ROBUST AND EFFICIENT AUTOMATED MACHINE LEARNING

Systems, Infrastructure and Advances in Hyperparameter Optimization

DISSERTATION ZUR ERLANGUNG DES DOKTORGRADES DER
TECHNISCHEN FAKULTÄT DER
ALBERT-LUDWIGS-UNIVERSITÄT FREIBURG IM BREISGAU

vorgelegt von

MATTHIAS FEURER

Dekan
Prof. Dr. Roland Zengerle

Erstgutachter und Betreuer der Arbeit
Prof. Dr. Frank Hutter

Zweitgutachter
Prof. Dr. Joaquin Vanschoren

Datum der mündlichen Prüfung
16.09.2022

License: CC-BY 4.0, see
https://creativecommons.org/licenses/by/4.0/

ABSTRACT

Automated Machine Learning (AutoML) is a new paradigm that democratizes machine learning and that will enable its widespread use. AutoML systems provide hands-free machine learning and automatically search for the best models, tune their hyperparameters, and ensemble them. Hyperparameter Optimization (HPO) is a core part of AutoML research and also a research field on its own as models and datasets increase in size and complexity. However, current AutoML systems and HPO methods are resource-hungry and not yet robust enough for all the settings in which we would like to deploy them.

While machine learning automates programming by learning programs from data, AutoML goes one step further and allows us to leave such tasks entirely to the computer. It reduces the burden on the human expert and automates many steps required to build well-performing machine learning models. We are in dire need of such systems because of the increased usage of machine learning and the simultaneous shortage of machine learning practitioners and experts.

In this thesis, we make three contributions to increase the accessibility of machine learning by developing efficient and robust AutoML methods.

First, we address the problem of optimizing hyperparameters efficiently. Concretely, we survey the field of HPO, focusing on so-called *multi-fidelity* methods that efficiently optimize expensive machine learning algorithms by using cheaper approximations. Then, we develop a new method for *transfer* HPO, i.e., an HPO method that leverages knowledge gained on previous optimization tasks. It features strong empirical performance, worst-case performance guarantees, and is hyperparameter-free.

Second, we introduce our work on the efficient and robust AutoML system Auto-sklearn and its successor Auto-sklearn 2.0. We start by proposing Auto-sklearn, an extensible AutoML system that constructs linear machine learning pipelines from 15 classifiers, 16 preprocessing methods, and four data cleaning methods from scikit-learn. To improve over previous AutoML systems, we propose meta-learning and ensembling. We extend on this with Auto-sklearn 2.0, in which we improve the meta-learning component and add new options to make Auto-sklearn suitable for a broader number of use cases. However, this opens up an HPO problem on the AutoML system level, and we introduce a meta-level meta-learning approach that adapts the AutoML system itself to the task at hand. Moreover, we describe how we used Auto-sklearn to win the 1st and 2nd ChaLearn AutoML competition.

Third, we enable AutoML research by contributing to the OpenML platform. Concretely, we create the OpenML-Python API, which gives us access to the OpenML platform with all its datasets. Then, to simplify access to datasets on OpenML and organize the thousands of datasets on OpenML, we develop OpenML benchmarking suites. These are curated datasets to standardize benchmarking practices in machine learning, and we have also used them in our works on HPO and AutoML.

Finally, we discuss how to improve the Auto-sklearn AutoML system further, pose several open questions to the field of AutoML to further increase the automation in machine learning practice and better understand the developed AutoML methods, and give suggestions on how to facilitate better benchmarking in AutoML using the OpenML platform.

In total, we demonstrate how machine learning can be made accessible by robust and efficient Automated Machine Learning and demonstrate substantial performance gains compared to previous AutoML systems.

ZUSAMMENFASSUNG

Automatisiertes maschinelles Lernen (AutoML) ist ein neues Paradigma, das maschinelles Lernen demokratisiert und es einer breiten Anwendergruppe zur Verfügung stellen wird. AutoML-Systeme ermöglichen maschinelles Lernen ohne händische Eingriffe und suchen automatisch nach den besten Modellen, optimieren ihre Hyperparameter und kombinieren sie in Ensembles. Hyperparameter-Optimierung (HPO) ist ein Kernstück der AutoML-Forschung und auch ein eigenständiges Forschungsfeld, da Modelle und Datensätze immer größer und komplexer werden. Aktuelle AutoML-Systeme und HPO-Methoden benötigen jedoch viele Resourcen und sind noch nicht robust genug.

Maschinelles Lernen automatisiert das Programmieren, indem Programme aus Daten gelernt werden. AutoML geht noch einen Schritt weiter und erlaubt uns, solche Aufgaben vollständig dem Computer zu überlassen. Es reduziert die Arbeitsbelastung von Experten und automatisiert viele Schritte, die zum Erstellen leistungsfähiger Modelle für maschinelles Lernen erforderlich sind. Es besteht ein großer Bedarf an solchen Systeme, da auf der einen Seite maschinelles Lernen immer öfters eingesetzt wird, aber andererseits auch Fachkräftemangel herrscht.

In dieser Arbeit leisten wir drei wichtige Beiträge, um die Zugang zu maschinellem Lernen durch die Entwicklung effizienter und robuster AutoML-Methoden zu erleichtern.

Zuerst widmen wir uns HPO. Wir erstellene eine Übersichtsarbeit, wobei wir uns auf sogenannte Multi-Fidelity-Methoden konzentrieren, die teure maschinelle Lernalgorithmen effizient optimieren, indem sie günstigere Näherungen mit niedrigerer Genauigkeit verwenden. Danach entwickeln wir eine neue Transfer-HPO-Methode, d. h. eine HPO-Methode, die Erkenntnisse aus vorherigen Optimierungsaufgaben nutzt. Sie zeichnet sich durch eine starke empirische Leistung und dem Vorhandensein von Worst-Case-Leistungsgarantien aus und kommt zudem komplett ohne Hyperparameter aus.

Als Zweites stellen wir unsere Arbeit am effizienten und robusten AutoML-System Auto-sklearn und seinem Nachfolger Auto-sklearn 2.0 vor. Wir beschreiben zuerst Autosklearn, ein erweiterbares AutoML-System, das lineare Pipelines für maschinelles Lernen aus 15 Klassifikatoren, 16 Vorverarbeitungsmethoden und vier Datenbereinigungsmethoden von scikit-learn erstellt. Zur Verbesserung gegenüber früheren AutoML-Systemen schlagen wir Meta-Learning und Ensembling vor. Mit Auto-sklearn 2.0 erweitern wir die Methodik und verbessern zum einen die Meta-Learning-Komponente und erweitern zum anderen die Einstellmöglichkeiten von Auto-sklearn um es für eine größere Menge von Anwendungsfällen geeignet zu machen. Dies eröffnet jedoch ein HPO-Problem auf der Ebene des AutoML-Systems, und wir führen einen Meta-Learning-Ansatz auf der Meta-Ebene ein, welcher das AutoML-System selbst an die jeweilige Aufgabe anpasst. Darüber hinaus beschreiben wir, wie wir mit Auto-sklearn den 1. und 2. ChaLearn AutoML-Wettbewerb gewonnen haben.

Als Drittes tragen wir zur OpenML-Plattform bei und ermöglichen somit AutoML-Forschung im großen Stil. Konkret erstellen wir die OpenML-Python-API, die uns Zugriff auf die OpenML-Plattform mit all ihren Datensätzen gibt. Anschließend entwickeln wir OpenML-Benchmarking-Suiten, um den Zugriff auf Datensätze auf OpenML zu vereinfachen und die Tausenden von Datensätzen auf OpenML zu organisieren. Dies sind kuratierte Datensätze zur Standardisierung von Benchmarking-Praktiken beim maschinellen Lernen, und wir haben sie auch in unseren Beiträgen zu HPO und AutoML verwendet.

Abschließend diskutieren wir, wie das AutoML-System Auto-sklearn weiter verbessert werden kann, erörtern offene Fragen zum Bereich AutoML, und machen Vorschläge, wie ein mit der OpenML-Plattform besseres Benchmarking für AutoML erreicht werden kann.

Zusammenfassend zeigen wir wie maschinelles Lernen durch robustes und effizientes AutoML besser verfügbar gemacht werden kann, und demonstrieren erhebliche Leistungssteigerungen im Vergleich zu früheren Ansätzen.

ACKNOWLEDGMENTS

First of all I am sincerely grateful to my supervisor and first examiner, Frank Hutter. His Master's seminar on tuning hyperparameters made me start working on what is now known as AutoML. I'm thankful that he gave me the chance to do the PhD in the first place, supported me throughout the whole time, allowed me to contribute to the AutoML workshop series and all the different projects that I had the honor to participate in. His genuine enthusiasm, his capability to come up with an endless stream of great ideas, and his great dedication to details, simplification, and always submitting the best possible paper had a great impact on me and my thesis.

Second, I would like to extend my sincere thanks to Joaquin Vanschoren for being not only the second examiner, but also for initiating, maintaining and advancing the OpenML project, organizing the OpenML workshops, and being a wonderful co-author.

Third, I would like to thank Professor Nebel and Josif Grabocka who completed my PhD thesis examination committee as chairperson and observer. Moreover, I would like to acknowledge the DFG, ERC and Robert Bosch GmbH for funding the projects at the University of Freiburg I was employed to work on, which made my thesis possible at all. Furthermore, special thanks to Ben Letham and Eytan Bakshy from Facebook (now Meta) for hiring me as an intern and guiding me through my internship project, and also to everyone from the AE team for giving me a great time.

Furthermore, I would like to express my deepest gratitude to all co-authors of the papers in this thesis for their work, support, comments, ideas and input; as well as all co-authors of the papers which are not part of this thesis for letting me contribute. Then I'd like to thank everyone from the machine learning lab (the former AAD group) and also the former machine learning lab of Professor Riedmiller for the great time and discussions. The AAD group started small with just Aaron Klein, Marius Lindauer, Tobias Domhan, Katharina Eggensperger and me (and Robin Schirrmeister sometimes joining), and we often met with Manuel Watter, Manuel Blum and Tobias Springenberg from the former machine learning lab. Over time the group grew, and Stefan Falkner, Ilya Loshchilov, Matilde Gargiani, Jan van Rijn and James Wilson joined; together we have formed Mark 1 of what is now Frank's machine learning lab and played a lot of Smash Brothers. For the Mark 2 formation of the lab Thomas Elsken, Arbër Zela, André Biedenkapp, Raghu Rajan, Jörg Franke, Noor Awad and Lucas Zimmer joined. Unfortunately, I haven't really seen those who joined during the pandemic, except for Eddie Bergman, who

is already shaping the future of Auto-sklearn. During my time at the University Petra Geiger, Steffi Beysel, Svenja Wittpoth, Morgan Bambey, Zwetelina Steinbach and Evelyn Rusdea took care of my (up to three parallel) contracts and made sure everything went smooth on the administrative side. The thesis would also not have been possible without our clusters *META* and *META2* which were administered by Uli Jakob, Samuel Weishaupt and Stefan Stäglich. Besides doing my own research I had the opportunity to supervise students doing their research and also learn from them, and in no particular would like to thank Hector, Konstantina, Misgana, Mohsin, Francisco, Jorn, Anatolii, Farooq, Jinu, Guilherme, Lukas, Arlind and Neeratyoy.

Next, I would like to thank the academic and open source communities. Throughout the thesis I developed and maintained several open source packages (*OpenML-Python* and *liac-arff*; *ConfigSpace*, *SMAC3* and *Auto-sklearn*). I am grateful for the opportunity to do so, and would like to thank everyone who contributed back to them to further improve the projects. They are a core part of my thesis and hopefully continue to be so for other researchers, too. I conducted the work on OpenML together with the great OpenML community. I had the great pleasure to attend a total of nine OpenML workshops in person (and the tenth one only online due to the pandemic) and always had a great time. These workshops have tremendously helped me moving forward with the papers and my thesis. Everyone from the OpenML community was always happy to discuss and explain all aspects of machine learning, and besides Joaquin who I already mentioned above, I would like to thank Jan van Rijn and Bernd Bischl for their devotion to make OpenML a huge success, and Andreas Müller for teaching me a lot about developing open source software. I also very much enjoyed developing SMAC3 and very much appreciate the great effort of Marius Lindauer and the whole SMAC3 team. Lastly in this block, I would like to thank ChaLearn and in particular Hugo Jair Escalante and Isabelle Guyon for organizing the 1st and 2nd AutoML challenge in which we could demonstrate our work, and which also challenged us to push the limits of our software and methods.

I would be remiss not to mention my family who always supported me, believed in me, and always encouraged me. Then, I am also thankful to my friends for distracting me from the PhD student life.

Last by not least, I would like to thank Katha for the joy and sunshine she brings into my life and for her endless support and patience during my time as a PhD student.

CONTENTS

I INTRODUCTION
1 INTRODUCTION 3
1.1 Motivation 3
1.2 Goals of this Thesis 5
1.3 How to read this Thesis 9
2 CONTRIBUTIONS 11
2.1 Contributions 11
2.1.1 Part ii: Hyperparameter optimization 11
2.1.2 Part iii: Automated Machine Learning 13
2.1.3 Part iv: Benchmarking and the OpenML Platform 16
2.2 List of Publications 19

II PUBLICATIONS ON HYPERPARAMETER OPTIMIZATION
3 HYPERPARAMETER OPTIMIZATION 23
4 PRACTICAL TRANSFER LEARNING FOR BAYESIAN OPTIMIZATION 57

III PUBLICATIONS ON AUTOMATED MACHINE LEARNING
5 EFFICIENT AND ROBUST AUTOMATED MACHINE LEARNING 99
6 AUTO-SKLEARN 2.0: HANDS-FREE AUTOML VIA META-LEARNING 111

IV PUBLICATIONS ON BENCHMARKING AND THE OPENML PLATFORM
7 OPENML-PYTHON: AN EXTENSIBLE PYTHON API FOR OPENML 161
8 OPENML BENCHMARKING SUITES 169

V CONCLUSION
9 CONCLUSION 187
9.1 Summary and Discussion 187
9.1.1 Hyperparameter optimization 187
9.1.2 Automated Machine Learning 188
9.1.3 Benchmarking and the OpenML Platform 190
9.2 Future Work 191
9.2.1 Future Work To Improve Auto-sklearn 191
9.2.2 What makes a well-performing AutoML system? 192
9.2.3 Extending the Scope of AutoML and Bringing the Human Back into the Loop 196

9.2.4 Relation to Deep Learning and Neural Architecture Search 197
9.2.5 OpenML and Benchmarking 198

VI APPENDIX

A APPENDIX FOR PRACTICAL TRANSFER LEARNING FOR BAYESIAN OPTIMIZATION 203

B APPENDIX FOR EFFICIENT AND ROBUST AUTOMATED MACHINE LEARNING 219

C APPENDIX FOR AUTO-SKLEARN 2.0: HANDS-FREE AUTOML VIA META-LEARNING 235

D APPENDIX FOR OPENML BENCHMARKING SUITES 253

BIBLIOGRAPHY 257

Part I

INTRODUCTION

1 INTRODUCTION

1.1 MOTIVATION

The problems we aim to solve with computer programs are growing in size and complexity beyond what can be efficiently programmed in a traditional manner. Two examples are winning the game of Go and image classification in computer vision. For playing Go, in principle, we know the algorithm to find the optimal move for every board position at every turn of the game: we simulate all possible games and pick the move with the highest chance of winning. Unfortunately, Go has too many board positions, and we cannot iterate over all of them because of their combinatorial explosion.[1] For image classification, things are even worse. Here we know what a classification system should output (the object visible in an image), but not how to create such a program. It would be infeasible to write rules for such a system yet to account for all edge cases. While the nature of these two example problems is different, they share the same solution: *machine learning*.

Machine learning has not only disrupted playing the game of Go (Silver, A. Huang, et al., 2016; Silver, Hubert, et al., 2018; Schrittwieser et al., 2020) and computer vision (LeCun et al., 1989; Krizhevsky, Sutskever, and Hinton, 2012; He et al., 2016). It has also disrupted other fields where there is an abundance of data but no feasible or known program to handle the problem, for example, biology (King, Hirst, and Sternberg, 1993; Jumper et al., 2021) and natural language processing (Jurafsky and Martin, 2022). There are many other applications, which can be found in numerous text books on machine learning (Hastie, Tibshirani, and Friedman, 2001; Bishop, 2006; Alpaydin, 2010; Murphy, 2012; Goodfellow, Bengio, and Courville, 2016).

The following two sentences from Alpaydin's book *Introduction to Machine Learning* (Alpaydin, 2010) are an excellent summary of the goal and inner working of machine learning:

> Machine learning is programming computers to optimize a performance criterion using example data or past experience. We have a model defined up to some parameters, and learning is the execution of a computer program to optimize the parameters of the model using the training data or past experience.

1 The exact number of legal Go positions was computed only in 2016 (Tromp, 2016); it is 208168199381979984699478633344862770286522453884530548425639456820927419612738015378525648451698519643907259916015628128546089888314427129715319317557736620397247064840935, and can be approximated by $2.081681994e^{170}$.

From a high-level point of view, machine learning has two ingredients: *data* and *models*. Data is required to train the models, and our goal is to extract patterns and rules from that data that cannot be programmed otherwise.

However, choosing the right model requires understanding them in-depth to choose the right one for the problem at hand. To make things worse, such models have hyperparameters that govern how the model handles the data that needs to be tuned to the problem at hand. Moreover, it is insufficient to apply only a model to new data. Instead, one must include several preprocessing steps that must be selected and tuned for the current data.

In practice, one experiments with multiple models and hyperparameter settings in a trial-and-error fashion which can be time and compute-intensive. Therefore, applying machine learning requires a thorough understanding of the target domain, machine learning models and pipelines, and their optimization. In addition, it also requires understanding experimentation in order to tune the hyperparameters properly.

Human experts use the algorithms they know about; for example, a user trained only in deep learning will apply it even though tree-based methods might perform superior for the given data (King, Feng, and Sutherland, 1995; LeDell, 2019). Also, the space of possible choices might be too large to have a good overview for a human practitioner (Fernández-Delgado et al., 2014). Besides these methodological problems, there is also a severe shortage of machine learning practitioners to make such modeling choices in all of the relevant applications (Teichmann, 2021).

To unlock the full potential of machine learning, it is therefore of paramount importance to reduce the barriers to machine learning. AutoML, short for *Automated Machine Learning*,[2] aims to overcome these barriers by researching and developing methods and tools that automatically search for the best machine learning model for a dataset at hand.

The problem of correctly setting the hyperparameters is referred to as Hyperparameter Optimization (HPO). HPO can be used to tune the hyperparameters of a model in a one-off fashion to improve its performance, and HPO has been shown to perform superior to manual tuning (Bergstra, Bardenet, et al., 2011; Snoek, Larochelle, and Adams, 2012; Melis, Dyer, and Blunsom, 2018). Advanced methods for HPO take properties of the model fitting process into account and improve the efficiency of such HPO procedures even further. Additionally, if one conducts HPO repeatedly for the same machine learning model but different datasets, one can use knowledge gained in earlier optimization runs.

2 Sometimes also autonomous machine learning (Escalante, 2021) or automatic machine learning (Hoffman, Shahriari, and Freitas, 2014).

AutoML systems, such as Auto-sklearn, which we will introduce in Part iii, are general-purpose machine learning algorithms themselves. They tune themselves for maximal performance on new datasets by automatically choosing between different machine learning models, tuning their hyperparameters, applying necessary preprocessing steps and following tricks of the trade. With such systems available, the machine learning expert can focus on other parts of the machine learning problem, for example, data preparation or better problem formulation. Also, they enable novice users or domain experts to use advanced machine learning in the first place. Under the hood AutoML systems combine an HPO tool with a machine learning library. As the developers of the AutoML system, we can flexibly design the individual components and optimize them jointly for maximal performance.

Our goal in this thesis is to develop AutoML methods to democratize machine learning. We will lay out the concrete research questions we that we wish to answer in the next section.

1.2 GOALS OF THIS THESIS

A lot of work exists on creating new methods for machine learning, as can be seen by the number of submissions to machine learning journals and conferences. The number of submissions to the *Journal of Machine Learning Research* (JMLR, the top journal for machine learning papers) doubled in the last three years, as shown in Figure 1.1. Similarly, for the *Neural Information Processing Systems* conference (NeurIPS), the number of submissions increased by 40% per year for 2017-2020, culminating in over 9400 submissions in 2020.

Yet, in order to develop new methods, to decide which of the many methods to use for a new problem and how to set their hyperparameters, we need to enable the users and developers to make these decisions in an efficient manner, i.e., by using efficient methods in a fully automated fashion. The definition of efficiency can vary, but it is often measured in terms of a computing budget.

The main question behind this thesis is *Can we make machine learning more accessible by efficient and robust Automated Machine Learning?* We have researched this question under the following three aspects:

1. *Hyperparameter Optimization (HPO)*: Can we improve the applicability of HPO to costly problems?

2. *Automated Machine Learning*: Can we improve the efficiency of Combined Algorithm Selection and Hyperparameter Optimization (CASH) for supervised learning?

3. *Benchmarking and the OpenML platform*: Can we facilitate AutoML research and development by providing better data to the researcher?

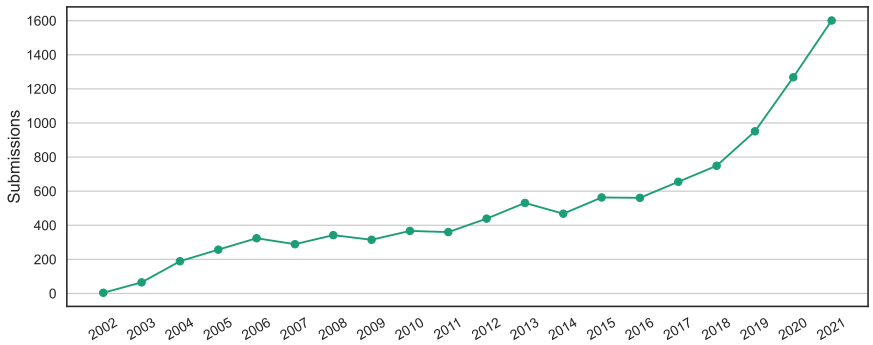


Figure 1.1: Number of yearly submissions to the Journal of Machine Learning Research (Image from (Pedregosa, Maharaj, et al., 2022)).

We have identified these three aspects during our research and found several open questions and challenges. In the remainder of this subsection, we will dive further into these three directions and give further details on what we aim to achieve with this thesis.

Hyperparameter Optimization

Hyperparameter Optimization (HPO) is the process of searching for the optimal hyperparameters for a machine learning model or pipeline for a dataset at hand. HPO methods are iterative algorithms that train and evaluate the model they optimize in each iteration.

Optimizing the hyperparameters of a machine learning model or pipeline can be expensive for different reasons:

1. because the model itself is huge and takes a long time to train,
2. because the data used to train the model is large, or
3. because one is constantly re-optimizing the model for new data.

Since datasets increase in size, models grow larger, and the number of datasets themselves explodes, researchers have started to extend HPO to tackle these challenges.

To make HPO feasible for the first two dimensions, HPO researchers make use of so-called *learning curves* of machine learning models. The term learning curve originally described how growing dataset sizes impact model performance; which can also be modeled (Provost, Jensen, and Oates, 1999). Recently, they also describe how the performance of iterative algorithm evolves, and researchers can often guesstimate the performance of such algorithms from the observation of just a few iterations (Domhan, Springenberg, and Hutter, 2015). Both types of learning curves are important to design so-called multi-fidelity methods. These multi-fidelity methods use cheap approximations, so-called low-fidelity versions, of the expensive, actual target. By this, they save computational budget with the goal of querying the expensive,

high-fidelity model (i.e., training on all data or training for the highest amount of iterations possible) as few times as possible. However, there is no literature overview of multi-fidelity methods available to make an informed decision on how to improve on them and which one to use in one's work. Therefore, we consider it to be an important step to summarize available work on standard and multi-fidelity HPO and provide such a survey in Chapter 3 of this thesis.

In contrast, for the third dimension, where one constantly re-optimizes the same model for new data, it is customary to re-use knowledge from earlier optimization runs to speed up the optimization. So far, all methods in the literature suffer from at least one issue that prevents easy applicability: they do not scale to a dozen tasks or more, they have a too high modeling overhead, they have hyperparameters themselves, require other task-specific settings, or provide no guarantees that re-using knowledge does not hurt the performance. Developing such a method would advance the practical applicability of HPO in large-scale settings; and we do so in Chapter 4.

Automated Machine Learning

As mentioned above, AutoML systems are off-the-shelf general-purpose machine learning algorithms and they draw their power from combining an HPO tool with a machine learning library. We can take this information into account when designing such an application (Hoos, 2012) to get the most out of the methods used within the AutoML system.

Previous work on AutoML falls into two realms: the field of meta-learning, which learns which machine learning algorithm to apply in which situation in a one-shot manner (see Brazdil, Gama, and Henery (1994) and Gama and Brazdil (1995) for two early examples and Brazdil, Giraud-Carrier, et al. (2008), Smith-Miles (2008) and Vanschoren (2019) for an overview) and the field of HPO which aims to find the correct algorithm and its hyperparameters by joint optimization (Escalante, Montes, and Sucar, 2009; Thornton et al., 2013; Q. Sun, Pfahringer, and Mayo, 2013; Komer, Bergstra, and Eliasmith, 2014; Bürger and Pauli, 2015).[3]

The first is limited (i.e., does not include hyperparameter tuning), while the second is slow for either large datasets or when many models need to be searched over. Furthermore, previous algorithms would return only a single model while it is known that ensembles generally improve predictive performance (Brazdil, Giraud-Carrier, et al., 2008,

3 There have been intermediate works, for example, meta-learning that predicts which algorithm would be best after HPO (Q. Sun and Pfahringer, 2013; Reif, Shafait, Goldstein, et al., 2014) and meta-learning to find the initial starting points for HPO (Reif, Shafait, and Dengel, 2012; Gomes et al., 2012; Feurer, Springenberg, and Hutter, 2015). Furthermore, the third realm of Neural Architecture Search (NAS) emerged in parallel to this thesis and we discuss it in Section 9.2.4.

Section 5). Lastly, the methods neither took direct optimization for generalization to new data into account nor did they study how to measure performance on the available data to maximize generalization performance within a given time horizon. We consider overcoming these limitations crucial for the success of AutoML systems and introduce our approaches for doing so, Auto-sklearn and Auto-sklearn 2.0, in Chapters 5 and 6 of this thesis.

Benchmarking and the OpenML platform

While we conducted research on AutoML methodology for this thesis, we encountered several roadblocks that we had to remove to work efficiently on the two previous subproblems:

1. Accessibility of a large amount of realistic data to develop meta-learning systems, and

2. well-defined and standardized sets of datasets for benchmarking.

When we started this thesis, there already existed several repositories dedicated to machine learning data: UCI machine learning repository (Dua and Graff, 2019), LibSVM (Chang and C.-J. Lin, 2011), MLData (Vanschoren, Braun, and Ong, 2014), skdata (Bergstra, Pinto, and Cox, 2015) and OpenML (Vanschoren, Rijn, et al., 2014). UCI did not allow easy interaction, i.e., there is neither a universal, machine-readable description of the datasets available nor an API for listing all datasets. The datasets on LibSVM are not very diverse, and the overall number is limited. MLData also allowed for arbitrary data formats. skdata had only a very limited amount of data available, with most datasets being for computer vision. OpenML, on the other hand, contained many diverse datasets, searchable via an accessible REST API, and all datasets were available in a unified, machine-readable format. Out of those, only OpenML allowed for easy and programmatic access to a large number of consistently formatted datasets. OpenML allowed access via a Java, R, and REST API, but not yet via a Python API.

However, researchers require easy access to large amounts of data, easily searchable, well-described. To provide researchers easy access to curated data within the Python programming language and enhance their capabilities in researching meta-learning and large-scale AutoML, we introduce the OpenML-Python package in Chapter 7 of this thesis.

In addition to the aforementioned repositories, several papers provided the datasets as supplementary material. To the date of when we started this thesis, the paper introducing Auto-WEKA (Thornton et al., 2013; Kotthoff et al., 2019) had the most extensive evaluation of an AutoML system, using a total of 21 datasets, which are available for download. In the field of meta-learning works were using up to 466 datasets (Q. Sun and Pfahringer, 2013). However, they did not come with a dataset-level description and were only available

in private communication. Furthermore, these datasets were heavily preprocessed (and consisted only of binary classification datasets, including binarized multiclass classification datasets and regression problems that were converted to binary classification problems) and downsampled to contain at most 5000 data points.[4] The paper asking the question *Do we need hundreds of classifiers to solve real world classification problems?* (Fernández-Delgado et al., 2014) used a total of 121 datasets and the authors provide access to them as a downloadable archive of files. They are, however, also preprocessed and only contain numerical and normalized features. Unfortunately, such preprocessing simplifies the machine learning problems and reduces the generality of the datasets. For example, using datasets without categorical features would not allow building a general-purpose AutoML system that can handle categorical features, or using only of up to 5000 samples would only allow us to build AutoML systems for small-scale data. While these are excellent data sources, they are, however, not sufficient for constructing large-scale AutoML systems.

Nonetheless, we would like to have an integration of defined benchmarks and a dataset repository. With access to OpenML, researchers now have hundreds of datasets in a machine-readable format and searchable via an API at their fingertips. As a next step, we integrate the definition of collections of benchmarks into OpenML and make them easy to access and easy to develop further with the concept of *OpenML benchmarking suites*, which we introduce in Chapter 8.

1.3 HOW TO READ THIS THESIS

This thesis is a *thesis by publication*, which combines several works published in leading venues that follow a common research question into a single thesis. We structure it into five parts. In this part, Part i, we have so far discussed the problem setup and the general research question. Additionally, we have introduced the three angles from which we tackle this research question. We will next describe the contributions of this thesis. We will follow up on this with the individual works in Parts ii (HPO), iii (AutoML) and iv (Benchmarking and the OpenML platform). Each of these parts consists of two chapters, and each chapter consists of one publication. We close this thesis in Part v by summarizing and discussing our contributions and by providing an extended discussion of the impact of our works to put each publication into the context of the latest research.

As mentioned above, this is a *thesis by publication*. Therefore, we wrote papers with different states of knowledge and audiences in mind and used different notations throughout the thesis. Moreover, we do

4 While summarizing this thesis, we learned that Macià and Bernadó-Mansilla (2014) had collected a total of 166 classification datasets in a common format. However, they were also available only upon request.

not order the papers chronologically, but rather by topic. However, the different chapters can be read independently and do not require knowledge of other chapters of this thesis. We suggest the interested reader proceed directly to the chapter of interest. Each chapter is self-contained and contains relevant pointers to related literature. However, for a reader who is new to this field, we suggest starting with Chapter 3, which is also the first chapter of the book *Automated Machine Learning: Methods, Systems, Challenges* (Hutter, Kotthoff, and Vanschoren, 2019), and then continue with Chapters 2 and 3 of said book (Vanschoren, 2019; Elsken, Metzen, and Hutter, 2019b) before returning to this thesis.

2 CONTRIBUTIONS

In this chapter, we will discuss the contributions we made to address the main research question we posed in the previous chapter: *Can we make machine learning more accessible by efficient and robust Automated Machine Learning?* We tackled this question by approaching it from three different directions: Hyperparameter Optimization (HPO), Automated Machine Learning (AutoML), and Benchmarking and the OpenML platform. Pushing the research frontier in each of these directions corresponds to the three goals we aim to achieve. During our quest to solve these goals, we published a total of six papers, which we consolidated in this thesis.

In this chapter, first, we discuss the contributions of our publications towards achieving our goals. Second, we provide the references for these. Third, we also provide references for publications the thesis author contributed to, which are related but not part of this thesis.

2.1 CONTRIBUTIONS

In this section, we will discuss the individual contributions part by part. Concretely, we will give a brief introduction for each part, followed by the two contributions we made to the goal of the respective part.

2.1.1 *Part ii: Hyperparameter optimization*

Hyperparameter Optimization (HPO) is a cornerstone of AutoML. It powers AutoML systems such as *particle swarm model selection* (Escalante, Montes, and Sucar, 2009, PSMS), *Auto-WEKA* (Thornton et al., 2013; Kotthoff et al., 2019), *hyperopt-sklearn* (Komer, Bergstra, and Eliasmith, 2019), *Auto-sklearn* (Chapter 5), *MOSAIC* (Rakotoarison, Schoenauer, and Sebag, 2019) and many more. Deep neural networks are another prime application for HPO and one that sparked interest in AutoML (Bergstra, Bardenet, et al., 2011; Snoek, Larochelle, and Adams, 2012).

HPO can lead to substantial performance gains and solely make the difference between a good model and state-of-the-art (Snoek, Larochelle, and Adams, 2012; Melis, Dyer, and Blunsom, 2018; Y. Chen et al., 2018; Bello et al., 2021) However, standard HPO can be too costly to conduct. As an example, training AlexNet took five to six days on recent hardware of that time (Krizhevsky, Sutskever, and Hinton, 2012). Such training times make regular, sequential HPO infeasible.

On a similar note, the cost for training the recent GPT3 model is estimated to be above $4.000.000 (C. Li, 2020).

We present two publications on making HPO more efficient to allow scaling it to larger problems: the first is a review of modern HPO methods that features cost-efficient multi-fidelity methods and upcoming trends for improving HPO's efficiency, and the second is a new HPO method that re-uses knowledge from earlier optimization tasks to speed up optimization on a new task.

Survey on Hyperparameter Optimization

Matthias Feurer and Frank Hutter (2019). "Hyperparameter Optimization." In: *AutoML: Methods, Systems, Challenges*. Ed. by Frank Hutter, Lars Kotthoff, and Joaquin Vanschoren. Springer. Chap. 1, pp. 3–38.

Our first contribution to the field of efficient Hyperparameter Optimization (HPO) is a survey on HPO. We assess modern HPO approaches and focus on multi-fidelity optimization methods that started gaining traction.

Concretely, we define the problem statement of HPO and discuss why it is a particularly hard problem. We also discuss the related approaches of ensembling and marginalization of hyperparameters. On the optimization side, we start by discussing popular techniques for black-box HPO: grid search and random search (Bergstra and Bengio, 2012), population-based algorithms such as CMA-ES (Hansen, 2006), and Bayesian optimization (Brochu, Cora, and de Freitas, 2010; Shahriari et al., 2016; Frazier, 2018; Garnett, 2022). Next, we thoroughly discuss multi-fidelity HPO. We identify three overarching techniques: learning curves, bandits, and adaptive methods. Furthermore, we discuss the applications of HPO to AutoML and give an overview of the first dedicated papers introducing methods for HPO. We conclude this chapter by posing open research questions that we believe are important to the field of HPO.

Practical Transfer Learning For Bayesian Optimization

Matthias Feurer, Benjamin Letham, Frank Hutter, and Eytan Bakshy (2022). "Practical Transfer Learning for Bayesian Optimization." In: *arXiv:1802.02219v4 [cs.LG]*.

An alternative angle to reduce the costs of HPO can be found in applications where problems are repeatedly solved, for example, for different datasets. As our second contribution to the field of HPO, we introduce new methods that re-use knowledge from earlier optimization tasks to speed up optimization on a new task. Because they transfer knowledge, they go by the name of *transfer HPO* methods.

We introduce properties of a desirable transfer HPO algorithm, such as scalability to many tasks, fast updates of the surrogate model, no dependence on task-specific hyperparameter settings or numerical task descriptors, and no performance degradation because of the transfer learning setup. These properties are inspired by the fact that the algorithm was developed in a setting where there was no direct access to the machine learning model and dataset, for example, because the machine learning model is maintained by a different department or there are issues adapting the code.

We base our algorithms, which follow these design principles, on a linear combination of Gaussian processes. Each Gaussian process models the performance of the target model on one previous optimization task. Additionally, we use one Gaussian process to model the performance for the current task. Our methods also feature theoretical guarantees about falling back to single task Bayesian optimization and being only a constant factor slower than single task Bayesian optimization. We explain how our algorithms generalize two existing algorithms and relate to an older meta-learning technique (Leite, Brazdil, and Vanschoren, 2012), which gives us a better understanding of the previous work.

We then introduce the largest benchmark to date, including six transfer HPO tasks and nine baselines, and conduct an extensive evaluation, including a thorough ablation study. Our evaluation shows that the proposed methods improve over the previous state-of-the-art. This thesis chapter also supplements the HPO survey as it contains a thorough overview of the related work on transfer HPO.

2.1.2 *Part iii: Automated Machine Learning*

In this next part of this thesis, we study whether we can increase the accessibility of machine learning by making AutoML systems more efficient and robust. Concretely, we study how we can combine HPO and a machine learning library into an efficient AutoML system by exploiting the fact that we can develop and configure both. We focus on supervised classification, and we aim to provide users with an automated problem solver if they formulate their problem in the required format. In the course of this work, we developed the popular AutoML system Auto-sklearn, which we validated by winning the 1st and 2nd ChaLearn AutoML competition (Guyon et al., 2019) and by using the OpenML AutoML benchmark (Gijsbers et al., 2019).

Robust and Efficient Automated Machine Learning

Matthias Feurer, Aaron Klein, Katharina Eggensperger, Jost Tobias Springenberg, Manuel Blum, and Frank Hutter (2015). "Efficient and Robust Automated Machine Learning." In: *Advances in Neu-*

ral Information Processing Systems. Ed. by Corinna Cortes, Neil D Lawrence, Daniel D Lee, Masashi Sugiyama, and Roman Garnett. Vol. 28, pp. 2962–2970.

Auto-WEKA (Thornton et al., 2013; Kotthoff et al., 2019) has demonstrated that one can build general-purpose AutoML systems by extending the HPO problem to also treating the choice of the machine learning models as a hyperparameter. This extended formulation is known as the Combined Algorithm Selection and Hyperparameter Optimization (CASH) problem. However, Auto-WEKA had to start each optimization from scratch, which is a significant drawback if evaluating a single machine learning pipeline takes minutes or hours. This drawback drastically reduced Auto-WEKA's applicability in settings that require fast execution, such as the setting of the ChaLearn AutoML competition, in which the contestants had to return a prediction within a 20-minute time limit.

To overcome these limitations we designed and implemented the robust, efficient, and extensible AutoML tool Auto-sklearn. Auto-sklearn constructs linear machine learning pipelines from 15 classifiers, 16 preprocessing methods, and four data cleaning methods from scikit-learn (Pedregosa, Varoquaux, et al., 2011). To make Auto-sklearn as resource-efficient as possible, we extend previous work of warmstarting Bayesian optimization for small search spaces to the high dimensionality of this new search space with more than 100 hyperparameters. Specifically, we use a k-nearest neighbor approach previously developed for smaller search spaces (Feurer, Springenberg, and Hutter, 2015).

Previous AutoML systems would only return the single best model, which is wasteful given that the AutoML systems train dozens or hundreds of models in the course of finding the best model. To overcome this limitation, we introduce post-hoc ensembling, which creates an ensemble from all models evaluated while running the AutoML procedure.[1]

We demonstrate that our tool outperformed the two existing AutoML systems of that time, Auto-WEKA and hyperopt-sklearn, on the benchmark defined by Thornton et al. (2013) and Kotthoff et al. (2019). We crawled OpenML for a total of 140 benchmark datasets from diverse domains, such as text classification, digit and letter recognition, gene sequence and RNA classification, advertisement, particle classification for telescope data, and cancer detection in tissue samples. We then did a leave-one-dataset out evaluation, e.g., running Auto-sklearn on one of the datasets and using the remaining 139 as meta-datasets. This evaluation was the broadest one to that date. In a detailed analysis on a representative subset of datasets, we showed that different com-

1 As we will discuss in Chapters 3 and 6, we later on learned that such ideas had been around before (Escalante, Montes, and Sucar, 2010; Bürger and Pauli, 2015). However, these methods did not construct ensembles in a principled fashion.

ponents are required for different datasets. Finally, we demonstrated Auto-sklearn's robustness in the ChaLearn AutoML competition, winning the first round of the ChaLearn AutoML competition (Guyon et al., 2019).[2]

Auto-sklearn 2.0: Hands-free AutoML via Meta-Learning

Matthias Feurer, Katharina Eggensperger, Stefan Falkner, Marius Lindauer, and Frank Hutter (2022). "Auto-Sklearn 2.0: Hands-free AutoML via Meta-Learning." In: *Journal of Machine Learning Research* 23.261. Ed. by Marc Schoenauer, pp. 1–61.

The AutoML system Auto-sklearn, which we introduced in the previous section, features impressive out-of-the-box performance. In this work, first, we manually tune and improve the Auto-sklearn system for the 2nd ChaLearn AutoML competition and then, second, introduce a new meta-learning layer on top of the existing AutoML scheme.

We tackled the 2nd ChaLearn AutoML competition by introducing the concepts of *budget allocation strategies* and *model selection strategies* and by developing an improved meta-learning strategy.

Budget allocation strategies define how the available budget is allocated between configurations, and we proposed using *successive halving* (Jamieson and Talwalkar, 2016). Besides, model selection strategies define how to estimate the generalization error, and we stuck to a standard holdout procedure. However, we also used a manually designed fallback if datasets had less than 1000 samples in the competition: assign all configurations the same budget and use cross-validation. The meta-learning strategy takes inspiration from portfolio construction for algorithm selection (Xu, Hoos, and Leyton-Brown, 2010; Xu, Hutter, et al., 2011) and is based on the greedy algorithm (Krause and Golovin, 2014). We relate the greedy portfolio construction to the *sensor placement problem* for minimizing detection time (Krause, Leskovec, et al., 2008), which allows us to apply existing results for the sensor set problem, such as submodularity and monotonicity of the optimization problem to our setting. These results provide us with a worst-case guarantee for the performance of the portfolio. We validated this manually improved AutoML system, which we dubbed *PoSH-Auto-sklearn* based on their two main components **PO**rtfolios

2 Up to the date the paper was submitted to the NeurIPS conference, we had only won the first out of five rounds of the ChaLearn AutoML competition. We continued to participate in the ChaLearn AutoML competition using Auto-sklearn and won two more first places and one second place. Moreover, using Auto-sklearn, we also won three first places and one third place in the *Tweakathon* track in which participants could tackle the problem locally without any time or hardware restrictions. In particular, for the two final phases, we placed first in both the Tweakathon and the AutoML track. We refer to Guyon et al. (2019) for details on the competition and leaderboards.

and **S**uccessive **H**alving, by winning the 2nd ChaLearn AutoML competition (Guyon et al., 2019). Furthermore, we experimentally studied the budget allocation and model selection strategies and found that the combination of successive halving and holdout was indeed the best option. Lastly, we demonstrate that the choice differs per dataset and time horizon, which is an important pre-condition for the remainder of this work.

When we manually manually configured the AutoML system for the 2nd ChaLearn AutoML competition we realized that the design of the AutoML system is an optimization problem itself, and that we should tackle it in a data-driven manner.

While the previous AutoML formalism has only targeted the performance on a single dataset, we explicitly formalize that AutoML systems should optimize generalization performance across a distribution of datasets, following the expected risk minimization principle (Vapnik, 1991). We then use this formalism to design a self-tuning AutoML system that can automatically select the appropriate *budget allocation strategy* and *model selection strategy* for the dataset at hand based on algorithm selection (Rice, 1976; Kerschke et al., 2019). Because the meta-learning now contains an offline stage, we can no longer perform leave-one-dataset out evaluation and rather use the OpenML AutoML benchmark (Gijsbers et al., 2019) for evaluations. To actually train the algorithm selector and also build the portfolios, we compiled a complementary set of 209 datasets for meta-learning. This is the first work that conducts hyperparameter optimization of an AutoML system, or hyper-hyperparameter optimization.

Again, we conduct a large-scale evaluation against popular competitor systems. We show that our new Auto-sklearn 2.0 outperforms the previous Auto-sklearn 1.0 and also TPOT (Olson and Moore, 2019), H2O (LeDell and Poirier, 2020) and Auto-WEKA (Thornton et al., 2013; Kotthoff et al., 2019). In a large-scale ablation study, we take a detailed look into the design decisions to give recommendations on how to improve such AutoML systems further.

2.1.3 *Part iv: Benchmarking and the OpenML Platform*

AutoML research is dataset-heavy. For our work on Auto-sklearn and Auto-sklearn 2.0 (see Part iii) we require machine learning datasets at two levels:

1. Evaluation: Because AutoML is a general-purpose tool, we require many diverse datasets, and,

2. Meta-learning: Because we aim to build a general-purpose tool, meta-learning also needs to provide solutions in the general-purpose case.

To access datasets, we opted for the collaborative science platform OpenML (Vanschoren, Rijn, et al., 2014). It allows researchers to share artifacts produced during machine learning research. OpenML is organized around four core concepts: datasets, tasks, flows and runs. A dataset is a tabular dataset, and a task describes how to conduct machine learning on that dataset. The task defines all aspects of the evaluation, and there can be multiple tasks for a single dataset. A flow describes an implementation of a machine learning algorithm, and a run is the outcome of executing a flow on a task. Furthermore, OpenML follows several *design patterns* for collaborative science (Nielsen, 2011; Vanschoren, Rijn, et al., 2014).

One such *design pattern* of OpenML is to "construct rich and structured information commons so that people can efficiently build on prior knowledge". Another design pattern is "Split[ting] up complex tasks into many small subtasks that can be attacked (nearly) individually". Following these two design patterns, OpenML provides datasets and tasks in a machine-readable fashion that researchers can exploit; and we made heavy use of this to design the AutoML systems we described in the previous section.

We now describe our contributions to the OpenML platform, the *OpenML-Python API*, and *OpenML benchmarking suites*, which organize datasets into reusable benchmarks. Even though our work on benchmarking and the OpenML platform forms the final methodological part of this thesis, we conducted this work in parallel to all other publications.

OpenML-Python: an extensible Python API for OpenML

Matthias Feurer, Jan N. van Rijn, Arlind Kadra, Pieter Gijsbers, Neeratyoy Mallik, Sahithya Ravi, Andreas Müller, Joaquin Vanschoren, and Frank Hutter (2021). "OpenML-Python: an extensible Python API for OpenML." In: *Journal of Machine Learning Research* 22.100. Ed. by Balazs Kegl, pp. 1–5.

Over the years, Python has become the de-facto standard programming language for machine learning and data science (Elliott, 2019). However, when we started working on Auto-sklearn, OpenML only offered a Java (Rijn, 2016) and an R (Casalicchio et al., 2017) API. With Python rising in popularity, this would have hindered networked science success factors "designed serendipity" and "dynamic division of labor" as researcher using the Python programming language would have had no access to OpenML. Also, this would have prevented our Python-based tools from interacting with OpenML. To overcome these limitations, we created the OpenML-Python API. It accesses the OpenML platform using its REST API. While our primary goal was having convenient access to datasets to construct meta-data for Auto-sklearn and Auto-sklearn 2.0, the API provides full access to

all concepts hosted on OpenML. To simplify its usage in machine learning experimentation, we implement a plugin mechanism to connect 3rd-party machine learning libraries directly to OpenML. We integrated scikit-learn (Pedregosa, Varoquaux, et al., 2011) as a prototypical plugin, and this integration allows running scikit-learn models so that the results can readily be shared on OpenML. We provide two examples of how to use the API. Then we also describe use cases of the package, such as using it to obtain meta-data for meta-learning (Chapter 5, Fusi, Sheth, and Elibol (2018)) and building hyperparameter surrogates (Eggensperger, Hutter, et al., 2015; Perrone, Jenatton, et al., 2018).

OpenML Benchmarking Suites

Bischl, Bernd, Giuseppe Casalicchio, *Matthias Feurer*, Pieter Gijsbers, Frank Hutter, Michel Lang, Rafael G. Mantovani, Jan N. van Rijn, and Joaquin Vanschoren (2021). "OpenML Benchmarking Suites." In: *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*. Ed. by J. Vanschoren and S. Yeung. Vol. 1.

Algorithm benchmarks are a pillar of machine learning research. To fulfill their duty of tracking progress, they need to be well-designed, challenging, easily accessible, and practical to use. Nonetheless, benchmarks in machine learning research were often restricted to a few datasets (Rahimi and Recht, 2007; T. Chen and Guestrin, 2016) or were crafted manually from multiple repositories in a one-off fashion (Thornton et al., 2013; Q. Sun, Pfahringer, and Mayo, 2013; Fernández-Delgado et al., 2014), making it hard to extend them. Therefore, researchers had to define their own set of tasks to use from OpenML, which was possible because of the aforementioned design patterns. Nevertheless, to create the datasets for Auto-sklearn and Auto-sklearn 2.0 (see Part iii), two 2 PhD students and one research student assistant spent countless hours on semi-automatically sorting through datasets to assemble dataset lists to use.

While OpenML contains the concepts of datasets and tasks, it did not provide mechanisms to organize these and provide benchmark suites, i.e., collections of OpenML tasks. In this paper, we advocate for the use of benchmarking suites and also provide a software tool to create and use the benchmark suites. To demonstrate benchmarking suites in action, we propose the *OpenML-CC18* benchmark suite for supervised classification. It is carefully curated and contains no subsampled or preprocessed datasets. Moreover, we review the OpenML AutoML benchmark, a popular benchmarking suite for comparing AutoML tools. We close the chapter by discussing the impact of benchmarking suites on machine learning research in general.

2.2 LIST OF PUBLICATIONS

We briefly restate the publications that we combine into this thesis. We discussed their contributions in the previous section and give the full papers in Parts ii, iii, and iv.

Matthias Feurer and Frank Hutter (2019). "Hyperparameter Optimization." In: *AutoML: Methods, Systems, Challenges*. Ed. by Frank Hutter, Lars Kotthoff, and Joaquin Vanschoren. Springer. Chap. 1, pp. 3–38.

Matthias Feurer, Benjamin Letham, Frank Hutter, and Eytan Bakshy (2022). "Practical Transfer Learning for Bayesian Optimization." In: *arXiv:1802.02219v4 [cs.LG]*.

Matthias Feurer, Aaron Klein, Katharina Eggensperger, Jost Tobias Springenberg, Manuel Blum, and Frank Hutter (2015). "Efficient and Robust Automated Machine Learning." In: *Advances in Neural Information Processing Systems*. Ed. by Corinna Cortes, Neil D Lawrence, Daniel D Lee, Masashi Sugiyama, and Roman Garnett. Vol. 28, pp. 2962–2970.

Matthias Feurer, Katharina Eggensperger, Stefan Falkner, Marius Lindauer, and Frank Hutter (2022). "Auto-Sklearn 2.0: Hands-free AutoML via Meta-Learning." In: *Journal of Machine Learning Research* 23.261. Ed. by Marc Schoenauer, pp. 1–61.

Matthias Feurer, Jan N. van Rijn, Arlind Kadra, Pieter Gijsbers, Neeratyoy Mallik, Sahithya Ravi, Andreas Müller, Joaquin Vanschoren, and Frank Hutter (2021). "OpenML-Python: an extensible Python API for OpenML." In: *Journal of Machine Learning Research* 22.100. Ed. by Balazs Kegl, pp. 1–5.

Bischl, Bernd, Giuseppe Casalicchio, *Matthias Feurer*, Pieter Gijsbers, Frank Hutter, Michel Lang, Rafael G. Mantovani, Jan N. van Rijn, and Joaquin Vanschoren (2021). "OpenML Benchmarking Suites." In: *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*. Ed. by J. Vanschoren and S. Yeung. Vol. 1.

For completeness, we list further publications by the thesis author that are not part of this thesis. We do not list intermediate workshop publications that are superseded by a follow-up publication.

Matthias Feurer, Aaron Klein, Katharina Eggensperger, Jost Tobias Springenberg, Manuel Blum, and Frank Hutter (2019). "Auto-sklearn: Efficient and Robust Automated Machine Learning." In: *AutoML: Methods, Systems, Challenges*. Ed. by Frank Hutter, Lars Kotthoff, and Joaquin Vanschoren. Springer. Chap. 6, pp. 113–134.

Mendoza, Hector, Aaron Klein, *Matthias Feurer*, Jost Tobias Springenberg, Matthias Urban, Michael Burkart, Maximilian Dippel, Marius Lindauer, and Frank Hutter (2019). "Towards Automatically-Tuned Deep Neural Networks." In: *AutoML: Methods, Systems, Challenges*. Ed. by Frank Hutter, Lars Kotthoff, and Joaquin Vanschoren. Springer. Chap. 7, pp. 135–149.

Lindauer, Marius, *Matthias Feurer*, Katharina Eggensperger, André Biedenkapp, and Frank Hutter (2019). "Towards Assessing the Impact of Bayesian Optimization's Own Hyperparameters." In: *IJCAI 2019 Data Science Meets Optimisation Workshop*.

Lindauer, Marius, Katharina Eggensperger, *Matthias Feurer*, André Biedenkapp, Joshua Marben, Philipp Müller, and Frank Hutter (2019). "BOAH: A Tool Suite for Multi-Fidelity Bayesian Optimization & Analysis of Hyperparameters." In: *arXiv:1908.06756 [cs.LG]*.

Awad, Noor, Gresa Shala, Difan Deng, Neeratyoy Mallik, *Matthias Feurer*, Katharina Eggensperger, André Biedenkapp, Diederick Vermetten, Hao Wang, Carola Doerr, Marius Lindauer, and Frank Hutter (2020). "Squirrel: A Switching Hyperparameter Optimizer Description of the entry by AutoML.org & IOHprofiler to the NeurIPS 2020 BBO challenge." In: *arXiv:2012.08180 [cs.LG]*.

Eggensperger, Katharina, Philipp Müller, Neeratyoy Mallik, *Feurer, Matthias*, René Sass, Aaron Klein, Noor Awad, Marius Lindauer, and Frank Hutter (2021). "HPOBench: A Collection of Reproducible Multi-Fidelity Benchmark Problems for HPO." In: *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*. Ed. by J. Vanschoren and S. Yeung. Vol. 1.

Lindauer, Marius, Katharina Eggensperger, *Feurer, Matthias*, André Biedenkapp, Difan Deng, Carolin Benjamins, Tim Ruhkopf, René Sass, and Frank Hutter (2022). "SMAC3: A Versatile Bayesian Optimization Package for Hyperparameter Optimization." In: *Journal of Machine Learning Research* 23.54, pp. 1–9.

Part II

PUBLICATIONS ON HYPERPARAMETER OPTIMIZATION

3 HYPERPARAMETER OPTIMIZATION

Matthias Feurer and Frank Hutter (2019). "Hyperparameter Optimization." In: *AutoML: Methods, Systems, Challenges*. Ed. by Frank Hutter, Lars Kotthoff, and Joaquin Vanschoren. Springer. Chap. 1, pp. 3–38.

PAPER SUMMARY. This book chapter carefully reviews the state-of-the-art in hyperparameter optimization. First, it introduces the traditional hyperparameter optimization problem and alternative problem setups. Second, it discusses the two dominant hyperparameter optimization paradigms, which are black-box hyperparameter optimization and multi-fidelity hyperparameter optimization. Third, it then discusses the application of these techniques to the research field of Automated Machine Learning before closing with a list of open problems and future research directions.

PROJECT IDEA. Frank Hutter proposed to write this book chapter. Matthias Feurer conducted the literature review for the survey and drafted the survey structure, which was then jointly developed by Matthias Feurer and Frank Hutter.

PAPER WRITING. Matthias Feurer wrote the first draft. Matthias Feurer and Frank Hutter revised the book chapter. The final version was, to a large extent, written by Matthias Feurer.

