	0.834 ±0.002*	0.829 ± 0.005	0.833 ± 0.002	0.837 ±0.003*	0.832 ± 0.001

Table C.6: Comparative learning performances on OpenML datasets over sampling 30 configurations of the **Random Forest** pipeline. Performances that are statistically significant compared to the second best are in bold. Statistically comparable performances are indicated with $(^*)$. Pairwise comparison and p-value along the iterations are presented in Figure C.2.

OpenML Task ID	Metabu MF	Auto-SklearnMF	Landmark MF	SCOT MF	Random1x
Average Rank	**2.48**	2.96	2.89	2.85	3.80
3	0.995 ±0.001*	0.994 ± 0.000	0.996 ±0.001*	0.994 ± 0.002	0.996 ± 0.001
6	0.970 ± 0.001	0.969 ± 0.002	0.967 ± 0.002	**0.972 ± 0.001**	0.967 ± 0.005
11	0.928 ±0.083*	0.891 ± 0.071	0.914 ± 0.069	0.973 ±0.017*	0.920 ± 0.093
12	0.977 ±0.001*	0.977 ±0.001*	0.976 ± 0.002	0.977 ± 0.001	0.975 ± 0.002
14	0.827 ±0.007*	0.829 ± 0.004	0.824 ± 0.006	0.825 ± 0.002	0.829 ±0.004*
15	0.964 ±0.004*	0.962 ± 0.006	0.969 ±0.004*	0.967 ± 0.005	0.967 ± 0.006
16	0.963 ±0.001*	0.966 ± 0.001	0.967 ±0.003*	0.964 ± 0.004	0.961 ± 0.001
18	0.691 ±0.015*	0.674 ± 0.003	0.695 ±0.014*	0.689 ± 0.012	0.680 ± 0.018
22	0.796 ±0.002*	0.800 ± 0.004	0.801 ±0.006*	0.794 ± 0.014	0.791 ± 0.005
23	0.572 ±0.009*	0.579 ± 0.011	0.582 ±0.015*	0.575 ± 0.004	0.581 ± 0.009
28	0.988 ±0.001*	0.988 ± 0.001	0.987 ± 0.001	0.989 ±0.001*	0.987 ± 0.001
29	0.865 ± 0.006	0.866 ± 0.008	0.875 ± 0.009	**0.882 ± 0.006**	0.845 ± 0.015
31	0.739 ±0.011*	0.726 ± 0.007	0.734 ± 0.011	0.739 ± 0.019	0.740 ±0.014*
32	0.995 ±0.001*	0.997 ±0.000*	0.996 ± 0.000	0.996 ± 0.001	0.994 ± 0.001
37	0.790 ±0.005*	0.794 ±0.018*	0.771 ± 0.008	0.782 ± 0.010	0.784 ± 0.002
43	0.936 ±0.001*	0.933 ± 0.003	0.934 ±0.003*	0.932 ± 0.004	0.933 ± 0.002
45	0.961 ±0.003*	0.951 ± 0.005	0.957 ±0.005*	0.954 ± 0.004	0.951 ± 0.003
49	0.993 ± 0.002	0.995 ± 0.002	0.997 ± 0.003	0.987 ± 0.012	**0.998 ± 0.002**
53	0.808 ±0.005*	0.783 ± 0.027	0.804 ±0.003*	0.801 ± 0.007	0.801 ± 0.012
219	**0.938 ± 0.001**	0.937 ± 0.000	0.931 ± 0.006	0.933 ± 0.003	0.924 ± 0.001
2074	0.903 ±0.002*	0.902 ±0.001*	0.901 ± 0.001	0.901 ± 0.002	0.901 ± 0.001
2079	0.657 ±0.011*	0.649 ±0.006*	0.648 ± 0.010	0.641 ± 0.007	0.627 ± 0.002
3021	0.955 ±0.005*	0.956 ±0.004*	0.951 ± 0.003	0.955 ± 0.002	0.953 ± 0.002
3022	0.952 ± 0.020	0.969 ± 0.001	0.966 ± 0.005	**0.972 ± 0.002**	0.960 ± 0.008
3549	0.988 ±0.002*	0.988 ±0.002*	0.988 ± 0.001	0.986 ± 0.001	0.985 ± 0.003
3560	0.255 ±0.017*	0.241 ± 0.001	0.258 ± 0.020	0.262 ±0.012*	0.253 ± 0.002
3902	0.762 ±0.004*	0.769 ±0.015*	0.754 ± 0.024	0.755 ± 0.012	0.760 ± 0.014
3903	0.604 ±0.027*	0.581 ± 0.010	0.575 ± 0.016	0.574 ± 0.015	0.598 ±0.015*
3904	0.615 ±0.001*	0.608 ± 0.006	0.612 ± 0.003	0.613 ±0.002*	0.613 ± 0.002
3913	0.665 ±0.037*	0.661 ±0.018*	0.656 ± 0.011	0.649 ± 0.006	0.659 ± 0.019
3917	0.681 ±0.007*	0.682 ± 0.003	0.682 ± 0.008	0.694 ±0.015*	0.681 ± 0.009
3918	0.683 ±0.019*	0.696 ±0.008*	0.688 ± 0.019	0.675 ± 0.018	0.685 ± 0.011
7592	0.798 ±0.005*	0.797 ± 0.000	0.796 ± 0.000	0.799 ±0.004*	0.798 ± 0.001
9910	0.798 ±0.001*	0.796 ± 0.002	0.797 ±0.004*	0.797 ± 0.003	0.793 ± 0.005
9946	0.977 ±0.010*	0.986 ± 0.003	0.993 ±0.007*	0.987 ± 0.011	0.990 ± 0.005
9952	0.899 ±0.004*	0.899 ± 0.002	0.900 ± 0.002	0.900 ±0.002*	0.896 ± 0.001
9957	0.871 ±0.005*	0.871 ± 0.010	0.867 ± 0.006	0.875 ±0.007*	0.868 ± 0.003
9960	0.997 ±0.001*	0.997 ± 0.001	0.997 ± 0.001	0.997 ±0.001*	0.996 ± 0.002
9964	0.932 ± 0.006	**0.943 ± 0.003**	0.940 ± 0.005	0.937 ± 0.008	0.928 ± 0.005
9971	0.563 ±0.033*	0.573 ± 0.042	0.576 ±0.010*	0.558 ± 0.007	0.565 ± 0.027
9976	0.846 ±0.003*	0.834 ± 0.004	0.844 ±0.008*	0.830 ± 0.012	0.833 ± 0.012
9977	0.964 ± 0.002	0.966 ± 0.002	0.964 ± 0.001	0.967 ± 0.001	**0.967 ± 0.001**
9978	0.699 ±0.026*	0.683 ± 0.008	0.696 ±0.021*	0.672 ± 0.016	0.692 ± 0.022
9981	0.885 ±0.011*	0.890 ± 0.004	0.888 ± 0.004	0.894 ±0.002*	0.881 ± 0.003
9985	0.475 ±0.005*	0.473 ± 0.005	0.475 ±0.001*	0.475 ± 0.002	0.473 ± 0.007
10093	0.997 ±0.003*	0.994 ±0.001*	0.991 ± 0.003	0.993 ± 0.004	0.994 ± 0.005
10101	0.619 ±0.011*	0.615 ± 0.000	0.621 ± 0.002	0.626 ±0.013*	0.619 ± 0.009
14952	0.964 ±0.002*	0.963 ± 0.001	0.963 ± 0.002	0.963 ± 0.002	0.964 ±0.001*
14954	0.869 ±0.007*	0.881 ±0.018*	0.872 ± 0.003	0.868 ± 0.002	0.863 ± 0.010
14965	0.711 ± 0.004	0.715 ± 0.002	0.714 ± 0.003	**0.716 ± 0.002**	0.714 ± 0.001
14969	0.617 ±0.004*	0.606 ± 0.015	0.610 ± 0.000	0.610 ±0.004*	0.607 ± 0.012
125920	0.572 ±0.019*	0.569 ± 0.015	0.569 ±0.015*	0.565 ± 0.005	0.555 ± 0.013
125922	0.992 ±0.001*	0.992 ±0.000*	0.991 ± 0.000	0.992 ± 0.001	0.989 ± 0.000
146800	0.996 ±0.002*	0.996 ± 0.003	0.998 ±0.001*	0.997 ± 0.001	0.990 ± 0.004
146817	0.817 ±0.008*	0.812 ± 0.005	0.810 ± 0.014	0.821 ±0.007*	0.805 ± 0.008
146819	0.797 ±0.029*	0.766 ± 0.039	0.782 ± 0.024	0.817 ±0.025*	0.806 ± 0.020
146820	0.859 ±0.004*	0.859 ± 0.008	0.849 ± 0.006	0.855 ± 0.016	0.860 ±0.002*
146821	0.980 ±0.015*	0.974 ± 0.003	0.979 ± 0.009	0.981 ±0.004*	0.970 ± 0.010
146822	0.943 ± 0.000	0.942 ± 0.003	**0.945 ± 0.000**	0.943 ± 0.004	0.941 ± 0.004
146824	0.977 ±0.001*	0.976 ±0.002*	0.976 ± 0.002	0.974 ± 0.001	0.972 ± 0.003
167119	0.818 ±0.002*	0.816 ± 0.003	0.817 ±0.001*	0.815 ± 0.003	0.815 ± 0.001
167125	0.914 ±0.002*	0.914 ±0.000*	0.910 ± 0.002	0.911 ± 0.002	0.912 ± 0.002
167140	0.954 ±0.005*	0.954 ±0.003*	0.953 ± 0.002	0.952 ± 0.003	0.948 ± 0.002
167141	0.824 ±0.004*	0.823 ± 0.002	0.825 ± 0.003	0.826 ±0.001*	0.822 ± 0.003