NOTE. This chapter is part of the book *Automated Machine Learning: Methods, Systems, Challenges* (Hutter, Kotthoff, and Vanschoren, 2019). Consequently, all references to other chapters in Chapter 3 reference chapters in the book, not in this thesis.

LICENSE. This chapter was published by Springer as part of the book *Automated Machine Learning: Methods, Systems, Challenges* (Hutter, Kotthoff, and Vanschoren, 2019) and is licensed under the terms of the Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/). No changes were made to the original publication.

Chapter 1
Hyperparameter Optimization

Matthias Feurer and Frank Hutter

Abstract Recent interest in complex and computationally expensive machine learning models with many hyperparameters, such as automated machine learning (AutoML) frameworks and deep neural networks, has resulted in a resurgence of research on hyperparameter optimization (HPO). In this chapter, we give an overview of the most prominent approaches for HPO. We first discuss blackbox function optimization methods based on model-free methods and Bayesian optimization. Since the high computational demand of many modern machine learning applications renders pure blackbox optimization extremely costly, we next focus on modern multi-fidelity methods that use (much) cheaper variants of the blackbox function to approximately assess the quality of hyperparameter settings. Lastly, we point to open problems and future research directions.

1.1 Introduction

Every machine learning system has hyperparameters, and the most basic task in automated machine learning (AutoML) is to automatically set these hyperparameters to optimize performance. Especially recent deep neural networks crucially depend on a wide range of hyperparameter choices about the neural network's architecture, regularization, and optimization. Automated hyperparameter optimization (HPO) has several important use cases; it can

- reduce the human effort necessary for applying machine learning. This is particularly important in the context of AutoML.

M. Feurer (✉)
Department of Computer Science, University of Freiburg, Freiburg, Baden-Württemberg, Germany
e-mail: feurerm@informatik.uni-freiburg.de

F. Hutter
Department of Computer Science, University of Freiburg, Freiburg, Germany

© The Author(s) 2019

F. Hutter et al. (eds.), *Automated Machine Learning*, The Springer Series on Challenges in Machine Learning, https://doi.org/10.1007/978-3-030-05318-5_1

- improve the performance of machine learning algorithms (by tailoring them to the problem at hand); this has led to new state-of-the-art performances for important machine learning benchmarks in several studies (e.g. [105, 140]).
- improve the reproducibility and fairness of scientific studies. Automated HPO is clearly more reproducible than manual search. It facilitates fair comparisons since different methods can only be compared fairly if they all receive the same level of tuning for the problem at hand [14, 133].

The problem of HPO has a long history, dating back to the 1990s (e.g., [77, 82, 107, 126]), and it was also established early that different hyperparameter configurations tend to work best for different datasets [82]. In contrast, it is a rather new insight that HPO can be used to adapt general-purpose pipelines to specific application domains [30]. Nowadays, it is also widely acknowledged that tuned hyperparameters improve over the default setting provided by common machine learning libraries [100, 116, 130, 149].

Because of the increased usage of machine learning in companies, HPO is also of substantial commercial interest and plays an ever larger role there, be it in company-internal tools [45], as part of machine learning cloud services [6, 89], or as a service by itself [137].

HPO faces several challenges which make it a hard problem in practice:

- Function evaluations can be extremely expensive for large models (e.g., in deep learning), complex machine learning pipelines, or large datesets.
- The configuration space is often complex (comprising a mix of continuous, categorical and conditional hyperparameters) and high-dimensional. Furthermore, it is not always clear which of an algorithm's hyperparameters need to be optimized, and in which ranges.
- We usually don't have access to a gradient of the loss function with respect to the hyperparameters. Furthermore, other properties of the target function often used in classical optimization do not typically apply, such as convexity and smoothness.
- One cannot directly optimize for generalization performance as training datasets are of limited size.

We refer the interested reader to other reviews of HPO for further discussions on this topic [64, 94].

This chapter is structured as follows. First, we define the HPO problem formally and discuss its variants (Sect. 1.2). Then, we discuss blackbox optimization algorithms for solving HPO (Sect. 1.3). Next, we focus on modern multi-fidelity methods that enable the use of HPO even for very expensive models, by exploiting approximate performance measures that are cheaper than full model evaluations (Sect. 1.4). We then provide an overview of the most important hyperparameter optimization systems and applications to AutoML (Sect. 1.5) and end the chapter with a discussion of open problems (Sect. 1.6).

1.2 Problem Statement

Let $\mathcal{A}$ denote a machine learning algorithm with N hyperparameters. We denote the domain of the n-th hyperparameter by Λ_n and the overall *hyperparameter configuration space* as $\mathbf{\Lambda} = \Lambda_1 \times \Lambda_2 \times \ldots \Lambda_N$. A vector of hyperparameters is denoted by $\boldsymbol{\lambda} \in \mathbf{\Lambda}$, and $\mathcal{A}$ with its hyperparameters instantiated to $\boldsymbol{\lambda}$ is denoted by $\mathcal{A}_{\boldsymbol{\lambda}}$.

The domain of a hyperparameter can be real-valued (e.g., learning rate), integer-valued (e.g., number of layers), binary (e.g., whether to use early stopping or not), or categorical (e.g., choice of optimizer). For integer and real-valued hyperparameters, the domains are mostly bounded for practical reasons, with only a few exceptions [12, 113, 136].

Furthermore, the configuration space can contain *conditionality*, i.e., a hyperparameter may only be relevant if another hyperparameter (or some combination of hyperparameters) takes on a certain value. Conditional spaces take the form of directed acyclic graphs. Such conditional spaces occur, e.g., in the automated tuning of machine learning pipelines, where the choice between different preprocessing and machine learning algorithms is modeled as a categorical hyperparameter, a problem known as *Full Model Selection* (FMS) or *Combined Algorithm Selection and Hyperparameter optimization problem* (CASH) [30, 34, 83, 149]. They also occur when optimizing the architecture of a neural network: e.g., the number of layers can be an integer hyperparameter and the per-layer hyperparameters of layer i are only active if the network depth is at least i [12, 14, 33].

Given a data set $\mathcal{D}$, our goal is to find

$$\boldsymbol{\lambda}^* = \underset{\boldsymbol{\lambda} \in \mathbf{\Lambda}}{\operatorname{argmin}}\, \mathbb{E}_{(D_{train}, D_{valid}) \sim \mathcal{D}} \mathbf{V}(\mathcal{L}, \mathcal{A}_{\boldsymbol{\lambda}}, D_{train}, D_{valid}), \tag{1.1}$$

where $\mathbf{V}(\mathcal{L}, \mathcal{A}_{\boldsymbol{\lambda}}, D_{train}, D_{valid})$ measures the loss of a model generated by algorithm $\mathcal{A}$ with hyperparameters $\boldsymbol{\lambda}$ on training data D_{train} and evaluated on validation data D_{valid}. In practice, we only have access to finite data $D \sim \mathcal{D}$ and thus need to approximate the expectation in Eq. 1.1.

Popular choices for the validation protocol $\mathbf{V}(\cdot, \cdot, \cdot, \cdot)$ are the holdout and cross-validation error for a user-given loss function (such as misclassification rate); see Bischl et al. [16] for an overview of validation protocols. Several strategies for reducing the evaluation time have been proposed: It is possible to only test machine learning algorithms on a subset of folds [149], only on a subset of data [78, 102, 147], or for a small amount of iterations; we will discuss some of these strategies in more detail in Sect. 1.4. Recent work on multi-task [147] and multi-source [121] optimization introduced further cheap, auxiliary tasks, which can be queried instead of Eq. 1.1. These can provide cheap information to help HPO, but do not necessarily train a machine learning model on the dataset of interest and therefore do not yield a usable model as a side product.

1.2.1 Alternatives to Optimization: Ensembling and Marginalization

Solving Eq. 1.1 with one of the techniques described in the rest of this chapter usually requires fitting the machine learning algorithm $\mathcal{A}$ with multiple hyperparameter vectors $\boldsymbol{\lambda}_t$. Instead of using the argmin-operator over these, it is possible to either construct an ensemble (which aims to minimize the loss for a given validation protocol) or to integrate out all the hyperparameters (if the model under consideration is a probabilistic model). We refer to Guyon et al. [50] and the references therein for a comparison of frequentist and Bayesian model selection.

Only choosing a single hyperparameter configuration can be wasteful when many good configurations have been identified by HPO, and combining them in an ensemble can improve performance [109]. This is particularly useful in AutoML systems with a large configuration space (e.g., in *FMS* or *CASH*), where good configurations can be very diverse, which increases the potential gains from ensembling [4, 19, 31, 34]. To further improve performance, *Automatic Frankensteining* [155] uses HPO to train a stacking model [156] on the outputs of the models found with HPO; the 2nd level models are then combined using a traditional ensembling strategy.

The methods discussed so far applied ensembling after the HPO procedure. While they improve performance in practice, the base models are not optimized for ensembling. It is, however, also possible to directly optimize for models which would maximally improve an existing ensemble [97].

Finally, when dealing with Bayesian models it is often possible to integrate out the hyperparameters of the machine learning algorithm, for example using *evidence maximization* [98], *Bayesian model averaging* [56], *slice sampling* [111] or *empirical Bayes* [103].

1.2.2 Optimizing for Multiple Objectives

In practical applications it is often necessary to trade off two or more objectives, such as the performance of a model and resource consumption [65] (see also Chap. 3) or multiple loss functions [57]. Potential solutions can be obtained in two ways.

First, if a limit on a secondary performance measure is known (such as the maximal memory consumption), the problem can be formulated as a constrained optimization problem. We will discuss constraint handling in Bayesian optimization in Sect. 1.3.2.4.

Second, and more generally, one can apply multi-objective optimization to search for the Pareto front, a set of configurations which are optimal tradeoffs between the objectives in the sense that, for each configuration on the Pareto front, there is no other configuration which performs better for at least one and at least as well for all other objectives. The user can then choose a configuration from the Pareto front. We refer the interested reader to further literature on this topic [53, 57, 65, 134].

1.3 Blackbox Hyperparameter Optimization

In general, every blackbox optimization method can be applied to HPO. Due to the non-convex nature of the problem, global optimization algorithms are usually preferred, but some locality in the optimization process is useful in order to make progress within the few function evaluations that are usually available. We first discuss model-free blackbox HPO methods and then describe blackbox Bayesian optimization methods.

1.3.1 Model-Free Blackbox Optimization Methods

Grid search is the most basic HPO method, also known as full factorial design [110]. The user specifies a finite set of values for each hyperparameter, and grid search evaluates the Cartesian product of these sets. This suffers from the curse of dimensionality since the required number of function evaluations grows exponentially with the dimensionality of the configuration space. An additional problem of grid search is that increasing the resolution of discretization substantially increases the required number of function evaluations.

A simple alternative to grid search is random search [13].[1] As the name suggests, random search samples configurations at random until a certain budget for the search is exhausted. This works better than grid search when some hyperparameters are much more important than others (a property that holds in many cases [13, 61]). Intuitively, when run with a fixed budget of B function evaluations, the number of different values grid search can afford to evaluate for each of the N hyperparameters is only $B^{1/N}$, whereas random search will explore B different values for each; see Fig. 1.1 for an illustration.

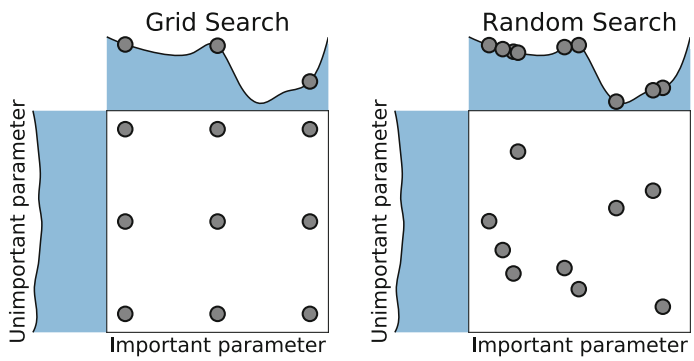


Fig. 1.1 Comparison of grid search and random search for minimizing a function with one important and one unimportant parameter. This figure is based on the illustration in Fig. 1 of Bergstra and Bengio [13]

[1] In some disciplines this is also known as *pure* random search [158].

Further advantages over grid search include easier parallelization (since workers do not need to communicate with each other and failing workers do not leave holes in the design) and flexible resource allocation (since one can add an arbitrary number of random points to a random search design to still yield a random search design; the equivalent does not hold for grid search).

Random search is a useful baseline because it makes no assumptions on the machine learning algorithm being optimized, and, given enough resources, will, in expectation, achieves performance arbitrarily close to the optimum. Interleaving random search with more complex optimization strategies therefore allows to guarantee a minimal rate of convergence and also adds exploration that can improve model-based search [3, 59]. Random search is also a useful method for initializing the search process, as it explores the entire configuration space and thus often finds settings with reasonable performance. However, it is no silver bullet and often takes far longer than guided search methods to identify one of the best performing hyperparameter configurations: e.g., when sampling without replacement from a configuration space with N Boolean hyperparameters with a good and a bad setting each and no interaction effects, it will require an expected 2^{N-1} function evaluations to find the optimum, whereas a guided search could find the optimum in $N + 1$ function evaluations as follows: starting from an arbitrary configuration, loop over the hyperparameters and change one at a time, keeping the resulting configuration if performance improves and reverting the change if it doesn't. Accordingly, the guided search methods we discuss in the following sections usually outperform random search [12, 14, 33, 90, 153].

Population-based methods, such as *genetic algorithms*, *evolutionary algorithms*, *evolutionary strategies*, and *particle swarm optimization* are optimization algorithms that maintain a population, i.e., a set of configurations, and improve this population by applying local perturbations (so-called mutations) and combinations of different members (so-called crossover) to obtain a new generation of better configurations. These methods are conceptually simple, can handle different data types, and are embarrassingly parallel [91] since a population of N members can be evaluated in parallel on N machines.

One of the best known population-based methods is the covariance matrix adaption evolutionary strategy (CMA-ES [51]); this simple evolutionary strategy samples configurations from a multivariate Gaussian whose mean and covariance are updated in each generation based on the success of the population's individuals. CMA-ES is one of the most competitive blackbox optimization algorithms, regularly dominating the *Black-Box Optimization Benchmarking* (BBOB) challenge [11].

For further details on population-based methods, we refer to [28, 138]; we discuss applications to hyperparameter optimization in Sect. 1.5, applications to neural architecture search in Chap. 3, and genetic programming for AutoML pipelines in Chap. 8.

1.3.2 Bayesian Optimization

Bayesian optimization is a state-of-the-art optimization framework for the global optimization of expensive blackbox functions, which recently gained traction in HPO by obtaining new state-of-the-art results in tuning deep neural networks for image classification [140, 141], speech recognition [22] and neural language modeling [105], and by demonstrating wide applicability to different problem settings. For an in-depth introduction to Bayesian optimization, we refer to the excellent tutorials by Shahriari et al. [135] and Brochu et al. [18].

In this section we first give a brief introduction to Bayesian optimization, present alternative surrogate models used in it, describe extensions to conditional and constrained configuration spaces, and then discuss several important applications to hyperparameter optimization.

Many recent advances in Bayesian optimization do not treat HPO as a blackbox any more, for example multi-fidelity HPO (see Sect. 1.4), Bayesian optimization with meta-learning (see Chap. 2), and Bayesian optimization taking the pipeline structure into account [159, 160]. Furthermore, many recent developments in Bayesian optimization do not directly target HPO, but can often be readily applied to HPO, such as new acquisition functions, new models and kernels, and new parallelization schemes.

1.3.2.1 Bayesian Optimization in a Nutshell

Bayesian optimization is an iterative algorithm with two key ingredients: a probabilistic surrogate model and an acquisition function to decide which point to evaluate next. In each iteration, the surrogate model is fitted to all observations of the target function made so far. Then the acquisition function, which uses the predictive distribution of the probabilistic model, determines the utility of different candidate points, trading off exploration and exploitation. Compared to evaluating the expensive blackbox function, the acquisition function is cheap to compute and can therefore be thoroughly optimized.

Although many acquisition functions exist, the *expected improvement* (EI) [72]:

$$\mathbb{E}[\mathbb{I}(\boldsymbol{\lambda})] = \mathbb{E}[\max(f_{min} - y, 0)] \tag{1.2}$$

is common choice since it can be computed in closed form if the model prediction y at configuration $\boldsymbol{\lambda}$ follows a normal distribution:

$$\mathbb{E}[\mathbb{I}(\boldsymbol{\lambda})] = (f_{min} - \mu(\boldsymbol{\lambda}))\, \Phi\left(\frac{f_{min} - \mu(\boldsymbol{\lambda})}{\sigma}\right) + \sigma\phi\left(\frac{f_{min} - \mu(\boldsymbol{\lambda})}{\sigma}\right), \tag{1.3}$$

where $\phi(\cdot)$ and $\Phi(\cdot)$ are the standard normal density and standard normal distribution function, and f_{min} is the best observed value so far.

Fig. 1.2 illustrates Bayesian optimization optimizing a toy function.

1.3.2.2 Surrogate Models

Traditionally, Bayesian optimization employs Gaussian processes [124] to model the target function because of their expressiveness, smooth and well-calibrated

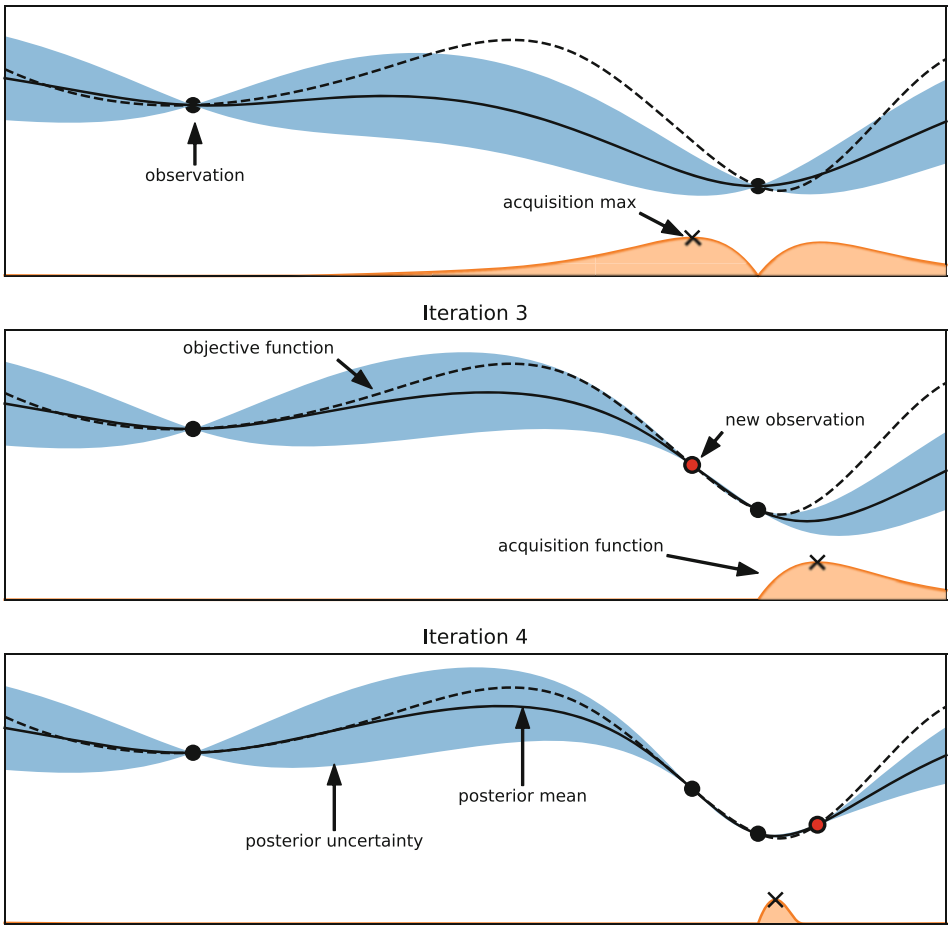


Fig. 1.2 Illustration of Bayesian optimization on a 1-d function. Our goal is to minimize the dashed line using a Gaussian process surrogate (predictions shown as black line, with blue tube representing the uncertainty) by maximizing the acquisition function represented by the lower orange curve. (Top) The acquisition value is low around observations, and the highest acquisition value is at a point where the predicted function value is low and the predictive uncertainty is relatively high. (Middle) While there is still a lot of variance to the left of the new observation, the predicted mean to the right is much lower and the next observation is conducted there. (Bottom) Although there is almost no uncertainty left around the location of the true maximum, the next evaluation is done there due to its expected improvement over the best point so far

uncertainty estimates and closed-form computability of the predictive distribution. A Gaussian process $\mathcal{G}\left(m(\boldsymbol{\lambda}), k(\boldsymbol{\lambda}, \boldsymbol{\lambda}')\right)$ is fully specified by a mean $m(\boldsymbol{\lambda})$ and a covariance function $k(\boldsymbol{\lambda}, \boldsymbol{\lambda}')$, although the mean function is usually assumed to be constant in Bayesian optimization. Mean and variance predictions $\mu(\cdot)$ and $\sigma^2(\cdot)$ for the noise-free case can be obtained by:

$$\mu(\boldsymbol{\lambda}) = \mathbf{k}_*^T \mathbf{K}^{-1} \mathbf{y}, \sigma^2(\boldsymbol{\lambda}) = k(\boldsymbol{\lambda}, \boldsymbol{\lambda}) - \mathbf{k}_*^T \mathbf{K}^{-1} \mathbf{k}_*, \tag{1.4}$$

where $\mathbf{k}_*$ denotes the vector of covariances between $\boldsymbol{\lambda}$ and all previous observations, $\mathbf{K}$ is the covariance matrix of all previously evaluated configurations and $\mathbf{y}$ are the observed function values. The quality of the Gaussian process depends solely on the covariance function. A common choice is the Mátern 5/2 kernel, with its hyperparameters integrated out by Markov Chain Monte Carlo [140].

One downside of standard Gaussian processes is that they scale cubically in the number of data points, limiting their applicability when one can afford many function evaluations (e.g., with many parallel workers, or when function evaluations are cheap due to the use of lower fidelities). This cubic scaling can be avoided by scalable Gaussian process approximations, such as sparse Gaussian processes. These approximate the full Gaussian process by using only a subset of the original dataset as *inducing points* to build the kernel matrix $\mathbf{K}$. While they allowed Bayesian optimization with GPs to scale to tens of thousands of datapoints for optimizing the parameters of a randomized SAT solver [62], there are criticism about the calibration of their uncertainty estimates and their applicability to standard HPO has not been tested [104, 154].

Another downside of Gaussian processes with standard kernels is their poor scalability to high dimensions. As a result, many extensions have been proposed to efficiently handle intrinsic properties of configuration spaces with large number of hyperparameters, such as the use of random embeddings [153], using Gaussian processes on partitions of the configuration space [154], cylindric kernels [114], and additive kernels [40, 75].

Since some other machine learning models are more scalable and flexible than Gaussian processes, there is also a large body of research on adapting these models to Bayesian optimization. Firstly, (deep) neural networks are a very flexible and scalable models. The simplest way to apply them to Bayesian optimization is as a feature extractor to preprocess inputs and then use the outputs of the final hidden layer as basis functions for Bayesian linear regression [141]. A more complex, fully Bayesian treatment of the network weights, is also possible by using a Bayesian neural network trained with stochastic gradient Hamiltonian Monte Carlo [144]. Neural networks tend to be faster than Gaussian processes for Bayesian optimization after ∼250 function evaluations, which also allows for large-scale parallelism. The flexibility of deep learning can also enable Bayesian optimization on more complex tasks. For example, a variational auto-encoder can be used to embed complex inputs (such as the structured configurations of the automated statistician, see Chap. 9) into a real-valued vector such that a regular Gaussian process can handle it [92]. For multi-source Bayesian optimization, a neural network architecture built on

factorization machines [125] can include information on previous tasks [131] and has also been extended to tackle the CASH problem [132].

Another alternative model for Bayesian optimization are random forests [59]. While GPs perform better than random forests on small, numerical configuration spaces [29], random forests natively handle larger, categorical and conditional configuration spaces where standard GPs do not work well [29, 70, 90]. Furthermore, the computational complexity of random forests scales far better to many data points: while the computational complexity of fitting and predicting variances with GPs for n data points scales as $O(n^3)$ and $O(n^2)$, respectively, for random forests, the scaling in n is only $O(n \log n)$ and $O(\log n)$, respectively. Due to these advantages, the SMAC framework for Bayesian optimization with random forests [59] enabled the prominent AutoML frameworks Auto-WEKA [149] and Auto-sklearn [34] (which are described in Chaps. 4 and 6).

Instead of modeling the probability $p(\mathbf{y}|\boldsymbol{\lambda})$ of observations $\mathbf{y}$ given the configurations $\boldsymbol{\lambda}$, the *Tree Parzen Estimator* (TPE [12, 14]) models density functions $p(\boldsymbol{\lambda}|\mathbf{y} < \alpha)$ and $p(\boldsymbol{\lambda}|\mathbf{y} \geq \alpha)$. Given a percentile α (usually set to 15%), the observations are divided in good observations and bad observations and simple 1-d Parzen windows are used to model the two distributions. The ratio $\frac{p(\boldsymbol{\lambda}|\mathbf{y}<\alpha)}{p(\boldsymbol{\lambda}|\mathbf{y}\geq\alpha)}$ is related to the expected improvement acquisition function and is used to propose new hyperparameter configurations. TPE uses a tree of Parzen estimators for conditional hyperparameters and demonstrated good performance on such structured HPO tasks [12, 14, 29, 33, 143, 149, 160], is conceptually simple, and parallelizes naturally [91]. It is also the workhorse behind the AutoML framework Hyperopt-sklearn [83] (which is described in Chap. 5).

Finally, we note that there are also surrogate-based approaches which do not follow the Bayesian optimization paradigm: Hord [67] uses a deterministic RBF surrogate, and Harmonica [52] uses a compressed sensing technique, both to tune the hyperparameters of deep neural networks.

1.3.2.3 Configuration Space Description

Bayesian optimization was originally designed to optimize box-constrained, real-valued functions. However, for many machine learning hyperparameters, such as the learning rate in neural networks or regularization in support vector machines, it is common to optimize the exponent of an exponential term to describe that changing it, e.g., from 0.001 to 0.01 is expected to have a similarly high impact as changing it from 0.1 to 1. A technique known as *input warping* [142] allows to automatically learn such transformations during the optimization process by replacing each input dimension with the two parameters of a Beta distribution and optimizing these.

One obvious limitation of the box-constraints is that the user needs to define these upfront. To avoid this, it is possible to dynamically expand the configuration space [113, 136]. Alternatively, the estimation-of-distribution-style algorithm TPE [12] is able to deal with infinite spaces on which a (typically Gaussian) prior is placed.

Integers and categorical hyperparameters require special treatment but can be integrated fairly easily into regular Bayesian optimization by small adaptations of the kernel and the optimization procedure (see Sect. 12.1.2 of [58], as well as [42]). Other models, such as factorization machines and random forests, can also naturally handle these data types.

Conditional hyperparameters are still an active area of research (see Chaps. 5 and 6 for depictions of conditional configuration spaces in recent AutoML systems). They can be handled natively by tree-based methods, such as random forests [59] and tree Parzen estimators (TPE) [12], but due to the numerous advantages of Gaussian processes over other models, multiple kernels for structured configuration spaces have also been proposed [4, 12, 63, 70, 92, 96, 146].

1.3.2.4 Constrained Bayesian Optimization

In realistic scenarios it is often necessary to satisfy constraints, such as memory consumption [139, 149], training time [149], prediction time [41, 43], accuracy of a compressed model [41], energy usage [43] or simply to not fail during the training procedure [43].

Constraints can be *hidden* in that only a binary observation (success or failure) is available [88]. Typical examples in AutoML are memory and time constraints to allow training of the algorithms in a shared computing system, and to make sure that a single slow algorithm configuration does not use all the time available for HPO [34, 149] (see also Chaps. 4 and 6).

Constraints can also merely be *unknown*, meaning that we can observe and model an auxiliary constraint function, but only know about a constraint violation after evaluating the target function [46]. An example of this is the prediction time of a support vector machine, which can only be obtained by training it as it depends on the number of support vectors selected during training.

The simplest approach to model violated constraints is to define a penalty value (at least as bad as the worst possible observable loss value) and use it as the observation for failed runs [34, 45, 59, 149]. More advanced approaches model the probability of violating one or more constraints and actively search for configurations with low loss values that are unlikely to violate any of the given constraints [41, 43, 46, 88].

Bayesian optimization frameworks using information theoretic acquisition functions allow decoupling the evaluation of the target function and the constraints to dynamically choose which of them to evaluate next [43, 55]. This becomes advantageous when evaluating the function of interest and the constraints require vastly different amounts of time, such as evaluating a deep neural network's performance and memory consumption [43].

1.4 Multi-fidelity Optimization

Increasing dataset sizes and increasingly complex models are a major hurdle in HPO since they make blackbox performance evaluation more expensive. Training a single hyperparameter configuration on large datasets can nowadays easily exceed several hours and take up to several days [85].

A common technique to speed up manual tuning is therefore to probe an algorithm/hyperparameter configuration on a small subset of the data, by training it only for a few iterations, by running it on a subset of features, by only using one or a few of the cross-validation folds, or by using down-sampled images in computer vision. Multi-fidelity methods cast such manual heuristics into formal algorithms, using so-called low fidelity approximations of the actual loss function to minimize. These approximations introduce a tradeoff between optimization performance and runtime, but in practice, the obtained speedups often outweigh the approximation error.

First, we review methods which model an algorithm's learning curve during training and can stop the training procedure if adding further resources is predicted to not help. Second, we discuss simple selection methods which only choose one of a finite set of given algorithms/hyperparameter configurations. Third, we discuss multi-fidelity methods which can actively decide which fidelity will provide most information about finding the optimal hyperparameters. We also refer to Chap. 2 (which discusses how multi-fidelity methods can be used across datasets) and Chap. 3 (which describes low-fidelity approximations for neural architecture search).

1.4.1 Learning Curve-Based Prediction for Early Stopping

We start this section on multi-fidelity methods in HPO with methods that evaluate and model learning curves during HPO [82, 123] and then decide whether to add further resources or stop the training procedure for a given hyperparameter configuration. Examples of learning curves are the performance of the same configuration trained on increasing dataset subsets, or the performance of an iterative algorithm measured for each iteration (or every i-th iteration if the calculation of the performance is expensive).

Learning curve extrapolation is used in the context of *predictive termination* [26], where a learning curve model is used to extrapolate a partially observed learning curve for a configuration, and the training process is stopped if the configuration is predicted to not reach the performance of the best model trained so far in the optimization process. Each learning curve is modeled as a weighted combination of 11 parametric functions from various scientific areas. These functions' parameters and their weights are sampled via Markov chain Monte Carlo to minimize the loss of fitting the partially observed learning curve. This yields a predictive distribution,

which allows to stop training based on the probability of not beating the best known model. When combined with Bayesian optimization, the predictive termination criterion enabled lower error rates than off-the-shelve blackbox Bayesian optimization for optimizing neural networks. On average, the method sped up the optimization by a factor of two and was able to find a (then) state-of-the-art neural network for CIFAR-10 (without data augmentation) [26].

While the method above is limited by not sharing information across different hyperparameter configurations, this can be achieved by using the basis functions as the output layer of a Bayesian neural network [80]. The parameters and weights of the basis functions, and thus the full learning curve, can thereby be predicted for arbitrary hyperparameter configurations. Alternatively, it is possible to use previous learning curves as basis function extrapolators [21]. While the experimental results are inconclusive on whether the proposed method is superior to pre-specified parametric functions, not having to manually define them is a clear advantage.

Freeze-Thaw Bayesian optimization [148] is a full integration of learning curves into the modeling and selection process of Bayesian optimization. Instead of terminating a configuration, the machine learning models are trained iteratively for a few iterations and then *frozen*. Bayesian optimization can then decide to *thaw* one of the frozen models, which means to continue training it. Alternatively, the method can also decide to start a new configuration. Freeze-Thaw models the performance of a converged algorithm with a regular Gaussian process and introduces a special covariance function corresponding to exponentially decaying functions to model the learning curves with per-learning curve Gaussian processes.

1.4.2 Bandit-Based Algorithm Selection Methods

In this section, we describe methods that try to determine the best algorithm out of a given finite set of algorithms based on low-fidelity approximations of their performance; towards its end, we also discuss potential combinations with adaptive configuration strategies. We focus on variants of the bandit-based strategies *successive halving* and *Hyperband*, since these have shown strong performance, especially for optimizing deep learning algorithms. Strictly speaking, some of the methods which we will discuss in this subsection also model learning curves, but they provide no means of selecting new configurations based on these models.

First, however, we briefly describe the historical evolution of multi-fidelity algorithm selection methods. In 2000, Petrak [120] noted that simply testing various algorithms on a small subset of the data is a powerful and cheap mechanism to select an algorithm. Later approaches used iterative algorithm elimination schemes to drop hyperparameter configurations if they perform badly on subsets of the data [17], if they perform significantly worse than a group of top-performing configurations [86], if they perform worse than the best configuration by a user-specified factor [143], or if even an optimistic performance bound for an algorithm is worse than the best known algorithm [128]. Likewise, it is possible to drop

hyperparameter configurations if they perform badly on one or a few cross-validation folds [149]. Finally, Jamieson and Talwalkar [69] proposed to use the *successive halving* algorithm originally introduced by Karnin et al. [76] for HPO.

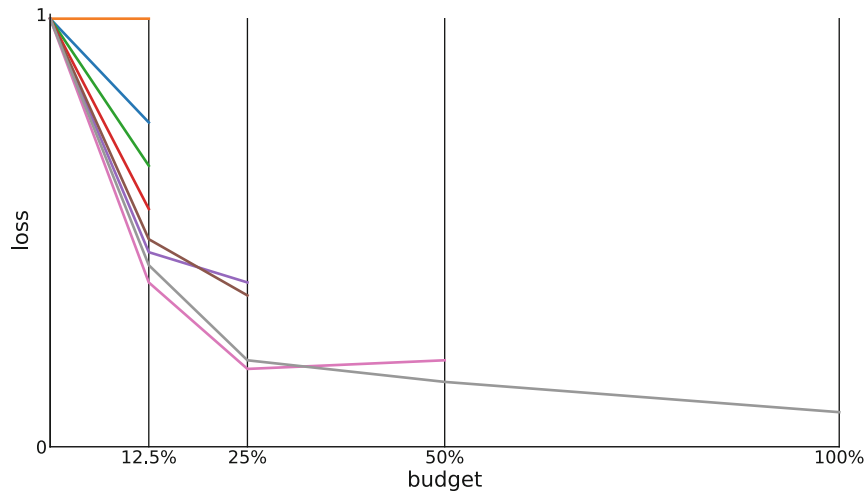


Fig. 1.3 Illustration of successive halving for eight algorithms/configurations. After evaluating all algorithms on $\frac{1}{8}$ of the total budget, half of them are dropped and the budget given to the remaining algorithms is doubled

Successive halving is an extremely simple, yet powerful, and therefore popular strategy for multi-fidelity algorithm selection: for a given initial budget, query all algorithms for that budget; then, remove the half that performed worst, double the budget [2] and successively repeat until only a single algorithm is left. This process is illustrated in Fig. 1.3. Jamieson and Talwalkar [69] benchmarked several common bandit methods and found that successive halving performs well both in terms of the number of required iterations and in the required computation time, that the algorithm theoretically outperforms a uniform budget allocation strategy if the algorithms converge favorably, and that it is preferable to many well-known bandit strategies from the literature, such as *UCB* and *EXP3*.

While successive halving is an efficient approach, it suffers from the budget-vs-number of configurations trade off. Given a total budget, the user has to decide beforehand whether to try many configurations and only assign a small budget to each, or to try only a few and assign them a larger budget. Assigning too small a budget can result in prematurely terminating good configurations, while assigning too large a budget can result in running poor configurations too long and thereby wasting resources.

[2] More precisely, drop the worst fraction $\frac{\eta-1}{\eta}$ of algorithms and multiply the budget for the remaining algorithms by η, where η is a hyperparameter. Its default value was changed from 2 to 3 with the introduction of HyperBand [90].

HyperBand [90] is a hedging strategy designed to combat this problem when selecting from randomly sampled configurations. It divides the total budget into several combinations of number of configurations vs. budget for each, to then call successive halving as a subroutine on each set of random configurations. Due to the hedging strategy which includes running some configurations only on the maximal budget, in the worst case, HyperBand takes at most a constant factor more time than vanilla random search on the maximal budget. In practice, due to its use of cheap low-fidelity evaluations, HyperBand has been shown to improve over vanilla random search and blackbox Bayesian optimization for data subsets, feature subsets and iterative algorithms, such as stochastic gradient descent for deep neural networks.

Despite HyperBand's success for deep neural networks it is very limiting to not adapt the configuration proposal strategy to the function evaluations. To overcome this limitation, the recent approach BOHB [33] combines Bayesian optimization and HyperBand to achieve the best of both worlds: strong anytime performance (quick improvements in the beginning by using low fidelities in HyperBand) and strong final performance (good performance in the long run by replacing HyperBand's random search by Bayesian optimization). BOHB also uses parallel resources effectively and deals with problem domains ranging from a few to many dozen hyperparameters. BOHB's Bayesian optimization component resembles TPE [12], but differs by using multidimensional kernel density estimators. It only fits a model on the highest fidelity for which at least $|\mathbf{\Lambda}| + 1$ evaluations have been performed (the number of hyperparameters, plus one). BOHB's first model is therefore fitted on the lowest fidelity, and over time models trained on higher fidelities take over, while still using the lower fidelities in successive halving. Empirically, BOHB was shown to outperform several state-of-the-art HPO methods for tuning support vector machines, neural networks and reinforcement learning algorithms, including most methods presented in this section [33]. Further approaches to combine HyperBand and Bayesian optimization have also been proposed [15, 151].

Multiple fidelity evaluations can also be combined with HPO in other ways. Instead of switching between lower fidelities and the highest fidelity, it is possible to perform HPO on a subset of the original data and extract the best-performing configurations in order to use them as an initial design for HPO on the full dataset [152]. To speed up solutions to the CASH problem, it is also possible to iteratively remove entire algorithms (and their hyperparameters) from the configuration space based on poor performance on small dataset subsets [159].

1.4.3 Adaptive Choices of Fidelities

All methods in the previous subsection follow a predefined schedule for the fidelities. Alternatively, one might want to actively choose which fidelities to evaluate given previous observations to prevent a misspecification of the schedule.

Multi-task Bayesian optimization [147] uses a multi-task Gaussian process to model the performance of related tasks and to automatically learn the tasks' correlation during the optimization process. This method can dynamically switch between cheaper, low-fidelity tasks and the expensive, high-fidelity target task based on a cost-aware information-theoretic acquisition function. In practice, the proposed method starts exploring the configuration space on the cheaper task and only switches to the more expensive configuration space in later parts of the optimization, approximately halving the time required for HPO. Multi-task Bayesian optimization can also be used to transfer information from previous optimization tasks, and we refer to Chap. 2 for further details.

Multi-task Bayesian optimization (and the methods presented in the previous subsection) requires an upfront specification of a set of fidelities. This can be suboptimal since these can be misspecified [74, 78] and because the number of fidelities that can be handled is low (usually five or less). Therefore, and in order to exploit the typically smooth dependence on the fidelity (such as, e.g., size of the data subset used), it often yields better results to treat the fidelity as continuous (and, e.g., choose a continuous percentage of the full data set to evaluate a configuration on), trading off the information gain and the time required for evaluation [78]. To exploit the domain knowledge that performance typically improves with more data, with diminishing returns, a special kernel can be constructed for the data subsets [78]. This generalization of multi-task Bayesian optimization improves performance and can achieve a 10–100 fold speedup compared to blackbox Bayesian optimization.

Instead of using an information-theoretic acquisition function, Bayesian optimization with the *Upper Confidence Bound* (UCB) acquisition function can also be extended to multiple fidelities [73, 74]. While the first such approach, MF-GP-UCB [73], required upfront fidelity definitions, the later BOCA algorithm [74] dropped that requirement. BOCA has also been applied to optimization with more than one continuous fidelity, and we expect HPO for more than one continuous fidelity to be of further interest in the future.

Generally speaking, methods that can adaptively choose their fidelity are very appealing and more powerful than the conceptually simpler bandit-based methods discussed in Sect. 1.4.2, but in practice we caution that strong models are required to make successful choices about the fidelities. When the models are not strong (since they do not have enough training data yet, or due to model mismatch), these methods may spend too much time evaluating higher fidelities, and the more robust fixed budget schedules discussed in Sect. 1.4.2 might yield better performance given a fixed time limit.

1.5 Applications to AutoML

In this section, we provide a historical overview of the most important hyperparameter optimization systems and applications to automated machine learning.

Grid search has been used for hyperparameter optimization since the 1990s [71, 107] and was already supported by early machine learning tools in 2002 [35]. The first adaptive optimization methods applied to HPO were greedy depth-first search [82] and pattern search [109], both improving over default hyperparameter configurations, and pattern search improving over grid search, too. Genetic algorithms were first applied to tuning the two hyperparameters C and γ of an RBF-SVM in 2004 [119] and resulted in improved classification performance in less time than grid search. In the same year, an evolutionary algorithm was used to learn a composition of three different kernels for an SVM, the kernel hyperparameters and to jointly select a feature subset; the learned combination of kernels was able to outperform every single optimized kernel. Similar in spirit, also in 2004, a genetic algorithm was used to select both the features used by and the hyperparameters of either an SVM or a neural network [129].

CMA-ES was first used for hyperparameter optimization in 2005 [38], in that case to optimize an SVM's hyperparameters C and γ, a kernel lengthscale l_i for each dimension of the input data, and a complete rotation and scaling matrix. Much more recently, CMA-ES has been demonstrated to be an excellent choice for parallel HPO, outperforming state-of-the-art Bayesian optimization tools when optimizing 19 hyperparameters of a deep neural network on 30 GPUs in parallel [91].

In 2009, Escalante et al. [30] extended the HPO problem to the *Full Model Selection* problem, which includes selecting a preprocessing algorithm, a feature selection algorithm, a classifier and all their hyperparameters. By being able to construct a machine learning pipeline from multiple off-the-shelf machine learning algorithms using HPO, the authors empirically found that they can apply their method to any data set as no domain knowledge is required, and demonstrated the applicability of their approach to a variety of domains [32, 49]. Their proposed method, particle swarm model selection (PSMS), uses a modified particle swarm optimizer to handle the conditional configuration space. To avoid overfitting, PSMS was extended with a custom ensembling strategy which combined the best solutions from multiple generations [31]. Since particle swarm optimization was originally designed to work on continuous configuration spaces, PSMS was later also extended to use a genetic algorithm to optimize the pipeline structure and only use particle swarm optimization to optimize the hyperparameters of each pipeline [145].

To the best of our knowledge, the first application of Bayesian optimization to HPO dates back to 2005, when Frohlich and Zell [39] used an online Gaussian process together with EI to optimize the hyperparameters of an SVM, achieving speedups of factor 10 (classification, 2 hyperparameters) and 100 (regression, 3 hyperparameters) over grid search. Tuned Data Mining [84] proposed to tune the hyperparameters of a full machine learning pipeline using Bayesian optimization; specifically, this used a single fixed pipeline and tuned the hyperparameters of the classifier as well as the per-class classification threshold and class weights.

In 2011, Bergstra et al. [12] were the first to apply Bayesian optimization to tune the hyperparameters of a deep neural network, outperforming both manual and random search. Furthermore, they demonstrated that TPE resulted in better

performance than a Gaussian process-based approach. TPE, as well as Bayesian optimization with random forests, were also successful for joint neural architecture search and hyperparameter optimization [14, 106].

Another important step in applying Bayesian optimization to HPO was made by Snoek et al. in the 2012 paper *Practical Bayesian Optimization of Machine Learning Algorithms* [140], which describes several tricks of the trade for Gaussian process-based HPO implemented in the Spearmint system and obtained a new state-of-the-art result for hyperparameter optimization of deep neural networks.

Independently of the Full Model Selection paradigm, Auto-WEKA [149] (see also Chap. 4) introduced the *Combined Algorithm Selection and Hyperparameter Optimization* (CASH) problem, in which the choice of a classification algorithm is modeled as a categorical variable, the algorithm hyperparameters are modeled as conditional hyperparameters, and the random-forest based Bayesian optimization system SMAC [59] is used for joint optimization in the resulting 786-dimensional configuration space.

In recent years, multi-fidelity methods have become very popular, especially in deep learning. Firstly, using low-fidelity approximations based on data subsets, feature subsets and short runs of iterative algorithms, Hyperband [90] was shown to outperform blackbox Bayesian optimization methods that did not take these lower fidelities into account. Finally, most recently, in the 2018 paper *BOHB: Robust and Efficient Hyperparameter Optimization at Scale*, Falkner et al. [33] introduced a robust, flexible, and parallelizable combination of Bayesian optimization and Hyperband that substantially outperformed both Hyperband and blackbox Bayesian optimization for a wide range of problems, including tuning support vector machines, various types of neural networks, and reinforcement learning algorithms.

At the time of writing, we make the following recommendations for which tools we would use in practical applications of HPO:

- If multiple fidelities are applicable (i.e., if it is possible to define substantially cheaper versions of the objective function of interest, such that the performance for these roughly correlates with the performance for the full objective function of interest), we recommend BOHB [33] as a robust, efficient, versatile, and parallelizable default hyperparameter optimization method.
- If multiple fidelities are not applicable:
  - If all hyperparameters are real-valued and one can only afford a few dozen function evaluations, we recommend the use of a Gaussian process-based Bayesian optimization tool, such as Spearmint [140].
  - For large and conditional configuration spaces we suggest either the random forest-based SMAC [59] or TPE [14], due to their proven strong performance on such tasks [29].
  - For purely real-valued spaces and relatively cheap objective functions, for which one can afford more than hundreds of evaluations, we recommend CMA-ES [51].

1.6 Open Problems and Future Research Directions

We conclude this chapter with a discussion of open problems, current research questions and potential further developments we expect to have an impact on HPO in the future. Notably, despite their relevance, we leave out discussions on hyperparameter importance and configuration space definition as these fall under the umbrella of meta-learning and can be found in Chap. 2.

1.6.1 Benchmarks and Comparability

Given the breadth of existing HPO methods, a natural question is what are the strengths and weaknesses of each of them. In order to allow for a fair comparison between different HPO approaches, the community needs to design and agree upon a common set of benchmarks that expands over time, as new HPO variants, such as multi-fidelity optimization, emerge. As a particular example for what this could look like we would like to mention the COCO platform (short for comparing continuous optimizers), which provides benchmark and analysis tools for continuous optimization and is used as a workbench for the yearly Black-Box Optimization Benchmarking (BBOB) challenge [11]. Efforts along similar lines in HPO have already yielded the hyperparameter optimization library (HPOlib [29]) and a benchmark collection specifically for Bayesian optimization methods [25]. However, neither of these has gained similar traction as the COCO platform.

Additionaly, the community needs clearly defined metrics, but currently different works use different metrics. One important dimension in which evaluations differ is whether they report performance on the validation set used for optimization or on a separate test set. The former helps to study the strength of the optimizer in isolation, without the noise that is added in the evaluation when going from validation to test set; on the other hand, some optimizers may lead to more overfitting than others, which can only be diagnosed by using the test set. Another important dimension in which evaluations differ is whether they report performance after a given number of function evaluations or after a given amount of time. The latter accounts for the difference in time between evaluating different hyperparameter configurations and includes optimization overheads, and therefore reflects what is required in practice; however, the former is more convenient and aids reproducibility by yielding the same results irrespective of the hardware used. To aid reproducibility, especially studies that use time should therefore release an implementation.

We note that it is important to compare against strong baselines when using new benchmarks, which is another reason why HPO methods should be published with an accompanying implementation. Unfortunately, there is no common software library as is, for example, available in deep learning research that implements all

the basic building blocks [2, 117]. As a simple, yet effective baseline that can be trivially included in empirical studies, Jamieson and Recht [68] suggest to compare against different parallelization levels of random search to demonstrate the speedups over regular random search. When comparing to other optimization techniques it is important to compare against a solid implementation, since, e.g., simpler versions of Bayesian optimization have been shown to yield inferior performance [79, 140, 142].

1.6.2 Gradient-Based Optimization

In some cases (e.g., least-squares support vector machines and neural networks) it is possible to obtain the gradient of the model selection criterion with respect to some of the model hyperparameters. Different to blackbox HPO, in this case each evaluation of the target function results in an entire hypergradient vector instead of a single float value, allowing for faster HPO.

Maclaurin et al. [99] described a procedure to compute the exact gradients of validation performance with respect to all continuous hyperparameters of a neural network by backpropagating through the entire training procedure of stochastic gradient descent with momentum (using a novel, memory-efficient algorithm). Being able to handle many hyperparameters efficiently through gradient-based methods allows for a new paradigm of hyperparametrizing the model to obtain flexibility over model classes, regularization, and training methods. Maclaurin et al. demonstrated the applicability of gradient-based HPO to many high-dimensional HPO problems, such as optimizing the learning rate of a neural network for each iteration and layer separately, optimizing the weight initialization scale hyperparameter for each layer in a neural network, optimizing the $l2$ penalty for each individual parameter in logistic regression, and learning completely new training datasets. As a small downside, backpropagating through the entire training procedure comes at the price of doubling the time complexity of the training procedure. The described method can also be generalized to work with other parameter update algorithms [36]. To overcome the necessity of backpropagating through the complete training procedure, later work allows to perform hyperparameter updates with respect to a separate validation set interleaved with the training process [5, 10, 36, 37, 93].

Recent examples of gradient-based optimization of simple model's hyperparameters [118] and of neural network structures (see Chap. 3) show promising results, outperforming state-of-the-art Bayesian optimization models. Despite being highly model-specific, the fact that gradient-based hyperparemeter optimization allows tuning several hundreds of hyperparameters could allow substantial improvements in HPO.

1.6.3 Scalability

Despite recent successes in multi-fidelity optimization, there are still machine learning problems which have not been directly tackled by HPO due to their scale, and which might require novel approaches. Here, scale can mean both the size of the configuration space and the expense of individual model evaluations. For example, there has not been any work on HPO for deep neural networks on the ImageNet challenge dataset [127] yet, mostly because of the high cost of training even a simple neural network on the dataset. It will be interesting to see whether methods going beyond the blackbox view from Sect. 1.3, such as the multi-fidelity methods described in Sect. 1.4, gradient-based methods, or meta-learning methods (described in Chap. 2) allow to tackle such problems. Chap. 3 describes first successes in learning neural network building blocks on smaller datasets and applying them to ImageNet, but the hyperparameters of the training procedure are still set manually.

Given the necessity of parallel computing, we are looking forward to new methods that fully exploit large-scale compute clusters. While there exists much work on parallel Bayesian optimization [12, 24, 33, 44, 54, 60, 135, 140], except for the neural networks described in Sect. 1.3.2.2 [141], so far no method has demonstrated scalability to hundreds of workers. Despite their popularity, and with a single exception of HPO applied to deep neural networks [91],[3] population-based approaches have not yet been shown to be applicable to hyperparameter optimization on datasets larger than a few thousand data points.

Overall, we expect that more sophisticated and specialized methods, leaving the blackbox view behind, will be needed to further scale hyperparameter to interesting problems.

1.6.4 Overfitting and Generalization

An open problem in HPO is overfitting. As noted in the problem statement (see Sect. 1.2), we usually only have a finite number of data points available for calculating the validation loss to be optimized and thereby do not necessarily optimize for generalization to unseen test datapoints. Similarly to overfitting a machine learning algorithm to training data, this problem is about overfitting the hyperparameters to the finite validation set; this was also demonstrated to happen experimentally [20, 81].

A simple strategy to reduce the amount of overfitting is to employ a different shuffling of the train and validation split for each function evaluation; this was shown to improve generalization performance for SVM tuning, both with a holdout and a cross-validation strategy [95]. The selection of the final configuration can

[3] See also Chap. 3 where population-based methods are applied to Neural Architecture Search problems.

be further robustified by not choosing it according to the lowest observed value, but according to the lowest predictive mean of the Gaussian process model used in Bayesian optimization [95].

Another possibility is to use a separate holdout set to assess configurations found by HPO to avoid bias towards the standard validation set [108, 159]. Different approximations of the generalization performance can lead to different test performances [108], and there have been reports that several resampling strategies can result in measurable performance differences for HPO of support vector machines [150].

A different approach to combat overfitting might be to find *stable optima* instead of *sharp optima* of the objective function [112]. The idea is that for stable optima, the function value around an optimum does not change for slight perturbations of the hyperparameters, whereas it *does* change for sharp optima. Stable optima lead to better generalization when applying the found hyperparameters to a new, unseen set of datapoints (i.e., the test set). An acquisition function built around this was shown to only slightly overfit for support vector machine HPO, while regular Bayesian optimization exhibited strong overfitting [112].

Further approaches to combat overfitting are the ensemble methods and Bayesian methods presented in Sect. 1.2.1. Given all these different techniques, there is no commonly agreed-upon technique for how to best avoid overfitting, though, and it remains up to the user to find out which strategy performs best on their particular HPO problem. We note that the best strategy might actually vary across HPO problems.

1.6.5 Arbitrary-Size Pipeline Construction

All HPO techniques we discussed so far assume a finite set of components for machine learning pipelines or a finite maximum number of layers in neural networks. For machine learning pipelines (see the AutoML systems covered in Part II of this book) it might be helpful to use more than one feature preprocessing algorithm and dynamically add them if necessary for a problem, enlarging the search space by a hyperparameter to select an appropriate preprocessing algorithm and its own hyperparameters. While a search space for standard blackbox optimization tools could easily include several extra such preprocessors (and their hyperparameters) as conditional hyperparameters, an unbounded number of these would be hard to support.

One approach for handling arbitrary-sized pipelines more natively is the tree-structured pipeline optimization toolkit (TPOT [115], see also Chap. 8), which uses genetic programming and describes possible pipelines by a grammar. TPOT uses multi-objective optimization to trade off pipeline complexity with performance to avoid generating unnecessarily complex pipelines.

A different pipeline creation paradigm is the usage of hierarchical planning; the recent ML-Plan [101, 108] uses hierarchical task networks and shows competitive performance compared to Auto-WEKA [149] and Auto-sklearn [34].

So far these approaches are not consistently outperforming AutoML systems with a fixed pipeline length, but larger pipelines may provide more improvement. Similarly, neural architecture search yields complex configuration spaces and we refer to Chap. 3 for a description of methods to tackle them.

Acknowledgements We would like to thank Luca Franceschi, Raghu Rajan, Stefan Falkner and Arlind Kadra for valuable feedback on the manuscript.

Bibliography

1. Proceedings of the International Conference on Learning Representations (ICLR'18) (2018), published online: iclr.cc
2. Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., Zheng, X.: TensorFlow: Large-scale machine learning on heterogeneous systems (2015), https://www.tensorflow.org/
3. Ahmed, M., Shahriari, B., Schmidt, M.: Do we need "harmless" Bayesian optimization and "first-order" Bayesian optimization. In: NeurIPS Workshop on Bayesian Optimization (BayesOpt'16) (2016)
4. Alaa, A., van der Schaar, M.: AutoPrognosis: Automated Clinical Prognostic Modeling via Bayesian Optimization with Structured Kernel Learning. In: Dy and Krause [27], pp. 139–148
5. Almeida, L.B., Langlois, T., Amaral, J.D., Plakhov, A.: Parameter Adaptation in Stochastic Optimization, p. 111–134. Cambridge University Press (1999)
6. Amazon: Automatic model tuning (2018), https://docs.aws.amazon.com/sagemaker/latest/dg/automatic-model-tuning.html
7. Bach, F., Blei, D. (eds.): Proceedings of the 32nd International Conference on Machine Learning (ICML'15), vol. 37. Omnipress (2015)
8. Balcan, M., Weinberger, K. (eds.): Proceedings of the 33rd International Conference on Machine Learning (ICML'17), vol. 48. Proceedings of Machine Learning Research (2016)
9. Bartlett, P., Pereira, F., Burges, C., Bottou, L., Weinberger, K. (eds.): Proceedings of the 26th International Conference on Advances in Neural Information Processing Systems (NeurIPS'12) (2012)
10. Baydin, A.G., Cornish, R., Rubio, D.M., Schmidt, M., Wood, F.: Online Learning Rate Adaption with Hypergradient Descent. In: Proceedings of the International Conference on Learning Representations (ICLR'18) [1], published online: iclr.cc
11. BBOBies: Black-box Optimization Benchmarking (BBOB) workshop series (2018), http://numbbo.github.io/workshops/index.html
12. Bergstra, J., Bardenet, R., Bengio, Y., Kégl, B.: Algorithms for hyper-parameter optimization. In: Shawe-Taylor, J., Zemel, R., Bartlett, P., Pereira, F., Weinberger, K. (eds.) Proceedings of the 25th International Conference on Advances in Neural Information Processing Systems (NeurIPS'11). pp. 2546–2554 (2011)
13. Bergstra, J., Bengio, Y.: Random search for hyper-parameter optimization. Journal of Machine Learning Research 13, 281–305 (2012)

14. Bergstra, J., Yamins, D., Cox, D.: Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. In: Dasgupta and McAllester [23], pp. 115–123
15. Bertrand, H., Ardon, R., Perrot, M., Bloch, I.: Hyperparameter optimization of deep neural networks: Combining hyperband with Bayesian model selection. In: Conférence sur l'Apprentissage Automatique (2017)
16. Bischl, B., Mersmann, O., Trautmann, H., Weihs, C.: Resampling methods for meta-model validation with recommendations for evolutionary computation. Evolutionary Computation 20(2), 249–275 (2012)
17. Van den Bosch, A.: Wrapped progressive sampling search for optimizing learning algorithm parameters. In: Proceedings of the sixteenth Belgian-Dutch Conference on Artificial Intelligence. pp. 219–226 (2004)
18. Brochu, E., Cora, V., de Freitas, N.: A tutorial on Bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning. arXiv:1012.2599v1 [cs.LG] (2010)
19. Bürger, F., Pauli, J.: A Holistic Classification Optimization Framework with Feature Selection, Preprocessing, Manifold Learning and Classifiers., pp. 52–68. Springer (2015)
20. Cawley, G., Talbot, N.: On Overfitting in Model Selection and Subsequent Selection Bias in Performance Evaluation. Journal of Machine Learning Research 11 (2010)
21. Chandrashekaran, A., Lane, I.: Speeding up Hyper-parameter Optimization by Extrapolation of Learning Curves using Previous Builds. In: Ceci, M., Hollmen, J., Todorovski, L., Vens, C., Džeroski, S. (eds.) Machine Learning and Knowledge Discovery in Databases (ECML/PKDD'17). Lecture Notes in Computer Science, vol. 10534. Springer (2017)
22. Dahl, G., Sainath, T., Hinton, G.: Improving deep neural networks for LVCSR using rectified linear units and dropout. In: Adams, M., Zhao, V. (eds.) International Conference on Acoustics, Speech and Signal Processing (ICASSP'13). pp. 8609–8613. IEEE Computer Society Press (2013)
23. Dasgupta, S., McAllester, D. (eds.): Proceedings of the 30th International Conference on Machine Learning (ICML'13). Omnipress (2014)
24. Desautels, T., Krause, A., Burdick, J.: Parallelizing exploration-exploitation tradeoffs in Gaussian process bandit optimization. Journal of Machine Learning Research 15, 4053–4103 (2014)
25. Dewancker, I., McCourt, M., Clark, S., Hayes, P., Johnson, A., Ke, G.: A stratified analysis of Bayesian optimization methods. arXiv:1603.09441v1 [cs.LG] (2016)
26. Domhan, T., Springenberg, J.T., Hutter, F.: Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves. In: Yang, Q., Wooldridge, M. (eds.) Proceedings of the 25th International Joint Conference on Artificial Intelligence (IJCAI'15). pp. 3460–3468 (2015)
27. Dy, J., Krause, A. (eds.): Proceedings of the 35th International Conference on Machine Learning (ICML'18), vol. 80. Proceedings of Machine Learning Research (2018)
28. Eberhart, R., Shi, Y.: Comparison between genetic algorithms and particle swarm optimization. In: Porto, V., Saravanan, N., Waagen, D., Eiben, A. (eds.) 7th International conference on evolutionary programming. pp. 611–616. Springer (1998)
29. Eggensperger, K., Feurer, M., Hutter, F., Bergstra, J., Snoek, J., Hoos, H., Leyton-Brown, K.: Towards an empirical foundation for assessing Bayesian optimization of hyperparameters. In: NeurIPS Workshop on Bayesian Optimization in Theory and Practice (BayesOpt'13) (2013)
30. Escalante, H., Montes, M., Sucar, E.: Particle Swarm Model Selection. Journal of Machine Learning Research 10, 405–440 (2009)
31. Escalante, H., Montes, M., Sucar, E.: Ensemble particle swarm model selection. In: Proceedings of the 2010 IEEE International Joint Conference on Neural Networks (IJCNN). pp. 1–8. IEEE Computer Society Press (2010)
32. Escalante, H., Montes, M., Villaseñor, L.: Particle swarm model selection for authorship verification. In: Bayro-Corrochano, E., Eklundh, J.O. (eds.) Progress in Pattern Recognition, Image Analysis, Computer Vision, and Applications. pp. 563–570 (2009)

33. Falkner, S., Klein, A., Hutter, F.: BOHB: Robust and Efficient Hyperparameter Optimization at Scale. In: Dy and Krause [27], pp. 1437–1446
34. Feurer, M., Klein, A., Eggensperger, K., Springenberg, J.T., Blum, M., Hutter, F.: Efficient and robust automated machine learning. In: Cortes, C., Lawrence, N., Lee, D., Sugiyama, M., Garnett, R. (eds.) Proceedings of the 29th International Conference on Advances in Neural Information Processing Systems (NeurIPS'15). pp. 2962–2970 (2015)
35. Fischer, S., Klinkenberg, R., Mierswa, I., Ritthoff, O.: Yale: Yet another learning environment – tutorial. Tech. rep., University of Dortmund (2002)
36. Franceschi, L., Donini, M., Frasconi, P., Pontil, M.: Forward and Reverse Gradient-Based Hyperparameter Optimization. In: Precup and Teh [122], pp. 1165–1173
37. Franceschi, L., Frasconi, P., Salzo, S., Grazzi, R., Pontil, M.: Bilevel Programming for Hyperparameter Optimization and Meta-Learning. In: Dy and Krause [27], pp. 1568–1577
38. Friedrichs, F., Igel, C.: Evolutionary tuning of multiple SVM parameters. Neurocomputing 64, 107–117 (2005)
39. Frohlich, H., Zell, A.: Efficient parameter selection for support vector machines in classification and regression via model-based global optimization. In: Prokhorov, D., Levine, D., Ham, F., Howell, W. (eds.) Proceedings of the 2005 IEEE International Joint Conference on Neural Networks (IJCNN). pp. 1431–1436. IEEE Computer Society Press (2005)
40. Gardner, J., Guo, C., Weinberger, K., Garnett, R., Grosse, R.: Discovering and Exploiting Additive Structure for Bayesian Optimization. In: Singh, A., Zhu, J. (eds.) Proceedings of the Seventeenth International Conference on Artificial Intelligence and Statistics (AISTATS). vol. 54, pp. 1311–1319. Proceedings of Machine Learning Research (2017)
41. Gardner, J., Kusner, M., Xu, Z., Weinberger, K., Cunningham, J.: Bayesian Optimization with Inequality Constraints. In: Xing and Jebara [157], pp. 937–945
42. Garrido-Merchán, E., Hernández-Lobato, D.: Dealing with integer-valued variables in Bayesian optimization with Gaussian processes. arXiv:1706.03673v2 [stats.ML] (2017)
43. Gelbart, M., Snoek, J., Adams, R.: Bayesian optimization with unknown constraints. In: Zhang, N., Tian, J. (eds.) Proceedings of the 30th conference on Uncertainty in Artificial Intelligence (UAI'14). AUAI Press (2014)
44. Ginsbourger, D., Le Riche, R., Carraro, L.: Kriging Is Well-Suited to Parallelize Optimization. In: Computational Intelligence in Expensive Optimization Problems, pp. 131–162. Springer (2010)
45. Golovin, D., Solnik, B., Moitra, S., Kochanski, G., Karro, J., Sculley, D.: Google Vizier: A service for black-box optimization. In: Matwin, S., Yu, S., Farooq, F. (eds.) Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD). pp. 1487–1495. ACM Press (2017)
46. Gramacy, R., Lee, H.: Optimization under unknown constraints. Bayesian Statistics 9(9), 229–246 (2011)
47. Gretton, A., Robert, C. (eds.): Proceedings of the Seventeenth International Conference on Artificial Intelligence and Statistics (AISTATS), vol. 51. Proceedings of Machine Learning Research (2016)
48. Guyon, I., von Luxburg, U., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., Garnett, R. (eds.): Proceedings of the 31st International Conference on Advances in Neural Information Processing Systems (NeurIPS'17) (2017)
49. Guyon, I., Saffari, A., Dror, G., Cawley, G.: Analysis of the IJCNN 2007 agnostic learning vs. prior knowledge challenge. Neural Networks 21(2), 544–550 (2008)
50. Guyon, I., Saffari, A., Dror, G., Cawley, G.: Model Selection: Beyond the Bayesian/Frequentist Divide. Journal of Machine Learning Research 11, 61–87 (2010)
51. Hansen, N.: The CMA evolution strategy: A tutorial. arXiv:1604.00772v1 [cs.LG] (2016)
52. Hazan, E., Klivans, A., Yuan, Y.: Hyperparameter optimization: A spectral approach. In: Proceedings of the International Conference on Learning Representations (ICLR'18) [1], published online: iclr.cc
53. Hernandez-Lobato, D., Hernandez-Lobato, J., Shah, A., Adams, R.: Predictive Entropy Search for Multi-objective Bayesian Optimization. In: Balcan and Weinberger [8], pp. 1492–1501

54. Hernández-Lobato, J., Requeima, J., Pyzer-Knapp, E., Aspuru-Guzik, A.: Parallel and distributed Thompson sampling for large-scale accelerated exploration of chemical space. In: Precup and Teh [122], pp. 1470–1479
55. Hernández-Lobato, J., Gelbart, M., Adams, R., Hoffman, M., Ghahramani, Z.: A general framework for constrained Bayesian optimization using information-based search. The Journal of Machine Learning Research 17(1), 5549–5601 (2016)
56. Hoeting, J., Madigan, D., Raftery, A., Volinsky, C.: Bayesian model averaging: a tutorial. Statistical science pp. 382–401 (1999)
57. Horn, D., Bischl, B.: Multi-objective parameter configuration of machine learning algorithms using model-based optimization. In: Likas, A. (ed.) 2016 IEEE Symposium Series on Computational Intelligence (SSCI). pp. 1–8. IEEE Computer Society Press (2016)
58. Hutter, F.: Automated Configuration of Algorithms for Solving Hard Computational Problems. Ph.D. thesis, University of British Columbia, Department of Computer Science, Vancouver, Canada (2009)
59. Hutter, F., Hoos, H., Leyton-Brown, K.: Sequential model-based optimization for general algorithm configuration. In: Coello, C. (ed.) Proceedings of the Fifth International Conference on Learning and Intelligent Optimization (LION'11). Lecture Notes in Computer Science, vol. 6683, pp. 507–523. Springer (2011)
60. Hutter, F., Hoos, H., Leyton-Brown, K.: Parallel algorithm configuration. In: Hamadi, Y., Schoenauer, M. (eds.) Proceedings of the Sixth International Conference on Learning and Intelligent Optimization (LION'12). Lecture Notes in Computer Science, vol. 7219, pp. 55–70. Springer (2012)
61. Hutter, F., Hoos, H., Leyton-Brown, K.: An efficient approach for assessing hyperparameter importance. In: Xing and Jebara [157], pp. 754–762
62. Hutter, F., Hoos, H., Leyton-Brown, K., Murphy, K.: Time-bounded sequential parameter optimization. In: Blum, C. (ed.) Proceedings of the Fourth International Conference on Learning and Intelligent Optimization (LION'10). Lecture Notes in Computer Science, vol. 6073, pp. 281–298. Springer (2010)
63. Hutter, F., Osborne, M.: A kernel for hierarchical parameter spaces. arXiv:1310.5738v1 [stats.ML] (2013)
64. Hutter, F., Lücke, J., Schmidt-Thieme, L.: Beyond Manual Tuning of Hyperparameters. KI - Künstliche Intelligenz 29(4), 329–337 (2015)
65. Igel, C.: Multi-objective Model Selection for Support Vector Machines. In: Coello, C., Aguirre, A., Zitzler, E. (eds.) Evolutionary Multi-Criterion Optimization. pp. 534–546. Springer (2005)
66. Ihler, A., Janzing, D. (eds.): Proceedings of the 32nd conference on Uncertainty in Artificial Intelligence (UAI'16). AUAI Press (2016)
67. Ilievski, I., Akhtar, T., Feng, J., Shoemaker, C.: Efficient Hyperparameter Optimization for Deep Learning Algorithms Using Deterministic RBF Surrogates. In: Sierra, C. (ed.) Proceedings of the 27th International Joint Conference on Artificial Intelligence (IJCAI'17) (2017)
68. Jamieson, K., Recht, B.: The news on auto-tuning (2016), http://www.argmin.net/2016/06/20/hypertuning/
69. Jamieson, K., Talwalkar, A.: Non-stochastic best arm identification and hyperparameter optimization. In: Gretton and Robert [47], pp. 240–248
70. Jenatton, R., Archambeau, C., González, J., Seeger, M.: Bayesian Optimization with Tree-structured Dependencies. In: Precup and Teh [122], pp. 1655–1664
71. John, G.: Cross-Validated C4.5: Using Error Estimation for Automatic Parameter Selection. Tech. Rep. STAN-CS-TN-94-12, Stanford University, Stanford University (1994)
72. Jones, D., Schonlau, M., Welch, W.: Efficient global optimization of expensive black box functions. Journal of Global Optimization 13, 455–492 (1998)
73. Kandasamy, K., Dasarathy, G., Oliva, J., Schneider, J., Póczos, B.: Gaussian Process Bandit Optimisation with Multi-fidelity Evaluations. In: Lee et al. [87], pp. 992–1000

74. Kandasamy, K., Dasarathy, G., Schneider, J., Póczos, B.: Multi-fidelity Bayesian Optimisation with Continuous Approximations. In: Precup and Teh [122], pp. 1799–1808
75. Kandasamy, K., Schneider, J., Póczos, B.: High Dimensional Bayesian Optimisation and Bandits via Additive Models. In: Bach and Blei [7], pp. 295–304
76. Karnin, Z., Koren, T., Somekh, O.: Almost optimal exploration in multi-armed bandits. In: Dasgupta and McAllester [23], pp. 1238–1246
77. King, R., Feng, C., Sutherland, A.: Statlog: comparison of classification algorithms on large real-world problems. Applied Artificial Intelligence an International Journal 9(3), 289–333 (1995)
78. Klein, A., Falkner, S., Bartels, S., Hennig, P., Hutter, F.: Fast bayesian hyperparameter optimization on large datasets. In: Electronic Journal of Statistics. vol. 11 (2017)
79. Klein, A., Falkner, S., Mansur, N., Hutter, F.: RoBO: A flexible and robust Bayesian optimization framework in Python. In: NeurIPS workshop on Bayesian Optimization (BayesOpt'17) (2017)
80. Klein, A., Falkner, S., Springenberg, J.T., Hutter, F.: Learning curve prediction with Bayesian neural networks. In: Proceedings of the International Conference on Learning Representations (ICLR'17) (2017), published online: iclr.cc
81. Koch, P., Konen, W., Flasch, O., Bartz-Beielstein, T.: Optimizing support vector machines for stormwater prediction. Tech. Rep. TR10-2-007, Technische Universität Dortmund (2010)
82. Kohavi, R., John, G.: Automatic Parameter Selection by Minimizing Estimated Error. In: Prieditis, A., Russell, S. (eds.) Proceedings of the Twelfth International Conference on Machine Learning, pp. 304–312. Morgan Kaufmann Publishers (1995)
83. Komer, B., Bergstra, J., Eliasmith, C.: Hyperopt-sklearn: Automatic hyperparameter configuration for scikit-learn. In: Hutter, F., Caruana, R., Bardenet, R., Bilenko, M., Guyon, I., Kégl, B., Larochelle, H. (eds.) ICML workshop on Automated Machine Learning (AutoML workshop 2014) (2014)
84. Konen, W., Koch, P., Flasch, O., Bartz-Beielstein, T., Friese, M., Naujoks, B.: Tuned data mining: a benchmark study on different tuners. In: Krasnogor, N. (ed.) Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation (GECCO'11). pp. 1995–2002. ACM (2011)
85. Krizhevsky, A., Sutskever, I., Hinton, G.: Imagenet classification with deep convolutional neural networks. In: Bartlett et al. [9], pp. 1097–1105
86. Krueger, T., Panknin, D., Braun, M.: Fast cross-validation via sequential testing. Journal of Machine Learning Research (2015)
87. Lee, D., Sugiyama, M., von Luxburg, U., Guyon, I., Garnett, R. (eds.): Proceedings of the 30th International Conference on Advances in Neural Information Processing Systems (NeurIPS'16) (2016)
88. Lee, H., Gramacy, R.: Optimization Subject to Hidden Constraints via Statistical Emulation. Pacific Journal of Optimization 7(3), 467–478 (2011)
89. Li, F.F., Li, J.: Cloud AutoML: Making AI accessible to every business (2018), https://www.blog.google/products/google-cloud/cloud-automl-making-ai-accessible-every-business/
90. Li, L., Jamieson, K., DeSalvo, G., Rostamizadeh, A., Talwalkar, A.: Hyperband: A novel bandit-based approach to hyperparameter optimization. Journal of Machine Learning Research 18(185), 1–52 (2018)
91. Loshchilov, I., Hutter, F.: CMA-ES for hyperparameter optimization of deep neural networks. In: International Conference on Learning Representations Workshop track (2016), published online: iclr.cc
92. Lu, X., Gonzalez, J., Dai, Z., Lawrence, N.: Structured Variationally Auto-encoded Optimization. In: Dy and Krause [27], pp. 3273–3281
93. Luketina, J., Berglund, M., Greff, K., Raiko, T.: Scalable Gradient-Based Tuning of Continuous Regularization Hyperparameters. In: Balcan and Weinberger [8], pp. 2952–2960
94. Luo, G.: A review of automatic selection methods for machine learning algorithms and hyperparameter values. Network Modeling Analysis in Health Informatics and Bioinformatics 5(1) (2016)

95. Lévesque, J.C.: Bayesian Hyperparameter Optimization: Overfitting, Ensembles and Conditional Spaces. Ph.D. thesis, Université Laval (2018)
96. Lévesque, J.C., Durand, A., Gagné, C., Sabourin, R.: Bayesian optimization for conditional hyperparameter spaces. In: Howell, B. (ed.) 2017 International Joint Conference on Neural Networks (IJCNN). pp. 286–293. IEEE (2017)
97. Lévesque, J.C., Gagné, C., Sabourin, R.: Bayesian Hyperparameter Optimization for Ensemble Learning. In: Ihler and Janzing [66], pp. 437–446
98. MacKay, D.: Hyperparameters: Optimize, or Integrate Out?, pp. 43–59. Springer (1996)
99. Maclaurin, D., Duvenaud, D., Adams, R.: Gradient-based Hyperparameter Optimization through Reversible Learning. In: Bach and Blei [7], pp. 2113–2122
100. Mantovani, R., Horvath, T., Cerri, R., Vanschoren, J., Carvalho, A.: Hyper-Parameter Tuning of a Decision Tree Induction Algorithm. In: 2016 5th Brazilian Conference on Intelligent Systems (BRACIS). pp. 37–42. IEEE Computer Society Press (2016)
101. Marcel Wever, F.M., Hüllermeier, E.: ML-Plan for unlimited-length machine learning pipelines. In: Garnett, R., Vanschoren, F.H.J., Brazdil, P., Caruana, R., Giraud-Carrier, C., Guyon, I., Kégl, B. (eds.) ICML workshop on Automated Machine Learning (AutoML workshop 2018) (2018)
102. Maron, O., Moore, A.: The racing algorithm: Model selection for lazy learners. Artificial Intelligence Review 11(1–5), 193–225 (1997)
103. McInerney, J.: An Empirical Bayes Approach to Optimizing Machine Learning Algorithms. In: Guyon et al. [48], pp. 2712–2721
104. McIntire, M., Ratner, D., Ermon, S.: Sparse Gaussian Processes for Bayesian Optimization. In: Ihler and Janzing [66]
105. Melis, G., Dyer, C., Blunsom, P.: On the state of the art of evaluation in neural language models. In: Proceedings of the International Conference on Learning Representations (ICLR'18) [1], published online: iclr.cc
106. Mendoza, H., Klein, A., Feurer, M., Springenberg, J., Hutter, F.: Towards automatically-tuned neural networks. In: ICML 2016 AutoML Workshop (2016)
107. Michie, D., Spiegelhalter, D., Taylor, C., Campbell, J. (eds.): Machine Learning, Neural and Statistical Classification. Ellis Horwood (1994)
108. Mohr, F., Wever, M., Höllermeier, E.: ML-Plan: Automated machine learning via hierarchical planning. Machine Learning 107(8–10), 1495–1515 (2018)
109. Momma, M., Bennett, K.: A Pattern Search Method for Model Selection of Support Vector Regression. In: Proceedings of the 2002 SIAM International Conference on Data Mining, pp. 261–274 (2002)
110. Montgomery, D.: Design and analysis of experiments. John Wiley & Sons, Inc, eighth edn. (2013)
111. Murray, I., Adams, R.: Slice sampling covariance hyperparameters of latent Gaussian models. In: Lafferty, J., Williams, C., Shawe-Taylor, J., Zemel, R., Culotta, A. (eds.) Proceedings of the 24th International Conference on Advances in Neural Information Processing Systems (NeurIPS'10). pp. 1732–1740 (2010)
112. Nguyen, T., Gupta, S., Rana, S., Venkatesh, S.: Stable Bayesian Optimization. In: Kim, J., Shim, K., Cao, L., Lee, J.G., Lin, X., Moon, Y.S. (eds.) Advances in Knowledge Discovery and Data Mining (PAKDD'17). Lecture Notes in Artificial Intelligence, vol. 10235, pp. 578–591 (2017)
113. Nguyen, V., Gupta, S., Rana, S., Li, C., Venkatesh, S.: Filtering Bayesian optimization approach in weakly specified search space. Knowledge and Information Systems (2018)
114. Oh, C., Gavves, E., Welling, M.: BOCK: Bayesian Optimization with Cylindrical Kernels. In: Dy and Krause [27], pp. 3865–3874
115. Olson, R., Bartley, N., Urbanowicz, R., Moore, J.: Evaluation of a Tree-based Pipeline Optimization Tool for Automating Data Science. In: Friedrich, T. (ed.) Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'16). pp. 485–492. ACM (2016)
116. Olson, R., La Cava, W., Mustahsan, Z., Varik, A., Moore, J.: Data-driven advice for applying machine learning to bioinformatics problems. In: Proceedings of the Pacific Symposium in Biocomputing 2018. pp. 192–203 (2018)

117. Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., Lerer, A.: Automatic differentiation in PyTorch. In: NeurIPS Autodiff Workshop (2017)
118. Pedregosa, F.: Hyperparameter optimization with approximate gradient. In: Balcan and Weinberger [8], pp. 737–746
119. Peng-Wei Chen, Jung-Ying Wang, Hahn-Ming Lee: Model selection of SVMs using GA approach. In: Proceedings of the 2004 IEEE International Joint Conference on Neural Networks (IJCNN). vol. 3, pp. 2035–2040. IEEE Computer Society Press (2004)
120. Petrak, J.: Fast subsampling performance estimates for classification algorithm selection. Technical Report TR-2000-07, Austrian Research Institute for Artificial Intelligence (2000)
121. Poloczek, M., Wang, J., Frazier, P.: Multi-Information Source Optimization. In: Guyon et al. [48], pp. 4288–4298
122. Precup, D., Teh, Y. (eds.): Proceedings of the 34th International Conference on Machine Learning (ICML'17), vol. 70. Proceedings of Machine Learning Research (2017)
123. Provost, F., Jensen, D., Oates, T.: Efficient progressive sampling. In: Fayyad, U., Chaudhuri, S., Madigan, D. (eds.) The 5th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'99). pp. 23–32. ACM Press (1999)
124. Rasmussen, C., Williams, C.: Gaussian Processes for Machine Learning. The MIT Press (2006)
125. Rendle, S.: Factorization machines. In: Webb, G., Liu, B., Zhang, C., Gunopulos, D., Wu, X. (eds.) Proceedings of the 10th IEEE International Conference on Data Mining (ICDM'06). pp. 995–1000. IEEE Computer Society Press (2010)
126. Ripley, B.D.: Statistical aspects of neural networks. Networks and chaos—statistical and probabilistic aspects 50, 40–123 (1993)
127. Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A., Fei-Fei, L.: Imagenet large scale visual recognition challenge. International Journal of Computer Vision 115(3), 211–252 (2015)
128. Sabharwal, A., Samulowitz, H., Tesauro, G.: Selecting Near-Optimal Learners via Incremental Data Allocation. In: Schuurmans, D., Wellman, M. (eds.) Proceedings of the Thirtieth National Conference on Artificial Intelligence (AAAI'16). AAAI Press (2016)
129. Samanta, B.: Gear fault detection using artificial neural networks and support vector machines with genetic algorithms. Mechanical Systems and Signal Processing 18(3), 625–644 (2004)
130. Sanders, S., Giraud-Carrier, C.: Informing the Use of Hyperparameter Optimization Through Metalearning. In: Gottumukkala, R., Ning, X., Dong, G., Raghavan, V., Aluru, S., Karypis, G., Miele, L., Wu, X. (eds.) 2017 IEEE International Conference on Big Data (Big Data). IEEE Computer Society Press (2017)
131. Schilling, N., Wistuba, M., Drumond, L., Schmidt-Thieme, L.: Hyperparameter optimization with factorized multilayer perceptrons. In: Appice, A., Rodrigues, P., Costa, V., Gama, J., Jorge, A., Soares, C. (eds.) Machine Learning and Knowledge Discovery in Databases (ECML/PKDD'15). Lecture Notes in Computer Science, vol. 9285, pp. 87–103. Springer (2015)
132. Schilling, N., Wistuba, M., Drumond, L., Schmidt-Thieme, L.: Joint Model Choice and Hyperparameter Optimization with Factorized Multilayer Perceptrons. In: 2015 IEEE 27th International Conference on Tools with Artificial Intelligence (ICTAI). pp. 72–79. IEEE Computer Society Press (2015)
133. Sculley, D., Snoek, J., Wiltschko, A., Rahimi, A.: Winner's curse? on pace, progress, and empirical rigor. In: International Conference on Learning Representations Workshop track (2018), published online: iclr.cc
134. Shah, A., Ghahramani, Z.: Pareto Frontier Learning with Expensive Correlated Objectives. In: Balcan and Weinberger [8], pp. 1919–1927
135. Shahriari, B., Swersky, K., Wang, Z., Adams, R., de Freitas, N.: Taking the human out of the loop: A review of Bayesian optimization. Proceedings of the IEEE 104(1), 148–175 (2016)
136. Shahriari, B., Bouchard-Cote, A., de Freitas, N.: Unbounded Bayesian optimization via regularization. In: Gretton and Robert [47], pp. 1168–1176

137. SIGOPT: Improve ML models 100x faster (2018), https://sigopt.com/
138. Simon, D.: Evolutionary optimization algorithms. John Wiley & Sons (2013)
139. Snoek, J.: Bayesian optimization and semiparametric models with applications to assistive technology. PhD Thesis, University of Toronto (2013)
140. Snoek, J., Larochelle, H., Adams, R.: Practical Bayesian optimization of machine learning algorithms. In: Bartlett et al. [9], pp. 2960–2968
141. Snoek, J., Rippel, O., Swersky, K., Kiros, R., Satish, N., Sundaram, N., Patwary, M., Prabhat, Adams, R.: Scalable Bayesian optimization using deep neural networks. In: Bach and Blei [7], pp. 2171–2180
142. Snoek, J., Swersky, K., Zemel, R., Adams, R.: Input warping for Bayesian optimization of non-stationary functions. In: Xing and Jebara [157], pp. 1674–1682
143. Sparks, E., Talwalkar, A., Haas, D., Franklin, M., Jordan, M., Kraska, T.: Automating model search for large scale machine learning. In: Balazinska, M. (ed.) Proceedings of the Sixth ACM Symposium on Cloud Computing - SoCC '15. pp. 368–380. ACM Press (2015)
144. Springenberg, J., Klein, A., Falkner, S., Hutter, F.: Bayesian optimization with robust Bayesian neural networks. In: Lee et al. [87]
145. Sun, Q., Pfahringer, B., Mayo, M.: Towards a Framework for Designing Full Model Selection and Optimization Systems. In: Multiple Classifier Systems, vol. 7872, pp. 259–270. Springer (2013)
146. Swersky, K., Duvenaud, D., Snoek, J., Hutter, F., Osborne, M.: Raiders of the lost architecture: Kernels for Bayesian optimization in conditional parameter spaces. In: NeurIPS Workshop on Bayesian Optimization in Theory and Practice (BayesOpt'14) (2014)
147. Swersky, K., Snoek, J., Adams, R.: Multi-task Bayesian optimization. In: Burges, C., Bottou, L., Welling, M., Ghahramani, Z., Weinberger, K. (eds.) Proceedings of the 27th International Conference on Advances in Neural Information Processing Systems (NeurIPS'13). pp. 2004–2012 (2013)
148. Swersky, K., Snoek, J., Adams, R.: Freeze-thaw Bayesian optimization arXiv:1406.3896v1 [stats.ML] (2014)
149. Thornton, C., Hutter, F., Hoos, H., Leyton-Brown, K.: Auto-WEKA: combined selection and hyperparameter optimization of classification algorithms. In: Dhillon, I., Koren, Y., Ghani, R., Senator, T., Bradley, P., Parekh, R., He, J., Grossman, R., Uthurusamy, R. (eds.) The 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'13). pp. 847–855. ACM Press (2013)
150. Wainer, J., Cawley, G.: Empirical Evaluation of Resampling Procedures for Optimising SVM Hyperparameters. Journal of Machine Learning Research 18, 1–35 (2017)
151. Wang, J., Xu, J., Wang, X.: Combination of hyperband and Bayesian optimization for hyperparameter optimization in deep learning. arXiv:1801.01596v1 [cs.CV] (2018)
152. Wang, L., Feng, M., Zhou, B., Xiang, B., Mahadevan, S.: Efficient Hyper-parameter Optimization for NLP Applications. In: Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing. pp. 2112–2117. Association for Computational Linguistics (2015)
153. Wang, Z., Hutter, F., Zoghi, M., Matheson, D., de Feitas, N.: Bayesian optimization in a billion dimensions via random embeddings. Journal of Artificial Intelligence Research 55, 361–387 (2016)
154. Wang, Z., Gehring, C., Kohli, P., Jegelka, S.: Batched Large-scale Bayesian Optimization in High-dimensional Spaces. In: Storkey, A., Perez-Cruz, F. (eds.) Proceedings of the 21st International Conference on Artificial Intelligence and Statistics (AISTATS). vol. 84. Proceedings of Machine Learning Research (2018)
155. Wistuba, M., Schilling, N., Schmidt-Thieme, L.: Automatic Frankensteining: Creating Complex Ensembles Autonomously. In: Proceedings of the 2017 SIAM International Conference on Data Mining (2017)
156. Wolpert, D.: Stacked generalization. Neural Networks 5(2), 241–259 (1992)
157. Xing, E., Jebara, T. (eds.): Proceedings of the 31th International Conference on Machine Learning, (ICML'14). Omnipress (2014)

158. Zabinsky, Z.: Pure Random Search and Pure Adaptive Search. In: Stochastic Adaptive Search for Global Optimization, pp. 25–54. Springer (2003)
159. Zeng, X., Luo, G.: Progressive sampling-based Bayesian optimization for efficient and automatic machine learning model selection. Health Information Science and Systems 5(1) (2017)
160. Zhang, Y., Bahadori, M.T., Su, H., Sun, J.: FLASH: Fast Bayesian Optimization for Data Analytic Pipelines. In: Krishnapuram, B., Shah, M., Smola, A., Aggarwal, C., Shen, D., Rastogi, R. (eds.) Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD). pp. 2065–2074. ACM Press (2016)

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.

4

PRACTICAL TRANSFER LEARNING FOR BAYESIAN OPTIMIZATION

Matthias Feurer, Benjamin Letham, Frank Hutter, and Eytan Bakshy (2022). "Practical Transfer Learning for Bayesian Optimization." In: *arXiv:1802.02219v4 [cs.LG]*.

PAPER SUMMARY. This paper proposes a novel ensemble of Gaussian processes for hyperparameter optimization in a transfer learning setting. It is hyperparameter-free and comes with theoretical guarantees that using transfer learning does not have a negative effect on hyperparameter optimization. Moreover, we show that it is superior to other transfer hyperparameter optimization algorithms in extensive experiments.

PROJECT IDEA. Matthias Feurer proposed the project and developed the RGPE methodology with feedback from Benjamin Letham. Matthias Feurer and Frank Hutter extended the project to contain the mixture of Gaussian processes ensemble and the relation of RGPE and the transfer acquisition function.

IMPLEMENTATION AND EXPERIMENTATION. The initial implementation was done by Matthias Feurer with help from Benjamin Letham while doing an internship at Facebook. The final code, including a re-implementation of the work done at Facebook, was conducted solely by Matthias Feurer. All final experiments were conducted by Matthias Feurer.

PAPER WRITING. Benjamin Letham wrote the first draft, which Matthias Feurer and Eytan Bakshy revised. Then, Matthias Feurer extended it with the related work sections, new methodology, and theoretical sections. Matthias Feurer, Benjamin Letham, and Frank Hutter edited the final version.

LICENSE. This chapter was published as an arXiv preprint and is licensed under the terms of the Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/). No changes were made to the original publication.

arXiv:1802.02219v4 [stat.ML] 24 Oct 2022

Practical Transfer Learning for Bayesian Optimization

Matthias Feurer FEURERM@CS.UNI-FREIBURG.DE
Department of Computer Science, Faculty of Engineering
University of Freiburg, 79110 Freiburg, Germany

Benjamin Letham BLETHAM@FB.COM
Meta
1 Hacker Way, Menlo Park, CA 94025, USA

Frank Hutter FH@CS.UNI-FREIBURG.DE
Department of Computer Science, Faculty of Engineering
University of Freiburg, 79110 Freiburg, Germany, and
Bosch Center for Artificial Intelligence, Germany

Eytan Bakshy EBAKSHY@FB.COM
Meta
1 Hacker Way, Menlo Park, CA 94025, USA

Abstract

When hyperparameter optimization of a machine learning algorithm is repeated for multiple datasets it is possible to transfer knowledge to an optimization run on a new dataset. We develop a new hyperparameter-free ensemble model for Bayesian optimization that is a generalization of two existing transfer learning extensions to Bayesian optimization and establish a worst-case bound compared to vanilla Bayesian optimization. Using a large collection of hyperparameter optimization benchmark problems, we demonstrate that our contributions substantially reduce optimization time compared to standard Gaussian process-based Bayesian optimization and improve over the current state-of-the-art for transfer hyperparameter optimization.

Keywords: Bayesian optimization, Hyperparameter optimization, Meta-learning, Transfer HPO, AutoML

1. Introduction

Bayesian optimization (BO) is a data-efficient blackbox optimization technique that is routinely used for hyperparameter optimization (HPO) of machine learning (ML) algorithms (Snoek et al., 2012; Feurer and Hutter, 2019). Given a small initial set of function evaluations, BO proceeds by fitting a surrogate model to those observations, typically a Gaussian process (GP), and then optimizing an acquisition function that balances exploration and exploitation in determining what point to evaluate next.

If the same ML algorithm with the same hyperparameters is routinely optimized for different datasets, for example in online machine learning services, we can lift the blackbox assumption and use data from prior BO runs as ancillary information. As the hyperparameter surfaces of an ML algorithm applied to different datasets are similar, we aim for a strategy to transfer this ancillary information across datasets to find good solutions faster than vanilla BO. To facilitate such transfer learning in BO we specify several desiderata to allow broad applicability to new applications:

©2022 Matthias Feurer, Benjamin Letham, Frank Hutter and Eytan Bakshy.

License: CC-BY 4.0, see https://creativecommons.org/licenses/by/4.0/.

1. Scalability to many tasks: having many potentially related past BO runs requires a method that is able to handle both many tasks and many observations in total across all tasks.

2. Fast update of the surrogate model: similarly, such a method needs to be able to easily update the surrogate model after adding a new observation on the current target task.

3. Easy and fast adaptability to different tasks: the method should neither depend on task-specific hyperparameter settings, nor depend on numerical task descriptors which have to be manually defined by experts.

4. No performance degradation: transfer learning must not substantially worsen performance compared to vanilla BO, even in the worst case of misleading prior information.

Even though several BO methods have been developed to borrow strength across runs (see Section 2.3 for a review), none of them satisfy all of the desiderata above. The main merit of this work is to introduce a set of methods that does so. Our specific contributions are:[1]

1. We develop a new ensemble model for BO, based on a linear combination of GPs and show that this is a generalization of two existing, scalable transfer learning extensions to BO and is similarly scalable (desiderata #1 and #2).

2. We introduce a novel, hyperparameter-free (desideratum #3) method to weight models in a transfer learning setting based on Agnostic Bayesian Learning of Ensembles, which allows us to bound the worst-case slowdown over BO to a multiplicative factor (desideratum #4).

3. We perform the largest experimental evaluation of transfer learning for hyperparameter optimization (transfer HPO) to date, using six hyperparameter optimization benchmarks with a total of 249 tasks and 9 competitor methods from the literature. Our contributions robustly improve over the current state of the art, without any need for hyperparameter tuning or benchmark-specific modifications.

4. We provide the implementation of our methods, the reimplementations of the competitor methods, and the collection of benchmarks as open-source code, to both simplify further research and reproducing our experiments.

The paper is structured as follows. First, in Section 2 we give background information on BO, introduce the problem setting and give a primer on related work. Then, in Section 3 we review related work on linear combinations of models for Bayesian optimization. Next, we present our methodology in Section 4. Then, we describe the baselines and benchmarks we release (Section 5) and provide experimental evidence that our proposed methodology is preferable (Section 6). Finally, we give a comprehensive overview of related work (Section 7) before concluding the paper (Section 8).

1. An earlier version of this work is available on arXiv and we compare against it in Appendix D.5.

2. Background and Problem Setup

In this section we provide the necessary background and introduce the problem setup for the related work on linear combinations of models in Section 3 and our own, new methodology in Section 4.

2.1 Bayesian Optimization

The goal of BO is to find a global minimum $\mathbf{x}_{opt}$ of a blackbox function in a bounded space $\mathcal{X}$ by iteratively querying the function at input configurations $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n$ and observing the corresponding outputs $y_1, y_2, \ldots, y_n$. In each iteration, we first fit a probabilistic model f on observations $\mathcal{D} = \{(\mathbf{x}_k, y_k)\}_{k=1}^n$ made so far. We then use an acquisition function $\alpha(\mathbf{x})$ to select a promising configuration to evaluate next, balancing exploration and exploitation.

In general, y_k may be a noisy estimate of the function value. We estimate the underlying function with GP regression, yielding a posterior $f(\mathbf{x}|\mathcal{D})$ with an analytical mean $\mu(\mathbf{x})$ and variance $\sigma^2(\mathbf{x})$ (Rasmussen and Williams, 2006, Eqs. 2.25 and 2.26):

$$\mu(\mathbf{x}_*) = \mathbf{k}_*^T(\mathbf{K} + \sigma_n^2 I)^{-1}\mathbf{y}, \qquad \sigma^2(\mathbf{x}_*) = k_\theta(\mathbf{x}_*, \mathbf{x}_*) - \mathbf{k}_*^T(\mathbf{K} + \sigma_n^2 I)^{-1}\mathbf{k}_*, \tag{1}$$

where $k_\theta(\cdot,\cdot)$ is a kernel function with hyperparameters θ, $\mathbf{k}_*$ is the vector of covariances between the new point $\mathbf{x}_*$ and all training points, and $\mathbf{K}$ is the kernel matrix of size $n \times n$ where each entry $\mathbf{K}_{i,j} = k_\theta(\mathbf{x}_i, \mathbf{x}_j)$.

The most commonly used acquisition function is the expected improvement (EI) (Jones et al., 1998), which can be computed in closed form and generally gives good results. Let $f(\mathbf{x}_{\text{best}})$ be the current best function value: $f(\mathbf{x}_{\text{best}}) = \min_{k\in(1,\ldots,n)} f(\mathbf{x}_k)$. The EI is

$$\alpha(\mathbf{x}|\mathcal{D}) = \mathbb{E}_{f(\mathbf{x}|\mathcal{D})}\left[\max(0, f(\mathbf{x}_{\text{best}}) - f(\mathbf{x}))\right] = \sigma(\mathbf{x})z\Phi(z) + \sigma(\mathbf{x})\phi(z), \tag{2}$$

with $z = \frac{f(\mathbf{x}_{\text{best}})-\mu(\mathbf{x})}{\sigma(\mathbf{x})}$, $\Phi(\cdot)$ being the CDF and $\phi(\cdot)$ being the PDF of the normal distribution. Thorough introductions to Bayesian optimization are given by Brochu et al. (2010), Shahriari et al. (2016), Frazier (2018) and Garnett (2022).

2.2 Problem Setup

We suppose that $t-1$ BO runs have been completed on previous tasks with the same search space $\mathcal{X}$. Let $\mathcal{D}^i = \left\{(\mathbf{x}_k^i, y_k^i)\right\}_{k=1}^{n_i}$ be the function evaluations made for past optimization run $i \in (1, \ldots, t-1)$. We fit a GP model to the observations of each past run i and refer to these models as *base models*. They have posterior $f^i(\mathbf{x}|\mathcal{D}^i)$, with mean and variance $\mu_i(\mathbf{x})$ and $\sigma_i^2(\mathbf{x})$, respectively. These remain fixed throughout the optimization, inasmuch as we do not obtain new observations for them. The current optimization problem we are trying to solve is task t. We fit a GP to observations from task t and call it the *target model* f^t. The target model is refit after each new function evaluation. We overload notation and define $\mathcal{D} = \{\mathcal{D}^1, \ldots, \mathcal{D}^t\}$. Our goal is to minimize the target function using the base models $f^1, \ldots, f^{t-1}$ and the target model f^t.

2.3 A Primer on Related Work

Borrowing strength from past runs is a form of meta-learning and transfer learning. Techniques for transfer learning for hyperparameter optimization (transfer HPO) can be catego-

rized into 9 approaches: 1) a single model that is trained on all tasks simultaneously, 2) a learned kernel that is trained on the base tasks and applied to the target task, 3) learning an adaptation of each base task to the current task, 4) an initial design learned from previous optimization runs, 5) reducing the design space based on previous optimization runs, 6) changing the acquisition function to include past data or models, 7) learning a separate model for each base task and combining those, 8) hybrid methods, and 9) prior-based methods. We review these techniques in Section 7 and refer to Vanschoren (2019) for an in-depth overview of meta-learning. We discuss two relevant methods which fall into categories 6 and 7 in Section 3, since they are special cases of the methods we introduce in this paper.

3. Weighting-Based Bayesian Optimization Transfer Learning

Several recent works in transfer learning for BO suggest modeling the current task t as a linear combination of a surrogate model fit to current task t, as well as one surrogate model fit to each of the base tasks $(1, \ldots, t-1)$ (Wistuba et al., 2016; Lindauer and Hutter, 2018; Wistuba et al., 2018). If the fitted base models were stored during past runs, they can be used directly without any refitting of prior results, and only the GP for the current task needs to be updated. As an additional advantage, such methods do not require the existence of meta-features. However, they depend crucially on the strategy for computing the weights. In this section, we review the ranking method employed by the two-stage transfer surrogate for learning this linear combination and an acquisition function that is used on top of such an ensemble, and assess them with respect to the design criteria given in the introduction.

3.1 Two-Stage Transfer Surrogate Model with Rankings

The two-stage transfer surrogate model with rankings (TST-R, Wistuba et al., 2016) suggests combining the mean predictions of the models as

$$\bar{\mu}(\mathbf{x}_*) = \sum_{i=1}^{t} w_i \mu_i(\mathbf{x}_*),$$

while dropping the variance of the base models, and using for a predictive variance just that of the target model. It uses a Nadaraya-Watson kernel weighting to linearly combine the predictions from each GP according to a normalized distance measure across tasks:

$$w_i = \frac{k_\rho(\chi_i, \chi_t)}{\sum_{j=1}^{t} k_\rho(\chi_j, \chi_t)},$$

with the kernel

$$k_\rho(\chi_i, \chi_j) = \gamma\left(\frac{||\chi_i - \chi_j||_2}{\rho}\right).$$

Here ρ is a bandwidth hyperparameter and γ is given by the Epanechnikov kernel:

$$\gamma(r) = \begin{cases} \frac{3}{4}(1 - r^2) & \text{if } r \leq 1 \\ 0 & \text{otherwise.} \end{cases}$$

The most important question is how to describe each task with a vector χ, and Wistuba et al. (2016) proposed two different methods to obtain them: (1) descriptive meta-features (Brazdil et al., 1994; Rivolli et al., 2018), and (2) directly modeling the distance as the proportion of discordant pairs when the base model is evaluated on configurations evaluated on the current task:

$$||\chi_i - \chi_t||_2 = \frac{2}{n_t(n_t-1)} \sum_{k=1}^{n_t-1} \sum_{l=k+1}^{n_t} \mathbb{1}((f^i(\mathbf{x}_k^t) < f^i(\mathbf{x}_l^t)) \oplus (y_k^t < y_l^t)),$$

where $\oplus$ is the exclusive-or operator.

The bandwidth ρ, a sensitive hyperparameter which must be chosen by the user (violating desideratum #3), gives the fraction of discordant pairs a previously observed dataset may share with the target dataset before being discarded. Updating the distance measure over time improves over the meta-feature based distance measure (as also shown by Leite et al. (2012) for a closely related method), and we will therefore not compare to any meta-feature-based method (requiring meta-features would also violate desideratum #3).

The kernel is used to combine mean predictions of base models with the mean prediction of the target model, but variances are not combined—the combined model is given the variance of the target model and variances of base models are ignored. Furthermore, TST-R assigns a constant value to the distance of the target model to itself; therefore the actual weight of the target model in the ensemble depends on the magnitude of the weights of the base models. As weights are computed independently of each other, correlated base tasks can shift the weight in the combined model. Due to the close relation to our proposed model, we include TST-R for comparison in our experiments of Section 6, where we will refer to using the model as a drop-in replacement of the standard GP in BO as TST-R(EI).

3.2 Transfer Acquisition Function

Another approach to handling meta-data in BO is to adapt the acquisition function to a so-called *transfer acquisition function* (TAF, Wistuba et al., 2018). Similarly to TST-R(EI), the TAF uses a weighting of the different tasks, but applies them to the acquisition function rather than to the surrogate model. The expected improvement acquisition function using only the target model f^t on the target task is combined linearly with an improvement term computed on each base task:

$$\begin{aligned}
\alpha(\mathbf{x}) &= w_t EI_t(\mathbf{x}) + \sum_{i=1}^{t-1} w_i I_i(\mathbf{x}) \\
&= w_t EI_t(\mathbf{x}) + \sum_{i=1}^{t-1} w_i \max\left(0, \left(\min_{\mathbf{x}_k \in \mathcal{D}^t} f^i(\mathbf{x}_k)\right) - f^i(\mathbf{x})\right) \\
&= w_t \mathbb{E}_{f^t(\mathbf{x})}\left[\max(0, f^t(\mathbf{x}_{\text{best}}) - f^t(\mathbf{x}))\right] + \sum_{i=1}^{t-1} w_i \max\left(0, \left(\min_{\mathbf{x}_k \in \mathcal{D}^t} f^i(\mathbf{x}_k)\right) - f^i(\mathbf{x})\right),
\end{aligned}$$

where $f(\mathbf{x}_{\text{best}}) = \min_{k \in (1,\ldots,n^t)} f^t(\mathbf{x}_k^t)$, i.e. the best observation on the target task as predicted by the target model.

The idea is to fade out base tasks more quickly than TST-R(EI) by not only measuring the similarity with the target task as discussed in the previous subsection, but also by measuring whether there is still information that can be transferred (i.e., whether the base model "knows" about a better hyperparameter setting than those tried so far).

The TAF uses the TST-R weighting scheme which we described in the previous subsection, and inherits the bandwidth hyperparameter ρ. As TAF is a special case of the model we develop in this paper, we will include it as a baseline in Section 6. We discuss this method in more detail in Appendix A, where we also relate it to the well-known meta-learning strategy of *active testing* (Leite et al., 2012).

4. Methodology

In this section, we describe our new approach for transfer learning in BO. We first develop a new probabilistic weighting scheme for combining base models with the target model (Section 4.1). Next, we demonstrate how it can be used in a linear combination of GPs and present how to use this ensemble model for BO (Section 4.2). Then, we introduce a regularization mechanism to deal with a potentially large number of unrelated base models (Section 4.3). We end this section by providing theoretical insight into the described weighting scheme and regularization (Section 4.4).

4.1 Computing Ensemble Weights

Intuitively, we wish to learn a stacking meta-model that combines all base models and the target model into a single, joint model (Pardoe and Stone, 2010). Training data for the stacking meta-model is generated in two ways. For base models, we evaluate their ability to predict the observations $\mathcal{D}^t$ for task t. For the target model, these would be in-sample predictions and thus overly optimistic, so we use leave-one-out cross-validation on $\mathcal{D}^t$ to get an estimate of the target model's ability to correctly predict the target task. Then, a combined model is learned to explain the target task observations in the best possible way. In the BO setting, the target model has access to data from the correct function (the target task), but it will not be the best model to use early in the optimization because it does not yet have enough data to accurately model the function. Therefore, for each timestep, we need to find out which models perform best at explaining the data observed so far. In the beginning, base models trained on previous tasks might perform better than the target model, so we would like to use them to transfer knowledge. As soon as we are certain that the target model is the best, we would like to rely on it instead.

A solution for this problem is given by agnostic Bayesian ensemble learning (*ABE*, Lacoste et al., 2014). Given a desired loss function, each predictor in the ensemble is weighted according to the probability that it is the best predictor in the ensemble for the target task.[2] This has the desirable property of taking into account our uncertainty about when to trust the target model and which base models to rely on. This strategy further allows us to also learn the weight of the target model, which is not possible with the TST-R

2. ABE was introduced to construct an ensemble from models trained on a single dataset where we have access to a validation dataset or can obtain it via cross-validation. In contrast, we use ABE here to transfer knowledge from prior tasks to new tasks in a transfer stacking manner and have therefore adjusted the way we obtain this validation set.

strategy. Concretely, ABE constructs an ensemble as

$$\bar{f}(\mathbf{x}) = \sum_{i=1}^{t} p(f^i == f^* | \mathcal{D}^t) f^i(\mathbf{x}),$$

where $f^* \in \{f^1, f^2, \ldots, f^t\}$ is the predictor that minimizes the desired loss function.

ABE either directly models the posterior risk of a potential ensemble member or uses the bootstrap of the predictions on the target tasks; both of these approaches are agnostic to the models used. We use the latter method due to its simplicity and because it only requires the models' predictions on the target task.

We first construct the loss function for use with ABE. Our goal in BO is to find the minimum function value and a model will be useful for optimization if it is able to correctly order observations according to their function value. For meta-learning, we wish to assess the ability of model i to generalize to the target function, and so construct a loss function that measures the degree to which each model is able to correctly rank the target observations $\mathcal{D}^t$. Given $n_t > 1$ target function evaluations, we define the loss as the number of misranked pairs:

$$\mathcal{L}(f, \mathcal{D}^t) = \sum_{k=1}^{n_t} \sum_{l=1}^{n_t} \mathbb{1}((f(\mathbf{x}_k^t) < f(\mathbf{x}_l^t)) \oplus (y_k^t < y_l^t)), \tag{3}$$

where $\oplus$ denotes the exclusive OR operator (XOR).

Note here that we are evaluating base models only on their predictions for the target task, and thus are assessing their ability to generalize to the target function. For the target model, as mentioned above, this would be an estimate of the in-sample error and would not accurately reflect generalization. We estimate generalization in the target model using cross-validation, in practice with leave-one-out models. Let f_{-j}^t indicate the target model with observation (x_j^t, y_j^t) left out. The loss for the target model is computed as

$$\mathcal{L}(f^t, \mathcal{D}^t) = \sum_{k=1}^{n_t} \sum_{l=1}^{n_t} \mathbb{1}((f_{-k}^t(\mathbf{x}_k^t) < y_l^t) \oplus (y_k^t < y_l^t)). \tag{4}$$

Models f_{-j}^t are only useful when we fit them with at least two observations. For this reason we start the weighting procedure only when we have gathered three observations. Before that we assign uniform weights to all models.

The ranking loss is preferable to other choices, such as squared error or model log-likelihood, because the actual values of the predictions do not matter for optimization—we only need to identify the location of an optimum. It is easy to see that if all of the models are able to correctly order a set of points then the ensemble will also correctly order those points.