Table C.7: Comparative learning performances on OpenML datasets over sampling 30 configurations of the **Adaboost** pipeline. Performances that are statistically significant compared to the second best are in bold. Statistically comparable performances are indicated with $(^*)$. Pairwise comparisons and the associated p-value along the iterations are reported in Figure C.3.

OpenML Task ID	Metabu MF	Auto-SklearnMF	Landmark MF	SCOT MF	Random1x
Average Rank	**2.34**	2.91	2.97	3.27	3.48
3	0.995 ±0.001*	0.993 ± 0.004	0.994 ±0.003*	0.982 ± 0.015	0.988 ± 0.001
6	0.827 ±0.253*	0.969 ± 0.008	0.973 ±0.000*	0.967 ± 0.010	0.937 ± 0.000
11	0.967 ±0.045*	0.929 ± 0.060	0.897 ± 0.104	0.996 ±0.007*	0.958 ± 0.015
12	0.975 ±0.007*	0.980 ± 0.001	0.982 ± 0.002	0.984 ±0.007*	0.936 ± 0.018
14	0.857 ±0.029*	0.843 ±0.010*	0.830 ± 0.018	0.824 ± 0.031	0.839 ± 0.014
15	0.980 ±0.009*	0.983 ±0.005*	0.980 ± 0.004	0.982 ± 0.006	0.953 ± 0.002
16	0.981 ±0.004*	0.979 ± 0.002	0.981 ±0.005*	0.981 ± 0.003	0.976 ± 0.011
18	0.728 ±0.013*	0.721 ± 0.025	0.736 ±0.010*	0.731 ± 0.013	0.717 ± 0.014
22	0.836 ±0.016*	0.806 ± 0.001	0.818 ± 0.031	0.807 ± 0.009	0.820 ±0.027*
23	0.561 ± 0.010	0.601 ± 0.001	0.583 ± 0.028	**0.607 ± 0.010**	0.585 ± 0.011
28	0.989 ±0.006*	0.991 ±0.000*	0.988 ± 0.002	0.990 ± 0.002	0.987 ± 0.003
29	0.895 ±0.008*	0.893 ± 0.012	0.895 ± 0.016	0.900 ±0.001*	0.878 ± 0.003
31	0.756 ±0.017*	0.767 ± 0.011	0.780 ±0.012*	0.732 ± 0.015	0.757 ± 0.004
32	0.994 ±0.001*	0.993 ± 0.005	0.993 ± 0.005	0.996 ±0.001*	0.985 ± 0.001
37	0.838 ±0.014*	0.841 ± 0.010	0.853 ± 0.008	0.859 ±0.018*	0.853 ± 0.015
43	0.932 ±0.004*	0.929 ±0.005*	0.922 ± 0.011	0.919 ± 0.013	0.917 ± 0.018
45	0.952 ±0.002*	0.935 ± 0.004	0.943 ± 0.021	0.945 ± 0.010	0.952 ±0.003*
49	0.972 ±0.013*	0.978 ±0.018*	0.950 ± 0.034	0.960 ± 0.035	0.940 ± 0.027
53	0.798 ± 0.059	**0.878 ± 0.004**	0.869 ± 0.022	0.867 ± 0.016	0.846 ± 0.013
219	0.897 ± 0.035	0.843 ± 0.018	**0.938 ± 0.007**	0.856 ± 0.053	0.871 ± 0.049
2074	0.905 ±0.011*	0.888 ± 0.008	0.905 ±0.012*	0.899 ± 0.004	0.887 ± 0.017
2079	0.639 ±0.032*	0.578 ± 0.051	0.648 ±0.008*	0.631 ± 0.034	0.556 ± 0.023
3021	0.962 ±0.011*	0.939 ± 0.017	0.955 ± 0.008	0.889 ± 0.041	0.958 ±0.007*
3022	0.968 ±0.023*	0.980 ± 0.005	0.982 ±0.005*	0.966 ± 0.018	0.878 ± 0.085
3549	0.985 ±0.004*	0.992 ± 0.000	0.987 ± 0.006	0.992 ±0.002*	0.986 ± 0.003
3560	0.235 ±0.002*	0.238 ± 0.012	0.213 ± 0.014	0.243 ±0.014*	0.191 ± 0.007
3902	0.876 ±0.013*	0.829 ± 0.028	0.858 ± 0.010	0.860 ±0.034*	0.851 ± 0.017
3903	0.754 ±0.033*	0.738 ±0.032*	0.676 ± 0.084	0.737 ± 0.026	0.691 ± 0.033
3904	0.642 ±0.016*	0.651 ± 0.017	0.655 ± 0.028	0.665 ±0.014*	0.664 ± 0.008
3913	0.815 ±0.009*	0.833 ±0.018*	0.812 ± 0.021	0.790 ± 0.041	0.804 ± 0.019
3917	0.739 ±0.013*	0.742 ± 0.020	0.739 ± 0.022	0.736 ± 0.028	0.754 ±0.011*
3918	0.758 ±0.029*	0.736 ± 0.059	0.761 ±0.011*	0.732 ± 0.034	0.717 ± 0.033
7592	0.844 ±0.002*	0.831 ± 0.004	0.838 ±0.015*	0.819 ± 0.009	0.822 ± 0.010
9910	0.798 ±0.003*	0.761 ± 0.024	0.774 ± 0.044	0.783 ± 0.017	0.795 ±0.003*
9946	0.982 ±0.012*	0.983 ± 0.005	0.982 ± 0.021	0.994 ±0.010*	0.993 ± 0.002
9952	0.894 ±0.006*	0.885 ±0.017*	0.858 ± 0.031	0.831 ± 0.094	0.877 ± 0.019
9957	0.865 ±0.021*	0.849 ± 0.003	0.850 ± 0.011	0.870 ±0.028*	0.859 ± 0.017
9960	0.995 ±0.001*	0.982 ± 0.017	0.942 ± 0.040	0.989 ± 0.007	0.996 ±0.001*
9964	0.925 ±0.011*	0.939 ±0.020*	0.909 ± 0.043	0.924 ± 0.036	0.922 ± 0.010
9971	0.698 ±0.053*	0.697 ±0.036*	0.674 ± 0.005	0.668 ± 0.006	0.666 ± 0.034
9976	0.756 ±0.121*	0.746 ± 0.131	0.746 ±0.125*	0.739 ± 0.110	0.670 ± 0.054
9977	0.971 ±0.001*	0.948 ± 0.011	0.967 ±0.006*	0.936 ± 0.009	0.965 ± 0.002
9978	0.840 ±0.019*	0.865 ±0.028*	0.865 ± 0.005	0.820 ± 0.053	0.816 ± 0.015
9981	0.980 ±0.009*	0.964 ±0.014*	0.955 ± 0.017	0.953 ± 0.030	0.888 ± 0.003
9985	0.484 ±0.021*	0.475 ± 0.013	0.488 ±0.011*	0.461 ± 0.028	0.477 ± 0.018
10093	1.000 ±0.000*	0.996 ± 0.001	0.993 ± 0.005	0.994 ± 0.007	0.998 ±0.003*
10101	0.675 ±0.003*	0.685 ±0.021*	0.665 ± 0.049	0.661 ± 0.019	0.675 ± 0.047
14952	0.959 ±0.009*	0.959 ± 0.003	0.956 ± 0.006	0.949 ± 0.022	0.963 ±0.002*
14954	0.854 ±0.025*	0.844 ±0.027*	0.800 ± 0.040	0.814 ± 0.014	0.799 ± 0.027
14965	0.857 ±0.014*	0.777 ± 0.072	0.856 ±0.014*	0.838 ± 0.015	0.839 ± 0.006
14969	0.545 ± 0.028	**0.641 ± 0.014**	0.585 ± 0.084	0.546 ± 0.119	0.605 ± 0.004
125920	0.572 ±0.026*	0.577 ±0.033*	0.535 ± 0.024	0.549 ± 0.050	0.566 ± 0.032
125922	0.997 ±0.001*	0.993 ± 0.003	0.996 ±0.002*	0.993 ± 0.002	0.987 ± 0.017
146195	0.694 ±0.042*	0.616 ± 0.111	0.720 ±0.029*	0.551 ± 0.130	0.670 ± 0.048
146800	0.999 ±0.001*	0.999 ± 0.002	0.979 ± 0.015	0.999 ±0.001*	0.994 ± 0.008
146817	0.795 ± 0.021	0.807 ± 0.019	**0.841 ± 0.020**	0.808 ± 0.056	0.802 ± 0.036
146819	0.848 ±0.015*	0.812 ± 0.039	0.818 ± 0.022	0.824 ± 0.011	0.836 ±0.016*
146820	0.924 ±0.030*	0.919 ± 0.045	0.963 ±0.005*	0.875 ± 0.107	0.955 ± 0.008
146821	0.963 ±0.040*	0.969 ± 0.021	0.942 ± 0.052	0.990 ±0.010*	0.983 ± 0.008
146822	0.927 ±0.012*	0.939 ±0.015*	0.935 ± 0.008	0.916 ± 0.024	0.926 ± 0.003
146824	0.982 ±0.002*	0.968 ± 0.012	0.954 ± 0.033	0.976 ±0.006*	0.974 ± 0.008
146825	0.857 ±0.014*	0.847 ±0.020*	0.828 ± 0.041	0.839 ± 0.022	0.810 ± 0.056
167119	0.890 ±0.006*	0.874 ± 0.036	0.882 ±0.026*	0.851 ± 0.013	0.872 ± 0.013
167121	**0.912 ± 0.025**	0.726 ± 0.158	0.619 ± 0.003	0.715 ± 0.180	0.731 ± 0.154
167125	0.897 ±0.005*	0.891 ± 0.002	0.889 ± 0.003	0.894 ±0.022*	0.884 ± 0.004
167140	0.944 ±0.005*	0.942 ± 0.006	0.946 ±0.004*	0.931 ± 0.000	0.886 ± 0.009
167141	0.801 ±0.092*	0.842 ± 0.031	0.838 ± 0.027	0.818 ± 0.027	0.846 ±0.027*