We now weight each model with the probability that it is the ensemble member with the lowest ranking loss. We estimate this probability with a Monte Carlo approximation. Namely, we draw S bootstrap samples $\ell_{i,s} \sim \mathcal{L}(f^i, \mathcal{D}_s^t)$ from each of our models $i = 1, \ldots, t$, and for each sample s assess which model aligns best with the observed data $\mathcal{D}^t$. The weight for model i is then computed as

$$w_i = \frac{1}{S} \sum_{s=1}^{S} \left(\frac{\mathbb{I}(i \in \arg\min_{i'} l_{i',s})}{\sum_{j=1}^{t} \mathbb{I}(j \in \arg\min_{i'} l_{i',s})} \right), \tag{5}$$

Algorithm 1: Learning weights with the agnostic Bayesian learning of ensembles

Input: Number of bootstrap samples S, all observations on all base tasks and the target task $\mathcal{D}^i \forall i \in (1, \ldots, t)$, all base models and the target model $f^i \forall i \in (1, \ldots, t)$
if $|\mathcal{D}^t| < 3$ **then**
  return weights $\left[\frac{1}{t} | i \in (1, \ldots, t)\right]$
end if
$L = [[l_{1,1}, \ldots, l_{1,S}], \ldots, [l_{t,1}, l_{t,S}]]$ `// Initialize` $2d$ `array to store the losses per model and bootstrap sample`
for $s \in (1, 2, \ldots, S)$ **do**
  $\mathcal{B} = \{randint(1, n_t)\}_{k=1}^{n_t}$ `// Draw bootstrap indices with replacement`
  $\mathcal{D}_s^t = \{(\mathbf{x}_k^t, y_k^t) | k \in \mathcal{B}\}$ `// Resample the target task data`
  for $i \in (1, 2, \ldots, t)$ **do**
    if $i < t$ **then**
      $l_{i,s} = \mathcal{L}(f^i, \mathcal{D}_s^t)$ `// Equation 3`
    else
      $l_{t,s} = \mathcal{L}(f^t, \mathcal{D}_s^t)$ `// Equation 4`
    end if
  end for
end for
Compute weights $\mathbf{w} = \left[\frac{1}{S} \sum_{s=1}^{S} \left(\frac{\mathbb{I}(i \in \arg\min_{i'} l_{i',s})}{\sum_{j=1}^{t} \mathbb{I}(j \in \arg\min_{i'} l_{i',s})}\right) | i \in (1, \ldots, t)\right]$ `// Equation 5`
return weights $\mathbf{w}$

which distributes the weight of a sample across all members of the argmin in case of a tie. Because base models compete against each other, correlated base models are weighted down as in the original ABE strategy (Lacoste et al., 2014). Using ABE induces a slight overhead compared to TST-R, but is still substantially faster than using a single Gaussian process for all data points (see Appendix B). We present pseudo-code in Algorithm 1.

4.2 Ranking-Weighted Gaussian Process Ensemble

Having established a weighting strategy, we propose to use a weighted combination of the predictions of each base model and the target model itself: $\bar{f}(\mathbf{x}|\mathcal{D}) = \sum_{i=1}^{t} w_i f^i(\mathbf{x}|\mathcal{D}^i)$. We will show that this is a very powerful formalism. First, this ensemble model remains a GP, and in particular

$$\bar{f}(\mathbf{x}|\mathcal{D}) \sim \mathcal{N}\left(\sum_{i=1}^{t} w_i \mu_i(\mathbf{x}), \sum_{i=1}^{t} w_i^2 \sigma_i^2(\mathbf{x})\right), \tag{6}$$

which leaves the usual computational machinery for BO with GPs valid, such as the applicability of arbitrary acquisition functions, a closed-form expression for EI, and the ability to draw joint samples for parallel optimization. We dub this model ranking-weighted Gaussian process ensemble, or short, RGPE.

Additionally, each base model remains unchanged throughout the optimization and can be loaded directly from the previous runs. The fitting cost is only the cost of fitting the target model and inferring the weights w_i. Finally, predictions are made independently for each GP, and at prediction time we obtain only a linear slowdown relative to using only the target model. In case there would be too many data points for the base tasks to fit a GP on them, one could also replace them with more scalable models, such as Bayesian neural

Abbreviation	Description	Section
RGPE(mean)	Ranking-weighted Gaussian process ensemble using only the variance of the target model	4.2.1
RGPE(NEI)	Ranking-weighted Gaussian process ensemble with noisy EI	4.2.2
RGPE(MoGP)	Mixture of Gaussian processes weighted by RGPE	4.2.3
RGPE(TAF)	Transfer acquisition function with the RGPE model	4.2.4

Table 1: Instantiations of RGPE.

networks (Schilling et al., 2015; Snoek et al., 2015; Springenberg et al., 2016) or random forests (Hutter et al., 2011), but this is beyond the scope of this paper.

To compensate for the fact that response values of different tasks can live on different scales, we standardize each model prior to inclusion in the ensemble (Yogatama and Mann, 2014). We study alternatives to this strategy in Appendix D.3.

We now present how we will use the RGPE model for BO and give an overview of the different instantiations of RGPE in Table 1.

4.2.1 Expected improvement and Dropping Base Model Variance

The most straight-forward application of RGPE for BO is to use the ensemble as a plug-in replacement for the standard GP. As the most simple strategy one can use regular expected improvement with the posterior mean as a plug-in estimate for $f(\mathbf{x}_{\text{best}})$. As a second ad-hoc strategy we consider using only the variance of the target model and drop the base model variances as suggested by TST-R(EI) (see Section 3).[3] We refer to these techniques as RGPE(EI) and RGPE(mean).

4.2.2 Noisy Expected Improvement

However, with the formulation in Equation 6, the base models can add uncertainty to points where the target model is certain. In particular, they can add posterior variance to points where the target model has noiseless observations, which can inflate the acquisition value around already-observed points. Because base models are not updated over time, this could lead to the optimization process stalling when using standard EI as we discussed in the previous subsection. Also, if observations are noisy or if there is uncertainty in base models at the current best, $f(\mathbf{x}_{\text{best}})$ in the expected improvement (see Equation 2) is not a constant, but rather is a random variable with uncertainty both in which point is the best, and what the function value at that point is. This can occur when the locations of the observations in the base models do not overlap with those of the target model or if there is overlap but noise in the base models, or if there is noise in the target model.

3. An alternative derivation of dropping the base model variances is using the weighted base models as a mean function in the GP model. If we define the mean function to be $m(\mathbf{x}) = \sum_{i=1}^{t-1} w_i f^i(\mathbf{x})$ and the target GP to regress $\tilde{y} = m(\mathbf{x}) + w_t y$ we obtain a predictive mean of the form

$$\mu(\mathbf{x}_*) = m(\mathbf{x}_*) + \mathbf{k}_*^T(\mathbf{K} + \sigma_n^2 I)^{-1}(\tilde{\mathbf{y}} - \mathbf{m}(\mathbf{X})), \tag{7}$$

while the variances of the base models are not considered at all.

Typical approaches for computing the expected improvement with noisy observations can be used with the RGPE; we use the so-called noisy expected improvement to account for the uncertainty in the best value (Letham et al., 2019). We refer to the resulting combination of RGPE and the noisy expected improvement as RGPE(NEI).

With this approach, we integrate out the uncertainty in the current best value, extending the expectation in Equation 2 to be over the joint posterior of $p(f(\mathbf{x}), f(\mathbf{x}_{\text{best}})|\mathcal{D})$. We first describe the procedure for a single Gaussian process before explaining how to use it with the RGPE model.

The expectation is computed by first using Monte-Carlo (MC) sampling to integrate over the posterior $p(f(\mathbf{x}_{\text{best}})|\mathcal{D})$, by drawing joint samples from the ensemble posterior at all of the observations $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n$. In each sample, $f(\mathbf{x}_{\text{best}})$ is deterministically computed as the best value in the sample draw. For each joint sample, we can then compute the conditional posterior $p(f(\mathbf{x})|f(\mathbf{x}_{\text{best}}), \mathcal{D})$ by conditioning the Gaussian process on the (noiseless) sample values at the observations. This means that we condition the GP on $\{(\mathbf{x}_k, \tilde{f}(\mathbf{x}_k)\}_{k=1}^n$ with $\tilde{f} \sim p(f|\mathcal{D})$.[4] Afterwards, $f(\mathbf{x}_{\text{best}})$ is deterministic and we can compute EI for each MC sample in the usual manner using Equation 2. Averaging EI over the MC samples produces an estimate for the joint expectation.

For use with an ensemble model the noisy EI requires us to sample from the ensemble posterior. Because all models in the ensembles we consider are independent, the sampling and conditioning described above can be done independently for each base model. The integration procedure does not change for the target model, but for the base models. Here we need to draw joint samples at the locations observed on the target task and the respective base tasks, i.e. conditioning the GP for task i on $\{(\mathbf{x}_k^i, \tilde{f}^i(\mathbf{x}_k^i)\}_{k=1}^{n^i} \cup \{(\mathbf{x}_k^t, \tilde{f}^i(\mathbf{x}_k^t)\}_{k=1}^{n^t}$ with $\tilde{f}^i \sim p(f^i|\mathcal{D}^i)$. For each draw we then computed EI using the ensemble. In practice we used 30 draws of quasi-Monte Carlo integration (using scrambled Sobol sequences as described by Letham et al. (2019)).

4.2.3 Ranking-Weighted Mixture of Gaussian Processes

Having introduced a combination of models based on the addition of random variables, we now establish a way in which the combination of GPs can be used in a mixture-of-GPs fashion (Tresp, 2001). The PDF of a Gaussian mixture model at $f(\mathbf{x})$ with t components is $p_{\text{mix}}(f(\mathbf{x})) = \sum_{i=1}^t w_i p_i(f(\mathbf{x}))$, where p_i is the (Gaussian) PDF of component i. Making use of the fact that we can sample from a mixture-of-GPs by first sampling a mixture component and then sampling from the GP, the expected improvement at $\mathbf{x}$ can be decomposed:

$$EI_{\text{mix}}(\mathbf{x}) = \mathbb{E}_{f(\mathbf{x})\sim p_{\text{mix}}}[I(\mathbf{x})] = \mathbb{E}_i \mathbb{E}_{f(\mathbf{x})\sim p_i}[I(\mathbf{x})] = \sum\nolimits_{i=1}^t w_i EI_i(\mathbf{x}), \tag{8}$$

where $EI_i(\mathbf{x})$ is the EI computed under mixture component i. Thus, EI for the mixture model is a weighted sum of the EI for each of member of the ensemble. In order to select $\mathbf{x}_{\text{best}}$ for EI_i we use the predictive mean of the respective model i. Importantly, we choose $\mathbf{x}_{\text{best}} \in \mathcal{D}^t$, so we aim to improve only over observations made on the target task. We call the resulting procedure RGPE(MoGP).

4. In practice, for numerical stability the noise levels at these conditioning points will not be exactly 0, so we set it to the minimum noise level our GP can take on during its hyperparameter optimization: e^{-25}.

In order to deal with noise in the base models we use the re-interpolation trick (Forrester et al., 2008). The re-interpolation trick can be regarded as a one-shot alternative to the noisy EI described in Section 4.2.2. Instead of using MC integration, we condition the base model GP f^i on the predictive mean at the target observations, i.e. $\mathcal{D}^i \cup \{(\mathbf{x}_k^t, f^i(\mathbf{x}_k^t)\}_{k=1}^{n_t}$. This ensures that the individual $f^i(\mathbf{x}_{best})$ per base model are constants and the EIs using the individual base models can be computed in closed form.

Additionally, the result in Equation 8 makes it clear how the mixture of GPs can be used with any other acquisition function that is based on the expectation of a quantity (Wilson et al., 2018), for example the knowledge gradient (Frazier, 2018) and max-value entropy search (Wang et al., 2017), while it is not clear whether this can be done with the TAF.

4.2.4 Transfer Acquisition Function

Finally, we notice a striking resemblance between this method and the transfer acquisition function from Section 3.2, with a single difference: Equation 8 uses the EI and not only the improvement for the base models. The transfer acquisition function is therefore a different approach of dealing with the variance of the base models in a mixture-of-GPs, and this line of reasoning gives us a derivation of the TAF from our RGPE model given in Equation 6. This combination of our RGPE model with the well-performing transfer acquisition function will be the overall best method in our experiments and we will refer to it as RGPE(TAF).

4.3 Preventing Weight Dilution

One challenge with this type of ensemble is preventing weight dilution by a large number of models. Due to the bootstrap sampling, even models with bad generalization performance have a chance of obtaining a non-zero weight. This is because when we consider a very large number of poor models, the chance of at least one of them producing the correct ranking on a sample goes to 1 as the number of models increases.

In order to prevent weight dilution we add a base model filtering step to drop base models which are unlikely to improve over the target model. The weight calculation mechanism presented in the previous section allows us to directly compare two models according to their quality, and in particular to compare the quality of any base model to the quality of the current target model. We prevent weight dilution by discarding base models with frequency proportional to their probability of producing a worse ranking than the target model. Base model $i \in (1, \dots, t-1)$ is discarded from the ensemble in each iteration with probability

$$p_{drop}(i) = 1 - \left(\left(1 - \frac{n_t}{H}\right) \frac{\sum_{s=1}^{S} \mathbb{1}(l_{i,s} < l_{t,s})}{S} \right), \tag{9}$$

where H is the optimization horizon. The factor $\frac{\sum_{s=1}^{S} \mathbb{1}(l_{i,s} < l_{t,s})}{S}$ is the probability that the base model outperforms the target model. Instead of a conjugate beta prior, we use a multiplicative prior $(1 - \frac{n_t}{H})$ that reduces the probability of keeping a base model linearly over time and encodes our belief that the target model is the correct model and should be favored. This ensures that at the end of the optimization we use only the target model. We highlight that this mechanism does not automatically give a higher weight to the target

model. Base models are discarded prior to computing the weights. Therefore, the target model still needs to outperform the remaining base models in order to obtain a high weight.

Preventing weight dilution also has computational benefits in that it results in fewer GP predictions for each ensemble prediction. Models are only removed from the ensemble proportionally to their probability of performing worse than the target model. In contrast, TST-R removes models if they perform badly on the observed function evaluations without taking into account the performance of the target model, rather based on the hyperparameter ρ. Furthermore, there is a non-zero probability that we only consider the target model in each iteration, which we will use in Section 4.4.

4.4 Theoretical Analysis

Here, we show that in each iteration the proposed weighting mechanism has a positive chance of performing vanilla BO, in which only the target model is used. Therefore, standard proofs for the convergence of BO apply (Bull, 2011) with a multiplicative slowdown factor.[5]

Theorem 1 *Bayesian optimization using a linear combination of Gaussian processes with weights learned according to Section 4.3 is at most a factor of*

$$1/\left(\frac{1}{H}\sum_{h=1}^{H}\left(\frac{h}{H}\right)^{t-1}\right)$$

slower than Bayesian optimization in the worst case.

As before, H is the optimization horizon, while we change the number of observed data points n^t to h, i.e. the current iteration.

Proof sketch: In order for the proposed method to fall back to vanilla BO, we need the weights of all models except that of the base model to be zero. Given the definition of p_{drop} in Equation 9, and setting n_t to h in $p_{drop}(i,h)$, we can calculate the probability of dropping all base models at step h as $\prod_{i=0}^{t-1} p_{drop}(i,h)$, and so the expected proportion of iterations that proceed as vanilla BO is

$$\frac{1}{H}\sum_{h=1}^{H}\prod_{i=1}^{t-1} p_{drop}(i,h) \tag{10}$$

$$= \frac{1}{H}\sum_{h=1}^{H}\prod_{i=1}^{t-1}\left(1-\left(\left(1-\frac{h}{H}\right)\frac{\sum_{s=1}^{S}\mathbb{1}\left(l_{i,s}<l_{t,s}\right)}{S}\right)\right) \tag{11}$$

$$\geq \frac{1}{H}\sum_{h=1}^{H}\left(1-\left(\left(1-\frac{h}{H}\right)\frac{S}{S}\right)\right)^{t-1} \tag{12}$$

$$= \frac{1}{H}\sum_{h=1}^{H}\left(\frac{h}{H}\right)^{t-1} \tag{13}$$

$$> 0 \tag{14}$$

5. This only applies to results that give guarantees in terms of the simple regret and not the cumulative regret.

The observations gathered in iterations when not all base models are dropped do not impose any issues on the convergence proof by Bull, which we show in Appendix C. ■

This analysis is for the worst-case setting in which every base model consistently outperforms the target model. In practice, base models are frequently worse than the target model, leading to an improved bound.

5. An Experimental Framework for Transfer Hyperparameter Optimization

In this section we describe the experimental framework we implemented to develop our new methods and compare them to prior work from the literature. For this, we give an overview of the benchmark problems we use and also detail our implementation of the proposed methods and prior work into the Bayesian optimization framework SMAC3.

We provide all code at `https://github.com/automl/transfer-hpo-framework`.

5.1 Benchmarks

We evaluate our methods on six benchmark problems we collected from the literature. We make them available with a uniform API to simplify their usage and allow them to be reused.

First, we use two HPO benchmark problems for optimizing the four hyperparameters of a support vector machine ($\text{SVM}_{s,4D}$) and the ten hyperparameters of gradient boosting ($\text{XGBoost}_{s,10D}$) following Perrone et al. (2018). These benchmark problems are surrogate benchmarks built on data from `OpenML.org` (Vanschoren et al., 2014). They were not released to the public, but data to construct the surrogates is available from OpenML.org, and so we have reconstructed them and made them publicly available as part of our framework release.[6] These are evaluations on 38 binary datasets without missing values from the OpenML 100 (Bischl et al., 2019). We follow the standard methodology of employing a random forest as a surrogate (Eggensperger et al., 2015), and optimize its hyperparameters using random search and 10-fold cross-validation to maximize the Spearman rank correlation. We then also use the data from the same source that is available for generalized linear models to construct a 2-d HPO problem ($\text{GLMNET}_{s,2D}$).

Second, we use LCBench, a neural network benchmark that contains the performance of deep neural networks implemented in Auto-Pytorch (Zimmer et al., 2021, $\text{NN}_{t,7D}$). These networks are trained on 35 different tasks from the OpenML AutoML benchmark (Gijsbers et al., 2019), excluding the four *large* ones. For each dataset, the benchmark contains 2000 randomly sampled hyperparameter settings from a 7-d hyperparameter space.

Third, we use two large sets of hyperparameter optimization benchmark problems for $\text{AdaBoost}_{g,2D}$ and $\text{SVM}_{g,6D}$ on a diverse set of 50 datasets, with sizes ranging from 35 to 250000 training examples, and from 2 to 7000 features from Schilling et al. (2016) and Wistuba et al. (2018). Both of them are grid benchmarks, meaning that they contain a discretized grid of hyperparameter settings and the corresponding test-set accuracies. The $\text{AdaBoost}_{g,2D}$ benchmark, inspired by a benchmark suggested by Bardenet et al. (2013),

6. We use the curated data by Kühn et al. (2018), which is actually a superset of what was used by Perrone et al. (2018). This data is available in multiple csv files created by the uploaders of the data to OpenML, while Perrone et al. (2018) downloaded the data from OpenML.org independently. These csv files contain all results required to construct the surrogates.

Name	# Dim	# Cat	# Cont	# Cond	# Tasks	# Samp	Reference
$\text{AdaBoost}_{g,2D}$	2	0	2	0	50	50	Schilling et al. (2016)
$\text{SVM}_{g,6D}$	6	3	3	0	50	50	Schilling et al. (2016)
$\text{GLMNET}_{s,2D}$	2	0	2	0	38	50	ours
$\text{SVM}_{s,4D}$	4	1	3	2	38	50	Perrone et al. (2018)
$\text{XGBoost}_{s,10D}$	10	1	9	4	38	50	Perrone et al. (2018)
$\text{NN}_{t,7D}$	7	0	7	0	35	50	Zimmer et al. (2021)

Table 2: Summary of the benchmark problems considered. We give the name; dimensionality; number of categorical, continuous, and conditional hyperparameters; number of tasks; and the number of samples used to train the base models.

contains a grid of two hyperparameters, the number of iterations and the number of product terms, for a total of 108 evaluations. The $\text{SVM}_{g,6D}$ benchmark contains a grid of six hyperparameters: three binary hyperparameters indicating a linear, polynomial, or RBF kernel; the penalty hyperparameter C; the degree of the polynomial kernel (0 if unused); and the RBF kernel bandwidth (0 if unused) for a total of 288 function evaluations. Note that this is a harder problem than the common 2-dimensional RBF SVM problem.

All of these are available under a unified interface for the first time which allows them to easily be reused by other researchers. We summarize the benchmarks in Table 5.1 and provide the exact search spaces in Appendix D.

5.2 Implementation of Transfer HPO Methods

We used the SMAC3 Bayesian optimization package (Lindauer et al., 2021), which to the best of our knowledge is the only BO package that natively supports Gaussian processes with categorical and conditional hyperparameters. SMAC3 implements GPs with scikit-learn (Pedregosa et al., 2011), and we used the ARD Matérn 5/2 kernel. Kernel hyperparameters θ were optimized in each iteration for the maximum a-posteriori estimate via L-BFGS-B using priors on the GP hyperparameters (a top-hat prior on the length-scales, a log-normal prior on the function scale and a horseshoe prior on the noise). We used a Latin-hypercube initial design of size 10 for the GP-based vanilla BO.

For categorical hyperparameters, SMAC3 uses a Hamming kernel (Hutter, 2009) and it uses a kernel separating the search spaces into subspaces if there are hyperparameters that are dependent on the exact setting of another hyperparameter (Lévesque et al., 2017). In order to optimize the acquisition function for such mixed spaces, SMAC3 uses a stochastic local search starting from promising points in the search space (Hutter et al., 2011; Lévesque et al., 2017). Obviously, this is not necessary for the grid-based benchmarks, for which we can simply compute the acquisition function value for each point. We used the expected improvement acquisition function (Jones et al., 1998) in all experiments unless the method provides an alternate acquisition function itself (TAF, RGPE(MoGP)) or requires special

treatment of uncertainty (RGPE). We give an overview over all implemented methods in Table 3.

5.3 Implemented Methods

We briefly describe the HPO methods we implemented. We start with single task *standard HPO* baselines, then go on to prior *transfer HPO* baselines from the literature, before finishing with the methods we propose in this paper. We give an overview of these methods in Table 3. Following the design of SMAC3 (Lindauer et al., 2021) we can implement them as four distinct building blocks of a Bayesian optimization algorithm: 1. the hyperparameter search space, 2. the initial design of Bayesian optimization, 3. the model, and 4. the acquisition function.

As standard HPO methods we provide a standard GP as described above and a Gaussian Copula process (GCP, Salinas et al., 2020), in which the responses for each task are transformed to follow a standard normal distribution using a Copula transformation. Furthermore, we provide random search (Bergstra and Bengio, 2012) and random search with simulated parallelism, which are suggested as sanity checks for hyperparameter optimization methods (Recht and Jamieson, 2016) (Random 2x, 4x and 10x).[7]

We implemented the competitor methods search space learning (Perrone and Shen, 2019), sequential model-free hyperparameter optimization (Wistuba et al., 2015a), warm-starting algorithm configuration (Lindauer and Hutter, 2018), ABLR (Perrone et al., 2018), GCP+Prior (Salinas et al., 2020), TST-R(EI) (Wistuba et al., 2016) and TAF (Wistuba et al., 2018) as closely as possible within our Python-based system. Methods not described in Section 3.1 are described in Section 7.

For the search space learning methods (Perrone and Shen, 2019) there is no code available. We reimplemented the low-volume bounding box using for-loops (Section 4.2) and the handling of outliers using scipy (Virtanen et al., 2020) (Section 5). We then use it together with Random Search (Random+Box) and a GP (GP(Box)).[8] Search space learning is implemented to run prior to calling Bayesian optimization and its only action is creating a new searchspace. Similarly, we had to reimplement sequential model-free hyperparameter optimization (Wistuba et al., 2015a, SMFO), which then is an initial design passed to SMAC3.

For the WAC method (Lindauer and Hutter, 2018) we used the publicly available code as a starting point for our implementation.

For the linear weighting of models using the Kullback-Leibler divergence (Ramachandran et al., 2019b) we used the MATLAB implementation provided by the authors as a starting point to reimplement the weighting scheme in Python.

For the ABLR method (Perrone et al., 2018) there is no code available which we could either reuse or compare against and therefore used a Pytorch reimplementation.

7. These are available as post-hoc methods. For each seed we perform random search once with 500 function evaluations and then postprocess each run to have 50 steps, where each step consists of running multiple evaluations (2, 4 or 10) in parallel in a synchronous fashion. We cannot run this for two of our grid-based benchmarks, $\text{AdaBoost}_{g,2D}$ and $\text{SVM}_{g,6D}$, as they have less than 500 points in the grid.
8. Our implementation currently does not allow combining search space pruning with other transfer HPO methods.

Abbreviation	Description	Reference
GP	Standard Gaussian process	Jones et al. (1998); Lindauer et al. (2021)
GCP	Gaussian process with Copula transform	Salinas et al. (2020)
Random (1x)	Random search	Bergstra and Bengio (2012)
Random (2x,4x,10x)	Random search with simulated parallelism	Recht and Jamieson (2016)
Random+Box	Random search with a learned box-shaped search space	Perrone and Shen (2019)
GP(Box)	Gaussian process with a learned box-shaped search space	Perrone and Shen (2019)
SMFO	sequential model-free optimization	Xu et al. (2010, 2011); Wistuba et al. (2015a); Pfisterer et al. (2021)
WAC	warmstarting algorithm configuration	Lindauer and Hutter (2018)
KL-weighting	Two-stage transfer surrogate model using KL-divergence based weighting	Ramachandran et al. (2019b)
ABLR	adaptive basis function linear regression	Perrone et al. (2018)
GCP+Prior	Gaussian Copula process with a neural network prior	Salinas et al. (2020)
TST-R(EI)	Two-stage transfer surrogate model using pairwise rankings, see Section 3.1	Wistuba et al. (2016)
TST-R(TAF)	Transfer acquisition function, see Section 3.2	Wistuba et al. (2018)
RGPE(mean)	Ranking-weighted Gaussian process ensemble using only the variance of the target model	ours, see Section 4.2.1
RGPE(NEI)	Ranking-weighted Gaussian process ensemble with noisy EI	ours, see Section 4.2.2
RGPE(MoGP)	Mixture of Gaussian processes weighted by RGPE	ours, see Section 4.2.3
RGPE(TAF)	Transfer acquisition function with the RGPE model	ours, see Section 4.2.4

Table 3: Methods compared throughout this work. Top: standard HPO baselines. Middle: prior work on transfer HPO. Bottom: our methods.

For GCP+Prior there is a reimplementation available by one of the authors[9], but it was made available only after we wrote our own reimplementation using Pytorch. As the author's reimplementation does not achieve the exact numbers from the paper either, we stick to our own reimplementation.

While Java code is publicly available for TST-R (Wistuba, 2016), we obtained the code for TAF via private communication with the authors. TST-R(EI) and TAF with the TST-R weighting (TST-R(TAF)) require setting a bandwidth hyperparameter ρ and we choose it per benchmark to be the best on the remaining five benchmarks ($\rho \in (0.01, 0.02, 0.05, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0)$). We refer to Appendix D.2 for further details on our reimplementation.

Our code for these is organized as follows. For the approaches that learn an ensemble (WAC, KL-weighting, TST-R) or are a single model (ABLR) we implement a new model class for SMAC3. The acquisition function is then implemented as a separate acquisition function class that can be used together with an ensemble. Because GCP+Prior requires both a model and an acquisition function, we provide two classes that can be used together. All model-based methods use the same initial design and the minimal number of observations required to start the method from SMFO as a starting point. The minimal number is two for all methods that scale the observations to have zero mean and unit variance or using the copula transform and one for all methods that do not scale the data.

Finally, we implemented our methods as follows. We first implemented a model class for the RGPE model described in Sections 4.1 and 4.3, and second, implemented two acquisition functions, the noisy EI (4.2.2) and the mixture of Gaussian processes (4.2.3). This allows us to combine our ensemble with the noisy EI (RGPE(NEI)), use our ensemble with regular EI (RGPE(EI)) and only the variance of the target model (RGPE(mean)), with the acquisition function based on the mixture of GPs (RGPE(MoGP)) and also with the transfer acquisition function (RGPE(TAF)). For our methods we used $S = 1000$ for each experiment and study its effect in Section 6.4.

6. Experiments

In this section we describe the experimental validation of the methods introduced so far. We first give the experimental setup and then we show and discuss the results. For better readability we split the presentation into three parts: 1) a comparison to prior work and baselines, 2) a study on the evolution of weights over time, and 3) an ablation study of our methods' components. We provide further experiments on how to scale data, an ablation study on the effect of the hyperparameters of the baselines and comparisons to versions we proposed in earlier preprints of this paper in Appendix D.

6.1 Experimental Setup

We optimize each function for a total of 50 iterations. For each meta-task we provide 50 function evaluations obtained by vanilla BO to the transfer HPO method. We then start each transfer HPO method with a single meta-learned configuration (or two if the method performs some sort of data scaling that requires at least two observations on the target task)

9. https://github.com/geoalgo/A-Quantile-based-Approach-for-Hyperparameter-Transfer-Learning

and each other method with a Latin hypercube design of size 10. To account for randomness we conducted 15 repetitions.

We evaluated each benchmark in a leave-one-task-out fashion: we use one task as the target task and the remaining ones as base tasks. For reporting comparable numbers, we first normalize the regret on each dataset between zero and one. Second, we average the normalized regret for all tasks within a benchmark. This metric is also known as the *average distance to the global minimum* (ADTM, (Wistuba et al., 2018)).

We provide results as tables and rescale the ADTM to be percentages (between zero and one hundred). We typeset the best method in boldface and underline all methods which are not significantly different from the best according to a paired Wilcoxon signed-rank. For comparing performance, the *paired* differences on the different tasks are the key quantity of interest, and the Wilcoxon signed-rank test provides a correct statistical test for assessing if one method performs better than another (Demšar, 2006). The standard deviations in performance often used in BO papers are not useful here (and so are not given) because they include the significant variance from the different meta-datasets.

All experiments were conducted on a standard compute cluster with Broadwell E5-2630v4 2.2GHz processors.

6.2 Comparison to Prior Work and Baselines

In the first part of our experiments we compare the methods proposed in this paper to other popular hyperparameter optimization methods (standard HPO) and transfer learning hyperparameter optimization methods (transfer HPO) from the literature. We already described all methods in Section 5.2 and gave an overview in Table 3. To fit all methods into a single table we have to split the results across three tables in total and present results for $\text{GLMNET}_{s,2D}$ and $\text{SVM}_{s,4D}$ in Table 4, results for $\text{XGBoost}_{s,10D}$ and $\text{NN}_{t,7D}$ in Table 5 and results for $\text{AdaBoost}_{g,2D}$ and $\text{SVM}_{g,6D}$ in Table 6. For each method we give the averaged distance to the minimum at 10, 20, 30, 40 and 50 iterations of BO.

On a high level, we find that our proposed methods (bottom groups) consistently improve over the standard HPO baselines (top groups) and transfer HPO baselines (middle groups). Remarkably, RGPE(TAF) is either the best model or performs statistically equivalent to the best model on all six benchmarks. RGPE(TAF) is also able to outperform Random(10x) on three out of four benchmarks and is always better than Random(4x) (we cannot apply Random(2x) etc. on $\text{AdaBoost}_{g,2D}$ and $\text{SVM}_{g,6D}$ as there are not sufficient points in the grid).

Our proposed methods consistently outperform the standard HPO baselines vanilla BO and random search. Comparing our proposed method to the TST-R(EI) baseline, we find that TST-R(EI) can perform well on some tasks, but fails to do so on the $\text{SVM}_{s,4D}$ and $\text{NN}_{t,7D}$ benchmarks. We find that our methods consistently outperform the additional baselines WAC, ABLR and KL-weighting. We find these baselines to have unreliable performance and suggest further research into their robustness. GCP+Prior showed good performance on some problems, but our proposed methods yield further improvements in mean performance on all six benchmarks, and significant improvements on three of them.

The SMFO baseline is a stronger baseline than the regular GP and is particularly strong on the $\text{NN}_{t,7D}$ benchmark, where it outperforms all methods except for RGPE(TAF). The

	10	20	30	40	50	10	20	30	40	50
	$\text{GLMNET}_{s,2D}$					$\text{SVM}_{s,4D}$				
GP	1.92	**0.59**	**0.45**	**0.38**	**0.35**	5.38	1.60	1.06	0.76	0.68
GCP	1.92	**0.55**	**0.44**	**0.39**	**0.36**	5.38	2.25	1.78	1.51	1.33
Random(1x)	4.80	1.40	0.89	0.70	0.52	5.60	2.99	2.12	1.77	1.58
Random(2x)	1.40	0.70	0.47	0.36	0.32	2.99	1.77	1.44	1.15	1.02
Random(4x)	0.70	0.36	0.29	0.24	0.21	1.77	1.15	0.89	0.78	0.69
Random(10x)	0.32	**0.21**	**0.16**	**0.13**	**0.10**	**1.02**	**0.69**	0.56	0.51	0.46
Random+Box	0.96	0.65	0.50	0.38	0.32	4.08	2.30	2.11	1.72	1.58
GP(Box)	0.73	**0.27**	**0.22**	**0.19**	**0.17**	5.10	1.31	0.68	0.52	0.45
SMFO	0.60	**0.32**	**0.29**	**0.27**	**0.25**	3.71	2.12	1.98	1.76	1.74
WAC	5.14	4.84	4.82	4.81	4.81	3.64	3.30	2.99	2.90	2.84
KL-weighting	0.94	0.68	0.61	0.28	0.19	-	-	-	-	-
ABLR	4.09	3.05	2.98	2.96	2.95	4.52	2.46	1.34	0.93	0.86
GCP+Prior	2.98	**2.48**	**1.69**	**0.30**	**0.26**	5.23	4.08	2.61	2.24	2.11
TST-R(EI)	**0.44**	**0.38**	**0.23**	**0.23**	**0.20**	**1.35**	**0.90**	0.64	0.52	0.49
TST-R(TAF)	0.36	**0.26**	**0.21**	**0.19**	**0.17**	2.76	1.15	0.85	0.60	0.48
RGPE(TAF)	**0.29**	**0.22**	**0.19**	**0.17**	**0.16**	**2.17**	**1.78**	**0.78**	**0.48**	**0.38**
RGPE(MoGP)	**0.39**	**0.23**	**0.20**	**0.18**	**0.16**	2.33	**1.08**	0.74	**0.50**	**0.30**
RGPE(mean)	**0.43**	**0.31**	**0.27**	**0.25**	**0.23**	1.78	**0.78**	**0.45**	**0.38**	**0.26**
RGPE(NEI)	3.29	**2.67**	**2.22**	**2.21**	**2.20**	2.97	**0.91**	**0.66**	**0.44**	**0.37**

Table 4: Final results on the $\text{GLMNET}_{s,2D}$ and $\text{SVM}_{s,4D}$ benchmarks. The numbers reported are the average normalized regret (Wistuba et al., 2018). We boldface the best value per benchmark and number of function evaluations and underline methods that are not significantly different according to a Wilcoxon signed-rank test with $\alpha = 0.05$ (Demšar, 2006). Top group: standard HPO baselines; middle group: transfer HPO baselines; **bottom group: our methods**.

small difference in performance between the GP and random search shows that modeling provides limited benefit on this benchmark. However, it is outperformed on the other five benchmarks by most transfer learning methods. We would like to note that the SMFO baseline outperforming the proposed methods on a single benchmark is not a drawback (Sculley et al., 2018) as one would not know that the baseline performs well on this benchmark in advance, and using the baseline on all benchmarks would lead to clearly inferior results.

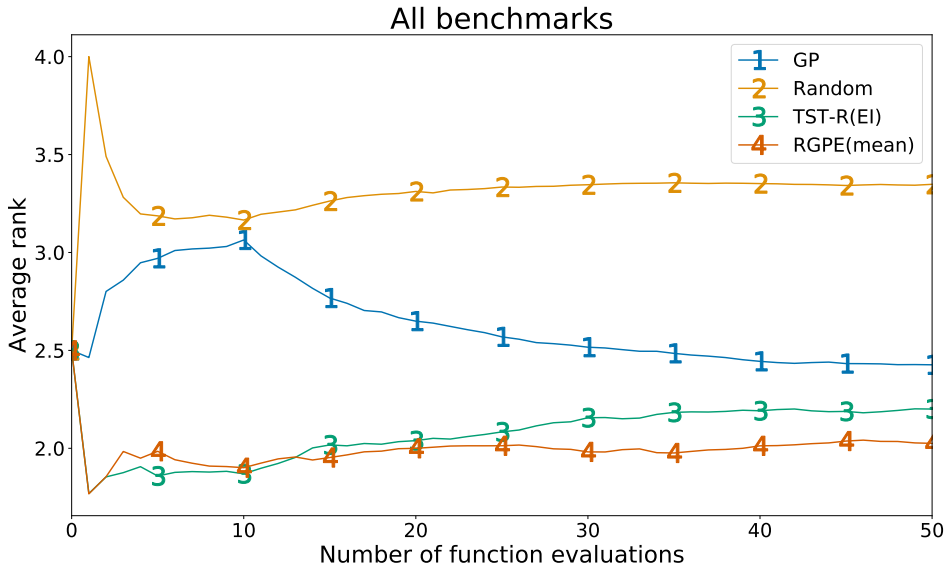
The search space pruning baselines (*Random+Box* and *GP(Box)*) show that shrinking the search space indeed improves performance. However, shrinking the search space does not simplify the search space and therefore does not improve the GP much on the $\text{NN}_{t,7D}$ benchmark. Nonetheless, there is still a performance improvement on this benchmark, and the reduced search space appears to be a good idea. We further find that search space pruning is consistently outperformed on the table-lookup benchmarks ($\text{AdaBoost}_{g,2D}$, $\text{SVM}_{g,6D}$ and $\text{NN}_{t,7D}$) while it performed competitively on the surrogate benchmarks. This hints at the two types of benchmarks (tabular vs. surrogates) having a different underlying structure, which is an interesting topic for further studies.

	10	20	30	40	50	10	20	30	40	50
	$NN_{t,7D}$					$XGBoost_{s,10D}$				
GP	16.40	11.51	9.69	8.62	7.84	5.31	2.36	1.46	1.22	1.09
GCP	16.40	11.61	9.80	8.56	7.76	5.31	2.75	2.10	1.77	1.50
Random(1x)	14.65	11.50	9.95	8.75	7.93	5.68	3.42	2.47	2.11	1.80
Random(2x)	11.50	8.75	7.11	6.13	5.42	3.42	2.11	1.62	1.53	1.36
Random(4x)	8.75	6.13	4.81	4.24	3.56	2.11	1.53	1.31	1.23	1.17
Random(10x)	5.42	3.56	**2.62**	**2.19**	**1.78**	1.36	1.17	1.00	0.92	0.88
Random+Box	12.73	9.78	8.38	7.23	6.51	3.92	2.60	1.62	1.53	1.43
GP(Box)	11.79	8.85	7.08	6.06	5.14	2.75	1.18	0.89	0.78	0.73
SMFO	**3.59**	**2.77**	**2.17**	**1.86**	**1.75**	1.25	1.09	0.94	0.91	0.87
WAC	7.44	6.68	5.98	5.79	5.51	1.88	1.85	1.84	1.75	1.75
KL-weighting	6.43	5.44	4.67	4.17	3.78	-	-	-	-	-
ABLR	7.88	6.60	6.02	5.45	5.03	2.29	1.88	1.75	1.61	1.58
GCP+Prior	5.93	4.30	3.28	2.74	2.38	1.37	**1.18**	**1.08**	**0.94**	**0.84**
TST-R(EI)	5.37	4.46	3.88	3.31	3.00	**1.14**	**0.88**	0.84	**0.79**	0.78
TST-R(TAF)	4.88	4.02	3.58	3.24	2.95	**0.98**	**0.82**	**0.73**	**0.69**	**0.64**
RGPE(TAF)	**3.60**	**2.54**	**2.09**	**1.70**	**1.55**	**0.97**	**0.80**	**0.70**	**0.67**	**0.67**
RGPE(MoGP)	**3.52**	**2.63**	2.34	2.07	1.93	1.55	1.16	0.89	0.83	0.80
RGPE(mean)	5.20	3.92	3.29	2.91	2.55	**1.12**	**0.79**	**0.68**	**0.67**	**0.65**
RGPE(NEI)	5.09	3.90	3.38	3.00	2.71	1.19	**0.98**	**0.87**	**0.73**	**0.64**

Table 5: Final results on the $NN_{t,7D}$ and $XGBoost_{s,10D}$ benchmarks. The numbers reported are the average normalized regret (Wistuba et al., 2018). We boldface the best value per benchmark and number of function evaluations and underline methods that are not significantly different according to a Wilcoxon signed-rank test with $\alpha = 0.05$ (Demšar, 2006). Top group: standard HPO baselines; middle group: transfer HPO baselines; **bottom group: our methods**.

Figure 1 provides a graphical representation of the results comparing our best method based on a single ensemble (RGPE(mean)) and our best method based on an improved acquisition function (RGPE(TAF)) against their respective baselines (TST-R(EI) and TST-R(TAF)). We present the baselines GP and random search together with TST-R(EI) and RGPE(mean) (top), and TST-R(TAF) and RGPE(TAF) (bottom) . We can observe all transfer HPO methods benefit from the learned initial design immediately, while the GP using the Latin hypercube initial design and random search cannot keep up. The ranks of the GP and random search appear to converge towards each other until iteration 10, where the GP's initial design is over and BO actually starts, and from that moment on the GP clearly outperforms random search. The transfer HPO methods maintain their lead over the single-task methods over the course of the 50 function evaluations. We can observe that our proposed methods outperform the respective competitors without requiring any hyperparameter optimization (in contrast to the methods based on TST-R).

Practical Transfer Learning for Bayesian Optimization



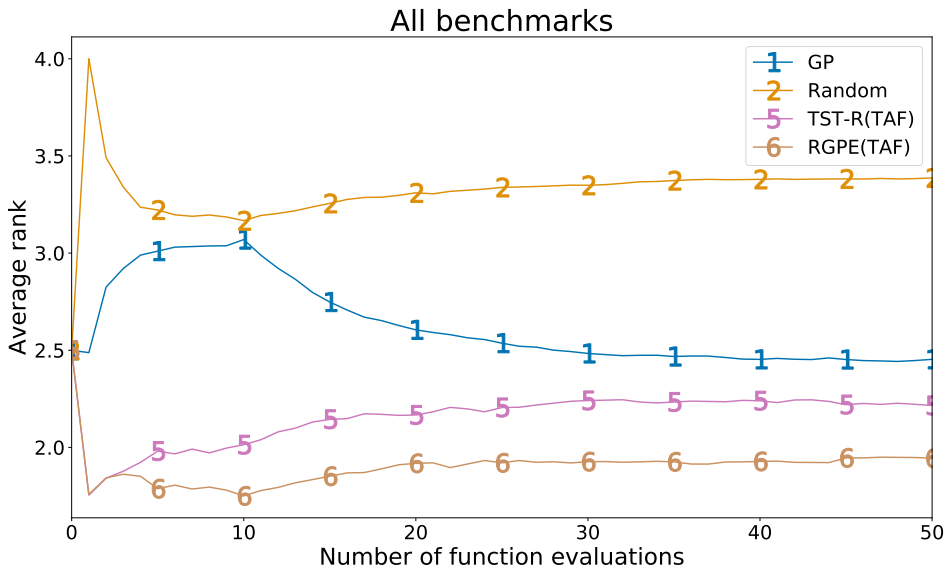


Figure 1: Ranking plots across all benchmarks (lower is better) comparing our methods (4,6) with small sets of baselines (1-3,5), as indicated in each panel legend. Our methods consistently outperformed baselines across all benchmarks.

	10	20	30	40	50	10	20	30	40	50
	$\text{AdaBoost}_{g,2D}$					$\text{SVM}_{g,6D}$				
GP	5.42	**2.26**	**1.26**	**0.82**	**0.66**	9.66	3.64	2.06	1.43	1.13
GCP	5.42	**2.26**	**1.18**	**0.92**	0.72	9.66	3.59	1.97	1.18	0.81
Random(1x)	6.27	3.62	2.53	1.72	1.32	11.52	6.44	5.07	4.07	3.24
Random+Box	5.43	3.61	2.37	1.79	1.45	10.64	6.09	4.26	3.37	2.89
GP(Box)	5.38	**2.31**	**1.27**	**0.81**	**0.72**	9.86	3.53	2.02	1.48	1.18
SMFO	**4.03**	2.76	1.96	1.52	1.11	4.30	2.63	1.97	1.37	1.21
WAC	5.97	4.11	3.01	2.18	1.42	8.49	6.02	3.67	2.63	1.90
KL-weighting	5.49	3.48	2.73	2.23	1.81					
ABLR	4.65	2.39	1.51	0.88	0.52	7.65	4.84	3.39	2.47	1.86
GCP+Prior	**5.37**	**3.38**	**1.92**	**1.25**	0.89	**4.23**	**2.59**	**2.02**	**1.85**	**1.61**
TST-R(EI)	4.75	**2.59**	**1.65**	0.94	**0.52**	3.69	2.07	1.36	0.89	**0.64**
TST-R(TAF)	**3.96**	**2.35**	**1.54**	1.14	**0.73**	3.89	2.24	1.39	0.78	**0.46**
RGPE(TAF)	**3.91**	**2.29**	**1.39**	**1.00**	**0.63**	**2.95**	**1.54**	0.91	**0.61**	**0.45**
RGPE(MoGP)	**4.01**	**2.26**	**1.47**	**0.95**	**0.76**	3.35	2.04	1.35	**0.92**	**0.58**
RGPE(mean)	4.71	**2.48**	**1.63**	**0.94**	**0.55**	**3.22**	**1.70**	**0.99**	**0.71**	**0.47**
RGPE(NEI)	4.55	**2.58**	1.76	1.09	**0.62**	**3.75**	**1.41**	**0.75**	**0.61**	**0.39**

Table 6: Final results on the $\text{AdaBoost}_{g,2D}$ and $\text{SVM}_{g,6D}$ benchmarks. The numbers reported are the average normalized regret (Wistuba et al., 2018). We boldface the best value per benchmark and number of function evaluations and underline methods that are not significantly different according to a Wilcoxon signed-rank test with $\alpha = 0.05$ (Demšar, 2006). Top group: standard HPO baselines; middle group: transfer HPO baselines; **bottom group: our methods**.

6.3 Evolution of Model Weights

Figure 2 shows how the ensemble weights evolved for the RGPE(NEI) method on the $\text{NN}_{t,7D}$ benchmark. The top panel depicts how the weight of the target model increases over the course of the 50 function evaluations and its relation to the weights of the base models. This plot shows the average of the weight over all 15 repetitions and 35 tasks. For the highest and 2nd highest weight of a base model, we identify it per time step before taking the average. We can see that the weight of the target model is initially close to zero before rapidly approaching a weight close to 1 by the final iteration. In early iterations the weight was spread across all of the base models, but with more data, weight quickly concentrated on a small number of base models, before finally concentrating on the target model. By iteration 11 the two most highly-weighted base models already had more weight than all of the other base models combined. On the bottom we can see the evolution of non-zero weights for the $\text{NN}_{t,7D}$ benchmark. For the first three iterations there are none, as the proposed weighting scheme requires three function evaluations before it can be used. As soon as the weighting kicks in, half of the weights are set to zero (in the median). The number of non-zero weights decreases monotonically and the median approaches one (all weight on the target model) at 28 function evaluations.

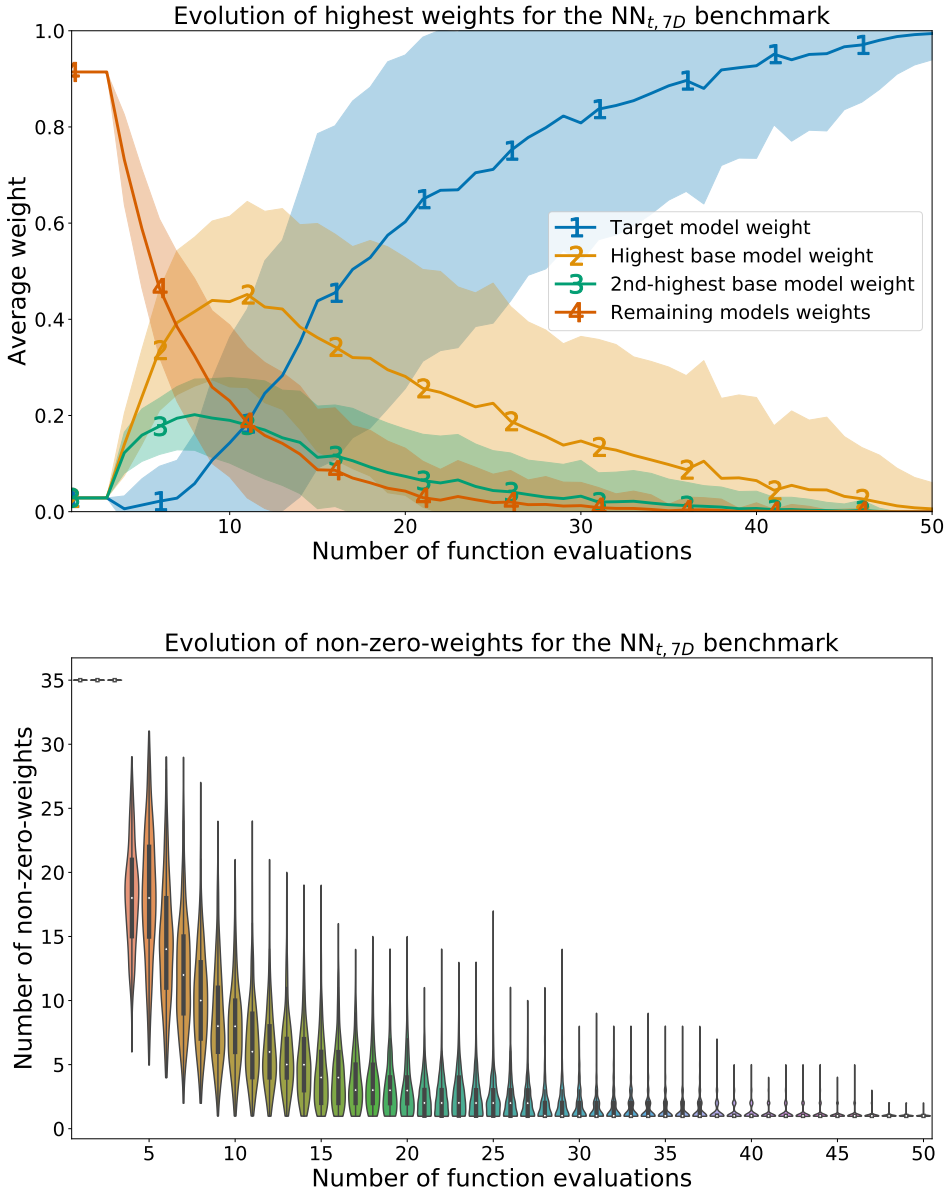


Figure 2: (Top) Evolution of the individual model weights over time for the RGPE(NEI) model. (Bottom) Evolution of the number of non-zero weights over time for the RGPE(NEI) model.

	$\text{GLMNET}_{s,2D}$	$\text{SVM}_{s,4D}$	$\text{NN}_{t,7D}$	$\text{XGBoost}_{s,10D}$	$\text{AdaBoost}_{g,2D}$	$\text{SVM}_{g,6D}$
RGPE(mean)	**0.23**	**0.26**	**2.55**	**0.65**	**0.55**	**0.47**
RGPE(EI)	**0.21**	**0.31**	**2.69**	**0.67**	**0.57**	**0.46**
RGPE(NEI)	**2.20**	**0.37**	**2.71**	**0.64**	**0.62**	**0.39**
RGPE(NEI,None)	**0.16**	**0.48**	3.27	**0.71**	**0.68**	**0.45**
RGPE(TAF)	0.16	**0.38**	**1.55**	**0.67**	**0.63**	**0.45**
RGPE(TAF,None)	**0.15**	**0.51**	**1.61**	**0.62**	0.71	**0.45**
RGPE(MoGP)	**0.16**	**0.30**	1.93	0.80	**0.76**	**0.58**
RGPE(MoGP,None)	**0.11**	**0.56**	1.96	0.75	**0.61**	0.58
RGPE(MoGP,$S = 100$)	**0.15**	0.55	2.00	**0.72**	**0.64**	**0.49**
RGPE(MoGP,$S = 1000$, default)	**0.16**	**0.30**	**1.93**	**0.80**	**0.76**	**0.58**
RGPE(MoGP,$S = 10000$)	**0.14**	**0.37**	**1.90**	**0.68**	0.58	**0.54**
RGPE(MoGP,$S = 100000$)	**0.17**	**0.31**	**1.83**	**0.73**	**0.40**	**0.50**

Table 7: Ablation study on the proposed methods. The numbers reported are the average normalized regret (Wistuba et al., 2018). We boldface the best value per benchmark and number of function evaluations and underline methods that are not significantly different according to a Wilcoxon signed-rank test with $\alpha = 0.05$ (Demšar, 2006); separately for RGPE(NEI), RGPE(TAF) and RGPE(MoGP) with different values of S.

6.4 Ablation Study

In this second part of the experimental validation we examine the effect of the proposed building blocks of our method on the performance. We conduct an ablation study and sensitivity analaysis for the TST-R(EI) and TST-R(TAF) baselines in Appendix D.4.

In the methodology section we have introduced the RGPE model and how it can be used for transfer HPO. Because the weighting mechanism contains a regularization mechanism that can be turned off, we conduct an ablation study about it. Moreover, it contains a hyperparameter (the number of bootstrap samples) of which we study the sensitivity. We present all results in Table 7. Furthermore, we provide an overview of how the method names and the exact contributions discussed in Section 4 correspond in Table 8.

First, we check whether using the weight dilution prevention is a useful addition, and we can compare RGPE(NEI) and RGPE(NEI,None), RGPE(TAF) and RGPE(TAF,None), and RGPE(MoGP) and RGPE(MoGP)(None). We can see that there is no large difference between using and not using the weight dilution mechanism, and a slight trend toward the methods with weight dilution prevention achieving lower regret values. However, there are two outliers. RGPE(NEI) with the weight dilution prevention has a drastically worse performance than RGPE(NEI,None) on $\text{GLMNET}_{s,2D}$, but this is not significant and caused by a single dataset. On the other hand, RGPE(NEI,None) is significantly worse than RGPE(NEI) on $\text{NN}_{t,7D}$. Therefore, we can conclude that using the weight dilution

Method name	4.1	4.3	Acquisition function
TST-R(EI)			EI
TST-R(EI,wd), see Appendix D.4		✓	EI
RGPE(NEI)	✓	✓	NoisyEI
RGPE(NEI,None)	✓		NoisyEI
RGPE(mean)	✓	✓	EI
RGPE(EI)	✓	✓	EI
TST-R(TAF)			TAF
TST-R(TAF,wd), see Appendix D.4		✓	TAF
RGPE(TAF)	✓	✓	TAF
RGPE(TAF,None)	✓		TAF
RGPE(MoGP)	✓	✓	Weighted sum of EIs
RGPE(MoGP)(None)	✓		Weighted sum of EIs

Table 8: Overview of how our proposed methods and their ablations compare against the baselines TST-R(EI) (line 1) and TST-R(TAF) (line 6).

prevention, which is also central to our proof that the proposed methods are in the worst case only a multiplicative factor slower than vanilla Bayesian optimization and improves optimization speed by more aggressively dropping base models, is a valuable contribution for methods combining multiple models into an ensemble.

Second, we analyze whether the methods that make use of the predictive variance in the base models have an edge over those that do not. This time we compare RGPE(NEI) and RGPE(mean), and RGPE(MoGP) and RGPE(TAF). Surprisingly, even though we have shown in a principled manner why the variance of the base models should matter, ignoring the variance of the base models as done by RGPE(mean) and RGPE(TAF) gives better results. While the difference in results for RGPE(mean) and RGPE(NEI) are not statistically significant, RGPE(mean) has lower regret on four out of six benchmarks and the lowest regret among the four compared methods on three out of six benchmarks. The situation for RGPE(MoGP) and RGPE(TAF) is slightly more nuanced, but similar overall. Our best explanations for this surprising result are 1) that predictive variance in base models can be misleading and increase the supposed variance of the combined model in regions where the target model is already certain, thereby misleading Bayesian optimization; and 2) that as there is no variance around already evaluated points in base models, these points (including the optima on the base models) appear less promising after combining all base models, repelling the transfer HPO methods from such locations.