Table C.8: Comparative learning performances on OpenML datasets over sampling 30 configurations of the **SVM** pipeline. Performances that are statistically significant compared to the second best are in bold. Statistically comparable performances are indicated with $(^*)$. Pairwise comparisons and the associated p-value along the iterations are presented in Figure C.4.

	Random Forest				Adaboost				SVM			
d \ NDCG@k	10	15	20	25	10	15	20	25	10	15	20	25
2	0.57	0.65	0.71	0.76	0.6	0.67	0.73	0.78	0.55	0.62	0.68	0.7
5	0.58	0.65	0.71	0.75	0.6	0.67	0.73	0.78	0.58	0.65	0.7	0.7
10	0.57	0.65	0.71	0.76	0.63	0.7	0.75	0.8	0.58	0.65	0.71	0.7
15	0.58	0.66	0.72	0.76	0.62	0.7	0.75	0.79	0.57	0.65	0.71	0.7
20	0.58	0.67	0.73	0.77	0.62	0.69	0.74	0.79	0.58	0.65	0.71	0.7
25	0.57	0.65	0.71	0.76	0.62	0.69	0.75	0.79	0.58	0.65	0.71	0.7
intrinsic	0.59	0.67	0.73	0.78	0.62	0.69	0.75	0.8	0.59	0.67	0.73	0.7

Table C.9: Sensitivity of Metabu w.r.t the number d on Task 1. The performance is the NDCG@k score measuring the relevance of the ranking induced by Metabu w.r.t. the target representation.

C.7 . Pairwise Comparisons

Figs. C.2-C.4 highlight a side-by-side comparison of Metabu with each baseline set of meta-features. These comparisons establish the relative improvement over each baseline, that may be lost in the general comparison, Figure 6.4.

On Random Forest pipeline, Metabu performs on par with SCOT meta-features. Whereas its improvement over Landmark MF is only significant between the (approximately) 8th and 23rd iteration, Metabu consistently outperforms Random and Auto-Sklearn meta-features along the iterations.

On Adaboost, Metabu performs similarly as Landmark meta-features. Interestingly, Metabu always has a better average rank than the baselines except for the first two iterations of the Auto-Sklearn baseline. It is seen that the p-value is most generally below the threshold .05, establishing the statistical significance of the rank performance.

Lastly, the gaps in performance for SVM are striking. Metabu consistently outperforms all the baselines meta-features with high confidence.

C.8 . Sensitivity Analysis of d

Table C.9 reports the NDCG@k performance of Metabu on Task 1 for varying values of d, showing that: i) the best results are obtained for the intrinsic dimension in the vast majority of cases; ii) the sensitivity w.r.t. d is very moderate.

The intrinsic dimension d of the OpenML benchmark is circa 6 for Auto-Sklearn, 8 for Adaboost, 9 for RandomForest and 14 for Support Vector Machines.

C.9 . Performance Curves

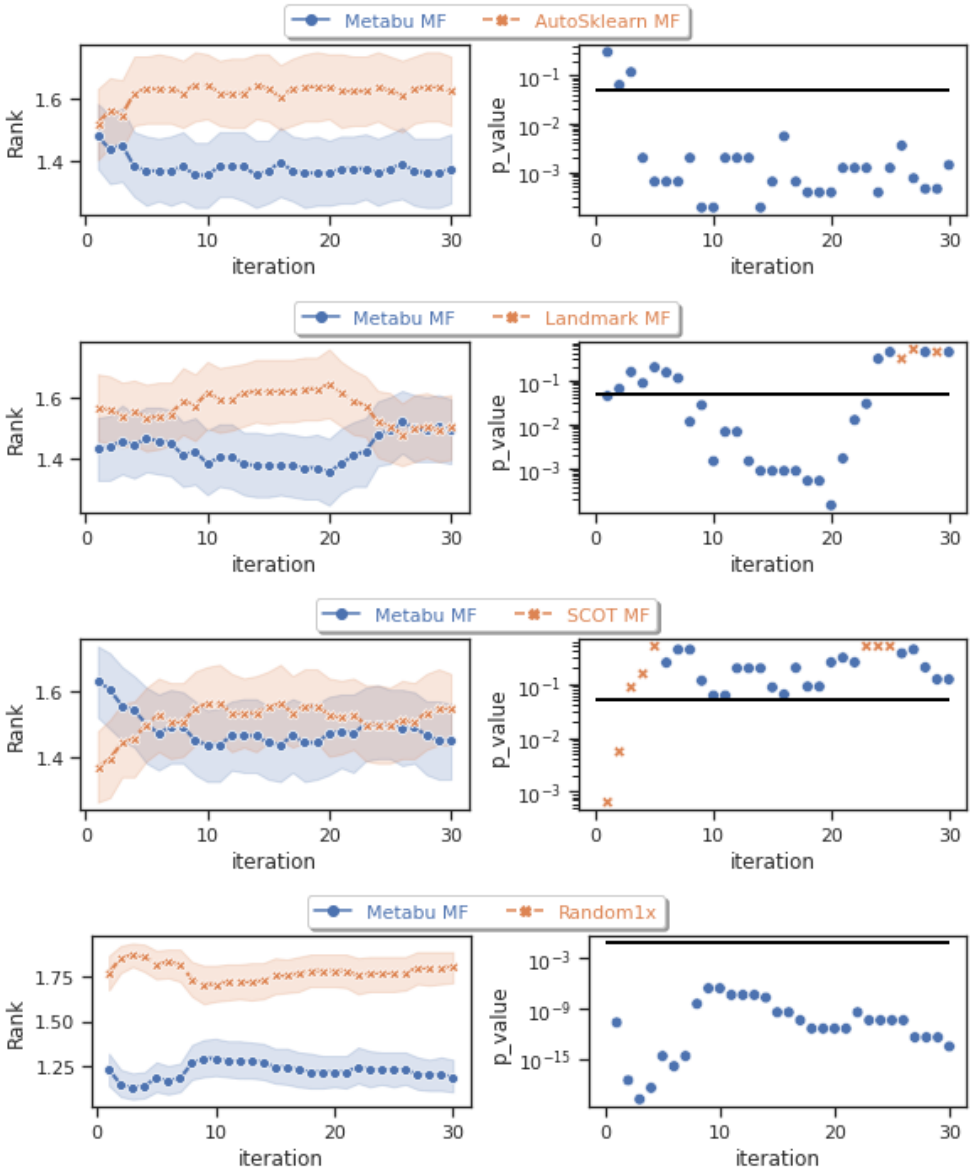


Figure C.2: Pairwise comparison of Metabu with baseline meta-features on **Random Forest** pipeline. Left: the average ranks. Right: the p-value assessing the statistical significance of the ranks according to the Mann-Whitney Wilcoxon test; the black horizontal line indicates the significance threshold p-value=0.05.