Third, we discuss whether the special treatment of noise in the RGPE(NEI) model is actually necessary or whether plain expected improvement would work, too (RGPE(EI), line 2 of Table 7). It turns out that there is no statistically significant degradation of the performance. We can currently not explain why there is no significant difference in the way we treat the uncertainty of the base model and suggest checking this finding when using the RGPE model with a different GP library.

Fourth, we have a look at using a different number of bootstrap samples than the default 1000, and also test 100, 10000 and 100000 bootstrap samples. Using as many as 100000 appears to perform consistently better or equivalent to using only 1000 samples. Contrarily, using only 100 samples results in significantly worse performance in two out of six cases. Therefore, we do not see a reason to tune the number of bootstrap samples but rather use the maximum number of samples that can be afforded. As a concrete example, it takes less than roughly 1.6 seconds to compute the weights with 1000 bootstrap samples after 50 observations on a $2d$ problem, while it takes 140 seconds for 100000 bootstrap samples. This increase in computational requirements would not have been feasible for the large experimental study we conducted, but can be used in practical settings.

Overall, we find our proposed methods to be very robust. We see some advantage to ignoring the predictive variance of the base models and using the weight dilution prevention. Based on the ablation study and the overall results in the previous section, our clear recommendation is to use RGPE(TAF) for transfer HPO.

7. Related Work

This section extends the primer on related work in Section 2.3 by discussing techniques of types 1-5 and 7-9 which are not required as background for our newly developed methods.

1. A single model that is trained on all tasks simultaneously. Several past methods have used manually defined meta-features to measure task similarity (Brazdil et al., 1994) and then adapted GP-based Bayesian optimization (Bardenet et al., 2013; Yogatama and Mann, 2014; Schilling et al., 2016). Besides the drawback of requiring additional hand-designed features (which violates desideratum #3), these methods share the issue that meta-features are non-adaptive throughout the optimization process and do not update task similarity based on the new observations (Leite et al., 2012; Wistuba et al., 2018).

Another set of methods attempts to learn the task similarity without the use of meta-features in order to fit a joint model. Swersky et al. (2013) and Poloczek et al. (2016) propose different kernels to jointly model all past runs and the current task. However, multitask GPs suffer from the same poor scaling as putting all observations into a single GP (violating desideratum #1) and cannot be used for the benchmarks we use in the main paper. Furthermore, Swersky et al. (2013) sample a $t \times t$ lower triangular matrix describing task correlations, which prohibits a large number of past runs as observed by Klein et al. (2017). We demonstrate this scaling issue in Appendix B where one can see that the fitting time for the multitask GP with only five tasks is already higher than that of RGPE for 50 tasks, and that fitting with 15 tasks already requires more than 1000 seconds.

Such scaling issues can be alleviated by using neural network models. One option is to learn a task embedding for every task inside a Bayesian neural network (Springenberg et al., 2016) and another option is to fine-tune a pre-trained deep kernel surrogate on the target task (Wistuba and Grabocka, 2021). Two recent works (Perrone et al., 2018; Law et al., 2019) circumvent both the scaling issue as well as the issue of explicitly learning task correlations by using a neural network as a feature extractor for tasks and hyperparameters and Bayesian linear regression as a scalable probabilistic output layer of the neural network. We compared to *adaptive basis function linear regression* (Perrone et al., 2018, ABLR) in Section 6.

When the hyperparameter search space is discrete it is also possible to use methods based on a matrix factorization model (Fusi et al., 2018; Yang et al., 2019, 2020). However, these methods require that there are overlapping observations between the datasets and were also only tested in settings where the metadata matrix (i.e. results of pipelines on all datasets) has between 10% of the entries (Fusi et al., 2018) or is even fully populated. Such approaches are related to algorithm selection, where the goal is to select a single algorithm from a discrete set without observing any feedback on the target task (Kerschke et al., 2019), see also the method we discuss in Appendix A.

2. A learned kernel that is trained on the base tasks and applied to the target task. As an alternative solution, Wang et al. (2018b) suggest to estimate the Gaussian process prior from offline data via empirical Bayes. While the learned kernel leads to performance improvements compared to a standard RBF prior, the size of the meta dataset required for the method to perform en-par with the RBF prior with optimized hyperparameters is between 2916 and 20000, which is above what we will have available during the experimental evaluation.

3. Learning an adaptation of each base task to the current task A number of papers propose to learn an adaptation of each past BO run to the current BO run. Schilling et al. (2016) learn a joint model for each past run and the target task and combine those models using meta-features as similarity descriptors in a Product of Gaussian Process Experts model. Shilton et al. (2017) model the difference between the past run and the target task with a GP, which is then used to adjust the past run observations for inclusion in the current model. Such approaches are not scalable as they require re-fitting a model for each meta-task in each iteration of the optimization process (violating desideratum #2). Ramachandran et al. (2018) then suggest a bandit method to select a base task to update within this transfer setting. Due to the necessary exploration this method is only applicable in the setting where the number of base tasks is substantially lower than the number of observations allowed on the target task (violating desideratum #1). Golovin et al. (2017) take a similar approach for an ordered set of past runs. Rather than fitting a GP to each run separately, a GP is fit to the residuals of each run relative to the predictions of the previous model in the stack. However, this method assumes an ordering of the runs, which would not be the case for transferring information from a collection of unrelated problems.

4. An initial design learned from previous optimization runs Orthogonal to optimization is learning the initial design based on past observations, which can be done both with meta-features (Reif et al., 2012; Gomes et al., 2012; Feurer et al., 2015b) and without (Wistuba et al., 2015a; Pfisterer et al., 2021). Using an adapted initial design is complementary to our strategy and would benefit our proposed method as well. We compared against an initial design-only baseline, also called *sequential model-free optimization* (Xu et al., 2010, 2011; Wistuba et al., 2015a; Pfisterer et al., 2021, SMFO), in Section 6.

5. Reducing the design space based on previous optimization runs Yet another orthogonal strategy is to use past optimization runs to shrink the considered search space to only contain promising areas from past runs (Wistuba et al., 2015b; Probst et al., 2019; Perrone and Shen, 2019). However, most of these strategies introduce at least one new hyperparameter (violating desideratum #4). The only exception is given by Perrone and

Shen (2019), which also introduces a hyperparameter-free method to reduce the search space to a box which only contains all prior observed optima. We compared to this technique in Section 6 (Random+Box,GP(Box)).

7. Linear combination of models In the related field of algorithm configuration (Hutter, 2009), Lindauer and Hutter (2018) use stochastic gradient descent to learn a weighted combination of random forests as surrogates for a runtime optimization task. The mean predictions are combined linearly as in TST-R (Wistuba et al., 2016), and the variances are combined according to $\bar{\sigma}^2(\mathbf{x}_*) = \sum_{i=1}^{t} w_i \sigma_i^2(\mathbf{x}_*)$. While the usage of a simple stacking approach is appealing, it introduces a new model selection problem to find a stacking regressor (violating desideratum #3). Without proper tuning, obtaining a sparse solution (sparse in terms of taking only relevant base tasks into account while ignoring unrelated ones) is not possible. Furthermore, minimizing the least squares error is not necessarily a good criterion when our goal is to obtain a good model for Bayesian optimization. We compared against this stacking approach abbreviated *WAC* in Section 6.

Another line of work extends the Predictive Entropy Search (PES) acquisition function to a multi-task Bayesian optimization algorithm in which the distribution over the minimum is a weighted linear combination of the distribution of the minimum on different tasks (Ramachandran et al., 2019b). The linear weighting is computed as $\exp\left(-\frac{KL(p_{opt}^{base}(\mathbf{x})||p_{opt}^{target}(\mathbf{x}))}{\eta}\right)$, where η is a hyperparameter and KL the Kullback-Leibler divergence estimated with a nearest-neighbor approach Perez-Cruz (2008). Later, they also proposed a modification that takes uncertainty on the source task into account at the cost of yet an additional hyperparameter (Ramachandran et al., 2019a). While here, we restrict ourselves to EI-based acquisition functions, we note that our proposed weighting scheme could also be used as a hyperparameter-free alternative in PES.

8. Hybrid methods While the ideas above can be categorized rather easily, it is also possible to define models and techniques that are combinations of the methods described above. One such method is to fit a joint neural network to all base tasks without taking the tasks into account for the network architecture to learn a global prior. To take the target task into account, this method uses a GP that only learns the differences of the current task to the global prior. To ensure all tasks live on the same scale, the data is scaled using copula transformations. As this changes the predictive distribution of the GP, they also derive a custom, closed-form solution for EI tailored to this model. We compare against this method *Copula Gaussian processes with prior*, or short GCP+Prior (Salinas et al., 2020), in the experimental section. Another combination involves meta-features and a learning a data-dependent default (Gijsbers et al., 2021).

9. Prior-based methods A transfer that we did not discussed yet is inferring a prior from previous task and applying to existing methods. An obvious choice is random search, where van Rijn and Hutter (2018) replace the uniform prior with a prior learned from hyperparameter optima on other datasets. An interesting line of work also aims to allow the users to manually specify their prior to guide BO (Souza et al., 2021; Anonymous, 2021).

Scalable models for standard HPO Bayesian optimization has also been done with models that scale better with the number of observations, such as random forests (Hutter et al., 2011), Parzen estimators (Bergstra et al., 2011) and neural networks (Snoek et al., 2015;

Springenberg et al., 2016). These models come with challenges of their own, such as poor uncertainty extrapolation with limited observations and hyperparameter sensitivity. There are also extensions of GPs for large datasets, most notably sparse GPs (Csató and Opper, 2002). However, sparse GPs can also have poor uncertainty extrapolation (Bauer et al., 2016; Wang et al., 2018a) and were shown to perform less well for Bayesian optimization than standard GPs (McIntire et al., 2016). Recently, there has been interest in scaling GPs by only optimizing in subspaces (Eriksson et al., 2019), however, it is unclear how to combine such approaches with transfer HPO. Therefore, GPs remain the standard model for practical Bayesian optimization, especially since most methods to jointly model several tasks require the definition of meta-features or expensive calculation of task similarities.

8. Discussion and Conclusion

Our goal was to have a transfer hyperparameter optimization method that scales beyond standard GPs, is hyperparameter free, does not require meta-features, and guarantees to not perform substantially worse than standard BO. We showed that a linear combination of GPs, weighted by their posterior probability of describing the observations, and probabilistically dropping underperforming models, gives rise to several novel methods to fulfill these four desiderata. Our experiments on six hyperparameter optimization benchmarks showed that the proposed methods performed better than alternatives from the literature, and they are scalable and effective methods for conducting transfer learning in GP-based BO. Because of the weight dilution prevention we also obtain a performance guarantee in terms of simple regret. In particular, using both our new RGPE weighting scheme and our novel weight dilution prevention mechanism together with the transfer acquisition function leads to very stable performance, and we recommend this RGPE(TAF) combination as the new default algorithm for transfer HPO.

A side-effect of our study is the better understanding of the previous transfer HPO methods TST-R (Wistuba et al., 2016) and TAF (Wistuba et al., 2018), which we showed are special cases of the linear combination of Gaussian processes proposed in this work.

Furthermore, we have made available the largest collection of transfer HPO methods and benchmarks in a single framework to date with a total of six benchmarks and 9 competitor methods from the literature. This will allow future researchers to more easily conduct research on transfer HPO by having a set of benchmarking problems and baselines to compare against readily available.

Finally, we acknowledge limitations of the proposed method which give directions for future work. First, they are limited to predictive models and cannot be used with models like the *Tree Parzen Estimator* (Bergstra et al., 2011). Second, our evaluations are limited to spaces that can be modeled well by regular Gaussian processes, which does not include the complex spaces of, e.g., Auto-sklearn (Feurer et al., 2015a), Auto-WEKA (Thornton et al., 2013), or Auto-PyTorch (Zimmer et al., 2021). As the only requirement for the model is being a probabilistic model, we expect that using random forests or neural networks for these is a promising direction. Third, all methods introduced hinge on the underlying BO library as they converge to vanilla BO. If the BO library cannot provide excellent performance by itself, putting transfer learning on top of it does not guarantee that it would outperform a well-tuned BO tool. Finally, it would be interesting to extend the proposed methods to

other acquisition functions to explicitly handle noise (Letham et al., 2019) or work in a multi-fidelity setting (Wu et al., 2020).

Acknowledgements

Thanks to Till Varoquaux for support in developing the method, Sam Daulton for help with the implementation, Martin Wistuba for help with the TST-R and TAF baselines, and Rodolphe Jenatton and Valerio Perrone for help with the ABLR baseline. We thank Josif Grabocka and the anonymous reviewers of previous versions for helpful feedback on the manuscript, and Carl Hvafner and Luigi Nardi for feedback on the theory. The authors acknowledge support by the state of Baden-Württemberg through bwHPC and the German Research Foundation (DFG) through grant no INST 39/963-1 FUGG (bwForCluster NEMO).

References

Anonymous. πBO: Augmenting acquisition functions with user beliefs for bayesian optimization. In *Under review at ICLR 2022: https://openreview.net/forum?id=MMAeCXIa89*, 2021.

M. Balandat, B. Karrer, D. Jiang, S. Daulton, B. Letham, A. Wilson, and E. Bakshy. BoTorch: A Framework for Efficient Monte-Carlo Bayesian Optimization. In H. Larochelle, M. Ranzato, R. Hadsell, M.-F. Balcan, and H. Lin, editors, *Proceedings of the 33rd International Conference on Advances in Neural Information Processing Systems (NeurIPS'20)*. Curran Associates, 2020.

R. Bardenet, M. Brendel, B. Kégl, and M. Sebag. Collaborative hyperparameter tuning. In S. Dasgupta and D. McAllester, editors, *Proceedings of the 30th International Conference on Machine Learning (ICML'13)*, pages 199–207. Omnipress, 2013.

M. Bauer, M. van der Wilk, and C. Rasmussen. Understanding probabilistic sparse Gaussian process approximations. In D. Lee, M. Sugiyama, U. von Luxburg, I. Guyon, and R. Garnett, editors, *Proceedings of the 29th International Conference on Advances in Neural Information Processing Systems (NeurIPS'16)*, pages 1533–1541. Curran Associates, 2016.

J. Bergstra and Y. Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13:281–305, 2012.

J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl. Algorithms for hyper-parameter optimization. In J. Shawe-Taylor, R. Zemel, P. Bartlett, F. Pereira, and K. Weinberger, editors, *Proceedings of the 24th International Conference on Advances in Neural Information Processing Systems (NeurIPS'11)*, pages 2546–2554. Curran Associates, 2011.

B. Bischl, G. Casalicchio, M. Feurer, F. Hutter, M. Lang, R. Mantovani, J. N. van Rijn, and J. Vanschoren. Openml benchmarking suites and the openml100. *arXiv:1708.03731v1*, 2019.

P. Brazdil, J. Gama, and B. Henery. Characterizing the applicability of classification algorithms using meta-level learning. In F. Bergadano and L. De Raedt, editors, *Machine Learning: ECML-94*, Lecture Notes in Artificial Intelligence, pages 83–102. Springer, 1994.

E. Brochu, V. Cora, and N. de Freitas. A tutorial on Bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning. *arXiv:1012.2599v1 [cs.LG]*, 2010.

A. Bull. Convergence rates of efficient global optimization algorithms. *Journal of Machine Learning Research*, 12:2879–2904, 2011.

L. Csató and M. Opper. Sparse on-line Gaussian processes. *Neural Computation*, 14(3): 641–668, 2002.

J. Demšar. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research*, 7:1–30, 2006.

K. Eggensperger, F. Hutter, H. Hoos, and K. Leyton-Brown. Efficient benchmarking of hyperparameter optimizers via surrogates. In B. Bonet and S. Koenig, editors, *Proceedings of the Twenty-ninth National Conference on Artificial Intelligence (AAAI'15)*, pages 1114–1120. AAAI Press, 2015.

D. Eriksson, M. Pearce, J. Gardner, R. Turner, and M. Poloczek. Scalable global optimization via local bayesian optimization. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Proceedings of the 32nd International Conference on Advances in Neural Information Processing Systems (NeurIPS'19)*. Curran Associates, 2019.

M. Feurer and F. Hutter. Hyperparameter optimization. In F. Hutter, L. Kotthoff, and J. Vanschoren, editors, *Automated Machine Learning: Methods, Systems, Challenges*, volume 5 of *The Springer Series on Challenges in Machine Learning*, chapter 1, pages 3–38. Springer, 2019. Available for free at http://automl.org/book.

M. Feurer, A. Klein, K. Eggensperger, J. Springenberg, M. Blum, and F. Hutter. Efficient and robust automated machine learning. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, editors, *Proceedings of the 28th International Conference on Advances in Neural Information Processing Systems (NeurIPS'15)*, pages 2962–2970. Curran Associates, 2015a.

M. Feurer, J. Springenberg, and F. Hutter. Initializing Bayesian hyperparameter optimization via meta-learning. In B. Bonet and S. Koenig, editors, *Proceedings of the Twenty-ninth National Conference on Artificial Intelligence (AAAI'15)*, pages 1128–1135. AAAI Press, 2015b.

M. Feurer, B. Letham, and E. Bakshy. Scalable meta-learning for Bayesian optimization. *arXiv:1802.02219v1 [stat.ML]*, 2018.

M. Feurer, B. Letham, F. Hutter, and E. Bakshy. Practical transfer learning for Bayesian optimization. *arXiv:1802.02219v2 [stat.ML]*, 2021.

A. Forrester, A. Sobester, and A. Keane. *Engineering design via surrogate modelling: a practical guide*. John Wiley & sons, 2008.

P. Frazier. A tutorial on Bayesian optimization. *arXiv:1807.02811 [stat.ML]*, 2018.

N. Fusi, R. Sheth, and M. Elibol. Probabilistic Matrix Factorization for Automated Machine Learning. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Proceedings of the 31st International Conference on Advances in Neural Information Processing Systems (NeurIPS'18)*, pages 3348–3357. Curran Associates, 2018.

R. Garnett. *Bayesian Optimization*. Cambridge University Press, 2022. in preparation.

P. Gijsbers, E. LeDell, S. Poirier, J. Thomas, B. Bischl, and J. Vanschoren. An open source automl benchmark. In K. Eggensperger, M. Feurer, F. Hutter, and J. Vanschoren, editors, *ICML workshop on Automated Machine Learning (AutoML workshop 2019)*, 2019.

P. Gijsbers, F. Pfisterer, J. van Rijn, B. Bischl, and J. Vanschoren. Meta-learning for symbolic hyperparameter defaults. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, page 151–152. Association for Computing Machinery, 2021.

D. Golovin, B. Solnik, S. Moitra, G. Kochanski, J. Karro, and D. Sculley. Google Vizier: A service for black-box optimization. In S. Matwin, S. Yu, and F. Farooq, editors, *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 1487–1495. ACM Press, 2017.

T.A.F. Gomes, R.B.C. Prudêncio, C. Soares, A. Rossi, and A. Carvalho. Combining meta-learning and search techniques to select parameters for support vector machines. *Neurocomputing*, 75(1):3–12, 2012.

F. Hutter. *Automated Configuration of Algorithms for Solving Hard Computational Problems*. PhD thesis, University of British Columbia, Department of Computer Science, Vancouver, Canada, 2009.

F. Hutter, H. Hoos, and K. Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In C. Coello, editor, *Proceedings of the Fifth International Conference on Learning and Intelligent Optimization (LION'11)*, volume 6683 of *Lecture Notes in Computer Science*, pages 507–523. Springer, 2011.

D. Jones, M. Schonlau, and W. Welch. Efficient global optimization of expensive black box functions. *Journal of Global Optimization*, 13:455–492, 1998.

P. Kerschke, H. Hoos, F. Neumann, and H. Trautmann. Automated algorithm selection: Survey and perspectives. *Evolutionary Computation*, 27(1):3–45, 2019.

A. Klein, S. Falkner, S. Bartels, P. Hennig, and F. Hutter. Fast Bayesian optimization of machine learning hyperparameters on large datasets. In A. Singh and J. Zhu, editors, *Proceedings of the Seventeenth International Conference on Artificial Intelligence and Statistics (AISTATS)*, volume 54. Proceedings of Machine Learning Research, 2017.

D. Kühn, P. Probst, J. Thomas, and B. Bischl. Automatic exploration of machine learning experiments on openml. *arXiv:1806.10961 [stats.ML]*, 2018.

A. Lacoste, M. Marchand, F. Laviolette, and H. Larochelle. Agnostic Bayesian learning of ensembles. In E. Xing and T. Jebara, editors, *Proceedings of the 31th International Conference on Machine Learning, (ICML'14)*, pages 611–619. Omnipress, 2014.

H. Law, P. Zhao, L. Chan, J. Huang, and D. Sejdinovic. Hyperparameter learning via distributional transfer. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Proceedings of the 32nd International Conference on Advances in Neural Information Processing Systems (NeurIPS'19)*, pages 6804–6815. Curran Associates, 2019.

R. Leite, P. Brazdil, and J. Vanschoren. Selecting classification algorithms with active testing. In P. Perner, editor, *Machine Learning and Data Mining in Pattern Recognition*, volume 7376 of *Lecture Notes in Computer Science*, pages 117–131. Springer, 2012.

B. Letham, K. Brian, G. Ottoni, and E. Bakshy. Constrained Bayesian optimization with noisy experiments. *Bayesian Analysis*, 14:495–519, 2019.

M. Lindauer and F. Hutter. Warmstarting of model-based algorithm configuration. In S. McIlraith and K. Weinberger, editors, *Proceedings of the Conference on Artificial Intelligence (AAAI'18)*, pages 1355–1362. AAAI Press, 2018.

M. Lindauer, K. Eggensperger, M. Feurer, A. Biedenkapp, D Deng, C. Benjamins, R. Sass, and F. Hutter. Smac3: A versatile bayesian optimization package for hyperparameter optimization, 2021. Accepted for publication.

J. Lévesque, A. Durand, C. Gagné, and R. Sabourin. Bayesian optimization for conditional hyperparameter spaces. In B. Howell, editor, *2017 International Joint Conference on Neural Networks (IJCNN)*, pages 286–293. IEEE Computer Society Press, 2017.

M. McIntire, D. Ratner, and S. Ermon. Sparse Gaussian Processes for Bayesian Optimization. In A. Ihler and D. Janzing, editors, *Proceedings of the 32nd conference on Uncertainty in Artificial Intelligence (UAI'16)*, pages 517–526. AUAI Press, 2016.

D. Pardoe and P. Stone. Boosting for regression transfer. In J. Fürnkranz and T. Joachims, editors, *Proceedings of the 27th International Conference on Machine Learning (ICML'10)*, pages 863–870. Omnipress, 2010.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

F. Perez-Cruz. Kullback-Leibler divergence estimation of continuous distributions. In *2008 IEEE International Symposium on Information Theory*, pages 1666–1670, 2008.

V. Perrone and H. Shen. Learning search spaces for Bayesian optimization: Another view of hyperparameter transfer learning. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Proceedings of the 32nd International Conference on Advances in Neural Information Processing Systems (NeurIPS'19)*, pages 12751–12761. Curran Associates, 2019.

V. Perrone, R. Jenatton, M. Seeger, and C. Archambeau. Scalable hyperparameter transfer learning. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Proceedings of the 31st International Conference on Advances in Neural Information Processing Systems (NeurIPS'18)*, pages 12751–12761. Curran Associates, 2018.

F. Pfisterer, J. van Rijn, P. Probst, A. Müller, and B. Bischl. Learning multiple defaults for machine learning algorithms. In F. Chicano, editor, *GECCO '21: Proceedings of the Genetic and Evolutionary Computation Conference Companion*, page 241–242. ACM Press, 2021.

M. Poloczek, J. Wang, and P. Frazier. Warm starting Bayesian optimization. In *Proceedings of the 2016 Winter Simulation Conference*, pages 770—-781. IEEE Computer Society Press, 2016.

P. Probst, A. Boulesteix, and B. Bischl. Tunability: Importance of hyperparameters of machine learning algorithms. *Journal of Machine Learning Research*, 20(53):1–32, 2019.

A. Ramachandran, S. Gupta, S. Rana, and S. Venkatesh. Selecting optimal source for transfer learning in Bayesian optimisation. In *PRICAI 2018: Trends in Artificial Intelligence*, pages 42–56. Springer, 2018.

A. Ramachandran, S. Gupta, S. Rana, and S. Venkatesh. Information-theoretic multi-task learning framework for Bayesian optimisation. In *AI 2019: Advances in Artificial Intelligence*, pages 497–509. Springer, 2019a.

A. Ramachandran, S. Gupta, S. Rana, and S. Venkatesh. Information-theoretic transfer learning framework for Bayesian optimisation. In M. Berlingerio, F. Bonchi, T. Gärtner, N. Hurley, and G. Ifrim, editors, *Machine Learning and Knowledge Discovery in Databases (ECML/PKDD'19)*, Lecture Notes in Computer Science, pages 827–842. Springer, 2019b.

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

B. Recht and K. Jamieson. The news on auto-tuning. `4http://www.argmin.net/2016/06/20/hypertuning/`, 2016. Accessed: 2020-02-06.

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

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

D. Salinas, H. Shen, and V. Perrone. A quantile-based approach for hyperparameter transfer learning. In H. Daume III and A. Singh, editors, *Proceedings of the 37th International Conference on Machine Learning (ICML'20)*, volume 98, pages 8438–8448. Proceedings of Machine Learning Research, 2020.

N. Schilling, M. Wistuba, L. Drumond, and L. Schmidt-Thieme. Hyperparameter optimization with factorized multilayer perceptrons. In A. Appice, P. Rodrigues, V. Costa, J. Gama, A. Jorge, and C. Soares, editors, *Machine Learning and Knowledge Discovery in Databases (ECML/PKDD'15)*, volume 9285 of *Lecture Notes in Computer Science*, pages 87–103. Springer, 2015.

N. Schilling, M. Wistuba, and L. Schmidt-Thieme. Scalable hyperparameter optimization with products of Gaussian process experts. In F. Paolo, N. Landwehr, G. Manco, and J. Vreeken, editors, *Machine Learning and Knowledge Discovery in Databases (ECML/PKDD'16)*, Lecture Notes in Computer Science, pages 33–48. Springer, 2016.

D. Sculley, J. Snoek, A. Wiltschko, and A. Rahimi. Winner's curse? on pace, progress, and empirical rigor. In *International Conference on Learning Representations Workshop track*, 2018. Published online: `iclr.cc`.

B. Shahriari, K. Swersky, Z. Wang, R. Adams, and N. de Freitas. Taking the human out of the loop: A review of Bayesian optimization. *Proceedings of the IEEE*, 104(1):148–175, 2016.

A. Shilton, S. Gupta, S. Rana, and S. Venkatesh. Regret bounds for transfer learning in Bayesian optimisation. In A. Singh and J. Zhu, editors, *Proceedings of the Seventeenth International Conference on Artificial Intelligence and Statistics (AISTATS)*, volume 54, pages 307–315. Proceedings of Machine Learning Research, 2017.

J. Snoek, H. Larochelle, and R. Adams. Practical Bayesian optimization of machine learning algorithms. In P. Bartlett, F. Pereira, C. Burges, L. Bottou, and K. Weinberger, editors, *Proceedings of the 25th International Conference on Advances in Neural Information Processing Systems (NeurIPS'12)*, pages 2960–2968. Curran Associates, 2012.

J. Snoek, O. Rippel, K. Swersky, R. Kiros, N. Satish, N. Sundaram, M. Patwary, Prabhat, and R. Adams. Scalable Bayesian optimization using deep neural networks. In F. Bach and D. Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning (ICML'15)*, volume 37, pages 2171–2180. Omnipress, 2015.

A. Souza, L. Nardi, L. Oliveira, K. Olukotun, M. Lindauer, and F. Hutter. Bayesian optimization with a prior for the optimum. In N. Oliver, F. Pérez-Cruz, S. Kramer, J. Read, and J.. Lozano, editors, *Machine Learning and Knowledge Discovery in Databases. Research Track (ECML/PKDD'21)*, volume 12977 of *Lecture Notes in Computer Science*, pages 265–296. Springer, 2021.

J. Springenberg, A. Klein, S. Falkner, and F. Hutter. Bayesian optimization with robust Bayesian neural networks. In D. Lee, M. Sugiyama, U. von Luxburg, I. Guyon, and R. Garnett, editors, *Proceedings of the 29th International Conference on Advances in Neural Information Processing Systems (NeurIPS'16)*. Curran Associates, 2016.

K. Swersky, J. Snoek, and R. Adams. Multi-task Bayesian optimization. In C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Weinberger, editors, *Proceedings of the 26th International Conference on Advances in Neural Information Processing Systems (NeurIPS'13)*, pages 2004–2012. Curran Associates, 2013.

C. Thornton, F. Hutter, H. Hoos, and K. Leyton-Brown. Auto-WEKA: combined selection and hyperparameter optimization of classification algorithms. In I. Dhillon, Y. Koren, R. Ghani, T. Senator, P. Bradley, R. Parekh, J. He, R. Grossman, and R. Uthurusamy, editors, *The 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'13)*, pages 847–855. ACM Press, 2013.

V. Tresp. Mixtures of Gaussian processes. In T. Leen, T. Dietterich, and V. Tresp, editors, *Proceedings of the 14th International Conference on Advances in Neural Information Processing Systems (NeurIPS'01)*, pages 654–660. MIT Press, 2001.

J. van Rijn and F. Hutter. Hyperparameter importance across datasets. In Y. Guo and F.Farooq, editors, *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 2367–2376. ACM Press, 2018.

J. Vanschoren. Meta-learning. In F. Hutter, L. Kotthoff, and J. Vanschoren, editors, *Automated Machine Learning: Methods, Systems, Challenges*, volume 5 of *The Springer Series on Challenges in Machine Learning*, chapter 2, pages 35–61. Springer, 2019. Available for free at http://automl.org/book.

J. Vanschoren, J. van Rijn, B. Bischl, and L. Torgo. OpenML: Networked science in machine learning. *SIGKDD Explor. Newsl.*, 15(2):49–60, 2014.

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

Z. Wang, C. Li, S. Jegelka, and P. Kohli. Batched High-dimensional Bayesian Optimization via Structural Kernel Learning. In D. Precup and Y. Teh, editors, *Proceedings of the 34th International Conference on Machine Learning (ICML'17)*, volume 70, pages 3656–3664. Proceedings of Machine Learning Research, 2017.

Z. Wang, C. Gehring, P. Kohli, and S. Jegelka. Batched Large-scale Bayesian Optimization in High-dimensional Spaces. In A. Storkey and F Perez-Cruz, editors, *Proceedings of the 21st International Conference on Artificial Intelligence and Statistics (AISTATS)*, volume 84, pages 745–754. Proceedings of Machine Learning Research, 2018a.

Z. Wang, B. Kim, and L. Kaelbling. Regret bounds for meta bayesian optimization with an unknown gaussian process prior. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Proceedings of the 31st International Conference*

on Advances in Neural Information Processing Systems (NeurIPS'18), pages 10477–10488. Curran Associates, 2018b.

J. Wilson, F. Hutter, and M. Deisenroth. Maximizing acquisition functions for Bayesian optimization. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Proceedings of the 31st International Conference on Advances in Neural Information Processing Systems (NeurIPS'18)*, pages 9906–9917. Curran Associates, 2018.

M. Wistuba. TST-R implementation, 2016. `https://github.com/wistuba/TST`.

M. Wistuba and J. Grabocka. Few-shot Bayesian optimization with deep kernel surrogates. In *Proceedings of the International Conference on Learning Representations (ICLR'21)*, 2021. Published online: `iclr.cc`.

M. Wistuba, N. Schilling, and L. Schmidt-Thieme. Sequential Model-Free Hyperparameter Tuning. In *ICDM '15: Proceedings of the 2015 IEEE International Conference on Data Mining (ICDM)*, pages 1033–1038. IEEE Computer Society Press, 2015a.

M. Wistuba, N. Schilling, and L. Schmidt-Thieme. Hyperparameter search space pruning - A new component for sequential model-based hyperparameter optimization. In A. Appice, P. Rodrigues, V. Costa, J. Gama, A. Jorge, and C. Soares, editors, *Machine Learning and Knowledge Discovery in Databases (ECML/PKDD'15)*, volume 9285 of *Lecture Notes in Computer Science*, pages 104–119. Springer, 2015b.

M. Wistuba, N. Schilling, and L. Schmidt-Thieme. Two-stage transfer surrogate model for automatic hyperparameter optimization. In F. Paolo, N. Landwehr, G. Manco, and J. Vreeken, editors, *Machine Learning and Knowledge Discovery in Databases (ECML/PKDD'16)*, Lecture Notes in Computer Science, pages 199–214. Springer, 2016.

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

J. Wu, S. Toscano-Palmerin, P. Frazier, and A. Wilson. Practical multi-fidelity Bayesian optimization for hyperparameter tuning. In R. Adams and V. Gogate, editors, *Proceedings of The 35th Uncertainty in Artificial Intelligence Conference (UAI'20)*, volume 115, pages 788–798. Proceedings of Machine Learning Research, 2020.

L. Xu, H. Hoos, and K. Leyton-Brown. Hydra: Automatically configuring algorithms for portfolio-based selection. In M. Fox and D. Poole, editors, *Proceedings of the Twenty-fourth National Conference on Artificial Intelligence (AAAI'10)*, pages 210–216. AAAI Press, 2010.

L. Xu, F. Hutter, H. Hoos, and K. Leyton-Brown. Hydra-MIP: Automated algorithm configuration and selection for mixed integer programming. In *Proc. of RCRA workshop at IJCAI*, 2011.

C. Yang, J. Akimoto, D. Kim, and M. Udell. OBOE: Collaborative filtering for AutoML model selection. In A. Teredesai, V. Kumar, Y. Li, R. Rosales, E. Terzi, and G. Karypis,

editors, *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD'19*, pages 1173–1183. ACM Press, 2019.

C. Yang, J. Fan, Z. Wu, and M. Udell. AutoML pipeline selection: Efficiently navigating the combinatorial space. In J. Tang and B. Prakash, editors, *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'20)*, pages 1446–1456. ACM Press, 2020.

D. Yogatama and G. Mann. Efficient transfer learning method for automatic hyperparameter tuning. In S. Kaski and J. Corander, editors, *Proceedings of the Seventeenth International Conference on Artificial Intelligence and Statistics (AISTATS)*, volume 33, pages 1077–1085. Proceedings of Machine Learning Research, 2014.

L. Zimmer, M. Lindauer, and F. Hutter. Auto-Pytorch: Multi-fidelity metalearning for efficient and robust AutoDL. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 1–1, 2021.

Part III

PUBLICATIONS ON AUTOMATED MACHINE LEARNING

5

EFFICIENT AND ROBUST AUTOMATED MACHINE LEARNING

Matthias Feurer, Aaron Klein, Katharina Eggensperger, Jost Tobias Springenberg, Manuel Blum, and Frank Hutter (2015). "Efficient and Robust Automated Machine Learning." In: *Advances in Neural Information Processing Systems*. Ed. by Corinna Cortes, Neil D Lawrence, Daniel D Lee, Masashi Sugiyama, and Roman Garnett. Vol. 28, pp. 2962–2970.

PAPER SUMMARY. This paper proposed to robustify automated machine learning systems and presents the winning solution to the 1st ChaLearn AutoML competition. Its contributions are:

1. Formally defining the AutoML problem.
2. Demonstrating that warm-starting can be applied to a fully-fledged AutoML systems to overcome the so-called cold-start problem when optimizing hyperparameters.
3. Introducing post-hoc ensemble building as a final step to not only return a single machine learning model, but an ensemble of the models constructed throughout the search for an optimal machine learning pipeline, robustifying the AutoML system.
4. A large-scale empirical study, comparing against two available competitor systems, AutoWEKA and Hyperopt-sklearn.

PROJECT IDEA. Writing this paper was proposed by Frank Hutter. It is based on the winning solution to the 1st ChaLearn AutoML competition, which was developed and researched by all co-authors spearheaded by Matthias Feurer.

IMPLEMENTATION AND EXPERIMENTATION. Matthias Feurer led the implementation. Matthias Feurer implemented the software framework with help from Katharina Eggensperger, Matthias Feurer implemented the meta-learning component and Aaron Klein implemented the ensembling component. Experiments were conducted by Matthias Feurer and Katharina Eggensperger.

PAPER WRITING. Aaron Klein, Katharina Eggensperger and Matthias Feurer prepared a first draft of this paper. All co-authors jointly revised and extended it. The final paper version was largely revised and edited by Frank Hutter.

Efficient and Robust Automated Machine Learning

Matthias Feurer **Aaron Klein** **Katharina Eggensperger**
Jost Tobias Springenberg **Manuel Blum** **Frank Hutter**
Department of Computer Science
University of Freiburg, Germany
{feurerm,kleinaa,eggenspk,springj,mblum,fh}@cs.uni-freiburg.de

Abstract

The success of machine learning in a broad range of applications has led to an ever-growing demand for machine learning systems that can be used off the shelf by non-experts. To be effective in practice, such systems need to automatically choose a good algorithm and feature preprocessing steps for a new dataset at hand, and also set their respective hyperparameters. Recent work has started to tackle this *automated machine learning (AutoML)* problem with the help of efficient Bayesian optimization methods. Building on this, we introduce a robust new AutoML system based on scikit-learn (using 15 classifiers, 14 feature preprocessing methods, and 4 data preprocessing methods, giving rise to a structured hypothesis space with 110 hyperparameters). This system, which we dub AUTO-SKLEARN, improves on existing AutoML methods by automatically taking into account past performance on similar datasets, and by constructing ensembles from the models evaluated during the optimization. Our system won the first phase of the ongoing ChaLearn AutoML challenge, and our comprehensive analysis on over 100 diverse datasets shows that it substantially outperforms the previous state of the art in AutoML. We also demonstrate the performance gains due to each of our contributions and derive insights into the effectiveness of the individual components of AUTO-SKLEARN.

1 Introduction

Machine learning has recently made great strides in many application areas, fueling a growing demand for machine learning systems that can be used effectively by novices in machine learning. Correspondingly, a growing number of commercial enterprises aim to satisfy this demand (*e.g.*, BigML.com, Wise.io, SkyTree.com, RapidMiner.com, Dato.com, Prediction.io, DataRobot.com, Microsoft's Azure Machine Learning, Google's Prediction API, and Amazon Machine Learning). At its core, every effective machine learning service needs to solve the fundamental problems of deciding which machine learning algorithm to use on a given dataset, whether and how to preprocess its features, and how to set all hyperparameters. This is the problem we address in this work.

More specifically, we investigate automated machine learning (AutoML), the problem of automatically (without human input) producing test set predictions for a new dataset within a fixed computational budget. Formally, this AutoML problem can be stated as follows:

Definition 1 (AutoML problem)**.** *For $i = 1, \dots, n+m$, let $\boldsymbol{x}_i \in \mathbb{R}^d$ denote a feature vector and $y_i \in Y$ the corresponding target value. Given a training dataset $D_{train} = \{(\boldsymbol{x}_1, y_1), \dots, (\boldsymbol{x}_n, y_n)\}$ and the feature vectors $\boldsymbol{x}_{n+1}, \dots, \boldsymbol{x}_{n+m}$ of a test dataset $D_{test} = \{(\boldsymbol{x}_{n+1}, y_{n+1}), \dots, (\boldsymbol{x}_{n+m}, y_{n+m})\}$ drawn from the same underlying data distribution, as well as a resource budget b and a loss metric $\mathcal{L}(\cdot,\cdot)$, the AutoML problem is to (automatically) produce test set predictions $\hat{y}_{n+1}, \dots, \hat{y}_{n+m}$. The loss of a solution $\hat{y}_{n+1}, \dots, \hat{y}_{n+m}$ to the AutoML problem is given by $\frac{1}{m}\sum_{j=1}^{m} \mathcal{L}(\hat{y}_{n+j}, y_{n+j})$.*

In practice, the budget b would comprise computational resources, such as CPU and/or wallclock time and memory usage. This problem definition reflects the setting of the ongoing ChaLearn AutoML challenge [1]. The AutoML system we describe here won the first phase of that challenge.

Here, we follow and extend the AutoML approach first introduced by AUTO-WEKA [2] (see `http://automl.org`). At its core, this approach combines a highly parametric machine learning framework F with a Bayesian optimization [3] method for instantiating F well for a given dataset.

The contribution of this paper is to extend this AutoML approach in various ways that considerably improve its *efficiency* and *robustness*, based on principles that apply to a wide range of machine learning frameworks (such as those used by the machine learning service providers mentioned above). First, following successful previous work for low dimensional optimization problems [4, 5, 6], we reason across datasets to identify instantiations of machine learning frameworks that perform well on a new dataset and warmstart Bayesian optimization with them (Section 3.1). Second, we automatically construct ensembles of the models considered by Bayesian optimization (Section 3.2). Third, we carefully design a highly parameterized machine learning framework from high-performing classifiers and preprocessors implemented in the popular machine learning framework scikit-learn [7] (Section 4). Finally, we perform an extensive empirical analysis using a diverse collection of datasets to demonstrate that the resulting AUTO-SKLEARN system outperforms previous state-of-the-art AutoML methods (Section 5), to show that each of our contributions leads to substantial performance improvements (Section 6), and to gain insights into the performance of the individual classifiers and preprocessors used in AUTO-SKLEARN (Section 7).

2 AutoML as a CASH problem

We first review the formalization of AutoML as a *Combined Algorithm Selection and Hyperparameter optimization (CASH)* problem used by AUTO-WEKA's AutoML approach. Two important problems in AutoML are that (1) no single machine learning method performs best on all datasets and (2) some machine learning methods (*e.g.*, non-linear SVMs) crucially rely on hyperparameter optimization. The latter problem has been successfully attacked using Bayesian optimization [3], which nowadays forms a core component of an AutoML system. The former problem is intertwined with the latter since the rankings of algorithms depend on whether their hyperparameters are tuned properly. Fortunately, the two problems can efficiently be tackled as a single, structured, joint optimization problem:

Definition 2 (CASH)**.** *Let $\mathcal{A} = \{A^{(1)}, \ldots, A^{(R)}\}$ be a set of algorithms, and let the hyperparameters of each algorithm $A^{(j)}$ have domain $\Lambda^{(j)}$. Further, let $D_{train} = \{(x_1, y_1), \ldots, (x_n, y_n)\}$ be a training set which is split into K cross-validation folds $\{D^{(1)}_{valid}, \ldots, D^{(K)}_{valid}\}$ and $\{D^{(1)}_{train}, \ldots, D^{(K)}_{train}\}$ such that $D^{(i)}_{train} = D_{train} \backslash D^{(i)}_{valid}$ for $i = 1, \ldots, K$. Finally, let $\mathcal{L}(A^{(j)}_{\boldsymbol{\lambda}}, D^{(i)}_{train}, D^{(i)}_{valid})$ denote the loss that algorithm $A^{(j)}$ achieves on $D^{(i)}_{valid}$ when trained on $D^{(i)}_{train}$ with hyperparameters $\boldsymbol{\lambda}$. Then, the* Combined Algorithm Selection and Hyperparameter optimization (CASH) *problem is to find the joint algorithm and hyperparameter setting that minimizes this loss:*

$$A^{\star}, \boldsymbol{\lambda}_{\star} \in \underset{A^{(j)} \in \mathcal{A}, \boldsymbol{\lambda} \in \Lambda^{(j)}}{\operatorname{argmin}} \frac{1}{K} \sum_{i=1}^{K} \mathcal{L}(A^{(j)}_{\boldsymbol{\lambda}}, D^{(i)}_{train}, D^{(i)}_{valid}). \tag{1}$$

This CASH problem was first tackled by Thornton et al. [2] in the AUTO-WEKA system using the machine learning framework WEKA [8] and tree-based Bayesian optimization methods [9, 10]. In a nutshell, Bayesian optimization [3] fits a probabilistic model to capture the relationship between hyperparameter settings and their measured performance; it then uses this model to select the most promising hyperparameter setting (trading off exploration of new parts of the space *vs.* exploitation in known good regions), evaluates that hyperparameter setting, updates the model with the result, and iterates. While Bayesian optimization based on Gaussian process models (*e.g.*, Snoek et al. [11]) performs best in low-dimensional problems with numerical hyperparameters, tree-based models have been shown to be more successful in high-dimensional, structured, and partly discrete problems [12] – such as the CASH problem – and are also used in the AutoML system HYPEROPT-SKLEARN [13]. Among the tree-based Bayesian optimization methods, Thornton et al. [2] found the random-forest-based SMAC [9] to outperform the tree Parzen estimator TPE [10], and we therefore use SMAC to solve the CASH problem in this paper. Next to its use of random forests [14], SMAC's main distinguishing feature is that it allows fast cross-validation by evaluating one fold at a time and discarding poorly-performing hyperparameter settings early.

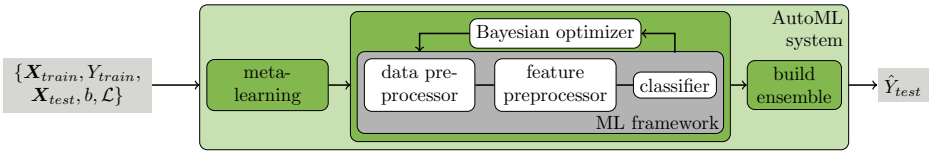


Figure 1: Our improved AutoML approach. We add two components to Bayesian hyperparameter optimization of an ML framework: meta-learning for initializing the Bayesian optimizer and automated ensemble construction from configurations evaluated during optimization.

3 New methods for increasing efficiency and robustness of AutoML

We now discuss our two improvements of the AutoML approach. First, we include a meta-learning step to warmstart the Bayesian optimization procedure, which results in a considerable boost in efficiency. Second, we include an automated ensemble construction step, allowing us to use all classifiers that were found by Bayesian optimization.

Figure 1 summarizes the overall AutoML workflow, including both of our improvements. We note that we expect their effectiveness to be greater for flexible ML frameworks that offer many degrees of freedom (*e.g.*, many algorithms, hyperparameters, and preprocessing methods).

3.1 Meta-learning for finding good instantiations of machine learning frameworks

Domain experts derive knowledge from previous tasks: They *learn about the performance of machine learning algorithms*. The area of meta-learning [15] mimics this strategy by reasoning about the performance of learning algorithms across datasets. In this work, we apply meta-learning to select instantiations of our given machine learning framework that are likely to perform well on a new dataset. More specifically, for a large number of datasets, we collect both performance data and a set of *meta-features*, i.e., characteristics of the dataset that can be computed efficiently and that help to determine which algorithm to use on a new dataset.

This meta-learning approach is complementary to Bayesian optimization for optimizing an ML framework. Meta-learning can quickly suggest some instantiations of the ML framework that are likely to perform quite well, but it is unable to provide fine-grained information on performance. In contrast, Bayesian optimization is slow to start for hyperparameter spaces as large as those of entire ML frameworks, but can fine-tune performance over time. We exploit this complementarity by selecting k configurations based on meta-learning and use their result to seed Bayesian optimization. This approach of warmstarting optimization by meta-learning has already been successfully applied before [4, 5, 6], but never to an optimization problem as complex as that of searching the space of instantiations of a full-fledged ML framework. Likewise, learning across datasets has also been applied in collaborative Bayesian optimization methods [16, 17]; while these approaches are promising, they are so far limited to very few meta-features and cannot yet cope with the high-dimensional partially discrete configuration spaces faced in AutoML.

More precisely, our meta-learning approach works as follows. In an offline phase, for each machine learning dataset in a dataset repository (in our case 140 datasets from the OpenML [18] repository), we evaluated a set of meta-features (described below) and used Bayesian optimization to determine and store an instantiation of the given ML framework with strong empirical performance for that dataset. (In detail, we ran SMAC [9] for 24 hours with 10-fold cross-validation on two thirds of the data and stored the resulting ML framework instantiation which exhibited best performance on the remaining third). Then, given a new dataset $\mathcal{D}$, we compute its meta-features, rank all datasets by their L_1 distance to $\mathcal{D}$ in meta-feature space and select the stored ML framework instantiations for the $k = 25$ nearest datasets for evaluation before starting Bayesian optimization with their results.

To characterize datasets, we implemented a total of 38 meta-features from the literature, including simple, information-theoretic and statistical meta-features [19, 20], such as statistics about the number of data points, features, and classes, as well as data skewness, and the entropy of the targets. All meta-features are listed in Table 1 of the supplementary material. Notably, we had to exclude the prominent and effective category of landmarking meta-features [21] (which measure the performance of simple base learners), because they were computationally too expensive to be helpful in the online evaluation phase. We note that this meta-learning approach draws its power from the availability of

a repository of datasets; due to recent initiatives, such as OpenML [18], we expect the number of available datasets to grow ever larger over time, increasing the importance of meta-learning.

3.2 Automated ensemble construction of models evaluated during optimization

While Bayesian hyperparameter optimization is data-efficient in finding the best-performing hyperparameter setting, we note that it is a very wasteful procedure when the goal is simply to make good predictions: all the models it trains during the course of the search are lost, usually including some that perform almost as well as the best. Rather than discarding these models, we propose to store them and to use an efficient post-processing method (which can be run in a second process on-the-fly) to construct an ensemble out of them. This automatic ensemble construction avoids to commit itself to a single hyperparameter setting and is thus more robust (and less prone to overfitting) than using the point estimate that standard hyperparameter optimization yields. To our best knowledge, we are the first to make this simple observation, which can be applied to improve any Bayesian hyperparameter optimization method.

It is well known that ensembles often outperform individual models [22, 23], and that effective ensembles can be created from a library of models [24, 25]. Ensembles perform particularly well if the models they are based on (1) are individually strong and (2) make uncorrelated errors [14]. Since this is much more likely when the individual models are different in nature, ensemble building is particularly well suited for combining strong instantiations of a flexible ML framework.

However, simply building a uniformly weighted ensemble of the models found by Bayesian optimization does *not* work well. Rather, we found it crucial to adjust these weights using the predictions of all individual models on a hold-out set. We experimented with different approaches to optimize these weights: *stacking* [26], gradient-free numerical optimization, and the method *ensemble selection* [24]. While we found both numerical optimization and stacking to overfit to the validation set and to be computationally costly, ensemble selection was fast and robust. In a nutshell, ensemble selection (introduced by Caruana et al. [24]) is a greedy procedure that starts from an empty ensemble and then iteratively adds the model that maximizes ensemble validation performance (with uniform weight, but allowing for repetitions). Procedure 1 in the supplementary material describes it in detail. We used this technique in all our experiments – building an ensemble of size 50.

4 A practical automated machine learning system

To design a robust AutoML system, as our underlying ML framework we chose scikit-learn [7], one of the best known and most widely used machine learning libraries. It offers a wide range of well established and efficiently-implemented ML algorithms and is easy to use for both experts and beginners. Since our AutoML system closely resembles AUTO-WEKA, but – like HYPEROPT-SKLEARN – is based on scikit-learn, we dub it AUTO-SKLEARN.

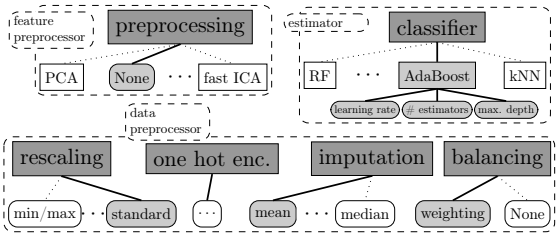


Figure 2: Structured configuration space. Squared boxes denote parent hyperparameters whereas boxes with rounded edges are leaf hyperparameters. Grey colored boxes mark active hyperparameters which form an example configuration and machine learning pipeline. Each pipeline comprises one *feature preprocessor*, *classifier* and up to three *data preprocessor* methods plus respective hyperparameters.

Figure 2 depicts AUTO-SKLEARN's overall components. It comprises 15 classification algorithms, 14 preprocessing methods, and 4 data preprocessing methods. We parameterized each of them, which resulted in a space of 110 hyperparameters. Most of these are conditional hyperparameters that are only active if their respective component is selected. We note that SMAC [9] can handle this conditionality natively.

All 15 classification algorithms in AUTO-SKLEARN are listed in Table 1a (and described in detail in Section A.1 of the supplementary material). They fall into different categories, such as general linear models (2 algorithms), support vector machines (2), discriminant analysis (2), nearest neighbors (1), naïve Bayes (3), decision trees (1) and ensembles (4). In contrast to AUTO-WEKA [2], we

name	#λ	cat (cond)	cont (cond)
AdaBoost (AB)	4	1 (-)	3 (-)
Bernoulli naïve Bayes	2	1 (-)	1 (-)
decision tree (DT)	4	1 (-)	3 (-)
extreml. rand. trees	5	2 (-)	3 (-)
Gaussian naïve Bayes	-	-	-
gradient boosting (GB)	6	-	6 (-)
kNN	3	2 (-)	1 (-)
LDA	4	1 (-)	3 (1)
linear SVM	4	2 (-)	2 (-)
kernel SVM	7	2 (-)	5 (2)
multinomial naïve Bayes	2	1 (-)	1 (-)
passive aggressive	3	1 (-)	2 (-)
QDA	2	-	2 (-)
random forest (RF)	5	2 (-)	3 (-)
Linear Class. (SGD)	10	4 (-)	6 (3)

(a) classification algorithms

name	#λ	cat (cond)	cont (cond)
extreml. rand. trees prepr.	5	2 (-)	3 (-)
fast ICA	4	3 (-)	1 (1)
feature agglomeration	4	3 ()	1 (-)
kernel PCA	5	1 (-)	4 (3)
rand. kitchen sinks	2	-	2 (-)
linear SVM prepr.	3	1 (-)	2 (-)
no preprocessing	-	-	-
nystroem sampler	5	1 (-)	4 (3)
PCA	2	1 (-)	1 (-)
polynomial	3	2 (-)	1 (-)
random trees embed.	4	-	4 (-)
select percentile	2	1 (-)	1 (-)
select rates	3	2 (-)	1 (-)
one-hot encoding	2	1 (-)	1 (1)
imputation	1	1 (-)	-
balancing	1	1 (-)	-
rescaling	1	1 (-)	-

(b) preprocessing methods

Table 1: Number of hyperparameters for each possible classifier (left) and feature preprocessing method (right) for a **binary classification** dataset in **dense** representation. Tables for sparse binary classification and sparse/dense multiclass classification datasets can be found in the Section E of the supplementary material, Tables 2a, 3a, 4a, 2b, 3b and 4b. We distinguish between categorical (cat) hyperparameters with discrete values and continuous (cont) numerical hyperparameters. Numbers in brackets are conditional hyperparameters, which are only relevant when another parameter has a certain value.

focused our configuration space on base classifiers and excluded meta-models and ensembles that are themselves parameterized by one or more base classifiers. While such ensembles increased AUTO-WEKA's number of hyperparameters by almost a factor of five (to 786), AUTO-SKLEARN "only" features 110 hyperparameters. We instead construct complex ensembles using our post-hoc method from Section 3.2. Compared to AUTO-WEKA, this is much more data-efficient: in AUTO-WEKA, evaluating the performance of an ensemble with 5 components requires the construction and evaluation of 5 models; in contrast, in AUTO-SKLEARN, ensembles come largely for free, and it is possible to mix and match models evaluated at arbitrary times during the optimization.