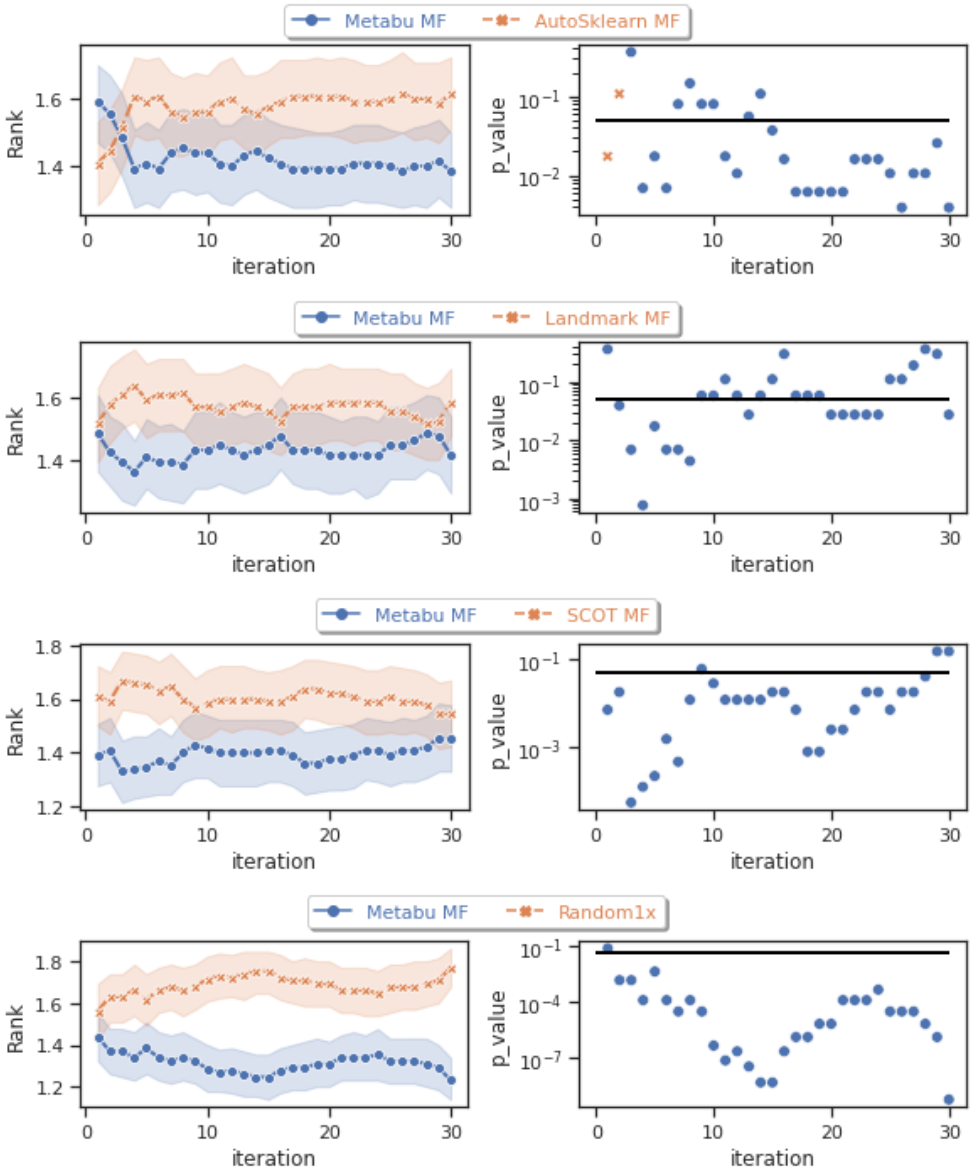


Figure C.3: Pairwise comparison of Metabu with baseline meta-features on **Adaboost** pipeline. Left: the average ranks. Right: the p-value assessing the statistical significance of the ranks according to the Mann-Whitney Wilcoxon test; the black horizontal line indicates the significance threshold p-value=0.05.

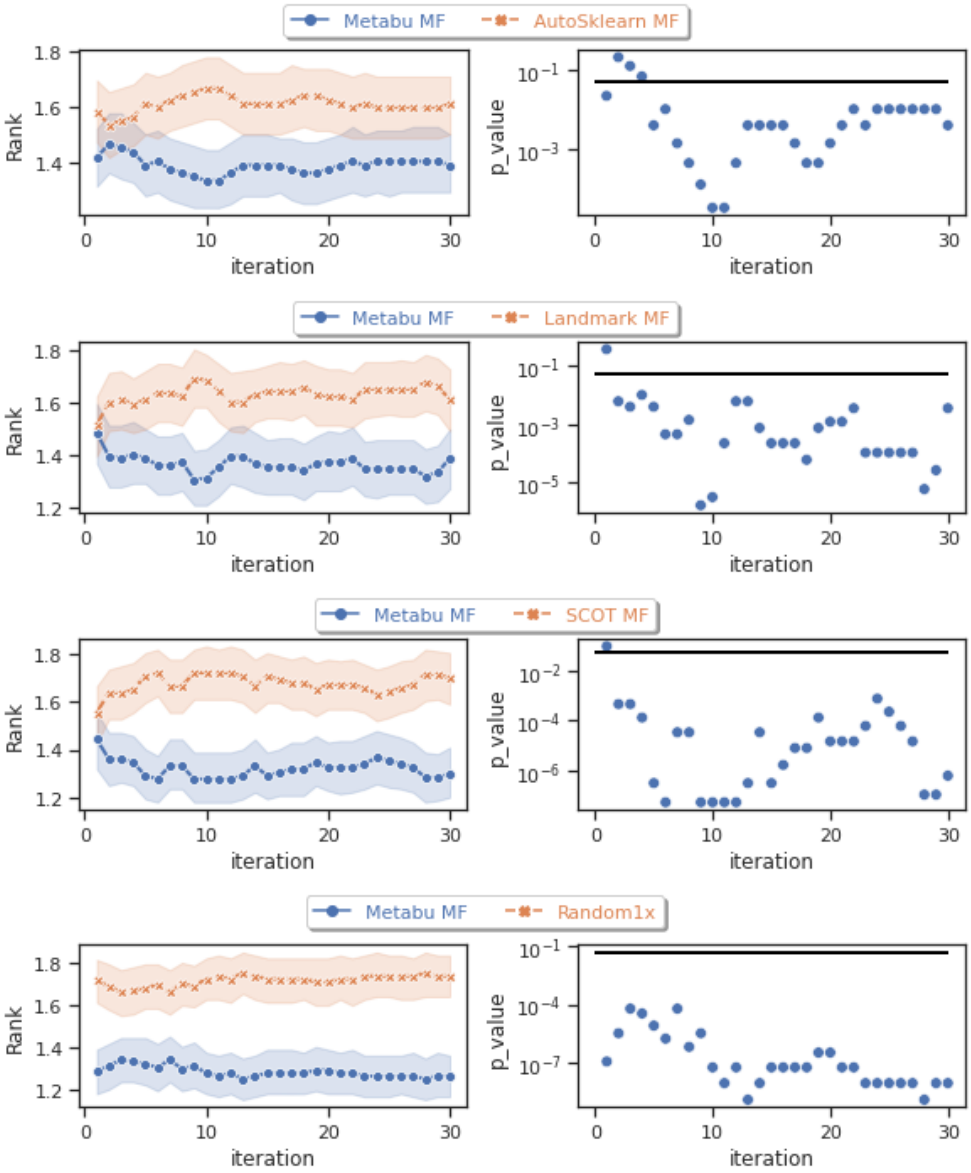


Figure C.4: Pairwise comparison of Metabu with baseline meta-features on **SVM** pipeline. Left: the average ranks. Right: the p-value assessing the statistical significance of the ranks according to the Mann-Whitney Wilcoxon test; the black horizontal line indicates the significance threshold p-value=0.05.

These curves, in addition to the rank results displayed in Figure 6.4b, display the performance values on 10 representative datasets from OpenML CC-18, in the context of Task 2 for respectively Random Forest (Figure C.5), Adaboost (Figure C.6), and SVM (Figure C.7). At each iteration, the curve reports the average performance value with its the standard deviation (on 3 runs).

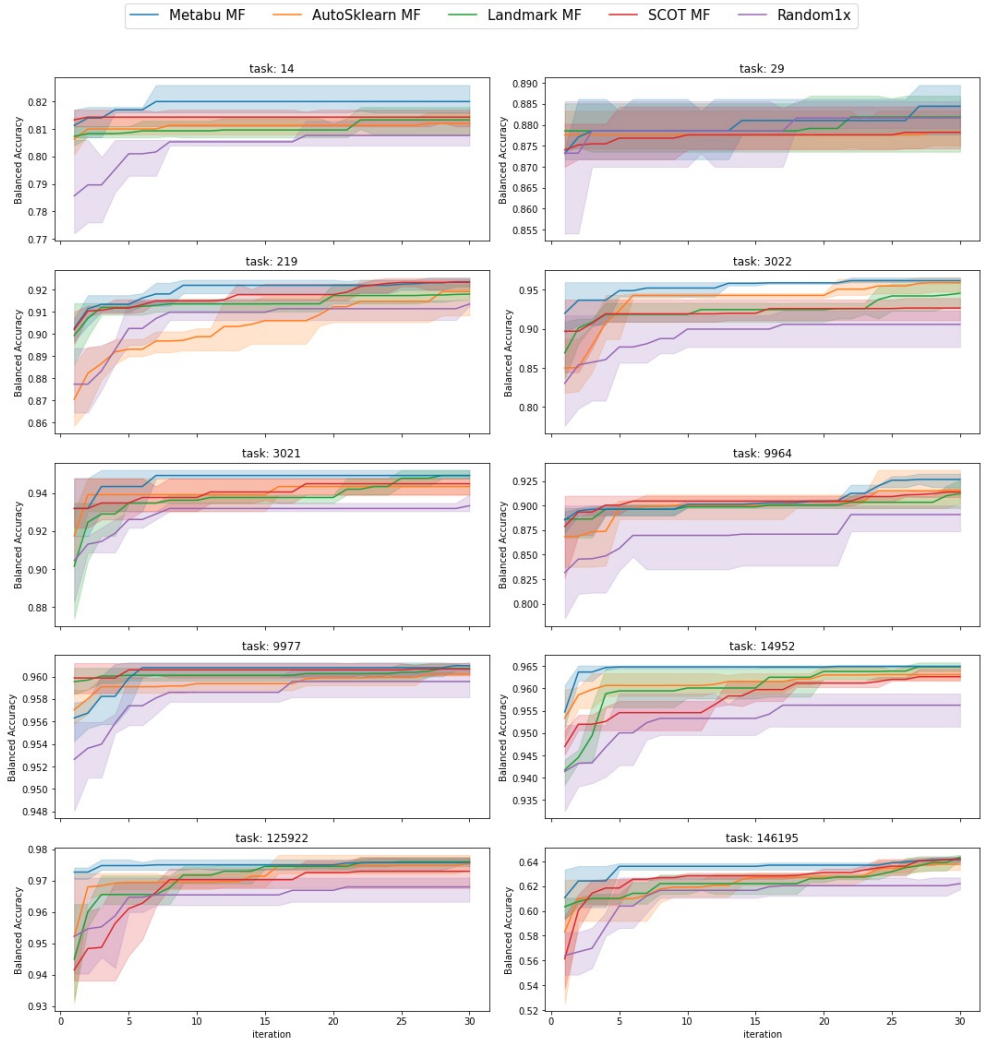


Figure C.5: Performance curves on Random Forest.