The preprocessing methods for datasets in dense representation in AUTO-SKLEARN are listed in Table 1b (and described in detail in Section A.2 of the supplementary material). They comprise data preprocessors (which change the feature values and are always used when they apply) and feature preprocessors (which change the actual set of features, and only one of which [or none] is used). Data preprocessing includes rescaling of the inputs, imputation of missing values, one-hot encoding and balancing of the target classes. The 14 possible feature preprocessing methods can be categorized into feature selection (2), kernel approximation (2), matrix decomposition (3), embeddings (1), feature clustering (1), polynomial feature expansion (1) and methods that use a classifier for feature selection (2). For example, L_1-regularized linear SVMs fitted to the data can be used for feature selection by eliminating features corresponding to zero-valued model coefficients.

As with every robust real-world system, we had to handle many more important details in AUTO-SKLEARN; we describe these in Section B of the supplementary material.

5 Comparing AUTO-SKLEARN to AUTO-WEKA and HYPEROPT-SKLEARN

As a baseline experiment, we compared the performance of vanilla AUTO-SKLEARN (without our improvements) to AUTO-WEKA and HYPEROPT-SKLEARN, reproducing the experimental setup with 21 datasets of the paper introducing AUTO-WEKA [2]. We describe this setup in detail in Section G in the supplementary material.

Table 2 shows that AUTO-SKLEARN performed statistically significantly better than AUTO-WEKA in 6/21 cases, tied it in 12 cases, and lost against it in 3. For the three datasets where AUTO-WEKA performed best, we found that in more than 50% of its runs the best classifier it chose is not implemented in scikit-learn (trees with a pruning component). So far, HYPEROPT-SKLEARN is more of a proof-of-concept – inviting the user to adapt the configuration space to her own needs – than a full AutoML system. The current version crashes when presented with sparse data and missing values. It also crashes on Cifar-10 due to a memory limit which we set for all optimizers to enable a

	Abalone	Amazon	Car	Cifar-10	Cifar-10 Small	Convex	Dexter	Dorothea	German Credit	Gisette	KDD09 Appetency	KR-vs-KP	Madelon	MNIST Basic	MRBI	Secom	Semeion	Shuttle	Waveform	Wine Quality	Yeast
AS	**73.50**	**16.00**	0.39	**51.70**	**54.81**	**17.53**	**5.56**	**5.51**	**27.00**	**1.62**	**1.74**	0.42	**12.44**	2.84	**46.92**	**7.87**	**5.24**	**0.01**	14.93	33.76	40.67
AW	**73.50**	30.00	**0.00**	56.95	56.20	21.80	8.33	6.38	28.33	2.29	**1.74**	**0.31**	18.21	2.84	60.34	8.09	5.24	**0.01**	14.13	**33.36**	**37.75**
HS	76.21	16.22	0.39	-	57.95	19.18	-	-	27.67	2.29	-	0.42	14.74	**2.82**	55.79	-	5.87	0.05	**14.07**	34.72	38.45

Table 2: Test set classification error of AUTO-WEKA (AW), vanilla AUTO-SKLEARN (AS) and HYPEROPT-SKLEARN (HS), as in the original evaluation of AUTO-WEKA [2]. We show median percent error across 100 000 bootstrap samples (based on 10 runs), simulating 4 parallel runs. Bold numbers indicate the best result. Underlined results are not statistically significantly different from the best according to a bootstrap test with $p = 0.05$.

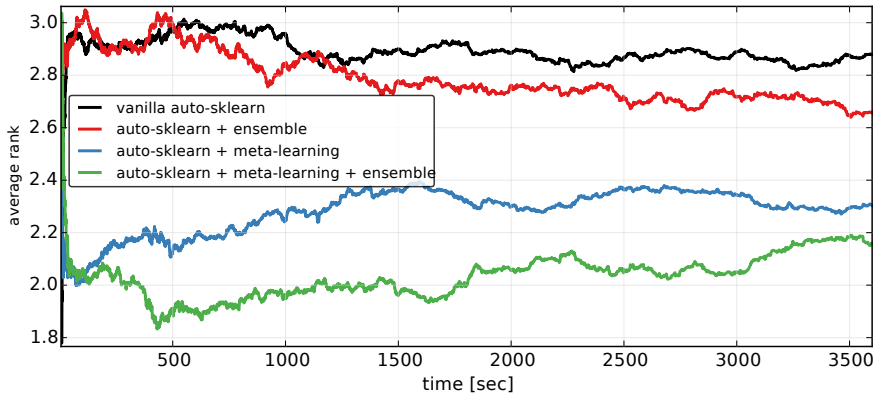


Figure 3: Average rank of all four AUTO-SKLEARN variants (ranked by balanced test error rate (BER)) across 140 datasets. Note that ranks are a relative measure of performance (here, the rank of all methods has to add up to 10), and hence an improvement in BER of one method can worsen the rank of another. The supplementary material shows the same plot on a log-scale to show the time overhead of meta-feature and ensemble computation.

fair comparison. On the 16 datasets on which it ran, it statistically tied the best optimizer in 9 cases and lost against it in 7.

6 Evaluation of the proposed AutoML improvements

In order to evaluate the robustness and general applicability of our proposed AutoML system on a broad range of datasets, we gathered 140 binary and multiclass classification datasets from the OpenML repository [18], only selecting datasets with at least 1000 data points to allow robust performance evaluations. These datasets cover a diverse range of applications, such as text classification, digit and letter recognition, gene sequence and RNA classification, advertisement, particle classification for telescope data, and cancer detection in tissue samples. We list all datasets in Table 7 and 8 in the supplementary material and provide their unique OpenML identifiers for reproducibility. Since the class distribution in many of these datasets is quite imbalanced we evaluated all AutoML methods using a measure called *balanced classification error rate* (BER). We define balanced error rate as the average of the proportion of wrong classifications in each class. In comparison to standard classification error (the average overall error), this measure (the average of the *class-wise* error) assigns equal weight to all classes. We note that balanced error or accuracy measures are often used in machine learning competitions (*e.g.*, the AutoML challenge [1] uses balanced accuracy).

We performed 10 runs of AUTO-SKLEARN both with and without meta-learning and with and without ensemble prediction on each of the datasets. To study their performance under rigid time constraints, and also due to computational resource constraints, we limited the CPU time for each run to 1 hour; we also limited the runtime for a single model to a tenth of this (6 minutes). To not evaluate performance on data sets already used for meta-learning, we performed a leave-one-dataset-out validation: when evaluating on dataset $\mathcal{D}$, we only used meta-information from the 139 other datasets.

Figure 3 shows the average ranks over time of the four AUTO-SKLEARN versions we tested. We observe that both of our new methods yielded substantial improvements over vanilla AUTO-SKLEARN. The most striking result is that meta-learning yielded drastic improvements starting with the first

OpenML dataset ID	AUTO-SKLEARN	AdaBoost	Bernoulli naïve Bayes	decision tree	extreml. rand. trees	Gaussian naïve Bayes	gradient boosting	kNN	LDA	linear SVM	kernel SVM	multinomial naïve Bayes	passive aggresive	QDA	random forest	Linear Class. (SGD)
38	2.15	2.68	50.22	2.15	18.06	11.22	**1.77**	50.00	8.55	16.29	17.89	46.99	50.00	8.78	2.34	15.82
46	3.76	4.65	-	5.62	4.74	7.88	**3.49**	7.57	8.67	8.31	5.36	7.55	9.23	7.57	4.20	7.31
179	**16.99**	17.03	19.27	18.31	17.09	21.77	17.00	22.23	18.93	17.30	17.57	18.97	22.29	19.06	17.24	17.01
184	**10.32**	10.52	-	17.46	11.10	64.74	10.42	31.10	35.44	15.76	12.52	27.13	20.01	47.18	10.98	12.76
554	1.55	2.42	-	12.00	2.91	10.52	3.86	2.68	3.34	2.23	**1.50**	10.37	100.00	2.75	3.08	2.50
772	46.85	49.68	47.90	47.75	**45.62**	48.83	48.15	48.00	46.74	48.38	48.66	47.21	48.75	47.67	47.71	47.93
917	10.22	9.11	25.83	11.00	10.22	33.94	10.11	11.11	34.22	18.67	**6.78**	25.50	20.67	30.44	10.83	18.33
1049	12.93	**12.53**	15.50	19.31	17.18	26.23	13.38	23.80	25.12	17.28	21.44	26.40	29.25	21.38	13.75	19.92
1111	23.70	23.16	28.40	24.40	24.47	29.59	**22.93**	50.30	24.11	23.99	23.56	27.67	43.79	25.86	28.06	23.36
1120	13.81	**13.54**	18.81	17.45	13.86	21.50	13.61	17.23	15.48	14.94	14.17	18.33	16.37	15.62	13.70	14.66
1128	4.21	4.89	4.71	9.30	3.89	4.77	4.58	4.59	4.58	4.83	4.59	4.46	5.65	5.59	**3.83**	4.33
293	2.86	4.07	24.30	5.03	3.59	32.44	24.48	4.86	24.40	14.16	100.00	24.20	21.34	28.68	**2.57**	15.54
389	19.65	22.98	-	33.14	19.38	29.18	19.20	30.87	19.68	**17.95**	22.04	20.04	20.14	39.57	20.66	17.99

Table 3: Median balanced test error rate (BER) of optimizing AUTO-SKLEARN subspaces for each classification method (and all preprocessors), as well as the whole configuration space of AUTO-SKLEARN, on 13 datasets. All optimization runs were allowed to run for 24 hours except for AUTO-SKLEARN which ran for 48 hours. Bold numbers indicate the best result; underlined results are not statistically significantly different from the best according to a bootstrap test using the same setup as for Table 2.

OpenML dataset ID	AUTO-SKLEARN	densifier	extreml. rand. trees prepr.	fast ICA	feature agglomeration	kernel PCA	rand. kitchen sinks	linear SVM prepr.	no preproc.	nystroem sampler	PCA	polynomial	random trees embed.	select percentile classification	select rates	truncatedSVD
38	**2.15**	-	4.03	7.27	2.24	5.84	8.57	2.28	2.28	7.70	7.23	2.90	18.50	2.20	2.28	-
46	**3.76**	-	4.98	7.95	4.40	8.74	8.41	4.25	4.52	8.48	8.40	4.21	7.51	4.17	4.68	-
179	16.99	-	17.83	17.24	16.92	100.00	17.34	**16.84**	16.97	17.30	17.64	16.94	17.05	17.09	16.86	-
184	10.32	-	55.78	19.96	11.31	36.52	28.05	**9.92**	11.43	25.53	21.15	10.54	12.68	45.03	10.47	-
554	1.55	-	1.56	2.52	1.65	100.00	100.00	2.21	1.60	2.21	1.65	100.00	3.48	**1.46**	1.70	-
772	**46.85**	-	47.90	48.65	48.62	47.59	47.68	47.72	48.34	48.06	47.30	48.00	47.84	47.56	48.43	-
917	10.22	-	**8.33**	16.06	10.33	20.94	35.44	8.67	9.44	37.83	22.33	9.11	17.67	10.00	10.44	-
1049	12.93	-	20.36	19.92	13.14	19.57	20.06	13.28	15.84	18.96	17.22	12.95	18.52	**11.94**	14.38	-
1111	23.70	-	23.36	24.69	23.73	100.00	25.25	23.43	**22.27**	23.95	23.25	26.94	26.68	23.53	23.33	-
1120	13.81	-	16.29	14.22	13.73	14.57	14.82	14.02	13.85	14.66	14.23	**13.22**	15.03	13.65	13.67	-
1128	4.21	-	4.90	4.96	4.76	4.21	5.08	4.52	4.59	**4.08**	4.59	50.00	9.23	4.33	**4.08**	-
293	2.86	24.40	3.41	-	-	100.00	19.30	3.01	**2.66**	20.94	-	-	8.05	2.86	2.74	4.05
389	19.65	20.63	21.40	-	-	**17.50**	19.66	19.89	20.87	18.46	-	-	44.83	20.17	19.18	21.58

Table 4: Like Table 3, but instead optimizing subspaces for each preprocessing method (and all classifiers).

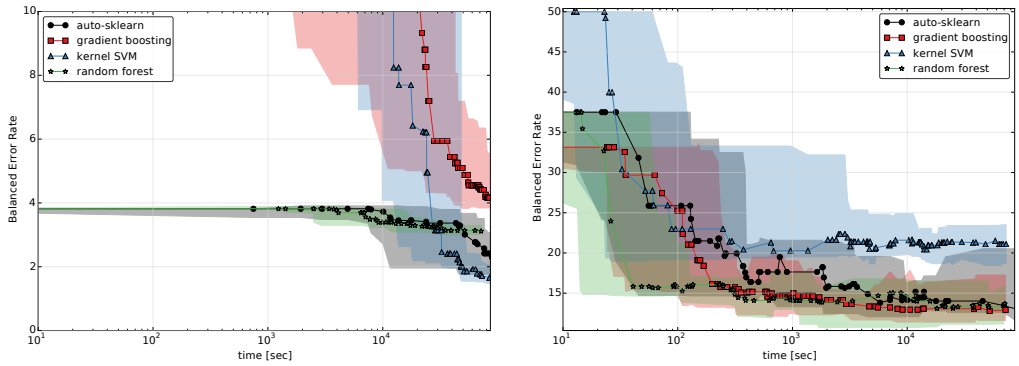
configuration it selected and lasting until the end of the experiment. We note that the improvement was most pronounced in the beginning and that over time, vanilla AUTO-SKLEARN also found good solutions without meta-learning, letting it catch up on some datasets (thus improving its overall rank).

Moreover, both of our methods complement each other: our automated ensemble construction improved both vanilla AUTO-SKLEARN and AUTO-SKLEARN with meta-learning. Interestingly, the ensemble's influence on the performance started earlier for the meta-learning version. We believe that this is because meta-learning produces better machine learning models earlier, which can be directly combined into a strong ensemble; but when run longer, vanilla AUTO-SKLEARN without meta-learning also benefits from automated ensemble construction.

7 Detailed analysis of AUTO-SKLEARN components

We now study AUTO-SKLEARN's individual classifiers and preprocessors, compared to jointly optimizing all methods, in order to obtain insights into their peak performance and robustness. Ideally, we would have liked to study all combinations of a single classifier and a single preprocessor in isolation, but with 15 classifiers and 14 preprocessors this was infeasible; rather, when studying the performance of a single classifier, we still optimized over all preprocessors, and vice versa. To obtain a more detailed analysis, we focused on a subset of datasets but extended the configuration budget for optimizing all methods from one hour to one day and to two days for AUTO-SKLEARN. Specifically, we clustered our 140 datasets with g-means [27] based on the dataset meta-features and used one dataset from each of the resulting 13 clusters (see Table 6 in the supplementary material for the list of datasets). We note that, in total, these extensive experiments required 10.7 CPU years.

Table 3 compares the results of the various classification methods against AUTO-SKLEARN. Overall, as expected, random forests, extremely randomized trees, AdaBoost, and gradient boosting, showed



(a) MNIST (OpenML dataset ID 554) (b) Promise pc4 (OpenML dataset ID 1049)

Figure 4: Performance of a subset of classifiers compared to AUTO-SKLEARN over time. We show median test error rate and the fifth and 95th percentile over time for optimizing three classifiers separately with optimizing the joint space. A plot with all classifiers can be found in Figure 4 in the supplementary material. While AUTO-SKLEARN is inferior in the beginning, in the end its performance is close to the best method.

the most robust performance, and SVMs showed strong peak performance for some datasets. Besides a variety of strong classifiers, there are also several models which could not compete: The decision tree, passive aggressive, kNN, Gaussian NB, LDA and QDA were statistically significantly inferior to the best classifier on most datasets. Finally, the table indicates that no single method was the best choice for all datasets. As shown in the table and also visualized for two example datasets in Figure 4, optimizing the joint configuration space of AUTO-SKLEARN led to the most robust performance. A plot of ranks over time (Figure 2 and 3 in the supplementary material) quantifies this across all 13 datasets, showing that AUTO-SKLEARN starts with reasonable but not optimal performance and effectively searches its more general configuration space to converge to the best overall performance over time.

Table 4 compares the results of the various preprocessors against AUTO-SKLEARN. As for the comparison of classifiers above, AUTO-SKLEARN showed the most robust performance: It performed best on three of the datasets and was not statistically significantly worse than the best preprocessor on another 8 of 13.

8 Discussion and Conclusion

We demonstrated that our new AutoML system AUTO-SKLEARN performs favorably against the previous state of the art in AutoML, and that our meta-learning and ensemble improvements for AutoML yield further efficiency and robustness. This finding is backed by the fact that AUTO-SKLEARN won the auto-track in the first phase of ChaLearn's ongoing AutoML challenge. In this paper, we did not evaluate the use of AUTO-SKLEARN for interactive machine learning with an expert in the loop and weeks of CPU power, but we note that that mode has also led to a third place in the human track of the same challenge. As such, we believe that AUTO-SKLEARN is a promising system for use by both machine learning novices and experts. The source code of AUTO-SKLEARN is available under an open source license at `https://github.com/automl/auto-sklearn`.

Our system also has some shortcomings, which we would like to remove in future work. As one example, we have not yet tackled regression or semi-supervised problems. Most importantly, though, the focus on scikit-learn implied a focus on small to medium-sized datasets, and an obvious direction for future work will be to apply our methods to modern deep learning systems that yield state-of-the-art performance on large datasets; we expect that in that domain especially automated ensemble construction will lead to tangible performance improvements over Bayesian optimization.

Acknowledgments

This work was supported by the German Research Foundation (DFG), under Priority Programme Autonomous Learning (SPP 1527, grant HU 1900/3-1), under Emmy Noether grant HU 1900/2-1, and under the BrainLinks-BrainTools Cluster of Excellence (grant number EXC 1086).

References

[1] I. Guyon, K. Bennett, G. Cawley, H. Escalante, S. Escalera, T. Ho, N.Macià, B. Ray, M. Saeed, A. Statnikov, and E. Viegas. Design of the 2015 ChaLearn AutoML Challenge. In *Proc. of IJCNN'15*, 2015.

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

[3] E. Brochu, V. Cora, and N. de Freitas. A tutorial on Bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning. *CoRR*, abs/1012.2599, 2010.

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

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

[6] T. Gomes, R. Prudêncio, C. Soares, A. Rossi, and A. Carvalho. Combining meta-learning and search techniques to select parameters for support vector machines. *Neurocomputing*, 75(1):3–13, 2012.

[7] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *JMLR*, 12:2825–2830, 2011.

[8] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. Witten. The WEKA data mining software: An update. *SIGKDD*, 11(1):10–18, 2009.

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

[10] J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl. Algorithms for hyper-parameter optimization. In *Proc. of NIPS'11*, pages 2546–2554, 2011.

[11] J. Snoek, H. Larochelle, and R. P. Adams. Practical Bayesian optimization of machine learning algorithms. In *Proc. of NIPS'12*, pages 2960–2968, 2012.

[12] K. Eggensperger, M. Feurer, F. Hutter, J. Bergstra, J. Snoek, H. Hoos, and K. Leyton-Brown. Towards an empirical foundation for assessing Bayesian optimization of hyperparameters. In *NIPS Workshop on Bayesian Optimization in Theory and Practice*, 2013.

[13] B. Komer, J. Bergstra, and C. Eliasmith. Hyperopt-sklearn: Automatic hyperparameter configuration for scikit-learn. In *ICML workshop on AutoML*, 2014.

[14] L. Breiman. Random forests. *MLJ*, 45:5–32, 2001.

[15] P. Brazdil, C. Giraud-Carrier, C. Soares, and R. Vilalta. *Metalearning: Applications to Data Mining*. Springer, 2009.

[16] R. Bardenet, M. Brendel, B. Kégl, and M. Sebag. Collaborative hyperparameter tuning. In *Proc. of ICML'13* [28], pages 199–207.

[17] D. Yogatama and G. Mann. Efficient transfer learning method for automatic hyperparameter tuning. In *Proc. of AISTATS'14*, pages 1077–1085, 2014.

[18] J. Vanschoren, J. van Rijn, B. Bischl, and L. Torgo. OpenML: Networked science in machine learning. *SIGKDD Explorations*, 15(2):49–60, 2013.

[19] D. Michie, D. Spiegelhalter, C. Taylor, and J. Campbell. *Machine Learning, Neural and Statistical Classification*. Ellis Horwood, 1994.

[20] A. Kalousis. *Algorithm Selection via Meta-Learning*. PhD thesis, University of Geneve, 2002.

[21] B. Pfahringer, H. Bensusan, and C. Giraud-Carrier. Meta-learning by landmarking various learning algorithms. In *Proc. of (ICML'00)*, pages 743–750, 2000.

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

[23] A. Lacoste, M. Marchand, F. Laviolette, and H. Larochelle. Agnostic Bayesian learning of ensembles. In *Proc. of ICML'14*, pages 611–619, 2014.

[24] R. Caruana, A. Niculescu-Mizil, G. Crew, and A. Ksikes. Ensemble selection from libraries of models. In *Proc. of ICML'04*, page 18, 2004.

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

[26] D. Wolpert. Stacked generalization. *Neural Networks*, 5:241–259, 1992.

[27] G. Hamerly and C. Elkan. Learning the k in k-means. In *Proc. of NIPS'04*, pages 281–288, 2004.

[28] *Proc. of ICML'13*, 2014.

6

AUTO-SKLEARN 2.0: HANDS-FREE AUTOML VIA META-LEARNING

Matthias Feurer, Katharina Eggensperger, Stefan Falkner, Marius Lindauer, and Frank Hutter (2022). "Auto-Sklearn 2.0: Hands-free AutoML via Meta-Learning." In: *Journal of Machine Learning Research* 23.261. Ed. by Marc Schoenauer, pp. 1–61.

PAPER SUMMARY. This paper aims to reduce the human involvement in setting up and developing AutoML systems. It proposes using a new, portfolio-based meta-learning strategy and subsampling for faster evaluation of individual machine learning models, which together form the technique of PoSH-Auto-sklearn. It then goes one step further and proposes using meta-level meta-learning to choose between different instantiations of an AutoML system, giving rise to Auto-sklearn 2.0.

PROJECT IDEA. This paper is a follow-up of two previous, peer-reviewed workshop papers (one by Matthias Feurer, Katharina Eggensperger, Stefan Falkner, Marius Lindauer and Frank Hutter, and the other one by Matthias Feurer and Frank Hutter), both published at the AutoML workshop at the *International Conference on Machine Learning* in 2018. PoSH-Auto-sklearn was developed by Matthias Feurer, Katharina Eggensperger, Stefan Falkner, Marius Lindauer and Frank Hutter for the 2nd ChaLearn AutoML challenge. The theoretical statements were derived by Matthias Feurer based on an idea by Frank Hutter. The meta-level meta-learning strategy was suggested by Frank Hutter and revised by Matthias Feurer.

IMPLEMENTATION AND EXPERIMENTATION. All software used in this paper was implemented by Matthias Feurer and Katharina Eggensperger and is based on the publicly available Auto-sklearn software (which is maintained by Matthias Feurer but also contains contributions from others, including numerous working students). An earlier version of PoSH-Auto-sklearn used for the 2nd ChaLearn AutoML competition was developed by Matthias Feurer, Katharina Eggensperger, Stefan Falkner and Marius Lindauer. Code for meta-level meta-learning was written solely by Matthias Feurer. An initial version of the experiment code was written by Matthias Feurer and Katharina Eggensperger, and revised by Matthias Feurer, who also conducted the final experiments.

PAPER WRITING. Matthias Feurer, Katharina Eggensperger and Marius Lindauer wrote the first draft. Marius Lindauer, Frank Hutter, Katharina Eggensperger and Matthias Feurer revised the first paper draft with feedback from Stefan Falkner. The final paper version was to a large extent written by Matthias Feurer and Katharina Eggensperger.

LICENSE. This chapter was published by the Journal of Machine Learning Research and is licensed under the terms of the Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/). No changes were made to the original publication.

Journal of Machine Learning Research 23 (2022) 1-61 Submitted 8/21; Revised 3/22; Published 8/22

Auto-Sklearn 2.0: Hands-free AutoML via Meta-Learning

Matthias Feurer[1] feurerm@cs.uni-freiburg.de
Katharina Eggensperger[1] eggenspk@cs.uni-freiburg.de
Stefan Falkner[2] Stefan.Falkner@de.bosch.com
Marius Lindauer[3] lindauer@tnt.uni-hannover.de
Frank Hutter[1,2] fh@cs.uni-freiburg.de
[1] *Department of Computer Science, Albert-Ludwigs-Universität Freiburg*
[2] *Bosch Center for Artificial Intelligence, Renningen, Germany*
[3] *Institute of Information Processing, Leibniz University Hannover*

Editor: Marc Schoenauer

Abstract

Automated Machine Learning (AutoML) supports practitioners and researchers with the tedious task of designing machine learning pipelines and has recently achieved substantial success. In this paper, we introduce new AutoML approaches motivated by our winning submission to the second ChaLearn AutoML challenge. We develop *PoSH Auto-sklearn*, which enables AutoML systems to work well on large datasets under rigid time limits by using a new, simple and meta-feature-free meta-learning technique and by employing a successful bandit strategy for budget allocation. However, *PoSH Auto-sklearn* introduces even more ways of running AutoML and might make it harder for users to set it up correctly. Therefore, we also go one step further and study the design space of AutoML itself, proposing a solution towards truly hands-free AutoML. Together, these changes give rise to the next generation of our AutoML system, *Auto-sklearn 2.0*. We verify the improvements by these additions in an extensive experimental study on 39 AutoML benchmark datasets. We conclude the paper by comparing to other popular AutoML frameworks and *Auto-sklearn 1.0*, reducing the relative error by up to a factor of 4.5, and yielding a performance in 10 minutes that is substantially better than what *Auto-sklearn 1.0* achieves within an hour.

Keywords: Automated machine learning, hyperparameter optimization, meta-learning, automated AutoML, benchmark

1. Introduction

The recent substantial progress in machine learning (ML) has led to a growing demand for hands-free ML systems that can support developers and ML novices in efficiently creating new ML applications. Since different datasets require different ML pipelines, this demand has given rise to the area of automated machine learning (AutoML; Hutter et al., 2019). Popular AutoML systems, such as *Auto-WEKA* (Thornton et al., 2013), *hyperopt-sklearn* (Komer et al., 2014), *Auto-sklearn* (Feurer et al., 2015a), *TPOT* (Olson et al., 2016a) and *Auto-Keras* (Jin et al., 2019) perform a combined optimization across different preprocessors, classifiers or regressors and their hyperparameter settings, thereby reducing the effort for users substantially.

©2022 Matthias Feurer and Katharina Eggensperger and Stefan Falkner and Marius Lindauer and Frank Hutter.

License: CC-BY 4.0, see https://creativecommons.org/licenses/by/4.0/. Attribution requirements are provided at http://jmlr.org/papers/v23/21-0992.html.

To assess the current state of AutoML and, more importantly, to foster progress in AutoML, ChaLearn conducted a series of AutoML challenges (Guyon et al., 2019), which evaluated AutoML systems in a systematic way under rigid time and memory constraints. Concretely, in these challenges, the AutoML systems were required to deliver predictions in less than 20 minutes. On the one hand, this would allow to efficiently integrate AutoML into the rapid prototype-driven workflow of many data scientists and, on the other hand, help to democratize ML by requiring less compute resources.

We won both the first and second AutoML challenge with modified versions of *Auto-sklearn*. In this work, we describe in detail how we improved *Auto-sklearn* from the first version (Feurer et al., 2015a) to construct *PoSH Auto-sklearn*, which won the second competition and then describe how we improved *PoSH Auto-sklearn* further to yield our current approach for *Auto-sklearn 2.0*.

Particularly, while AutoML relieves the user from making low-level design decisions (e.g. which model to use), AutoML itself opens a myriad of high-level design decisions, e.g. which model selection strategy to use (Guyon et al., 2010, 2015; Raschka, 2018) or how to allocate the given time budget (Jamieson and Talwalkar, 2016). Whereas our submissions to the AutoML challenges were mostly hand-designed, in this work, we go one step further by automating AutoML itself to fully unfold the potential of AutoML in practice.[1]

After detailing the AutoML problem we consider in Section 2, we present two main parts making the following contributions:

Part I: Portfolio Successive Halving in *PoSH Auto-sklearn*. In this part (see Section 3), we introduce budget allocation strategies as a complementary design choice to model selection strategies (holdout (HO) and cross-validation (CV)) for AutoML systems. We suggest using the budget allocation strategy successive halving (SH) as an alternative to always using the full budget (FB) to evaluate a configuration to allocate more resources to promising ML pipelines. Furthermore, we introduce both the practical approach as well as the theory behind building better portfolios for the meta-learning component of *Auto-sklearn*. We show that this combination substantially improves performance, yielding stronger results in 10 minutes than *Auto-sklearn 1.0* achieved in 60 minutes.

Part II: Automating AutoML in *Auto-sklearn 2.0*. In this part (see Section 4), we propose a meta-learning technique based on algorithm selection to automatically select the best setting of the AutoML system itself for a given dataset. We dub the resulting system *Auto-sklearn 2.0* and depict the evolution from *Auto-sklearn 1.0* via *PoSH Auto-sklearn* to *Auto-sklearn 2.0* in Figure 1.

In Section 5, we additionally use the AutoML benchmark (Gijsbers et al., 2019) to evaluate *Auto-sklearn 2.0* against other popular AutoML systems and show improved performance under rigid time constraints. Section 6 then puts our work into the context of related works, and Section 7 concludes the paper with open questions, limitations and future work.

1. The work presented in this paper is in part based on two earlier workshop papers introducing some of the presented ideas in preliminary form (Feurer et al., 2018; Feurer and Hutter, 2018).

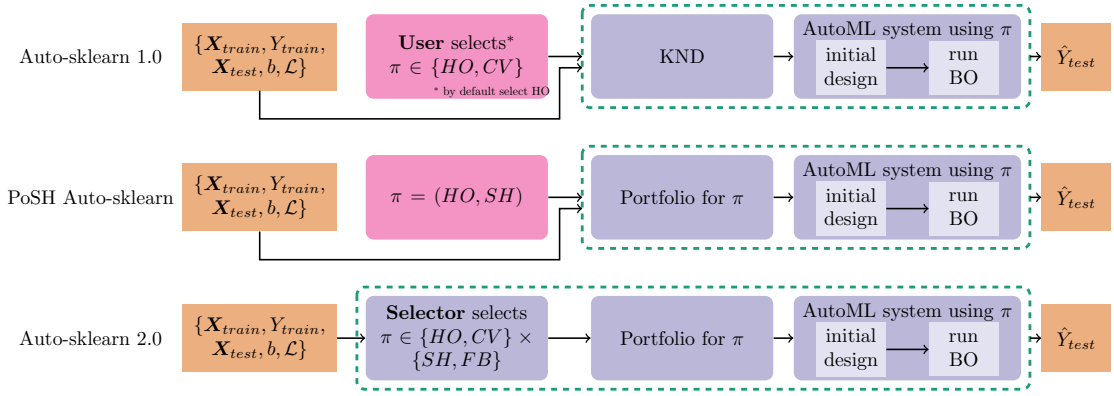


Figure 1: Schematic overview of *Auto-sklearn 1.0*, *PoSH Auto-sklearn*, and *Auto-sklearn 2.0*. Orange rectangular boxes refer to input and output data, while rounded purple boxes denote parts of the AutoML system (surrounded by a green dashed line). The pink, rounded box refers to a human in the loop required for manual design decisions. The newer AutoML systems simplify the usage of *Auto-sklearn* and reduce the required user input. We describe *PoSH Auto-sklearn* in Section 3 and give a schematic overview in Figure 2. Similarly, we describe *Auto-sklearn 2.0* in Section 4 and provide a schematic overview in Figure 5.

2. Problem Statement

AutoML is a widely used term, so, here we first define the problem we consider in this work. Let $P(\mathcal{D})$ be a distribution of datasets from which we can sample an individual dataset's distribution $P_d = P_d(\mathbf{x}, y)$. The AutoML problem we consider is to generate a trained pipeline $\mathcal{M}_{\boldsymbol{\lambda}} : \mathbf{x} \mapsto y$, hyperparameterized by $\boldsymbol{\lambda} \in \boldsymbol{\Lambda}$ that automatically produces predictions for samples from the distribution P_d minimizing the expected generalization error:[2]

$$GE(\mathcal{M}_{\boldsymbol{\lambda}}) = \mathbb{E}_{(\mathbf{x},y)\sim P_d}\left[\mathcal{L}(\mathcal{M}_{\boldsymbol{\lambda}}(\mathbf{x}), y)\right]. \tag{1}$$

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

$$\widehat{GE}(\mathcal{M}_{\boldsymbol{\lambda}}, \mathcal{D}_d) = \frac{1}{n}\sum_{i=1}^{n} \mathcal{L}(\mathcal{M}_{\boldsymbol{\lambda}}(\mathbf{x}_i), y_i). \tag{2}$$

In practice we have access to two disjoint, finite samples which we from now on denote $\mathcal{D}_{\text{train}}$ and $\mathcal{D}_{\text{test}}$ ($\mathcal{D}_{d,\text{train}}$ and $\mathcal{D}_{d,\text{test}}$ in case we reference a specific dataset P_d). For searching the best ML pipeline, we only have access to $\mathcal{D}_{\text{train}}$, however, in the end performance is estimated once on $\mathcal{D}_{\text{test}}$. AutoML systems use this to automatically search for the best $\mathcal{M}_{\boldsymbol{\lambda}^*}$:

$$\mathcal{M}_{\boldsymbol{\lambda}^*} \in \operatorname*{argmin}_{\boldsymbol{\lambda}\in\boldsymbol{\Lambda}} \widehat{GE}(\mathcal{M}_{\boldsymbol{\lambda}}, \mathcal{D}_{\text{train}}), \tag{3}$$

2. Our notation follows Vapnik (1991).

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

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

where $\mathcal{M}_{\boldsymbol{\lambda}}^{\mathcal{D}_{\mathrm{train}}^{(\mathrm{train},k)}}$ denotes that $\mathcal{M}_{\boldsymbol{\lambda}}$ was trained on the training split of k-th fold $\mathcal{D}_{\mathrm{train}}^{(\mathrm{train},k)} \subset \mathcal{D}_{\mathrm{train}}$, and it is then evaluated on the validation split of the k-th fold $\mathcal{D}_{\mathrm{train}}^{(\mathrm{val},k)} = \mathcal{D}_{\mathrm{train}} \setminus \mathcal{D}_{\mathrm{train}}^{(\mathrm{train},k)}$.[3] Assuming that, via $\boldsymbol{\lambda}$, an AutoML system can select both, an algorithm and its hyperparameter settings, this definition using $\widehat{GE}_{\mathrm{CV}}$ is equivalent to the definition of the CASH (*C*ombined *A*lgorithm *S*election and *H*yperparameter optimization) problem (Thornton et al., 2013; Feurer et al., 2015a). However, it is unlikely that, whatever optimization algorithm we use, the AutoML system finds the exact optimum location $\boldsymbol{\lambda}^*$. Instead, the AutoML system will return the best ML pipeline it has trained during the search process, which we denote by $\mathcal{M}_{\hat{\boldsymbol{\lambda}}^*}$, and the hyperparameter settings it was trained with by $\hat{\boldsymbol{\lambda}}^*$.

2.1 Time-bounded AutoML

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

$$\mathcal{M}_{\boldsymbol{\lambda}^*} \in \operatorname*{argmin}_{\boldsymbol{\lambda} \in \boldsymbol{\Lambda}} \widehat{GE}(\mathcal{M}_{\boldsymbol{\lambda}}, \mathcal{D}_{\mathrm{train}}) \text{ s.t. } \left(\sum t_{\boldsymbol{\lambda}_i}\right) < T \tag{5}$$

where the sum is over all evaluated pipelines $\boldsymbol{\lambda}_i$, explicitly honouring the optimization budget T. As before, the AutoML system will return the best model it has found within the optimization budget, $\mathcal{M}_{\hat{\boldsymbol{\lambda}}^*}$.

2.2 Generalization of AutoML

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

$$GE(\mathcal{A}) = \mathbb{E}_{\mathcal{D}_d \sim P(\mathcal{D})} \left[\widehat{GE}(\mathcal{A}(\mathcal{D}_d), \mathcal{D}_d) \right], \tag{6}$$

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

$$\widehat{GE}(\mathcal{A}, \mathbf{D}_{\mathrm{meta}}) = \frac{1}{\mid \mathbf{D}_{\mathrm{meta}} \mid} \sum_{d=1}^{|\mathbf{D}_{\mathrm{meta}}|} \widehat{GE}(\mathcal{A}(\mathcal{D}_d), \mathcal{D}_d). \tag{7}$$

3. Alternatively, one could use holdout to estimate GE with $\widehat{GE}_{\mathrm{HO}}(\mathcal{M}_{\boldsymbol{\lambda}}, \mathcal{D}_{\mathrm{train}}) = \widehat{GE}(\mathcal{M}_{\boldsymbol{\lambda}}^{\mathcal{D}_{\mathrm{train}}^{\mathrm{train}}}, \mathcal{D}_{\mathrm{train}}^{\mathrm{val}})$.

Having set up the problem statement, we can use this to further formalize our goals. Instead of using a single, fixed AutoML system $\mathcal{A}$, we will introduce optimization policies π, a combination of hyperparameters of the AutoML system and specific components to be used in a run, which can be used to configure an AutoML system for specific use cases. We then denote such a configured AutoML system as $\mathcal{A}_\pi$.

We will first construct π manually in Section 3, introduce a novel system for designing π from data in Section 4 and then extend this to a (learned) mapping $\Xi : \mathcal{D} \to \pi$ which automatically suggests an optimization policy for a new dataset using algorithm selection. This problem setup can also be used to introduce generalizations of the algorithm selection problem such as algorithm configuration (Birattari et al., 2002; Hutter et al., 2009; Kleinberg et al., 2017), per-instance algorithm configuration (Xu et al., 2010; Malitsky et al., 2012) and dynamic algorithm configuration (Biedenkapp et al., 2020) on a meta-level; but we leave these for future work. In addition, instead of selecting between multiple policies of a single AutoML system, the presented method can be applied to choose between different AutoML systems without adjustments. However, instead of maximizing performance by invoking many AutoML systems, thereby increasing the complexity, our goal is to improve single AutoML systems to make them easier to use by decreasing complexity for the user.

3. Part I: Portfolio Successive Halving in *PoSH Auto-sklearn*

In this section we introduce our winning solution for the second AutoML competition (Guyon et al., 2019), *PoSH Auto-sklearn*, short for **Po**rtfolio **S**uccessive **H**alving. We first describe our use of portfolios to warmstart an AutoML system and then motivate using the successive halving bandit strategy. Next, we describe practical considerations for building *PoSH Auto-sklearn*, give a schematic overview and recap additional handcrafted techniques we used in the competition. We end this first part of our main contributions with an experimental evaluation demonstrating the performance of *PoSH Auto-sklearn*.

3.1 Portfolio Building

Finding the optimal solution to the time-bounded optimization problem from Equation (5) requires searching a large space of possible ML pipelines as efficiently as possible. BO is a strong approach for this, but its vanilla version starts from scratch for every new problem. A better solution is to warmstart BO with ML pipelines that are expected to work well, as done in the k-nearest dataset (KND) approach of *Auto-sklearn 1.0* (Reif et al., 2012; Feurer et al., 2015b,a; see also the related work in Section 6.4.1). However, we found this solution to introduce new problems:

1. It is time-consuming since it requires to compute meta-features describing the characteristics of datasets.
2. It adds complexity to the system as the computation of the meta-features must also be done with a time and memory limit.
3. Many meta-features are not defined with respect to categorical features and missing values, making them hard to apply for most datasets.
4. It is not immediately clear which meta-features work best for which problem.

5. In the KND approach, there is no mechanism to guarantee that we do not execute redundant ML pipelines.

We indeed suffered from these issues in the first AutoML challenge, failing on one track due to running over time for the meta-feature generation, although we had already removed landmarking meta-features due to their potentially high runtime. Therefore, here we propose a meta-feature-free approach that does not warmstart with a set of configurations specific to a new dataset, but which uses a static *portfolio* – a set of complementary configurations that covers as many diverse datasets as possible and minimizes the risk of failure when facing a new task.

So, instead of evaluating configurations chosen *online* by the KND method, we construct a portfolio, consisting of high-performing and complementary ML pipelines to perform well on as many datasets as possible, *offline*. Then, for a dataset at hand, all pipelines in this portfolio are simply evaluated one after the other. If time is left afterwards, we continue with pipelines suggested by BO warmstarted with the evaluated portfolio pipelines. We introduce portfolio-based warmstarting to avoid computing meta-features for a new dataset. However, the portfolios also work inherently differently. While the KND method is aimed at using only well-performing configurations, a portfolio is built such that there is a diverse set of configurations, starting with ones that perform well on average and then moving to more specialized ones. Thus, it can be seen as an optimized initial design for the BO method.

In the following, we describe our offline procedure for constructing such a portfolio and give theoretical underpinning by a performance bound.

3.1.1 Approach

We first describe how we construct a portfolio given a finite set of candidate pipelines $\mathcal{C} = \{\boldsymbol{\lambda}_1, ..., \boldsymbol{\lambda}_l\}$. Additionally, we assume that there exists a set of datasets $\mathbf{D}_{\text{meta}} = \{\mathcal{D}_1, \ldots, \mathcal{D}_{|\mathbf{D}_{\text{meta}}|}\}$ and we wish to build a portfolio $\mathcal{P}$ consisting of a subset of the pipelines in $\mathcal{C}$ that performs well on $\mathbf{D}_{\text{meta}}$.

We outline the process to build such a portfolio in Algorithm 1. First, we initialize our portfolio $\mathcal{P}$ to the empty set (Line 2). Then, we repeat the following procedure until $|\mathcal{P}|$ reaches a pre-defined limit: From a set of candidate ML pipelines $\mathcal{C}$, we greedily add a candidate $\boldsymbol{\lambda}^+ \in \mathcal{C}$ to $\mathcal{P}$ that reduces the estimated generalization error over all meta-datasets most (Line 4), and then remove $\boldsymbol{\lambda}^+$ from $\mathcal{C}$ (Line 5).

The estimated generalization error of a portfolio $\mathcal{P}$ on a single dataset $\mathcal{D}$ is the performance of the best pipeline $\boldsymbol{\lambda} \in \mathcal{P}$ on $\mathcal{D}$ according to the model selection and budget allocation strategy. This can be described via a function $S(\cdot, \cdot, \cdot)$, which takes as input a function to compute the estimated generalization error (e.g., as defined in Equation 4), a set of machine learning pipelines to train, and a dataset. It then returns the pipeline with the lowest estimated generalization error as

$$\mathcal{M}^{\mathcal{D}}_{\boldsymbol{\lambda}^*} = S(\widehat{GE}, \mathcal{P}, \mathcal{D}) \in \operatorname*{argmin}_{\mathcal{M}^{\mathcal{D}}_{\boldsymbol{\lambda}} \in \mathcal{P}} \widehat{GE}(\mathcal{M}^{\mathcal{D}}_{\boldsymbol{\lambda}}, \mathcal{D}). \tag{8}$$

Algorithm 1: Greedy Portfolio Building

1: **Input:** Set of candidate ML pipelines $\mathcal{C}$, $\mathbf{D}_{\text{meta}} = \{\mathcal{D}_1, \ldots, \mathcal{D}_{|\mathbf{D}_{\text{meta}}|}\}$, maximal portfolio size p, model selection strategy S
2: $\mathcal{P} = \emptyset$
3: **while** $|\mathcal{P}| < p$ **do**
4: $\quad \boldsymbol{\lambda}^+ = \text{argmin}_{\boldsymbol{\lambda} \in \mathcal{C}} \widehat{GE}_S(\mathcal{P} \cup \{\boldsymbol{\lambda}\}, \mathbf{D}_{\text{meta}})$
`// Ties are broken favoring the model trained first.`
5: $\quad \mathcal{P} = \mathcal{P} \cup \boldsymbol{\lambda}^+,\ \mathcal{C} = \mathcal{C} \setminus \{\boldsymbol{\lambda}^+\}$
6: **end while**
7: **return** Portfolio $\mathcal{P}$

In case the result of argmin is not unique, we return the model that was evaluated first. The estimated generalization error of $\mathcal{P}$ across all meta-datasets $\mathbf{D}_{\text{meta}} = \{\mathcal{D}_1, \ldots, \mathcal{D}_{|\mathbf{D}_{\text{meta}}|}\}$ is then

$$\widehat{GE}_S(\mathcal{P}, \mathbf{D}_{\text{meta}}) = \sum_{d=1}^{|\mathbf{D}_{\text{meta}}|} \widehat{GE}\left(S\left(\widehat{GE}, \mathcal{P}, \mathcal{D}_d\right), \mathcal{D}_d^{\text{val}}\right), \tag{9}$$

Here, we give the equation for using holdout, and in Appendix A we provide the exact notation for cross-validation and successive halving.

We now detail how to construct the set of candidate pipelines $\mathcal{C}$ and describe how finding the candidate pipelines and constructing the portfolio fit in the larger picture. We give a schematic overview of this process in Figure 2. It consists of a training (TR1–TR3) and a testing stage (TE1–TE2).

Having collected datasets $\mathbf{D}_{\text{meta}}$ (we describe in Section 3.4.1 how we did this for our experiments), we obtain the candidate ML pipelines (TR1) by running *Auto-sklearn* without meta-learning and without ensembling on each dataset. We limit ourselves to a finite set of portfolio candidates $\mathcal{C}$, and pick one candidate per dataset. Then, we build a performance matrix of size $|\mathcal{C}| \times |\mathbf{D}_{\text{meta}}|$ by evaluating each of these candidate pipelines on each dataset (TR2, we refer to Section 3.4.2 for a detailed description of the meta-data generation). Finally, we then use this matrix to build a portfolio using Algorithm 1 for the combination of model selection strategy holdout and budget allocation strategy SH in training step TR3.

For a new dataset $\mathcal{D}_{\text{new}} \in \mathbf{D}_{\text{test}}$, we apply the AutoML system using SH, holdout and the portfolio to $\mathcal{D}_{\text{new}}$ (TE1). Finally, we return the best found pipeline $\mathcal{M}_{\hat{\boldsymbol{\lambda}}^*}$, or an ensemble of the evaluated pipelines, based on the training set of $\mathcal{D}_{\text{new}}$ (TE2.1). Optionally, we can then compute the loss of $\mathcal{M}_{\hat{\boldsymbol{\lambda}}^*}$ on the test set of $\mathcal{D}_{\text{new}}$ (TE2.2); we emphasize that this would be the only time we ever access the test set of $\mathcal{D}_{\text{new}}$.

To build a portfolio across datasets, we need to take into account that the generalization errors for different datasets live on different scales (Bardenet et al., 2013). Thus, before taking averages, for each dataset, we transform the generalization errors to the distance to the best observed performance scaled between zero and one, a metric named *distance to minimum*; which when averaged across all datasets is known as *average distance to the minimum* (ADTM) (Wistuba et al., 2015a, 2018). We compute the statistics for zero-one scaling individually for each combination of model selection and budget allocation (i.e., we use the lowest observed test loss and the largest observed test loss for each meta-dataset).

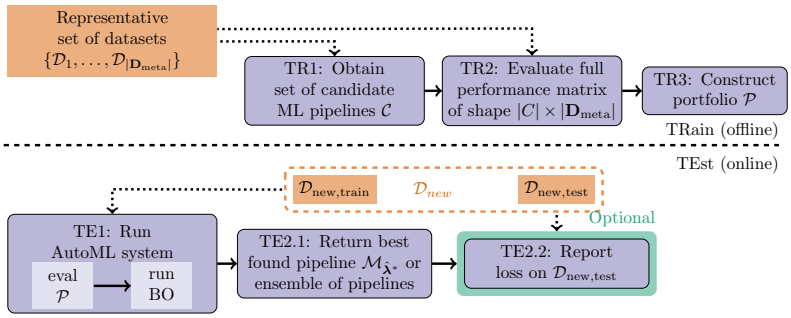


Figure 2: Schematic Overview of *PoSH Auto-sklearn* with the offline portfolio building phase (TR1-TR3) above and the test phase (TE1-TE2) below the dashed line. Rounded, purple boxes refer to computational steps while rectangular, orange boxes depict the input data to the AutoML system.

For each meta-dataset $\mathcal{D}_d \in \mathbf{D}_{\text{meta}}$ we have access to both $\mathcal{D}_{d,\text{train}}$ and $\mathcal{D}_{d,\text{test}}$. In the case of holdout, we split the training set $\mathcal{D}_{d,\text{train}}$ into two smaller disjoint sets $\mathcal{D}^{\text{train}}_{d,\text{train}}$ and $\mathcal{D}^{\text{val}}_{d,\text{train}}$. We usually train models using $\mathcal{D}^{\text{train}}_{d,\text{train}}$ and use $\mathcal{D}^{\text{val}}_{d,\text{train}}$ to choose a ML pipeline $\mathcal{M}_{\boldsymbol{\lambda}}$ from the portfolio by means of the model selection strategy S (instead of holdout we can of course also use cross-validation to compute the validation loss). However, if we instead choose the ML pipeline on the test set $\mathcal{D}_{d,\text{test}}$, Equation 9 becomes a monotone and submodular set function, which results in favorable guarantees for the greedy algorithm that we detail in Section 3.1.2. We follow this approach for the portfolio construction in the offline phase; we emphasize that for a new dataset $\mathcal{D}_{\text{new}}$, we of course do not require access to the test set $\mathcal{D}_{\text{new,test}}$.

3.1.2 Theoretical Properties of the Greedy Algorithm

Besides the already mentioned practical advantages of the proposed greedy algorithm, this algorithm also enjoys a bounded worst-case error.

Proposition 1 *Minimizing the test loss of a portfolio $\mathcal{P}$ on a set of datasets $\mathcal{D}_1, \ldots, \mathcal{D}_{|\mathbf{D}_{meta}|}$, when choosing an ML pipeline from $\mathcal{P}$ for $\mathcal{D}_d$ using holdout or cross-validation based on its performance on $\mathcal{D}_{d,test}$, is equivalent to the* sensor placement problem *for minimizing detection time (Krause et al., 2008).*

We detail this equivalence in Appendix C.2. Thereby, we can apply existing results for the sensor placement problem to our problem. Using the test set of the meta-datasets $\mathbf{D}_{\text{meta}}$ to construct a portfolio is perfectly fine as long as we do not use new datasets $\mathcal{D}_{\text{new}} \in \mathbf{D}_{\text{test}}$ which we use for testing the approach.

Corollary 1 *The penalty function for all meta-datasets is submodular.*

We can directly apply the proof from Krause et al. (2008) that the so-called penalty function (i.e., maximum estimated generalization error minus the observed estimated generalization error) is submodular and monotone to our problem setup. Since linear combinations of

submodular functions are also submodular (Krause and Golovin, 2014), the penalty function is also submodular.

Corollary 2 *The problem of finding an optimal portfolio $\mathcal{P}^*$ is NP-hard (Nemhauser et al., 1978; Krause et al., 2008).*

Corollary 3 *Let R denote the expected penalty reduction of a portfolio across all datasets, compared to the empty portfolio (which yields the worst possible score for each dataset). The greedy algorithm returns a portfolio $\mathcal{P}$ such that $R(\mathcal{P}^*) \geq R(\mathcal{P}) \geq (1 - \frac{1}{e})R(\mathcal{P}^*)$.*

This means that the greedy algorithm closes at least 63% of the gap between the worst ADTM score (1.0) and the score the best possible portfolio $\mathcal{P}^*$ of size $|\mathcal{P}|$ would achieve (Nemhauser et al., 1978; Krause and Golovin, 2014). A generalization of this result given by Krause and Golovin (2014, Theorem 1.5) also tightens this bound to close 99% of the gap between the worst ADTM score and the score the optimal portfolio $\mathcal{P}^*$ of size $|\mathcal{P}^*|$ would achieve, by extending the portfolio constructed by the greedy algorithm to size $5 \cdot |\mathcal{P}|$. Please note that a portfolio of size $5 \cdot |\mathcal{P}|$ could be better than the optimal portfolio of size $|\mathcal{P}|$. This means that we can find a close-to-optimal portfolio on the meta-train datasets $\mathbf{D}_{\text{meta}}$ at the very least. Under the assumption that we apply the portfolio to datasets from the same distribution of datasets, we have a strong set of default ML pipelines.

We could also apply other strategies for the sensor set placement in our setting, such as mixed integer programming strategies, which can solve it optimally; however, these do not scale to portfolio sizes of a dozen ML pipelines (Krause et al., 2008; Pfisterer et al., 2018).

The same proposition (with the same proof) and corollaries apply if we select an ML pipeline based on an intermediate step in a learning curve or use cross-validation instead of holdout. We discuss using the validation set and other model selection and budget allocation strategies in Appendix C.3 and Appendix C.4.

3.2 Budget Allocation using Successive Halving

A key issue we identified during the last AutoML challenge was that training expensive configurations on the complete training set, combined with a low time budget, does not scale well to large datasets. At the same time, we noticed that our (then manual) strategy to run predefined pipelines on subsets of the data already yielded predictions good enough for ensemble building. This questions the common choice of assigning the same amount of resources to all pipeline evaluations, i.e. time, compute and data.

For this reason we introduce the principle of *budget allocation strategies* to AutoML, that describe how the resources are allocated to the pipeline evaluations. This is an orthogonal design decision to the *model selection strategy*, which approximates the generalization error of a single ML pipeline, and which is typically tackled by holdout or K-fold cross-validation (see Section 6.4.1).

As a principled alternative to always using the full budget, we used the successive halving bandit strategy (SH; Karnin et al., 2013; Jamieson and Talwalkar, 2016), which assigns more budget to promising machine learning pipelines and can easily be combined with iterative algorithms.

3.2.1 Approach

AutoML systems evaluate each pipeline under the same resource limitations and on the same budget (e.g., number of iterations using iterative algorithms). To increase efficiency for cases with tight resource limitations, we suggest allocating more resources to promising pipelines by using SH (Karnin et al., 2013; Jamieson and Talwalkar, 2016) to prune poor-performing pipelines aggressively.

Given a minimal and maximal budget per ML pipeline, SH starts by training a fixed number of ML pipelines for the smallest budget. Then, it iteratively selects $\frac{1}{\eta}$ of the pipelines with the lowest generalization error, multiplies their budget by η, and re-evaluates. This process is continued until only a single ML pipeline is left or the maximal budget is spent, and replaces the standard holdout procedure in which every ML pipeline is trained for the full budget.

While SH itself chooses new pipelines $\mathcal{M}_{\boldsymbol{\lambda}}$ to evaluate at random, we aim to extend on our work on *Auto-sklearn 1.0* and continue to use BO. To do so, we follow work combining SH with BO (Falkner et al., 2018).[4] Specifically, we use BO to iteratively suggest new ML pipelines $\mathcal{M}_{\boldsymbol{\lambda}}$, which we evaluate on the lowest budget until a fixed number of pipelines has been evaluated. Then, we run SH as described above. We are using *Auto-sklearn*'s standard random forest-based BO method SMAC and, according to the methodology of Falkner et al. (2018) build the model for BO on the highest available budget for which we have sufficient datapoints. While the original model had a mathematical requirement for $n + 1$ finished pipelines, where n is the number of hyperparameters to be optimized, the random forest model can guide the optimization with fewer datapoints, and we define sufficient as $\frac{n}{2}$. The portfolios we have introduced in Section 3.1 integrate seamlessly into this scheme: as long as not all members of the portfolio have been evaluated, we suggest them instead of asking BO for a new suggestion.