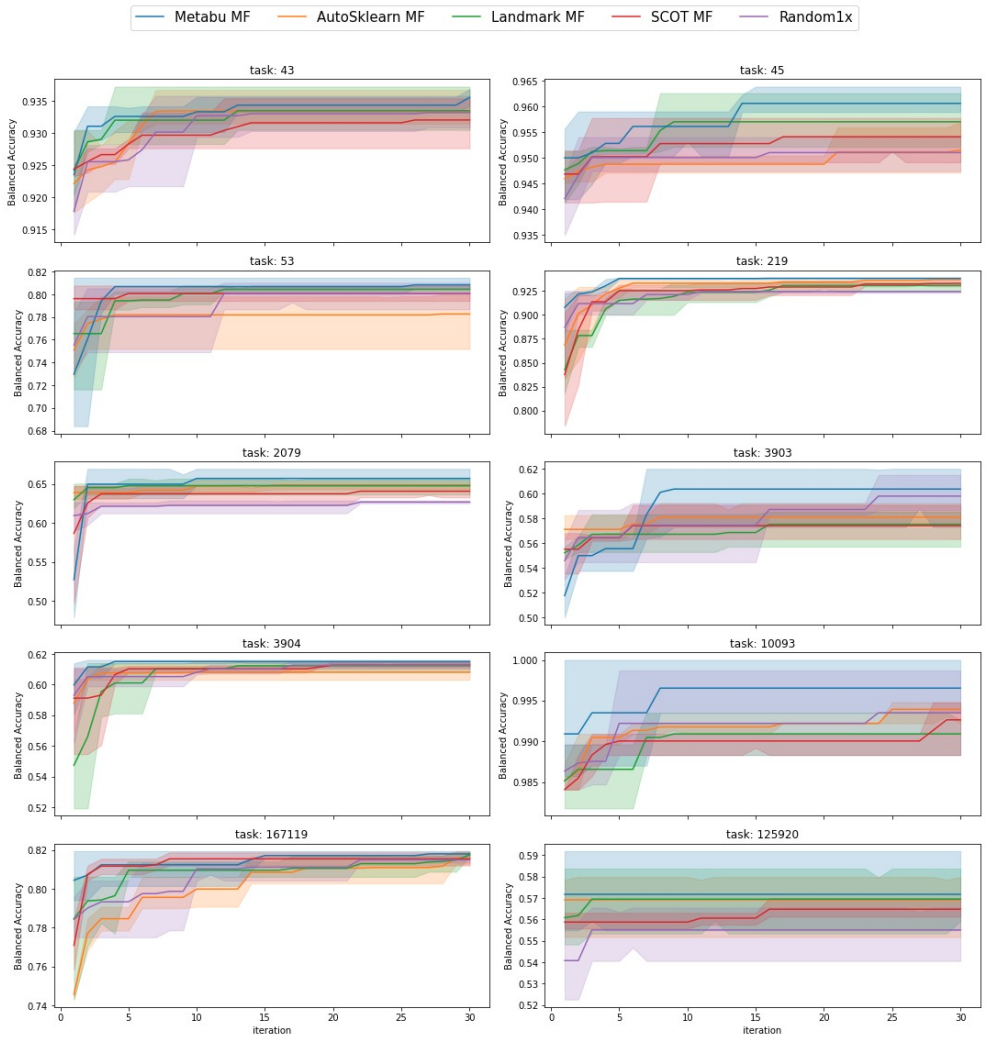


Figure C.6: Performance curves on Adaboost.

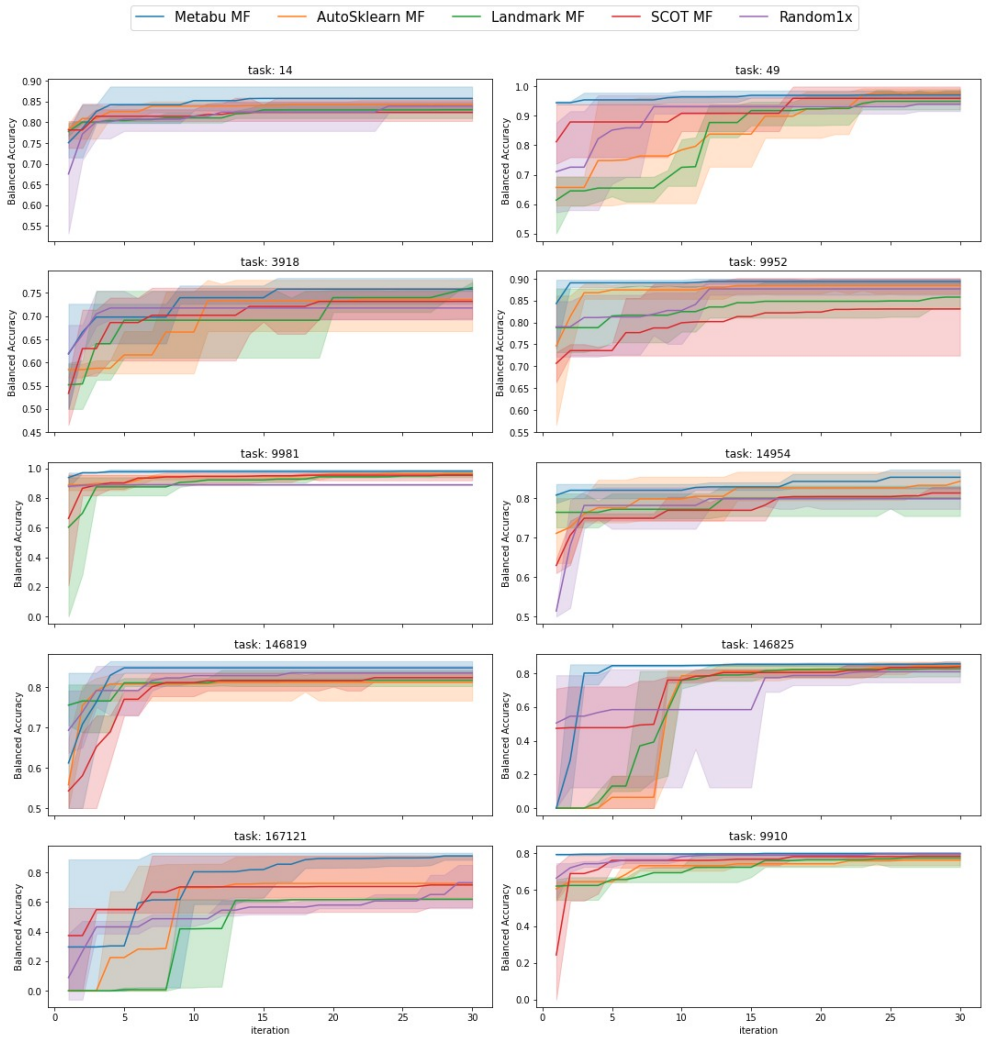


Figure C.7: Performance curves on SVM.