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

We note that SH can be used in combination with both, holdout or cross-validation, and thus indeed adds another hyper-hyperparameter to the AutoML system, namely whether to use SH or FB. However, it also adds more flexibility to tackle a broader range of problems.

3.3 Practical Considerations and Challenge Results

In order to make best use of the successive halving algorithm we had to do certain adjustments to obtain high performance.

First, we restricted the search space to contain only iterative algorithms and no more feature preprocessing. This simplifies the usage of SH as we only have to deal with a single type of fidelity, the number of iterations, while we would otherwise have to also consider dataset subsets as an alternative. This leaves us with extremely randomized trees (Geurts

4. Falkner et al. (2018) proposed using Hyperband (Li et al., 2018) together with BO; however, we use only SH as we expect it to work better in the extreme of having very little time, as it more aggressively reduces the budget per ML pipeline.

et al., 2006), random forests (Breimann, 2001), histogram-based gradient boosting (Friedman, 2001; Ke et al., 2017), a linear model fitted with a passive aggressive algorithm (Crammer et al., 2006) or stochastic gradient descent and a multi-layer perceptron. The exact configuration space can be found in Table 18 of the appendix.

Second, because of using only iterative algorithms, we are able to store partially fitted models to disk to prevent having no predictions in case of time- and memouts. That is, after $2, 4, 8, \ldots$ iterations, we make predictions for the validation set and dump the model for later usage. We provide further details, such as the restricted search space, in Appendix B.

For our submission to the second AutoML challenge, we implemented the following safeguards and tricks (Feurer et al., 2018), which we do not use in this paper since we instead focus on automatically designing a robust AutoML system:

- For the submission, we also employed support vector machines using subsets of the dataset as lower fidelities. Since none of the five final ensembles in the competition contained support vector machines, we did not consider them anymore for this paper, simplifying our methodology.

- We developed an additional library pruning method for ensemble selection. However, in preliminary experiments, we found that this, in the best case, provides an insignificant boost for the area under curve and not balanced accuracy, which we use in this work and thus did not follow up on that any further.

- To increase robustness against arbitrarily large datasets, we reduced all datasets to have at most 500 features using univariate feature selection. Similarly, we also reduced all datasets to have at most 45 000 datapoints using stratified subsampling. We do not think these are good strategies in general and only implemented them because we had no information about the dimensionality of the datasets used in the challenge, and to prevent running out of time and memory. Retrospectively, only one out of five datasets triggered this feature selection step. Now, we have instead a fallback strategy that is defined by data, see Section 4.1.1.

- In case the datasets had less than 1000 datapoints, we would have reverted from holdout to cross-validation. However, this fallback was not triggered due to the datasets being larger in the competition.

- We manually added a linear regression fitted with stochastic gradient descent with its hyperparameters optimized for fast runtime as the first entry in the portfolio to maximize the chances of fitting a model within the given time. We had implemented this strategy because we did not know the time limit of the competition. However, as for the paper at hand and future applications of *Auto-sklearn*, we expect to know the optimization budget we are optimizing the portfolio for, we no longer require such a safeguard.

Our submission, *PoSH Auto-sklearn*, was the overall winner of the second AutoML challenge. We give the results of the competition in Table 1 and refer to Feurer et al. (2018) and Guyon et al. (2019) for further details, especially for information on our competitors.

Name	Rank	Dataset #1	Dataset #2	Dataset #3	Dataset #4	Dataset #5
PoSH Auto-sklearn	**2.8**	0.5533(3)	0.2839(4)	**0.3932**(**1**)	**0.2635**(**1**)	0.6766(5)
narnars0	3.8	0.5418(5)	0.2894(2)	0.3665(2)	0.2005(9)	**0.6922**(**1**)
Malik	5.4	0.5085(7)	0.2297(7)	0.2670(6)	0.2413(5)	0.6853(2)
wlWangl	5.4	**0.5655**(**2**)	**0.4851**(**1**)	0.2829(5)	−0.0886(16)	0.6840(3)
thanhdng	5.4	0.5131(6)	0.2256(8)	0.2605(7)	0.2603(2)	0.6777(4)

Table 1: Results for the second AutoML challenge (Guyon et al., 2019). *Name* is the team name, *Rank* the final rank of the submission, followed by the individual results on the five datasets used in the competition. All performances are the normalized area under the ROC curve (Guyon et al., 2015) with the per-dataset rank in brackets. In case a rank is missing, for example, rank 1 for dataset 1, this rank was achieved by a contestant who did not place within the top 5.

3.4 Experimental Setup

So far, AutoML systems were designed without any optimization budget or with a single, fixed optimization budget T in mind (see Equation 5).[5] Our system takes the optimization budget into account when constructing the portfolio. We will study two optimization budgets: a short, 10 minute optimization budget and a long, 60 minute optimization budget as in the original *Auto-sklearn* paper. To have a single metric for binary classification, multiclass classification and unbalanced datasets, we report the balanced error rate (1 − balanced accuracy), following the 1st AutoML challenge (Guyon et al., 2019). As different datasets can live on different scales, we apply a linear transformation to obtain comparable values. Concretely, we obtain the minimal and maximal error obtained by executing *Auto-sklearn* with portfolios and using ensembles for each combination of model selection and budget allocation strategies per dataset. Then, we rescale by subtracting the minimal error and dividing by the difference between the maximal and minimal error (ADTM, as introduced in Section 3.1.1).[6] With this transformation, we obtain a normalized error which can be interpreted as the regret of our method.

We also limit the time and memory for each ML pipeline evaluation. For the time limit, we allow for at most 1/10 of the optimization budget, while for the memory, we allow the pipeline 4GB before forcefully terminating the execution.

3.4.1 Datasets

We require two disjoint sets of datasets for our setup: (i) $\mathbf{D}_{\text{meta}}$, on which we build portfolios and the model-based policy selector that we will introduce in Section 4, and (ii) $\mathbf{D}_{\text{test}}$, on which we evaluate our method. The distribution of both sets ideally spans a wide variety of problem domains and dataset characteristics. For $\mathbf{D}_{\text{test}}$, we rely on 39 datasets selected for

5. The OBOE AutoML system (Yang et al., 2019) is a potential exception that takes the optimization budget into consideration, but the experiments by Yang et al. (2019) were only conducted for a single optimization budget, not demonstrating that the system adapts itself to multiple optimization budgets.
6. We would like to highlight that this is slightly different than in Section 3.1.1 where we did not have access to the ensemble performance and also only normalized per model selection strategy.

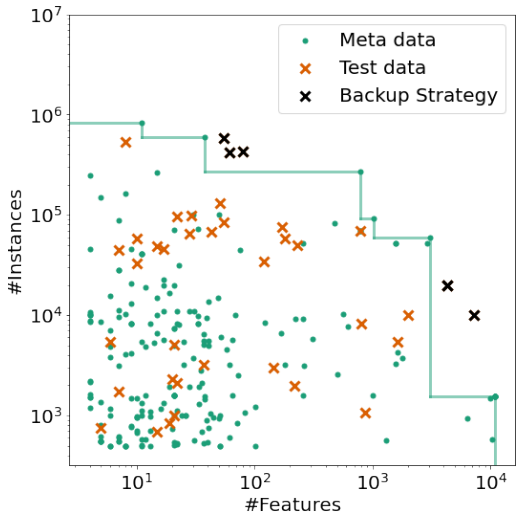


Figure 3: Distribution of meta and test datasets. We visualize each dataset w.r.t. its meta-features and highlight the datasets outside our meta distribution using black crosses.

the AutoML benchmark proposed by Gijsbers et al. (2019), which consists of datasets for comparing classifiers (Bischl et al., 2021) and datasets from the AutoML challenges (Guyon et al., 2019).

We collected the meta datasets $\mathbf{D}_{\text{meta}}$ based on OpenML (Vanschoren et al., 2014) using the OpenML-Python API (Feurer et al., 2021). To obtain a representative set, we considered all datasets on OpenML with more than 500 and less than 1 000 000 samples with at least two attributes. Next, we dropped all datasets that are sparse, contain time attributes or string type attributes as $\mathbf{D}_{\text{test}}$ does not contain any such datasets. Then, we dropped synthetic datasets and subsampled clusters of highly similar datasets. Finally, we manually checked for overlap with $\mathbf{D}_{\text{test}}$ and ended up with a total of 208 training datasets and used them to train our method.

We show the distribution of the datasets in Figure 3. Green points refer to $\mathbf{D}_{\text{meta}}$ and orange crosses to $\mathbf{D}_{\text{test}}$. We can see that $\mathbf{D}_{\text{meta}}$ spans the underlying distribution of $\mathbf{D}_{\text{test}}$ quite well, but several datasets are outside the $\mathbf{D}_{\text{meta}}$ distribution indicated by the green lines, marked with a black cross. We give the full list of datasets for $\mathbf{D}_{\text{meta}}$ and $\mathbf{D}_{\text{test}}$ in Appendix E.

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

3.4.2 Meta-data Generation

For each optimization budget we created four performance matrices of size $|\mathbf{D}_{\text{meta}}| \times |\mathcal{C}|$, see Section 3.1.1 for details on performance matrices. Each matrix refers to one way of assessing the generalization error of a model: holdout, 3-fold CV, 5-fold CV or 10-fold CV. To obtain each matrix, we did the following. For each dataset $\mathcal{D}$ in $\mathbf{D}_{\text{meta}}$, we used combined algorithm selection and hyperparameter optimization to find a customized ML pipeline. In practice, we ran *Auto-sklearn* without meta-learning and without ensemble building three times and picked the best resulting ML pipeline on the test split of $\mathcal{D}$. To ensure that *Auto-sklearn* finds a good configuration, we ran it for ten times the optimization budget given by the user (see Equation 5). Then, we ran the cross-product of all candidate ML pipelines and datasets to obtain the performance matrix. We also stored intermediate results for the iterative algorithms so that we could build custom portfolios for SH, too.

3.4.3 Other Experimental Details

We always report results averaged across 10 repetitions to account for randomness and report the mean and standard deviation over these repetitions. To check whether performance differences are significant, where possible, we ran the Wilcoxon signed-rank test as a statistical hypothesis test with $\alpha = 0.05$ (Demšar, 2006). In addition, we plot the average rank as follows. For each dataset, we draw one run per method (out of 10 repetitions) and rank these draws according to performance, using the average rank in case of ties. We then average over all 39 dataset and repeat this sampling 500 times to and then plot the median and the 10th and 90th percentile of these samples. In the case of only three methods to compare, we can enumerate all 1000 combinations of the seeds and do so. We use the exact method for Figure 6 and the sampling method for Figure 7 in the appendix.

We conducted all experiments using ensemble selection, and we constructed ensembles of size 50 with replacement. We give results without ensemble selection in the Appendix B.2.

All experiments were conducted on a compute cluster with machines equipped with 2 *Intel Xeon Gold 6242 CPUs* with 2.8GHz (32 cores) and 192 GB RAM, running Ubuntu 20.04.01. We provide scripts for reproducing all our experimental results at `https://github.com/automl/ASKL2.0_experiments` and provide a clean integration of our methods into the official *Auto-sklearn* repository.

3.5 Experimental Results

In this subsection, we now validate the improvements for *PoSH Auto-sklearn*. First, we will compare using a portfolio to the previous KND approach and no warmstarting and second, we will compare *PoSH Auto-sklearn* to the previous *Auto-sklearn 1.0*.

3.5.1 Portfolio vs. KND

Here, we study the performance of the learned portfolio and compare it against *Auto-sklearn 1.0*'s default meta-learning strategy using 25 configurations. Additionally, we also study how pure BO would perform. We give results in Table 2.

For the new AutoML-hyperparameter $|\mathcal{P}|$, we chose 32 to allow two full iterations of SH with our hyperparameter setting of SH. Unsurprisingly, warmstarting, in general, improves

AUTO-SKLEARN 2.0

	10 minutes			60 minutes		
	BO	KND	Port	BO	KND	Port
FB; holdout	5.98	5.29	**3.70**	3.84	3.98	**3.08**
SH; holdout	5.15	4.82	**4.11**	3.77	3.55	**3.19**
FB; 3CV	8.52	7.76	**6.90**	6.42	6.31	**4.96**
SH; 3CV	7.82	7.67	**6.16**	6.08	5.91	**5.17**
FB; 5CV	9.48	9.45	**7.93**	6.64	6.47	**5.05**
SH; 5CV	9.48	8.85	**7.05**	6.19	5.83	**5.40**
FB; 10CV	16.10	15.11	**12.42**	10.82	10.44	**9.68**
SH; 10CV	16.14	15.10	**12.61**	10.54	10.33	**9.23**

Table 2: Averaged normalized balanced error rate. We report the aggregated performance across 10 repetitions and 39 datasets of our AutoML system using only Bayesian optimization (*BO*), or BO warmstarted with k-nearest-datasets (*KND*) or a greedy portfolio (*Port*). Per line, we boldface the best mean value (per model selection and budget allocation strategy and optimization budget, and underline results that are not statistically different according to a Wilcoxon-signed-rank Test ($\alpha = 0.05$)).

	10MIN		60MIN	
	∅	std	∅	std
(1) PoSH-Auto-sklearn	**4.11**	0.09	**3.19**	0.12
(2) Auto-sklearn (1.0)	16.21	0.27	7.17	0.30

Table 3: Final performance of *PoSH Auto-sklearn* and *Auto-sklearn 1.0*. We report the normalized balanced error rate averaged across 10 repetitions on 39 datasets. We boldface the best mean value (per optimization budget) and underline results that are not statistically different according to a Wilcoxon signed-rank test ($\alpha = 0.05$).

the performance on all optimization budgets and most model selection strategies, often by a large margin. The portfolios always improve over BO, while KND does so in all but one case. When comparing the portfolios to KND, we find that the raw results are always favorable and that for half of the settings, the differences are also significant.

3.5.2 *PoSH Auto-sklearn* VS *Auto-sklearn 1.0*

We can also look at the performance of *PoSH Auto-sklearn* compared to *Auto-sklearn 1.0*.

First, we compare the performance of *PoSH Auto-sklearn* to *Auto-sklearn 1.0* using the full search space, and we provide those numbers in Table 3. For both time horizons, there is a strong reduction in the loss (10min: $16.21 \rightarrow 4.11$ and 60min: $7.17 \rightarrow 3.19$), indicating that the proposed *PoSH Auto-sklearn* is indeed an improvement over the existing solution and is able to fit better machine learning models in the given time limit.

Second, we compare the performance of *PoSH Auto-sklearn* (SH; holdout and Port) to *Auto-sklearn 1.0* (FB; holdout and KND) using only the reduced search space based on the results in Table 2. Again, there is a strong reduction in the loss for both time horizons (10min: 5.29 → 4.11 and 60min: 3.98 → 3.19), confirming abovementioned findings. Combined with the portfolio, the average results are inconclusive about whether our use of successive halving was the right choice or whether plain holdout would have been better. We also provide the raw numbers in Appendix B.3, but they are inconclusive, too.

4. Part II: Automating Design Decisions in AutoML

The goal of AutoML is to yield state-of-the-art performance without requiring the user to make low-level decisions, e.g., which model and hyperparameter configurations to apply. Using a portfolio and SH, *PoSH Auto-sklearn* is already an improvement over *Auto-sklearn 1.0* in terms of efficiency and scalability. However, high-level design decisions, such as choosing between cross-validation and holdout or whether to use SH or not, remain. Thus, *PoSH Auto-sklearn*, and AutoML systems in general, suffer from a similar problem as they are trying to solve, as users have set their arguments on a per-dataset basis manually.

To highlight this dilemma, in Figure 4 we show exemplary results comparing the balanced error rates of the best ML pipeline found by searching our configuration space with BO using holdout, 3CV, 5CV and 10CV with SH and FB on different optimization budgets and datasets. The top row shows results obtained using the same optimization budget of 10 minutes on two different datasets. While *FB; 10CV* is best on dataset *sylvine* (top left) the same strategy on median performs amongst the worst strategies on dataset *adult* (top right). Also, on *sylvine*, SH performs overall slightly worse in contrast to *adult*, where SH performs better on average. The bottom rows show how the given time-limit impacts the performance on the dataset *jungle_chess_2pcs_raw_endgame_complete*. Using a quite restrictive optimization budget of 10 minutes (bottom left), *SH; 3CV*, which aggressively cuts ML pipelines on lower budgets, performs best on average. With a higher optimization budget (bottom right), the overall results improve and more strategies become competitive.

Therefore, we propose to extend AutoML systems with a policy selector to automatically choose an optimization policy given a dataset (see Figure 1 in Section 1 for a schematic overview). In this second part, we discuss the resulting approach, which led to *Auto-sklearn 2.0* as the first implementation of it.

4.1 Automated Policy Selection

Specifically, we consider the case, where an AutoML system can be run with different optimization policies $\pi \in \Pi$ and study how to further automate AutoML using algorithm selection on this meta-meta level. In practice, we extend the formulation introduced in Equation 7 to not use an AutoML system $\mathcal{A}_\pi$ with a fixed policy π, but to contain a policy selector $\Xi : \mathcal{D} \rightarrow \pi$:

$$\widehat{GE}(\mathcal{A}, \Xi, \mathbf{D}_{\text{meta}}) = \frac{1}{\mid \mathbf{D}_{\text{meta}} \mid} \sum_{d=1}^{|\mathbf{D}_{\text{meta}}|} \widehat{GE}(\mathcal{A}_{\Xi(\mathcal{D}_d)}(\mathcal{D}_d), \mathcal{D}_d). \tag{10}$$

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

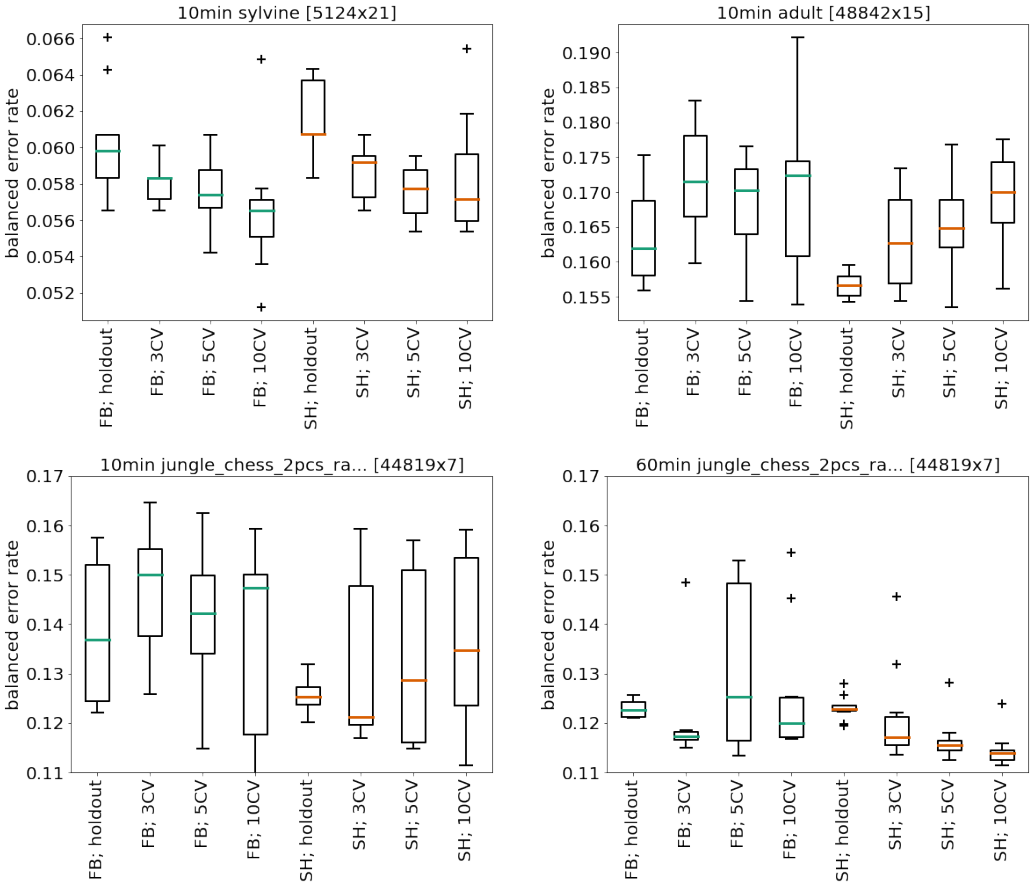


Figure 4: Final balanced error rate of BO using different model selection strategies averaged across 10 repetitions. Top row: Results for a optimization budget of 10 minutes on two different datasets. Bottom row: Results for a optimization budget of 10 and 60 minutes on the same dataset.

4.1.1 Approach

AutoML systems themselves are often heavily hyperparameterized. In our case, we deem the model selection strategy and budget allocation strategy (see Sections 3.2 and 6.4.1) as important choices the user has to make when using an AutoML system to obtain high performance. These decisions depend on both the given dataset and the available resources. As there is also an interaction between the two strategies and the optimal portfolio $\mathcal{P}$, we consider here that the optimization policy π is parameterized by a combination of (i) model selection strategy, (ii) budget allocation strategy and (iii) a portfolio constructed for the choice of the two strategies. In our case, these are eight different policies ({3-fold CV, 5-fold CV, 10-fold CV, holdout} $\times$ {SH, FB}).

We introduce a new layer on top of AutoML systems that automatically selects a policy π for a new dataset. We show an overview of this system in Figure 5 which consists of a

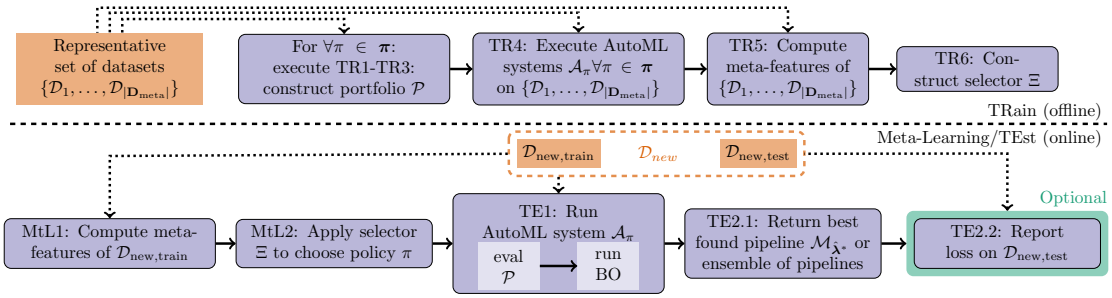


Figure 5: Schematic overview of the proposed *Auto-sklearn 2.0* system with the training phase (TR1–TR6) above and the test phase (MtL1–MtL2&TE1–TE2) below the dashed line. Rounded, purple boxes refer to computational steps, while rectangular, orange boxes depict the input data to the AutoML system.

training (TR1–TR6) and a testing stage (MtL1–2 and TE1–TE2). In brief, in training steps TR1–TR3, we perform the same steps that we have already outlined in Figure 2. However, we now do so for each combination of model selection and budget allocation strategy. Our policies are combinations of a portfolio, a model selection strategy and a budget allocation strategy. We then execute the full AutoML system for each such policy in step TR4 to obtain a realistic performance estimate. In step TR5, we compute meta-features and use them together with the performance estimate from TR4 in step TR6 to train a model-based policy selector Ξ, which we will use in the online test phase.

In order to not overestimate the performance of π on a dataset $\mathcal{D}_d$, dataset $\mathcal{D}_d$ must not be part of the meta-data for constructing the portfolio. To overcome this issue, we perform an inner 5-fold cross-validation and build each π on four fifths of the *meta*-datasets $\mathbf{D}_{\text{meta}}$ and evaluate it on the left-out fifth of *meta*-datasets $\mathbf{D}_{\text{meta}}$. For the final AutoML system we then use a portfolio built on all *meta*-datasets $\mathbf{D}_{\text{meta}}$.

For a new dataset $\mathcal{D}_{\text{new}} \in \mathbf{D}_{\text{test}}$, we first compute meta-features describing $\mathcal{D}_{\text{new}}$ (MtL1) and use the model-based policy selector from step TR6 to automatically select an appropriate policy for $\mathcal{D}_{\text{new}}$ based on the meta-features (MtL2). This will relieve users from making this decision on their own. Given an optimization policy π, we then apply the AutoML system $\mathcal{A}_\pi$ to $\mathcal{D}_{\text{new}}$ (TE1). Finally, we return the best found pipeline $\mathcal{M}_{\hat{\boldsymbol{\lambda}}^*}$ based on the training set of $\mathcal{D}_{\text{new}}$ (TE2.1). Optionally, we can then compute the loss of $\mathcal{M}_{\hat{\boldsymbol{\lambda}}^*}$ on the test set of $\mathcal{D}_{\text{new}}$ (TE2.2); we emphasize that this would be the only time we ever access the test set of $\mathcal{D}_{\text{new}}$. Steps TE1–TE2 are the same as in Figure 2, and the only difference at evaluation time is that we use algorithm selection to decide which policy π to use at test time instead of relying on a hand-picked one.

In the following, we describe two ways to construct a policy selector and introduce an additional backup strategy to make it robust towards failures.

Constructing the single best policy A straightforward way to construct a selector relies on the assumption that the *meta*-datasets $\mathbf{D}_{\text{meta}}$ are homogeneous and that a new dataset is similar to these. In such a case, we can use per-set algorithm selection (Kerschke et al., 2019), which aims to find the single algorithm that performs best on average on a set

hyperparameter	type	values
Min. number of samples to create a further split	int	$[3, 20]$
Min. number of samples to create a new leaf	int	$[2, 20]$
Max. depth of a tree	int	$[0, 20]$
Max. number of features to be used for a split	int	$[1, 2]$
Bootstrapping in the random forest	cat	$\{yes, no\}$
Soft or hard voting when combining models	cat	$\{soft, hard\}$
Error value scaling to compute dataset weights	cat	see text

Table 4: configuration space of the model-based policy selector.

of problem instances. In our context, it aims to find the combination of model selection and budget allocation that is best on average for the given set of *meta*-datasets $\mathbf{D}_{\text{meta}}$. This single best policy is then the automated replacement for our manual selection of SH and holdout in *PoSH Auto-sklearn*. While this seems to be a trivial baseline, it actually requires the same amount of compute power as the more elaborate strategy we introduce next.

Constructing the per-dataset Policy Selector Instead of using a fixed, learned policy, we now propose to adapt the policy to the dataset at hand by using per-instance algorithm selection, which means we select the appropriate algorithm for each dataset by taking its properties into account. To construct the meta selection model (TR6), we follow the policy selector design of HydraMIP (Xu et al., 2011): for each pair of AutoML policies, we fit a random forest to predict whether policy π_A outperforms policy π_B given the current dataset's meta-features. Since the misclassification loss depends on the difference between the losses of the two policies (i.e. the ADTM when choosing the wrong policy), we weight each meta-observation by their loss difference. To make errors comparable across different datasets (Bardenet et al., 2013), we scale the individual error values for each dataset. At test time (TE2), we query all pairwise models for the given meta-features and use voting for Ξ to choose a policy π. We will refer to this strategy as the *Policy Selector*.

To improve the performance of the model-based policy selector, we applied BO to optimize the model-based policy selector's hyperparameters to minimize the cross-validation error (Lindauer et al., 2015). We optimized in total seven hyperparameters, five of which are related to the random forest, one is how to combine the pairwise models to get a prediction, and the last one is the strategy of how to scale error values to compute weights for comparing datasets, i.e. using the raw observations, scale with $[min, max]$ / $[min, 1]$ across a pair or all policies or use the difference in ranks as the weight (see Table 4). Hyperparameters are shared between all pairwise models to avoid factorial growth of the number of hyperparameters with the number of new model selection strategies. We allow a tree depth of 0, i.e., a tree with all data in a single leaf, which is equivalent to the single best strategy described above.

Meta-Features. To train our model-based policy selector and to select a policy, as well to use the backup strategy, we use meta-features (Brazdil et al., 2008; Vanschoren, 2019) describing all meta-train datasets (TR5) and new datasets (TE1). To avoid the problems discussed in Section 3.1 we only use very simple and robust meta-features, which can be

reliably computed in linear time for every dataset: 1) the number of datapoints and 2) the number of features. In fact, these are already stored as meta-data for the data structure holding the dataset. Using only these two meta-features for the selector can be regarded as learning the manually-designed fallbacks that we discussed in Section 3.3. In our experiments, we will show that even with only these trivial and cheap meta-features, we can substantially improve over a static policy.

Backup strategy. Since there is no guarantee that our model-based policy selector will extrapolate well to datasets outside of the meta-datasets, we implement a fallback measure to avoid failures. Such failures can be harmful if a new dataset is, e.g., much larger than any dataset in the meta-dataset, and the model-based policy selector proposes to use a policy that would time out without any solution. More specifically, if there is no dataset in the meta-datasets that has higher or equal values for each meta-feature (i.e. dominates the dataset meta-features), our system falls back to use *holdout* with SH, which is the most aggressive and cheapest policy we consider. We visualize this in Figure 3 where we mark datasets outside our meta distribution using black crosses.

4.2 Experimental Results

To study the performance of the policy selector, we compare it to *PoSH Auto-sklearn* as described in Section 3 and *Auto-sklearn 1.0*. From now on we refer to *PoSH Auto-sklearn* + policy selector as *Auto-sklearn 2.0*. As before, we study two horizons, 10 minutes and 60 minutes, and use versions of *PoSH Auto-sklearn* and *Auto-sklearn 2.0* that were constructed with these time horizons in mind. Similarly, we use the same 208 datasets for constructing our AutoML systems and the same 39 for evaluating them.

Looking at Table 5, we see that *Auto-sklearn 2.0* achieves the lowest error, being significantly better for both optimization budgets. Most notably, *Auto-sklearn 2.0* reduces the relative error compared to *Auto-sklearn 1.0* by 78% (10MIN) and 65%, respectively, which means a reduction by a factor of 4.5 and three.

It turns out that these results are skewed by several large datasets (task IDs 189873 and 75193 for both horizons; 189866, 189874, 168796 and 168797 only for the ten minutes horizon) on which the KND initialization of *Auto-sklearn 1.0* only suggests ML pipelines

	10MIN		60MIN	
	∅	std	∅	std
(1) Auto-sklearn (2.0)	**3.58**	0.23	**2.47**	0.18
(2) PoSH-Auto-sklearn	4.11	0.09	3.19	0.12
(3) Auto-sklearn (1.0)	16.21	0.27	7.17	0.30

Table 5: Average normalized balanced error (ADTM, lower is better) of *Auto-sklearn 2.0*, *PoSH Auto-sklearn* and *Auto-sklearn 1.0* averaged across 10 repetitions on 39 datasets. We boldface the best mean value (per optimization budget) and underline results that are not statistically different according to a Wilcoxon-signed-rank Test ($\alpha = 0.05$).

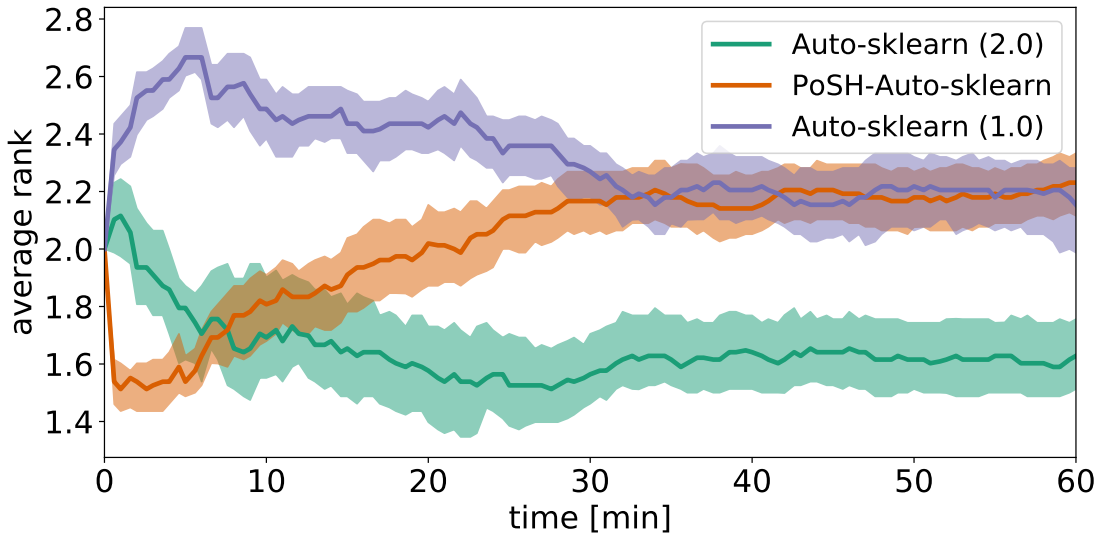


Figure 6: Performance over time. We report the median ranks (lower is better) and the 10th and 90th percentiles over time for *Auto-sklearn 2.0* and the previous AutoML systems. Concretely, we compute the mean rank for for all 39 for all 1000 combinations of the 10 seeds of the 3 AutoML systems, and compute the median and percentiles of these 1000 average ranks.

that time out or hit the memory limit and thus exhaust the optimization budget for the full configuration space. Our new AutoML system does not suffer from this problem as it a) selects SH to avoid spending too much time on unpromising ML pipelines and b) can return predictions and results even if an ML pipeline was not evaluated for the full budget or converged early; and even after removing the datasets in question from the average, the performance of *Auto-sklearn 1.0* is substantially worse than that *Auto-sklearn 2.0*.

When looking at the intermediate system, i.e. *PoSH Auto-sklearn*, we find that it outperforms *Auto-sklearn 1.0* in terms of the normalized balanced error rate, but that the additional step of selecting the model selection and budget allocation strategy gives *Auto-sklearn 2.0* an edge. When not considering the large datasets *Auto-sklearn 1.0* failed on, their performance becomes very similar.

Figure 6 provides another view on the results, presenting average ranks (where failures obtain less weight compared to the averaged performance). *Auto-sklearn 2.0* is still able to deliver best results, *PoSH Auto-sklearn* should be preferred to *Auto-sklearn 1.0* for the first 30 minutes and then converges to roughly the same ranking.

4.3 Ablation

Now, we study the contribution of each of our improvements in an ablation study. We iteratively disable one component and compare the performance to the entire system using the 39 datasets from the AutoML benchmark as done in the previous experimental sections.

These components are (1) using a per-dataset model-based policy selector to choose a policy, (2) using only a subset of the available policies, and (3) warmstarting BO with a portfolio.

4.3.1 Do we need per-dataset selection?

We first examine how much performance we gain by having a model-based policy selector to decide between different AutoML strategies based on meta-features and how to construct this model-based policy selector, or whether it is sufficient to select a single strategy based on meta-training datasets. We compare the performance of the entire system using a model-based policy selector to using a single, static strategy (single best) and both, the model-based policy selector and the single best, without the fallback mechanism for out-of-distribution datasets and give all results in Table 6. We also provide two additional baselines: a random baseline, which randomly assigns a policy to a run and an oracle baseline, which marks the lowest possible error that can be achieved by any of the policies.[7]

First, we compare the performance of the model-based policy selector with the single best. We can observe that for 10 minutes, there is a slight improvement in terms of performance, while the performance for 60 minutes is almost equal. While there is no significant difference to the single best for 10 minutes, there is for 60 minutes. These numbers can be compared with Table 2 to see how we fare against picking a single policy by hand. We find that our proposed algorithm selection compares favorably, especially for the longer time horizon.

Second, to study how much resources we need to spend on generating training data for our model-based policy selector, we consider three approaches: (P) only using the portfolio performance which we pre-computed and stored in the performance matrices as described in Section 3.1.1, (P+BO) actually running *Auto-sklearn* using the portfolio and BO for 10 and 60 minutes, respectively, and (P+BO+E) additionally also constructing ensembles, which yields the most realistic meta-data. Running BO on all 208 datasets (P+BO) is by far more expensive than the table lookups (P); building an ensemble (P+BO+E) adds only several seconds to minutes on top compared to (P+BO).

For both optimization budgets using P+BO yields the best results using the model-based policy selector closely followed by P+BO+ENS, see Table 6. The cheapest method, P, yields the worst results showing that it is worth investing resources into computing good meta-data. Surprisingly, looking at the single best, performance gets worse when using seemingly better meta-data. We investigated the reason why P+BO performs slightly better than P+BO+ENS. When using a model-based policy selector, this can be explained by a single dataset for both time horizons for which the policy chosen by the model-based policy selector is worse than the single best policy. When looking at the single best, there is no single dataset which stands out. To summarize, investing additional resources to compute realistic meta-data results in improved performance, but so far, it appears that having the effect of BO in the meta-data is sufficient, while the ensemble seems to lead to lower meta-data quality.

7. We would like to note that the oracle performance can be unequal to zero because we normalize the results using the single best test loss found for a single model to normalize the results. When evaluating the best policy on a dataset, this most likely results in selecting a model on the validation set that is not the single best model on the test set we use to normalize data.

AUTO-SKLEARN 2.0

	10 Min			60 Min		
trained on	P	P+BO	P+BO+E	P	P+BO	P+BO+E
model-based policy selector	3.58	**3.56**	3.58	2.53	**2.32**	2.47
model-based policy selector w/o fallback	5.43	5.68	4.79	4.98	5.36	5.43
single best	3.88	3.67	3.69	2.49	2.38	2.44
single best w/o fallback	5.18	6.38	6.40	5.10	5.01	5.07
oracle		2.33			1.22	
random		8.32			6.18	

Table 6: Average normalized balanced error (ADTM, lower is better) for 10 and 60 minutes. We report the performance for the *model-based policy selector* policy and the *single best* when trained on different data obtained on $\mathbf{D}_{\text{meta}}$ (P = Portfolio, BO = Bayesian Optimization, E = Ensemble) as well as the *model-based policy selector without the fallback*. The second part of the table shows the results of always choosing the best policy on the test set (*oracle*) and results for choosing a random policy (*random*) as baselines. We boldface the best mean value (per optimization budget) and underline results that are not statistically different according to a Wilcoxon-signed-rank Test ($\alpha = 0.05$).

Finally, we also take a closer look at the impact of the fallback mechanism to verify that our improvements are not solely due to this component. We observe that the performance drops for all policy selection strategies that do not use the fallback mechanism. For the shorter 10 minutes setting, we find that the model-based policy selector still outperforms the single best, while for the longer 60 minutes setting, the single best leads to better performance. The rather stark performance degradation compared to the regular model-based policy selector can mainly be explained by a few, huge datasets, to which the model-based policy selector cannot extrapolate (and which the single best does not account for). Based on these observations, we suggest research into an adaptive fallback strategy that can change the model selection strategy during the execution of the AutoML system so that a policy selector can be used on out-of-distribution datasets. We conclude that using a model-based policy selector is beneficial, and using a fallback strategy to cope with out-of-distribution datasets can substantially improve performance.

4.3.2 Do we need different model selection strategies?

Next, we study whether we need the different model selection strategies. For this, we build model-based policy selectors on different subsets of the available eight combinations of model selection strategies and budget allocations: {3-fold CV, 5-fold CV, 10-fold CV, holdout} × {SH, FB}. *Only Holdout* consists of holdout with SH or FB (2 combinations), *Only CV* comprises 3-fold CV, 5-fold CV and 10-fold CV, all of them with SH or FB (6 combinations), *FB* contains both holdout and cross-validation and assigns each pipeline evaluation the same budget (4 combinations) and *Only SH* uses SH to assign budgets (4 combinations).

		Selector		Random		Oracle	
		∅	std	∅	std	∅	std
10 Min	All	3.58	0.23	7.46	2.02	**2.33**	0.06
	Only Holdout	4.03	0.14	**3.78**	0.23	3.23	0.10
	Only CV	6.11	0.11	8.66	0.70	5.28	0.06
	Only FB	**3.50**	0.20	7.64	2.00	2.59	0.09
	Only SH	3.63	0.19	6.95	1.98	2.75	0.07
60 Min	All	2.47	0.18	5.64	1.95	**1.22**	0.08
	Only Holdout	3.18	0.15	**3.13**	0.12	2.62	0.07
	Only CV	5.09	0.19	6.85	0.86	3.94	0.10
	Only FB	**2.39**	0.18	5.46	1.52	1.51	0.06
	Only SH	2.44	0.24	5.13	1.72	1.68	0.12

Table 7: Average Normalized balanced error (ADTM, lower is better) for the full system and when not considering all model selection strategies.

In Table 7, we compare the performance of selecting a policy at random (random), the performance of selecting the best policy on the test set and thus giving a lower bound on the ADTM (oracle) and our model-based policy selector. The oracle indicates the best possible performance with each of these subsets of model selection strategies. It turns out that both *Only Holdout* and *Only CV* have a much worse oracle performance than *All*, with the oracle performance of *Only CV* being even worse than the performance of the model-based policy selector for *All*. Looking at *Full budget* (FB), it turns out that this subset would be slightly preferable in terms of performance with a policy selector. However, the oracle performance is worse than that of *All* which shows that there is some complementarity between the different policies which cannot yet be exploited by the policy selector. For *Only Holdout*, surprisingly, the random policy selector performs slightly better than the model-based policy selector. We attribute this to the fact that holdout with both SH and FB performs similarly and that the choice between these two cannot yet be learned, possibly also indicated by the close performance of the random selector.

These results show that a large variety of available model selection strategies to choose from increases best possible performances. However, they also show that a model-based policy selector cannot yet necessarily leverage this potential. This questions the usefulness of choosing from all model selection strategies, similar to a recent finding which proves that increasing the number of different policies a policy selector can choose from leads to reduced generalization (Balcan et al., 2021). However, we believe this points to the research question of whether we can learn on the meta-datasets which model selection and budget allocation strategies to include in the set of strategies to choose from. Also, with an ever-growing availability of meta-datasets and continued research on robust policy selectors, we expect this flexibility to eventually yield improved performance.

		10min		60min	
		∅	std	∅	std
With Portfolio	Policy selector	**3.58**	0.23	2.47	0.18
	Single best	3.69	0.14	**2.44**	0.12
Without Portfolio	Policy selector	5.63	0.89	3.42	0.32
	Single best	5.37	0.58	3.61	0.61

Table 8: Average normalized balanced error (ADTM, lower is better) after 10 and after 60 minutes with portfolios (top) and without (bottom). The row "with portfolio" and "policy selector" constitutes the full AutoML system including portfolios, BO and ensembles) and the row "without portfolios" and "policy selector" only removes the portfolios (both from the meta-data for model-based policy selector construction and at runtime). We boldface the best mean value (per optimization budget) and underline results that are not statistically different according to a Wilcoxon-signed-rank Test ($\alpha = 0.05$).

4.3.3 Do we still need to warm-start Bayesian optimization?

Last, we analyze the impact of the portfolio. Given the other improvements, we now discuss whether we still need to add the additional complexity and invest resources to warm-start BO (and can therefore save the time to build the performance matrices to construct the portfolios). For this study, we completely remove the portfolio from our AutoML system, meaning that we directly start with BO and construct ensembles – both for creating the data we train our policy selector on and for reporting performance. We report the results in Table 8.

Comparing the performance of an AutoML system with a model-based policy selector with and without portfolios (Row 1 and 3), there is a clear drop in performance when disabling the portfolios. Comparing Rows 2 and 4 also demonstrates that a portfolio is necessary when using the single best policy. This ablation highlights the importance of initializing the search procedure of AutoML systems with well-performing pipelines.

5. Comparison to other AutoML systems

Having established that *Auto-sklearn 2.0* does indeed improve over *Auto-sklearn 1.0*, we now compare our system to other well established AutoML systems. For this, we use the publicly available AutoML benchmark suite which defines a fixed benchmarking environment for AutoML systems (Gijsbers et al., 2019) comparisons. We use the original implementation of the benchmark and compare *Auto-sklearn 1.0* and *Auto-sklearn 2.0* to the provided implementations of *Auto-WEKA* (Thornton et al., 2013), *TPOT* (Olson et al., 2016a,b), *H2O* AutoML (LeDell and Poirier, 2020) and a random forest baseline with hyperparameter tuning on 39 datasets as implemented by the benchmark suite. These 39 datasets are the same datasets as in $\mathbf{D}_{\text{test}}$ and we provide details in Table 20 in the appendix.

5.1 Integration and setup

To avoid hardware-dependent performance differences, we (re-)ran all AutoML systems on our local hardware (see Section 3.4.3). We used the pre-defined *1h8c* setting, which divides each dataset into ten folds and gives each framework one hour on eight CPU cores to produce a final model. We furthermore assigned each run 32GB of RAM, which a SLURM cluster manager controlls. In addition, we conducted five repeats to account for randomness. The benchmark comes with *Docker* containers (Merkel, 2014). However, *Docker* requires superuser access on the execution nodes, which is not available on our compute cluster. Therefore, we extended the AutoML benchmark with support for *Singularity* images (Kurtzer et al., 2017), and used them to isolate the framework installations from each other. For reproducibility, we give the exact versions we used in Table 17 in the Appendix.

The default resource allocation of the AutoML benchmark is a highly parallel setting with eight cores. We chose the most straightforward way of making use of these resources for *Auto-sklearn* and evaluated eight ML pipelines in parallel, assigning each $total_memory/num_cores$ RAM, which are 4GB. This allows us to evaluate configurations obtained from the portfolio or KND in parallel but also requires a parallel strategy for running BO afterwards. We extended the Bayesian optimization package SMAC3 (Lindauer et al., 2022) to allow for asynchronous parallel optimization. In preliminary experiments, we found that the inherent randomness of the random forest used by SMAC combined with the interleaved random search of SMAC is sufficient to obtain results that perform a lot better than the previous parallelism implemented in *Auto-sklearn* via SMAC (Ramage, 2015). Whenever a pipeline finishes training, *Auto-sklearn* checks whether there is an instance of the ensemble construction running, and if not, it uses one of the eight slots to conduct ensemble building and otherwise continues to fit a new pipeline. We implemented this version of parallel *Auto-sklearn* using *Dask* (Dask Development Team, 2016).

5.2 Results

We give results for the AutoML benchmark in Table 9. For each dataset, we give the average performance of the AutoML systems across all ten folds and five repetitions and boldface the one with the lowest error (we cannot give any information about whether differences are significant as we cannot compute significances on cross-validation folds as described by Bengio and Grandvalet, 2004).

We report the log loss for multiclass datasets and $1 - AUC$ for binary datasets (lower is better). In addition, we provide the average rank as an aggregate measure (computed by averaging all folds and repetitions per dataset and then computing the rank). Furthermore, we count how often each framework is the winner on a dataset (champion), and give the losses, wins and ties against *Auto-sklearn 2.0*. We then use these to perform a binomial sign test (Demšar, 2006) to compare the individual algorithms against *Auto-sklearn 2.0*.

The results in Table 9 show that none of the AutoML systems is best on all datasets, and even the TunedRF performs best on a few datasets. However, we can also observe that the proposed *Auto-sklearn 2.0* has the lowest average rank. It is followed by *H2O* AutoML and *Auto-sklearn 1.0* which perform roughly en par wrt the ranking scores and the number of times they are the winner on a dataset. According to both aggregate metrics, the TunedRF, *Auto-WEKA* and *TPOT* cannot keep up and lead to substantially worse results. Finally,

AUTO-SKLEARN 2.0

	AS 2.0	AS 1.0	AW	TPOT	H2O	TunedRF
adult	0.0692	0.0701	0.0920	0.0750	**0.0690**	0.0902
airlines	0.2724	0.2726	0.3241	0.2758	**0.2682**	-
albert	0.2413	**0.2381**	-	0.2681	0.2530	0.2616
amazon	0.1233	0.1412	0.1836	0.1345	**0.1218**	0.1377
apsfailure	0.0085	**0.0081**	0.0365	0.0099	0.0081	0.0087
australian	**0.0594**	0.0702	0.0709	0.0670	0.0607	0.0610
bank-marketing	**0.0607**	0.0616	0.1441	0.0664	0.0610	0.0692
blood-transfusion	**0.2428**	0.2474	0.2619	0.2761	0.2430	0.3122
car	**0.0012**	0.0046	0.1910	2.7843	0.0032	0.0421
christine	0.1821	**0.1703**	0.2026	0.1821	0.1763	0.1908
cnae-9	**0.1424**	0.1779	0.7045	0.1483	0.1807	0.3119
connect-4	0.3387	0.3535	1.7083	0.3856	**0.3127**	0.4777
covertype	**0.1103**	0.1435	3.3515	0.5332	0.1281	-
credit-g	0.2031	0.2159	0.2505	0.2144	0.2078	**0.1985**
dilbert	0.0399	**0.0332**	2.0791	0.1153	0.0359	0.3283
dionis	**0.5620**	0.7171	-	-	4.7758	-
fabert	0.7386	0.7466	5.4784	0.8431	**0.7274**	0.8060
fashion-mnist	**0.2511**	0.2524	0.9505	0.4314	0.2762	0.3613
guillermo	0.0945	**0.0871**	0.1251	0.1680	0.0911	0.0973
helena	**2.4974**	2.5432	14.3523	2.8738	2.7578	-
higgs	**0.1824**	0.1846	0.3379	0.1969	0.1846	0.1966
jannis	0.6709	**0.6637**	2.9576	0.7244	0.6695	0.7288
jasmine	0.1141	0.1196	0.1356	0.1123	0.1141	**0.1118**
jungle_chess	0.2104	0.1956	1.6969	0.9557	**0.1479**	0.4020
kc1	0.1611	0.1594	0.1780	**0.1530**	0.1745	0.1590
kddcup09	**0.1580**	0.1632	-	0.1696	0.1636	0.2058
kr-vs-kp	**0.0001**	0.0003	0.0217	0.0003	0.0002	0.0004
mfeat-factors	**0.0726**	0.0901	0.5678	0.1049	0.1009	0.2091
miniboone	**0.0121**	0.0128	0.0352	0.0177	0.0129	0.0183
nomao	**0.0035**	0.0039	0.0157	0.0047	0.0036	0.0049
numerai28.6	0.4696	0.4705	0.4729	0.4741	**0.4695**	0.4792
phoneme	**0.0299**	0.0366	0.0416	0.0307	0.0325	0.0347
riccardo	0.0002	**0.0002**	0.0020	0.0021	0.0003	0.0002
robert	1.4302	**1.3800**	-	1.8600	1.4927	1.6877
segment	**0.1482**	0.1749	1.2497	0.1660	0.1580	0.1718
shuttle	**0.0002**	0.0004	0.0100	0.0008	0.0004	0.0006
sylvine	0.0105	0.0091	0.0290	**0.0075**	0.0106	0.0159
vehicle	0.3341	0.3754	2.0662	0.4402	**0.3067**	0.4839
volkert	**0.7477**	0.7862	3.4235	0.9852	0.8121	0.9792
Rank	**1.79**	2.64	5.72	4.08	2.38	4.38
Best performance	**19**	8	0	2	8	2
Wins/Losses/Ties of AS 2.0	-	28/11/0	39/0/0	35/4/0	26/13/0	36/3/0
P-values (AS 2.0 vs. other methods), based on a Binomial sign test	-	.009	$< .000$	$< .000$	.053	$< .000$

Table 9: Results of the AutoML benchmark averaged across five repetitions. We report log loss for multiclass datasets and $1 - AUC$ for binary classification datasets (lower is better). AS is short for *Auto-sklearn* and AW for *Auto-WEKA*. *Auto-sklearn* has the best overall rank, the best performance in most datasets and, based on the P-values of a Binomial sign test, we gain further confidence in its strong performance.

both versions of *Auto-sklearn* appear to be quite robust as they reliably provide results on all datasets, including the largest ones where several of the other methods fail.

6. Related Work

We now present related work on our individual contributions (portfolios, model selection strategies, and algorithm selection) as well as on related AutoML systems.

6.1 Related Work on Portfolios

Portfolios were introduced for hard combinatorial optimization problems, where the runtime between different algorithms varies drastically and allocating time shares to multiple algorithms instead of allocating all available time to a single one reduces the average cost for solving a problem (Huberman et al., 1997; Gomes and Selman, 2001), and had applications in different sub-fields of AI (Smith-Miles, 2008; Kotthoff, 2014; Kerschke et al., 2019). Algorithm portfolios were introduced to ML by the name of *algorithm ranking* to reduce the required time to perform model selection compared to running all algorithms under consideration (Brazdil and Soares, 2000; Soares and Brazdil, 2000), ignoring redundant ones (Brazdil et al., 2001). ML portfolios can be superior to hyperparameter optimization with Bayesian optimization (Wistuba et al., 2015b), Bayesian optimization with a model which takes past performance data into account (Wistuba et al., 2015a) or can be applied when there is simply no time to perform full hyperparameter optimization (Feurer et al., 2018). Furthermore, such a portfolio-based model-free optimization is both easier to implement than regular Bayesian optimization and meta-feature based solutions, and the portfolio can be shared easily across researchers and practitioners without the necessity of sharing meta-data (Wistuba et al., 2015a,b; Pfisterer et al., 2018) or additional hyperparameter optimization software. Here, our goal is to have strong hyperparameter settings when there is no time to optimize with a typical black-box algorithm.

The efficient creation of algorithm portfolios is an active area of research with the Greedy Algorithm being a popular choice (Xu et al., 2010, 2011; Seipp et al., 2015; Wistuba et al., 2015b; Lindauer et al., 2017; Feurer et al., 2018; Feurer and Hutter, 2018) due to its simplicity. Wistuba et al. (2015b) first proposed the use of the Greedy Algorithm for pipelines of ML portfolios, minimizing the average rank on meta-datasets for a single ML algorithm. Later, they extended their work to update the members of a portfolio in a round-robin fashion, this time using the average normalized misclassification error as a loss function and relying on a Gaussian process model (Wistuba et al., 2015a). The loss function of the first method does not optimize the metric of interest, while the second method requires a model and does not guarantee that well-performing algorithms are executed early on, which could be harmful under time constraints.

Research into the Greedy Algorithm continued after our submission to the second AutoML challenge and the publication of the employed methods (Feurer et al., 2018). Pfisterer et al. (2018) suggested using a set of default values to simplify hyperparameter optimization. They argued that constructing an optimal portfolio of hyperparameter settings is a generalization of the *Maximum coverage problem* and propose two solutions based on *Mixed Integer Programming* and the *Greedy Algorithm* which we also use as the base of our algorithm. The greedy algorithm recently also drew interest in deep learning research,

where it was applied in its basic form for tuning the hyperparameters of the popular ADAM algorithm (Metz et al., 2020).

Extending these portfolio strategies, which are learned offline, there are online portfolios that can select from a fixed set of machine learning pipelines, taking previous evaluations into account (Leite et al., 2012; Wistuba et al., 2015a,b; Fusi et al., 2018; Yang et al., 2019, 2020). However, such methods cannot be directly combined with all budget allocation strategies as they require the definition of a special model for extrapolating learning curves (Klein et al., 2017b; Falkner et al., 2018) and also introduce additional complexity into AutoML systems.

There exists other work on building portfolios without prior discretization (which we do for our work and was done for most work mentioned above), which directly optimizes the hyperparameters of ML pipelines to add next to the portfolio in a greedy fashion (Xu et al., 2010, 2011; Seipp et al., 2015), to jointly optimize all configurations of the portfolio with global optimization (Winkelmolen et al., 2020), and also to build parallel portfolios (Lindauer et al., 2017). We consider these to be orthogonal to using portfolios in the first place and plan to study improved optimization strategies in future work.

6.2 Related Work on Successive Halving

Large datasets, expensive ML pipelines and tight resource limitations demand sophisticated methods to speed up pipeline selection. One line of research, multi-fidelity optimization methods, tackle this problem by using cheaper approximations of the objective of interest. Practical examples are evaluating a pipeline only on a subset of the dataset or for iterative algorithms limiting the number of iterations. There exists a large body of research on optimization methods leveraging lower fidelities, for example working with a fixed set of auxiliary tasks (Forrester et al., 2007; Swersky et al., 2013; Poloczek et al., 2017; Moss et al., 2020), solutions for specific model classes (Swersky et al., 2014; Domhan et al., 2015; Chandrashekaran and Lane, 2017) and selecting a fidelity value from a continuous range (Klein et al., 2017a; Kandasamy et al., 2017; Wu et al., 2020; Takeno et al., 2020). Here, we focus on a methodologically simple bandit strategy, SH (Karnin et al., 2013; Jamieson and Talwalkar, 2016), which successively reduces the number of candidates and at the same time increases the allocated resources per run till only one candidate remains. Our use of SH in the 2nd AutoML challenge also inspired work on combining a genetic algorithm with SH (Parmentier et al., 2019). Another way of quickly discarding unpromising pipelines is the intensify procedure which was used by *Auto-WEKA* (Thornton et al., 2013) to speed up cross-validation. Instead of evaluating all folds at once, it evaluates the folds in an iterative fashion. After each evaluation, the average performance on the evaluated folds is compared to the performance of the so-far best pipeline on these folds. The evaluation is only continued if the performance is equal or better. While this allows evaluating many configurations in a short time, it cannot be combined with post-hoc ensembling and reduces the cost of a pipeline to, at most, the cost of holdout, which might already be too high.

6.3 Related Work on Algorithm Selection

Automatically choosing a model selection strategy to assess the performance of an ML pipeline for hyperparameter optimization has not previously been tackled, and only Guyon

et al. (2015) acknowledge the lack of such an approach. However, treating the choice of model selection strategy as an algorithm selection problem allows us to apply methods from the field of algorithm selection (Smith-Miles, 2008; Kotthoff, 2014; Kerschke et al., 2019) and we can in future work reuse existing techniques besides the pairwise classification we employ in this paper (Xu et al., 2011), such as the AutoAI system AutoFolio (Lindauer et al., 2015).

6.4 Background on AutoML Systems and Their Components

AutoML systems have recently gained traction in the research community, and there exists a multitude of approaches, often accompanied by open-source software. In the following, we provide background on the main components of AutoML frameworks before describing several prominent instantiations in more depth.

6.4.1 Components of AutoML systems

AutoML systems require a flexible pipeline configuration space and are driven by an efficient method to search this space. Furthermore, they rely on model selection and budget allocation strategies when evaluating different pipelines. Additionally, to speed up the search procedure, information gained on other datasets can be used to kick-start or guide the search procedure (i.e. meta-learning). Finally, one can also combine the models trained during the search phase in a post-hoc ensembling step.

Configuration Space and Search Mechanism While there are configuration space formulations that allow the application of multiple search mechanisms, not all formulations of a configuration space and a search mechanism can be mixed and matched, and we, therefore, describe the different formulations and the applicable search mechanisms in turn.

The most common description of the search space is the CASH formulation. There is a fixed amount of hyperparameters, each with a range of legal values or categorical choices, and some of them can be conditional, meaning that they are only active if other hyperparameters fulfill certain conditions. One such example is the choice of a classification algorithm and its hyperparameters. The hyperparameters of an SVM are only active if the categorical hyperparameter of the classification algorithm is set to SVM.

Standard black-box optimization algorithms can solve the CASH problem, and SMAC (Hutter et al., 2011) and TPE (Bergstra et al., 2011) were proposed first for this task. Others proposed the use of evolutionary algorithms (Bürger and Pauli, 2015) and random search (LeDell and Poirier, 2020). It is also known as the full model selection problem (Escalante et al., 2009), and solutions in that strain of work proposed the use of particle swarm optimization (Escalante et al., 2009) and a combination of a genetic algorithm with particle swarm optimization (Sun et al., 2013). To improve performance one can prune the configuration space to reduce the size of the space the optimization algorithm has to search through (Zhang et al., 2016), split the configuration space into smaller, more manageable subspaces (Alaa and van der Schaar, 2018; Liu et al., 2020), or heavily employ expert knowledge (LeDell and Poirier, 2020).

Instead of a fixed configuration space, genetic programming can make use of a flexible and possibly infinite space of components to be connected (Olson et al., 2016b,a). This approach can be formalized further by using grammar-based genetic programming (de Sa

et al., 2017). Context-free grammars can also be searched by model-based reinforcement learning algorithms (Drori et al., 2019).

Formalizing the search problem as a search tree allows the application of a custom Monte-Carlo tree search (Rakotoarison et al., 2019) and hierarchical task networks with best-first search (Mohr et al., 2018). With discrete spaces it is also possible to use combinations of meta-learning and matrix factorization (Yang et al., 2019, 2020; Fusi et al., 2018). In the special case of using only neural networks in an AutoML system it is possible to stick with standard black-box optimization (Mendoza et al., 2016, 2019; Zimmer et al., 2021), but one can also employ recent advances in neural architecture search (Elsken et al., 2019).

Meta-Learning. When there is knowledge about previous runs of the AutoML system on other datasets available, it is possible to employ meta-learning. One option is to define a dataset similarity measure, often by using hand-crafted meta-features which describe the datasets (Brazdil et al., 1994), to use the best solutions on the closest seen datasets to warmstart the search algorithm (Feurer et al., 2015a). While this way of meta-learning can be seen as an add-on to existing methods, other works use search strategies designed to take meta-learning into account, for example matrix factorization (Yang et al., 2019, 2020; Fusi et al., 2018) or reinforcement learning (Drori et al., 2019; Heffetz et al., 2020).

Model Selection. Given training data, the goal of an AutoML system is to find the best performing ML pipeline. Doing so requires to best approximate the generalization error to 1) provide a reliable and precise signal for the optimization procedure[8] and 2) select the model to be returned in the end. Typically, the generalization error is assessed via the *train-validation-test* protocol (Bishop, 1995; Raschka, 2018). This means that several models are trained on a *training set*, the best one is selected via holdout (using a single split) or the K-fold cross-validation, and the generalization error is then reported on the test set. The AutoML system then returns a single model in case of holdout and a combination of K models in case of K-fold cross-validation (Caruana et al., 2006). One could also use model selection strategies aiming to reduce the effect of overfitting to the validation set (Dwork et al., 2015; Tsamardinos et al., 2018), but while such model selection strategies are an important area of research, houldout or K-fold cross-validation remain the most prominent choices (Henery, 1994; Kohavi and John, 1995; Hastie et al., 2001; Guyon et al., 2010; Bischl et al., 2012; Raschka, 2018).

The influence of the model selection strategy on the performance is well known (Kalousis and Hilario, 2003), and researchers have studied their impact (Kohavi, 1995). However, there is no single best strategy, since there is a tradeoff between approximation quality and time required to compute the validation loss.

Post-hoc Ensembling. AutoML systems evaluate dozens or hundreds of models during their optimization procedure. Thus, it is a natural next step to not only use a single model at the end but to ensemble multiple for improved performance and reduced overfitting.

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

This was first proposed to combine the solutions found by particle swarm optimization (Escalante et al., 2010) and then by an evolutionary algorithm (Bürger and Pauli, 2015). While these works used heuristic methods to combine multiple models into a final ensemble, it is also possible to treat this as another optimization problem (Feurer et al., 2015a) and solve it with ensemble selection (Caruana et al., 2004) or stacking (LeDell and Poirier, 2020).

Instead of using a single layer of machine learning models, Automatic Frankensteining (Wistuba et al., 2017) proposed two-layer stacking, applying AutoML to the outputs of an AutoML system instead of a single layer of ML algorithms followed by an ensembling mechanism. Auto-Stacker went one step further, directly optimizing for a two-layer AutoML system (Chen et al., 2018).

6.4.2 AutoML systems

To the best of our knowledge, the first AutoML system which tunes both hyperparameters and chooses algorithms was an ensemble method (Caruana et al., 2004). This system randomly produces 2 000 classifiers from a wide range of ML algorithms and constructs a post-hoc ensemble. It was later robustified (Caruana et al., 2006) and employed in a winning submission to the KDD challenge (Niculescu-Mizil et al., 2009).

The first AutoML system to jointly optimize the whole pipeline was *Particle Swarm Model Selection* (Escalante et al., 2007, 2009). It used a fixed-length representation of the pipeline and contained feature selection, feature processing, classification and post-processing implemented in the CLOP package[9] and was developed for the IJCNN 2007 agnostic learning vs. prior knowledge challenge (Guyon et al., 2007). It placed 2nd among the solutions using the CLOP package provided by the organizers, only losing to a submission based on robust hyperparameter optimization and ensembling (Reunanen, 2007). Later systems started employing model-based global optimization algorithms, such as *Auto-WEKA* (Thornton et al., 2013; Kotthoff et al., 2019), which is built around the WEKA software (Hall et al., 2009) and SMAC (Hutter et al., 2011) and uses cross-validation with racing for model evaluation, and Hyperopt-sklearn (Komer et al., 2014), which was the first tool to use the now-popular scikit-learn (Pedregosa et al., 2011) and paired it with the TPE algorithm from the hyperopt package (Bergstra et al., 2011, 2013) and holdout.

We extended the approach of parametrizing a popular machine learning library and optimizing its hyperparameters with a black-box optimization algorithm using meta-learning and post-hoc ensembles in *Auto-sklearn* (Feurer et al., 2015a, 2019). For classification, the space of possible ML pipelines currently spans 16 classifiers, 14 feature preprocessing methods and numerous data preprocessing methods, adding up to 122 hyperparameters for the latest release. *Auto-sklearn* uses holdout as a default model selection strategy but allows for other strategies such as cross-validation. *Auto-sklearn* was the dominating solution of the first AutoML challenge (Guyon et al., 2019).

The *tree-based pipeline optimization tool* (*TPOT*; Olson et al., 2016b; Olson and Moore, 2019) uses grammatical evolution to construct ML pipelines of arbitrary length. Currently, it uses scikit-leearn (Pedregosa et al., 2011) and XGBoost (Chen and Guestrin, 2016) for its ML building blocks and 5-fold cross-validation to evaluate individual solutions. TPOT-

9. `http://clopinet.com/CLOP/`

SH (Parmentier et al., 2019), inspired by our submission to the second AutoML challenge, uses successive halving to speed up *TPOT* on large datasets.

There are also multiple AutoML systems that exploit stacking (Wolpert, 1992). First, Automatic Frankensteining (Wistuba et al., 2017) introduces a two-stage optimization process to build a two-layer stacking model. Second, AutoStacker directly optimizes a two-layer stacking model with a genetic algorithm (Chen et al., 2018). Third, *H2O AutoML* package builds on a manually designed set of defaults and random search and combines them in a post-hoc stacking step, using building blocks from the H2O library (H2O.ai, 2020) and XGBoost (Chen and Guestrin, 2016), and employing cross-validation. Lastly, AutoGluon takes a radically different approach and completely drops hyperparameter optimization and invests all available time into building a robust stacking model (Erickson et al., 2020).

Recently, there also have been works that aim to use dataset subsets to speed up the evaluation (Parmentier et al., 2019; Wang et al., 2021).

Finally, there is also work on creating AutoML systems that can leverage recent advancements in deep learning, using either black-box optimization (Mendoza et al., 2016; Zimmer et al., 2021) or neural architecture search (Jin et al., 2019).

Of course, there are also many techniques related to AutoML which are not used in one of the AutoML systems discussed in this section, and we refer to Hutter et al. (2019) for an overview of the field of Automated Machine Learning, to Brazdil et al. (2008) for an overview on meta-learning research which pre-dates the work on AutoML and to Escalante (2021) for a discussion on the history of AutoML.

7. Discussion and Conclusion

In this paper, we introduced our winning entry to the 2nd ChaLearn AutoML challenge, *PoSH Auto-sklearn*, and automated its internal settings further, resulting in the next generation of our AutoML system: *Auto-sklearn 2.0*. It provides a truly hands-free solution, which, given a new task and resource limitations, automatically chooses the best setup. Specifically, we introduce three improvements for faster and more efficient AutoML: (i) to get strong results quickly, we propose to use portfolios, which can be built offline and thus reduce startup costs, (ii) to reduce time spent on poorly performing pipelines we propose to add successive halving as a budget allocation strategy to the configuration space of our AutoML system and (iii) to close the design space we opened up for AutoML we propose to automatically select the best configuration of our system.

We conducted a large-scale study based on 208 meta-datasets for constructing our AutoML systems and 39 datasets for evaluating them and obtained substantially improved performance compared to *Auto-sklearn 1.0*, reducing the ADTM by up to a factor of 4.5 and achieving a lower loss after 10 minutes than *Auto-sklearn 1.0* after 60 minutes. Our ablation study showed that using a model-based policy selector to choose the model selection strategy has the largest impact on performance and allows *Auto-sklearn 2.0* to run robustly on new, unseen datasets. Furthermore, we showed that our method is highly competitive and outperforms other state-of-the-art AutoML systems in the OpenML AutoML benchmark.

However, our system also introduces some shortcomings since it optimizes performance towards a given optimization budget, performance metric and configuration space. Although

all of these, along with the meta datasets, could be provided by a user to automatically build a customized version of *Auto-sklearn 2.0*, it would be interesting whether we can learn how to transfer a specific AutoML system to different optimization budgets and metrics. Although we have observed strong empirical performance using SH, we do not have any performance guarantee when we combine SH with BO. Therefore, we deem developing approaches that increase successive halving's lower budget over time promising next steps. Also, there remain several hand-picked hyperparameters on the level of the AutoML system, which we plan to automate in future work. These are, for example, automatically learning the portfolio size, learning more hyper-hyperparameters of the different budget allocation strategies (for instance, of SH) and proposing suitable configuration spaces given a dataset and resources. Besides these, our use of two meta-features for the selector opens up the research question of whether other meta-features could result in better performance. We expect that we can tackle many of these problems by performing an additional optimization loop on the training data. Finally, building the training data is currently quite expensive. Even though this has to be done only once, it will be interesting to see whether we can take shortcuts here, for example, by using a joint ranking model (Tornede et al., 2020) or non-linear collaborative filtering (Fusi et al., 2018).

Acknowledgments

The authors acknowledge support by the state of Baden-Württemberg through bwHPC and the German Research Foundation (DFG) through grant no INST 39/963-1 FUGG. This work has partly been supported by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme under grant no. 716721. Robert Bosch GmbH is acknowledged for financial support. We furthermore thank all contributors to *Auto-sklearn* for their help in making it a useful AutoML tool and also thank Francisco Rivera for providing a Singularity integration for the AutoML benchmark.

References

2007 International Joint Conference on Neural Networks (IJCNN'07), 2007. IEEE Computer Society Press.

A. Alaa and M. van der Schaar. AutoPrognosis: Automated clinical prognostic modeling via Bayesian optimization with structured kernel learning. In Dy and Krause (2018), pages 139–148.

M.-F. Balcan, T. Sandholm, and E. Vitercik. Generalization in portfolio-based algorithm selection. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI'21)*, volume 35, pages 12225–12232, 2021.

R. Bardenet, M. Brendel, B. Kégl, and M. Sebag. Collaborative hyperparameter tuning. In Dasgupta and McAllester (2013), pages 199–207.

Y. Bengio and Y. Grandvalet. No unbiased estimator of the variance of k-fold cross-validation. *Journal of Machine Learning Research*, 4:1089–1105, 2004.

J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl. Algorithms for hyper-parameter optimization. In J. Shawe-Taylor, R. Zemel, P. Bartlett, F. Pereira, and K. Weinberger, editors, *Proceedings of the 24th International Conference on Advances in Neural Information Processing Systems (NeurIPS'11)*, pages 2546–2554. Curran Associates, 2011.

J. Bergstra, D. Yamins, and D. Cox. Making a science of model search: Hyperparameter Optimization in hundreds of dimensions for vision architectures. In Dasgupta and McAllester (2013), pages 115–123.

A. Biedenkapp, H. F. Bozkurt, T. Eimer, F. Hutter, and M. Lindauer. Dynamic algorithm configuration: Foundation of a new meta-algorithmic framework. In J. Lang, G. De Giacomo, B. Dilkina, and M. Milano, editors, *Proceedings of the Twenty-fourth European Conference on Artificial Intelligence (ECAI'20)*, pages 427–434, June 2020.

M. Birattari, T. Stützle, L. Paquete, and K. Varrentrapp. A racing algorithm for configuring metaheuristics. In W. Langdon, E. Cantu-Paz, K. Mathias, R. Roy, D. Davis, R. Poli, K. Balakrishnan, V. Honavar, G. Rudolph, J. Wegener, L. Bull, M. Potter, A. Schultz, J. Miller, E. Burke, and N. Jonoska, editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'02)*, pages 11–18. Morgan Kaufmann Publishers, 2002.

B. Bischl, O. Mersmann, H. Trautmann, and C. Weihs. Resampling methods for meta-model validation with recommendations for evolutionary computation. *Evolutionary Computation*, 20(2):249–275, 2012.

B. Bischl, G. Casalicchio, M. Feurer, F. Hutter, M. Lang, R. Mantovani, J. van Rijn, and J. Vanschoren. OpenML benchmarking suites. In J. Vanschoren, S. Yeung, and M. Xenochristou, editors, *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, 2021.

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

B. Bonet and S. Koenig, editors. *Proceedings of the Twenty-ninth National Conference on Artificial Intelligence (AAAI'15)*, 2015. AAAI Press.

P. Brazdil and C. Soares. A comparison of ranking methods for classification algorithm selection. In R. Lopez de Mantaras and E. Plaza, editors, *Machine Learning: ECML 2000*, volume 1810 of *Lecture Notes in Computer Science*, pages 63–74. Springer, 2000.

P. Brazdil, J. Gama, and B. Henery. Characterizing the applicability of classification algorithms using meta-level learning. In F. Bergadano and L. De Raedt, editors, *Machine Learning: ECML-94*, pages 83–102. Springer Berlin Heidelberg, 1994.

P. Brazdil, C. Soares, and R. Pereira. Reducing rankings of classifiers by eliminating redundant classifiers. In P. Brazdil and A. Jorge, editors, *Progress in Artificial Intelligence: Knowledge Extraction, Multi-agent Systems, Logic Programming, and Constraint Solving*, Lecture Notes in Artificial Intelligence, pages 14–21. Springer, 2001.

P. Brazdil, C. Giraud-Carrier, C. Soares, and R. Vilalta. *Metalearning: Applications to Data Mining.* Springer, 1 edition, 2008.

L. Breimann. Random forests. *Machine Learning Journal*, 45:5–32, 2001.

F. Bürger and J. Pauli. A holistic classification optimization framework with feature selection, preprocessing, manifold learning and classifiers. In A. Fred, M. De Marsico, and M. Figueiredo, editors, *Procceedings of 4th International Conference on Pattern Recognition: Applications and Methods (ICPRAM'15)*, volume 9493 of *Lecture Notes in Computer Science*, pages 52–68. Springer, 2015.

R. Caruana, A. Niculescu-Mizil, G. Crew, and A. Ksikes. Ensemble selection from libraries of models. In R. Greiner, editor, *Proceedings of the 21st International Conference on Machine Learning (ICML'04)*. Omnipress, 2004.

R. Caruana, A. Munson, and A. Niculescu-Mizil. Getting the most out of ensemble selection. In *Proceedings of the 6th IEEE International Conference on Data Mining (ICDM'06)*, pages 828–833. IEEE Computer Society Press, 2006.

A. Chandrashekaran and I. Lane. Speeding up Hyper-parameter Optimization by Extrapolation of Learning Curves using Previous Builds. In M. Ceci, J. Hollmen, L. Todorovski, C. Vens, and S. Džeroski, editors, *Machine Learning and Knowledge Discovery in Databases (ECML/PKDD'17)*, volume 10534 of *Lecture Notes in Computer Science*, pages 477–492. Springer, 2017.

B. Chen, H. Wu, W. Mo, I. Chattopadhyay, and H. Lipson. Autostacker: A Compositional Evolutionary Learning System. In H. Aguirre and K. Takadama, editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'18)*, pages 402–409. ACM, 2018.

T. Chen and C. Guestrin. Xgboost: A scalable tree boosting system. In Krishnapuram et al. (2016), pages 785–794.

K. Crammer, O. Dekel, J. Keshet, S. Shalev-Shwartz, and Y. Singer. Online passive-aggressive algorithms. *Journal of Machine Learning Research*, 7(19):551–585, 2006.

S. Dasgupta and D. McAllester, editors. *Proceedings of the 30th International Conference on Machine Learning (ICML'13)*, 2013. Omnipress.

Dask Development Team. *Dask: Library for dynamic task scheduling*, 2016. URL `https://dask.org`.

A. de Sa, W. Pinto, L. Oliveira, and G. Pappa. RECIPE: A grammar-based framework for automatically evolving classification pipelines. In M. Castelli, J. McDermott, and L. Sekanina, editors, *EuroGP 2017: Proceedings of the 20th European Conference on Genetic Programming*, volume 10196 of *LNCS*, pages 246–261. Springer Verlag, 2017.

J. Demšar. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research*, 7:1–30, 2006.

T. Domhan, J. Springenberg, and F. Hutter. Speeding up automatic Hyperparameter Optimization of deep neural networks by extrapolation of learning curves. In Q. Yang and M. Wooldridge, editors, *Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI'15)*, pages 3460–3468, 2015.

I. Drori, Y. Krishnamurthy, R. Lourenco, R. Rampin, K. Cho, C. Silva, and J. Freire. Automatic machine learning by pipeline synthesis using model-based reinforcement learning and a grammar. In K. Eggensperger, M. Feurer, F. Hutter, and J. Vanschoren, editors, *ICML workshop on Automated Machine Learning (AutoML workshop 2019)*, 2019.

C. Dwork, V. Feldman, M. Hardt, T. Pitassi, O. Reingold, and A. Roth. The reusable holdout: Preserving validity in adaptive data analysis. *Science*, 349(6248):636–638, 2015.

J. Dy and A. Krause, editors. *Proceedings of the 35th International Conference on Machine Learning (ICML'18)*, volume 80, 2018. Proceedings of Machine Learning Research.

T. Elsken, J. Metzen, and F. Hutter. Neural architecture search. In Hutter et al. (2019), chapter 3, pages 63–77. Available for free at http://automl.org/book.

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

H. Escalante. Automated machine learning—a brief review at the end of the early years. In N. Pillay and R. Qu, editors, *Automated Design of Machine Learning and Search Algorithms*, pages 11–28. Springer, 2021.

H. Escalante, M. Gomez, and L. Sucar. PSMS for neural networks on the ijcnn 2007 agnostic vs prior knowledge challenge. In *2007 International Joint Conference on Neural Networks (IJCNN'07)* ijc (2007), pages 678–683.

H. Escalante, M. Montes, and E. Sucar. Particle Swarm Model Selection. *Journal of Machine Learning Research*, 10:405–440, 2009.

H. Escalante, M. Montes, and E. Sucar. Ensemble particle swarm model selection. In *2010 International Joint Conference on Neural Networks (IJCNN'10)*, pages 1–8. IEEE Computer Society Press, 2010.

S. Falkner, A. Klein, and F. Hutter. BOHB: Robust and efficient Hyperparameter Optimization at scale. In Dy and Krause (2018), pages 1437–1446.

M. Feurer and F. Hutter. Towards further automation in automl. In Garnett et al. (2018).

M. Feurer, A. Klein, K. Eggensperger, J. Springenberg, M. Blum, and F. Hutter. Efficient and robust automated machine learning. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, editors, *Proceedings of the 28th International Conference on Advances in Neural Information Processing Systems (NeurIPS'15)*, pages 2962–2970. Curran Associates, 2015a.

M. Feurer, J. Springenberg, and F. Hutter. Initializing Bayesian Hyperparameter Optimization via meta-learning. In Bonet and Koenig (2015), pages 1128–1135.

M. Feurer, K. Eggensperger, S. Falkner, M. Lindauer, and F. Hutter. Practical automated machine learning for the automl challenge 2018. In Garnett et al. (2018).

M. Feurer, A. Klein, K. Eggensperger, J. Springenberg, M. Blum, and F. Hutter. Auto-sklearn: Efficient and robust automated machine learning. In Hutter et al. (2019), chapter 6, pages 113–134. Available for free at http://automl.org/book.

M. Feurer, J. van Rijn, A. Kadra, P. Gijsbers, N. Mallik, S. Ravi, A. Müller, J. Vanschoren, and F. Hutter. OpenML-Python: an extensible Python API for OpenML. *Journal of Machine Learning Research*, 22(100):1–5, 2021.

A. Forrester, A. Sóbester, and A. Keane. Multi-fidelity optimization via surrogate modelling. *Proceedings of the royal society A: mathematical, physical and engineering sciences*, 463 (2088):3251–3269, 2007.

J. Friedman. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, 29:1189–1232, 2001.

N. Fusi, R. Sheth, and M. Elibol. Probabilistic matrix factorization for automated machine learning. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Proceedings of the 31st International Conference on Advances in Neural Information Processing Systems (NeurIPS'18)*, pages 3348–3357. Curran Associates, 2018.

R. Garnett, F. Hutter J. Vanschoren, P. Brazdil, R. Caruana, C. Giraud-Carrier, I. Guyon, and B. Kégl, editors. *ICML workshop on Automated Machine Learning (AutoML workshop 2018)*, 2018.

P. Geurts, D. Ernst, and L. Wehenkel. Extremely randomized trees. *Machine Learning Journal*, 63(1):3–42, 2006.

P. Gijsbers, E. LeDell, S. Poirier, J. Thomas, B. Bischl, and J. Vanschoren. An open source automl benchmark. In K. Eggensperger, M. Feurer, F. Hutter, and J. Vanschoren, editors, *ICML workshop on Automated Machine Learning (AutoML workshop 2019)*, 2019.

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

I. Guyon, A. Saffari, G. Dror, and G. Cawley. Analysis of the IJCNN 2007 agnostic learning vs. prior knowledge challenge. In *2007 International Joint Conference on Neural Networks (IJCNN'07)* ijc (2007), pages 544–550.

I. Guyon, A. Saffari, G. Dror, and G. Cawley. Model selection: Beyond the Bayesian/Frequentist divide. *Journal of Machine Learning Research*, 11:61–87, 2010.

I. Guyon, K. Bennett, G. Cawley, H. J. Escalante, S. Escalera, Tin Kam Ho, N. Macià, B. Ray, M. Saeed, A. Statnikov, and E. Viegas. Design of the 2015 ChaLearn AutoML challenge. In *2015 International Joint Conference on Neural Networks (IJCNN'15)*, pages 1–8. IEEE Computer Society Press, 2015.

I. Guyon, U. von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors. *Proceedings of the 30th International Conference on Advances in Neural Information Processing Systems (NeurIPS'17)*, 2017. Curran Associates.

I. Guyon, L. Sun-Hosoya, M. Boullé, H. Escalante, S. Escalera, Z. Liu, D. Jajetic, B. Ray, M. Saeed, M. Sebag, A. Statnikov, W. Tu, and E. Viegas. Analysis of the AutoML Challenge Series 2015-2018. In Hutter et al. (2019), chapter 10, pages 177–219. Available for free at http://automl.org/book.

H2O.ai. *H2O: Scalable Machine Learning Platform*, 2020. URL `https://github.com/h2oai/h2o-3`. version 3.30.0.6.

M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. Witten. The WEKA data mining software: An update. *SIGKDD Explorations*, 11(1):10–18, 2009.

C. Harris, K. Millman, S. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. Smith, R. Kern, M. Picus, S. Hoyer, M. van Kerkwijk, M. Brett, A. Haldane, J. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. Oliphant. Array programming with numpy. *Nature*, 585(7825):357–362, 2020.

T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning*. Springer, 2001.

Y. Heffetz, R. Vainshtein, G. Katz, and L. Rokach. DeepLine: AutoML Tool for Pipelines Generation using Deep Reinforcement Learning and Hierarchical Actions Filtering. In Tang and Prakash (2020), pages 2103–2113.

R. Henery. Methods for comparison. In *Machine Learning, Neural and Statistical Classification*, chapter 7, pages 107–124. Ellis Horwood, 1994.

B. Huberman, R. Lukose, and T. Hogg. An economic approach to hard computational problems. *Science*, 275:51–54, 1997.

J. Hunter. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3):90–95, 2007.

F. Hutter, H. Hoos, K. Leyton-Brown, and T. Stützle. ParamILS: An automatic algorithm configuration framework. *Journal of Artificial Intelligence Research*, 36:267–306, 2009.

F. Hutter, H. Hoos, and K. Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In C. Coello, editor, *Proceedings of the Fifth International Conference on Learning and Intelligent Optimization (LION'11)*, volume 6683 of *Lecture Notes in Computer Science*, pages 507–523. Springer, 2011.

F. Hutter, L. Kotthoff, and J. Vanschoren, editors. *Automated Machine Learning: Methods, Systems, Challenges*. Springer, 2019. Available for free at http://automl.org/book.

K. Jamieson and A. Talwalkar. Non-stochastic best arm identification and Hyperparameter Optimization. In A. Gretton and C. Robert, editors, *Proceedings of the Seventeenth International Conference on Artificial Intelligence and Statistics (AISTATS'16)*, volume 51. Proceedings of Machine Learning Research, 2016.

H. Jin, Q. Song, and X. Hu. Auto-Keras: An efficient neural architecture search system. In Teredesai et al. (2019), pages 1946–1956.

A. Kalousis and M. Hilario. Representational Issues in Meta-Learning. In T. Fawcett and N. Mishra, editors, *Proceedings of the 20th International Conference on Machine Learning (ICML'03)*, pages 313–320. Omnipress, 2003.

K. Kandasamy, G. Dasarathy, J. Schneider, and B. Póczos. Multi-fidelity Bayesian Optimisation with Continuous Approximations. In D. Precup and Y. Teh, editors, *Proceedings of the 34th International Conference on Machine Learning (ICML'17)*, volume 70, pages 1799–1808. Proceedings of Machine Learning Research, 2017.

Z. Karnin, T. Koren, and O. Somekh. Almost optimal exploration in multi-armed bandits. In Dasgupta and McAllester (2013), pages 1238–1246.

G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu. Lightgbm: A highly efficient gradient boosting decision tree. In Guyon et al. (2017).

P. Kerschke, H. Hoos, F. Neumann, and H. Trautmann. Automated algorithm selection: Survey and perspectives. *Evolutionary Computation*, 27(1):3–45, 2019.

A. Klein, S. Falkner, S. Bartels, P. Hennig, and F. Hutter. Fast Bayesian optimization of machine learning hyperparameters on large datasets. In A. Singh and J. Zhu, editors, *Proceedings of the Seventeenth International Conference on Artificial Intelligence and Statistics (AISTATS'17)*, volume 54. Proceedings of Machine Learning Research, 2017a.

A. Klein, S. Falkner, J. Springenberg, and F. Hutter. Learning curve prediction with Bayesian neural networks. In *Proceedings of the International Conference on Learning Representations (ICLR'17)*, 2017b. Published online: `iclr.cc`.

R. Kleinberg, K. Leyton-Brown, and B. Lucier. Efficiency through procrastination: Approximately optimal algorithm configuration with runtime guarantees. In C. Sierra, editor, *Proceedings of the 26th International Joint Conference on Artificial Intelligence (IJCAI'17)*, pages 2023–2031, 2017.

R. Kohavi. A study of cross-validation and bootstrap for accuracy estimation and model selection. In C. Mellish, editor, *Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI'95)*, pages 1137–1143. Morgan Kaufmann Publishers, 1995.

R. Kohavi and G. John. Automatic Parameter Selection by Minimizing Estimated Error. In A. Prieditis and S. Russell, editors, *Proceedings of the Twelfth International Conference on Machine Learning (ICML'95)*, pages 304–312. Morgan Kaufmann Publishers, 1995.

B. Komer, J. Bergstra, and C. Eliasmith. Hyperopt-sklearn: Automatic hyperparameter configuration for scikit-learn. In F. Hutter, R. Caruana, R. Bardenet, M. Bilenko, I. Guyon, B. Kégl, and H. Larochelle, editors, *ICML workshop on Automated Machine Learning (AutoML workshop 2014)*, 2014.

L. Kotthoff. Algorithm selection for combinatorial search problems: A survey. *AI Magazine*, 35(3):48–60, 2014.

L. Kotthoff, C. Thornton, H. Hoos, F. Hutter, and K. Leyton-Brown. Auto-WEKA: automatic model selection and hyperparameter optimization in WEKA. In Hutter et al. (2019), chapter 4, pages 81–95. Available for free at http://automl.org/book.

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

A. Krause and D. Golovin. Submodular function maximization. In L. Bordeaux, Y. Hamadi, and P. Kohli, editors, *Tractability: Practical Approaches to Hard Problems*, pages 71–104. Cambridge University Press, 2014.

A. Krause, J. Leskovec, C. Guestrin, J. VanBriesen, and C. Faloutsos. Efficient sensor placement optimization for securing large water distribution networks. *Journal of Water Resources Planning and Management*, 134:516–526, 2008.

B. Krishnapuram, M. Shah, A. Smola, C. Aggarwal, D. Shen, and R. Rastogi, editors. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'16)*, 2016. ACM Press.

G. Kurtzer, V. Sochat, and M. Bauer. Singularity: Scientific containers for mobility of compute. *PloS one*, 12(5), 2017.

E. LeDell and S. Poirier. H2O AutoML: Scalable automatic machine learning. In K. Eggensperger, M. Feurer, C. Weill, M.Lindauer, F. Hutter, and J. Vanschoren, editors, *ICML workshop on Automated Machine Learning (AutoML workshop 2020)*, 2020.

R. Leite, P. Brazdil, and J. Vanschoren. Selecting classification algorithms with active testing. In P. Perner, editor, *Machine Learning and Data Mining in Pattern Recognition*, volume 7376 of *Lecture Notes in Computer Science*, pages 117–131. Springer, 2012.

L. Li, K. Jamieson, G. DeSalvo, A. Rostamizadeh, and A. Talwalkar. Hyperband: A novel bandit-based approach to Hyperparameter Optimization. *Journal of Machine Learning Research*, 18(185):1–52, 2018.

M. Lindauer, H. Hoos, F. Hutter, and T. Schaub. Autofolio: An automatically configured algorithm selector. *Journal of Artificial Intelligence Research*, 53:745–778, 2015.

M. Lindauer, H. Hoos, K. Leyton-Brown, and T. Schaub. Automatic construction of parallel portfolios via algorithm configuration. *Artificial Intelligence*, 244:272–290, 2017.

M. Lindauer, K. Eggensperger, M. Feurer, A. Biedenkapp, D. Deng, C. Benjamins, T. Ruhkopf, R. Sass, and F. Hutter. SMAC3: A versatile bayesian optimization package for Hyperparameter Optimization. *Journal of Machine Learning Research (JMLR) – MLOSS*, 23(54):1–9, 2022.

S. Liu, P. Ram, D. Vijaykeerthy, D. Bouneffouf, G. Bramble, H. Samulowitz, D. Wang, A. Conn, and A. Gray. An ADMM based framework for automl pipeline configuration. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI'20)*, volume 34, pages 4892–4899, 2020.

Y. Malitsky, A. Sabharwal, H. Samulowitz, and M. Sellmann. Parallel SAT solver selection and scheduling. In M. Milano, editor, *Proceedings of the Eighteenth International Conference on Principles and Practice of Constraint Programming (CP'12)*, volume 7514 of *Lecture Notes in Computer Science*, pages 512–526. Springer, 2012.

H. Mendoza, A. Klein, M. Feurer, J. Springenberg, and F. Hutter. Towards automatically-tuned neural networks. In F. Hutter, L. Kotthoff, and J. Vanschoren, editors, *ICML workshop on Automated Machine Learning (AutoML workshop 2016)*, volume 64. PMLR, 2016.

H. Mendoza, A. Klein, M. Feurer, J. Springenberg, M. Urban, M. Burkart, M. Dippel, M. Lindauer, and F. Hutter. Towards automatically-tuned deep neural networks. In Hutter et al. (2019), chapter 7, pages 135–149. Available for free at http://automl.org/book.

D. Merkel. Docker: lightweight linux containers for consistent development and deployment. *Linux journal*, 2014(239), 2014.

L. Metz, N. Maheswaranathan, C. Freeman, B. Poole, and J. Sohl-Dickstein. Tasks, stability, architecture, and compute: Training more effective learned optimizers, and using them to train themselves. *arXiv:2009.11243[cs.LG]*, 2020.

F. Mohr, M. Wever, and E. Hüllermeier. ML-Plan: Automated machine learning via hierarchical planning. *Machine Learning*, 107(8-10):1495–1515, 2018.

H. Moss, D. Leslie, and P. Rayson. MUMBO: Multi-task max-value Bayesian optimization. In F. Hutter, K. Kersting, J. Lijffijt, and I. Valera, editors, *Machine Learning and Knowledge Discovery in Databases (ECML/PKDD'20)*, Lecture Notes in Computer Science. Springer, 2020.

G. Nemhauser, L. Wolsey, and M. Fisher. An analysis of approximations for maximizing submodular set functions. *Mathematical Programming*, 14(1):265–294, 1978.

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

R. Olson and J. Moore. TPOT: A tree-based pipeline optimization tool for automating machine learning. In Hutter et al. (2019), pages 151–160. Available for free at http://automl.org/book.

R. Olson, N. Bartley, R. Urbanowicz, and J. Moore. Evaluation of a Tree-based Pipeline Optimization Tool for Automating Data Science. In T. Friedrich, editor, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'16)*, pages 485–492. ACM, 2016a.

R. Olson, R. Urbanowicz, P. Andrews, N. Lavender, L. Kidd, and J. Moore. Automating biomedical data science through tree-based pipeline optimization. In G. Squillero and P. Burelli, editors, *Proceedings of the 19th European Conference on Applications of Evolutionary Computation (EvoApplications'16)*, pages 123–137. Springer, 2016b.

L. Parmentier, O. Nicol, L. Jourdan, and M. Kessaci. TPOT-SH: A faster optimization algorithm to solve the automl problem on large datasets. In *IEEE 31st International Conference on Tools with Artificial Intelligence (ICTAI'19)*, pages 471–478, 2019.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

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

M. Poloczek, J. Wang, and P. Frazier. Multi-Information Source Optimization. In Guyon et al. (2017), pages 4288–4298.

H. Rakotoarison, M. Schoenauer, and M. Sebag. Automated machine learning with Monte-Carlo tree search. In S. Kraus, editor, *Proceedings of the 28th International Joint Conference on Artificial Intelligence (IJCAI'19)*, pages 3296–3303, 2019.

S. Ramage. *Advances in meta-algorithmic software libraries for distributed automated algorithm configuration*. PhD thesis, University of British Columbia, 2015. URL `https://open.library.ubc.ca/collections/ubctheses/24/items/1.0167184`.

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

J. Reback, jbrockmendel, W. McKinney, J. Van den Bossche, T. Augspurger, P. Cloud, S. Hawkins, gfyoung, Sinhrks, M. Roeschke, and et al. pandas-dev/pandas: Pandas 1.2.5, 2021.

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

J. Reunanen. Model selection and assessment using cross-indexing. In *2007 International Joint Conference on Neural Networks (IJCNN'07)* ijc (2007), pages 2581–2585.

J. Seipp, S. Sievers, M. Helmert, and F. Hutter. Automatic configuration of sequential planning portfolios. In Bonet and Koenig (2015).

K. Smith-Miles. Cross-disciplinary perspectives on meta-learning for algorithm selection. *ACM Computing Surveys*, 41(1), 2008.

C. Soares and P. Brazdil. Zoomed ranking: Selection of classification algorithms based on relevant performance information. In D. Zighed, J. Komorowski, and J. Żytkow, editors, *Principles of Data Mining and Knowledge Discovery*, volume 1910 of *Lecture Notes in Computer Science*, pages 126–135. Springer, 2000.

Q. Sun, B. Pfahringer, and M. Mayo. Towards a Framework for Designing Full Model Selection and Optimization Systems. In *Multiple Classifier Systems*, volume 7872, pages 259–270. Springer, 2013.

K. Swersky, J. Snoek, and R. Adams. Multi-task Bayesian optimization. In C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Weinberger, editors, *Proceedings of the 26th International Conference on Advances in Neural Information Processing Systems (NeurIPS'13)*, pages 2004–2012. Curran Associates, 2013.

K. Swersky, J. Snoek, and R. Adams. Freeze-thaw Bayesian optimization. *arXiv:1406.3896 [stats.ML]*, 2014.

S. Takeno, H. Fukuoka, Y. Tsukada, T. Koyama, M. Shiga, I. Takeuchi, and M. Karasuyama. Multi-fidelity Bayesian optimization with max-value entropy search and its parallelization. In H. Daume III and A. Singh, editors, *Proceedings of the 37th International Conference on Machine Learning (ICML'20)*, volume 98, pages 9334–9345. Proceedings of Machine Learning Research, 2020.

J. Tang and B. Prakash, editors. *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'20)*, 2020. ACM Press.

A. Teredesai, V. Kumar, Y. Li, R. Rosales, E. Terzi, and G. Karypis, editors. *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD'19)*, 2019. ACM Press.

C. Thornton, F. Hutter, H. Hoos, and K. Leyton-Brown. Auto-WEKA: combined selection and Hyperparameter Optimization of classification algorithms. In I. Dhillon, Y. Koren, R. Ghani, T. Senator, P. Bradley, R. Parekh, J. He, R. Grossman, and R. Uthurusamy, editors, *The 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'13)*, pages 847–855. ACM Press, 2013.

A. Tornede, M. Wever, and E. Hüllermeier. Extreme algorithm selection with dyadic feature representation. In A. Appice, G. Tsoumakas, Y. Manolopoulos, and S. Matwin, editors, *Discovery Science (DS)*, volume 12323 of *Lecture Notes in Computer Science*, pages 309–324. Springer, 2020.

I. Tsamardinos, E. Greasidou, and G. Borboudakis. Bootstrapping the out-of-sample predictions for efficient and accurate cross-validation. *Machine Learning*, 107(12):1895–1922, 2018.

J. Vanschoren. Meta-learning. In Hutter et al. (2019), pages 35–61. Available for free at http://automl.org/book.

J. Vanschoren, J. van Rijn, B. Bischl, and L. Torgo. OpenML: Networked science in machine learning. *SIGKDD Explorations*, 15(2):49–60, 2014.

V. Vapnik. Principles of risk minimization for learning theory. In J. Moody, S. Hanson, and R. Lippmann, editors, *Proceedings of the 4th International Conference on Advances in Neural Information Processing Systems (NeurIPS'91)*. Morgan Kaufmann Publishers, 1991.

P. Virtanen, R. Gommers, T. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. van der Walt, M. Brett, J. Wilson, K. Millman, N. Mayorov, A. Nelson, E. Jones, R. Kern, E. Larson, C. Carey, İ. Polat, Y. Feng, E. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. Quintero, C. Harris, A. Archibald, A. Ribeiro, F. Pedregosa, P. van Mulbregt, A. Vijaykumar, Alessandro P. Bardelli, A. Rothberg, A. Hilboll, A. Kloeckner, A. Scopatz, A. Lee, A. Rokem, C. Woods, C. Fulton, C. Masson, C. Häggström, C. Fitzgerald, D. Nicholson, D. Hagen, D. Pasechnik, E. Olivetti, E Martin, E. Wieser, F. Silva, F. Lenders, F. Wilhelm, G. Young, G. Price, G.-L. Ingold, G. Allen, G. Lee, H. Audren, I. Probst, J. Dietrich, J. Silterra, J. Webber, J. Slavič, J. Nothman, J. Buchner, J. Kulick, J. Schönberger, J. de Miranda Cardoso, J. Reimer, J. Harrington, J. Rodríguez, J. Nunez-Iglesias, J. Kuczynski, K. Tritz, M. Thoma, M. Newville, M. Kümmerer, M. Bolingbroke, M. Tartre, M. Pak, N. Smith, N. Nowaczyk, N. Shebanov, O. Pavlyk, P. Brodtkorb, P. Lee, R. McGibbon, R. Feldbauer, S. Lewis, S. Tygier, S. Sievert, S. Vigna, S. Peterson, S. More, T. Pudlik, T. Oshima, T. Pingel, T. Robitaille, T. Spura, T. Jones, T. Cera, T. Leslie, T. Zito, T. Krauss, U. Upadhyay, Y. Halchenko, Y. Vázquez-Baeza, and SciPy 1.0 Contributors. SciPy 1.0: fundamental algorithms for scientific computing in Python. *Nature Methods*, 17(3):261–272, 2020.

C. Wang, Q. Wu, M. Weimer, and E. Zhu. Flaml: A fast and lightweight automl library. In A. Smola, A. Dimakis, and I. Stoica, editors, *Proceedings of Machine Learning and Systems*, volume 3, pages 434–447, 2021.

Wes McKinney. Data Structures for Statistical Computing in Python. In Stéfan van der Walt and Jarrod Millman, editors, *Proceedings of the 9th Python in Science Conference*, pages 56 – 61, 2010.

F. Winkelmolen, N. Ivkin, H. Bozkurt, and Z. Karnin. Practical and sample efficient zero-shot HPO. *arXiv:2007.13382 [stat.ML]*, 2020.

M. Wistuba, N. Schilling, and L. Schmidt-Thieme. Learning Hyperparameter Optimization initializations. In *Proceedings of the International Conference on Data Science and Advanced Analytics (DSAA)*, pages 1–10. IEEE, 2015a.

M. Wistuba, N. Schilling, and L. Schmidt-Thieme. Sequential Model-Free Hyperparameter Tuning. In *ICDM '15: Proceedings of the 2015 IEEE International Conference on Data Mining (ICDM)*, pages 1033–1038. IEEE Computer Society Press, 2015b.

M. Wistuba, N. Schilling, and L. Schmidt-Thieme. Automatic Frankensteining: Creating Complex Ensembles Autonomously. In N. Chawla and W. Wang, editors, *Proceedings of the 2017 SIAM International Conference on Data Mining (SDM'17)*, pages 741–749. Society for Industrial and Applied Mathematics, 2017.

M. Wistuba, N. Schilling, and L. Schmidt-Thieme. Scalable Gaussian process-based transfer surrogates for Hyperparameter Optimization. *Machine Learning*, 107(1):43–78, 2018.

D. Wolpert. Stacked generalization. *Neural Networks*, 5(2):241–259, 1992.

J. Wu, S. Toscano-Palmerin, P. Frazier, and A. Wilson. Practical multi-fidelity Bayesian optimization for hyperparameter tuning. In J. Peters and D. Sontag, editors, *Proceedings of The 36th Uncertainty in Artificial Intelligence Conference (UAI'20)*, pages 788–798. PMLR, 2020.

L. Xu, H. Hoos, and K. Leyton-Brown. Hydra: Automatically configuring algorithms for portfolio-based selection. In M. Fox and D. Poole, editors, *Proceedings of the Twenty-fourth National Conference on Artificial Intelligence (AAAI'10)*, pages 210–216. AAAI Press, 2010.

L. Xu, F. Hutter, H. Hoos, and K. Leyton-Brown. Hydra-MIP: Automated algorithm configuration and selection for mixed integer programming. In *Proceedings of the RCRA workshop at IJCAI 2011*, 2011.

C. Yang, J. Akimoto, D. Kim, and M. Udell. OBOE: Collaborative filtering for AutoML model selection. In Teredesai et al. (2019), pages 1173–1183.

C. Yang, J. Fan, Z. Wu, and M. Udell. AutoML pipeline selection: Efficiently navigating the combinatorial space. In Tang and Prakash (2020), pages 1446–1456.

Y. Zhang, M. Bahadori, H. Su, and J. Sun. FLASH: Fast Bayesian Optimization for Data Analytic Pipelines. In Krishnapuram et al. (2016), pages 2065–2074.

L. Zimmer, M. Lindauer, and F. Hutter. Auto-Pytorch: Multi-fidelity metalearning for efficient and robust AutoDL. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 43(9):1–1, 2021.

Part IV

PUBLICATIONS ON BENCHMARKING AND THE OPENML PLATFORM

7

OPENML-PYTHON: AN EXTENSIBLE PYTHON API FOR OPENML

Matthias Feurer, Jan N. van Rijn, Arlind Kadra, Pieter Gijsbers, Neeratyoy Mallik, Sahithya Ravi, Andreas Müller, Joaquin Vanschoren, and Frank Hutter (2021). "OpenML-Python: an extensible Python API for OpenML." In: *Journal of Machine Learning Research* 22.100. Ed. by Balazs Kegl, pp. 1–5.

PAPER SUMMARY. This paper describes the *OpenML*-Python package. *OpenML*-Python is a Python API to the popular `OpenML.org` platform and also the backbone to many studies in the field of AutoML and meta-learning. The paper discusses the API design, the API's extensibility, and also how the API was used in previous publications.

PROJECT IDEA. This project was initiated by Matthias Feurer based on experiences during his Master's thesis. Over time, the project was extended and improved based on the ideas from all authors, community involvement on github.com, and discussions with the OpenML community at OpenML workshops.

IMPLEMENTATION. Matthias Feurer provided the first implementation. He then extended and maintained the project during his time as a PhD student. Matthias Feurer was the main contributor, and further implementations were mostly conducted by Jan N. van Rijn, Andreas Müller, Neeratyoy Mallik, Pieter Gijsbers, Sahithya Ravi, Joaquin Vanschoren, and Arlind Kadra. Detailed contribution statistics and the exact contribution of each author can be obtained on `github.com/openml/openml-python`.

PAPER WRITING. Matthias Feurer wrote the first paper draft. Jan N. van Rijn, Pieter Gijsbers, Joaquin Vanschoren, Frank Hutter and Matthias Feurer prepared the final version. All co-authors provided valuable feedback.

LICENSE. This chapter was published by the Journal of Machine Learning Research and is licensed under the terms of the Creative Commons Attribution 4.0 International License (`http://creativecommons.org/licenses/by/4.0/`). No changes were made to the original publication.

Journal of Machine Learning Research 22 (2021) 1-5 Submitted 11/19; Published 5/21

OpenML-Python: an extensible Python API for OpenML

Matthias Feurer FEURERM@CS.UNI-FREIBURG.DE
University of Freiburg, Freiburg, Germany

Jan N. van Rijn J.N.VAN.RIJN@LIACS.LEIDENUNIV.NL
Leiden University, Leiden, Netherlands

Arlind Kadra KADRAA@CS.UNI-FREIBURG.DE
University of Freiburg, Freiburg, Germany

Pieter Gijsbers P.GIJSBERS@TUE.NL
Eindhoven University of Technology, Eindhoven, Netherlands

Neeratyoy Mallik MALLIK@CS.UNI-FREIBURG.DE
University of Freiburg, Freiburg, Germany

Sahithya Ravi S.RAVI@TUE.NL
Eindhoven University of Technology, Eindhoven, Netherlands

Andreas Müller ANDREAS.MUELLER.ML@GMAIL.COM
Microsoft, Sunnyvale, USA

Joaquin Vanschoren J.VANSCHOREN@TUE.NL
Eindhoven University of Technology, Eindhoven, Netherlands

Frank Hutter FH@CS.UNI-FREIBURG.DE
University of Freiburg & Bosch Center for Artificial Intelligence, Freiburg, Germany

Editor: Balazs Kegl

Abstract

OpenML is an online platform for open science collaboration in machine learning, used to share datasets and results of machine learning experiments. In this paper, we introduce *OpenML-Python*, a client API for Python, which opens up the OpenML platform for a wide range of Python-based machine learning tools. It provides easy access to all datasets, tasks and experiments on OpenML from within Python. It also provides functionality to conduct machine learning experiments, upload the results to OpenML, and reproduce results which are stored on OpenML. Furthermore, it comes with a scikit-learn extension and an extension mechanism to easily integrate other machine learning libraries written in Python into the OpenML ecosystem. Source code and documentation are available at https://github.com/openml/openml-python/.

Keywords: Python, Collaborative Science, Meta-Learning, Reproducible Research

1. Introduction

OpenML is a collaborative online machine learning (ML) platform, meant for sharing and building on prior empirical machine learning research (Vanschoren et al., 2014).

It goes beyond open data repositories, such as UCI (Dua and Graff, 2019), PMLB (Olson et al., 2017), the 'datasets' submodules in scikit-learn and tensorflow (Pedregosa et al.,

©2021 Matthias Feurer, Jan N. van Rijn, Arlind Kadra, Pieter Gijsbers, Neeratyoy Mallik, Sahithya Ravi, Andreas Müller, Joaquin Vanschoren and Frank Hutter.

License: CC-BY 4.0, see https://creativecommons.org/licenses/by/4.0/. Attribution requirements are provided at http://jmlr.org/papers/v22/19-920.html.

2011; Abadi et al., 2016), and the closed-source data sharing platform at Kaggle.com, since OpenML also collects millions of shared experiments on these datasets, linked to the exact ML pipelines and hyperparameter settings, and includes comprehensive logging and uploading functionalities which can be accessed programmatically via a REST API. An introduction and detailed information can be found on https://docs.openml.org.

OpenML-Python is a seamless integration of OpenML into the popular Python ML ecosystem,[1] that takes away this complexity by providing easy programmatic access to all OpenML data and by automating the sharing of new experiments.[2] In this paper, we introduce OpenML-Python's core design, showcase its extensibility to new ML libraries, and give code examples for several common research tasks.

2. Use cases for the OpenML-Python API

OpenML-Python allows for easy dataset and experiment sharing and reuse by handling all communication with OpenML's REST API. In this section, we briefly describe how the package can be used in several common machine learning tasks and highlight recent uses.

Working with datasets. OpenML-Python can retrieve the thousands of datasets on OpenML (all of them, or specific subsets) in a unified format, retrieve meta-data describing them, and search through them with filters. Datasets are converted from OpenML's internal format into *numpy*, *scipy* or *pandas* data structures, which are standard for ML in Python. To facilitate contributions from the community, it allows people to upload new datasets in only two function calls, and to define new *tasks* on them (combinations of a dataset, train/test split and target attribute).

Publishing and retrieving results. Sharing empirical results allows anyone to search and download them in order to reproduce and reuse them in their own research. One goal of OpenML is to simplify the comparison of new algorithms and implementations to existing approaches by comparing to the results on OpenML. To this end we also provide an interface for integrating new machine learning libraries with OpenML and we have already integrated *scikit-learn*. OpenML-Python can then be used to set up and conduct machine learning experiments for a given *task* and *flow* (an ML pipeline), and publish reproducible results (including hyperparameter settings and random states).

Use cases in published works. OpenML-Python has already been used to scale up studies with hundreds of consistently formatted datasets (Feurer et al., 2015; Fusi et al., 2018), supply large amounts of meta-data for meta-learning (Perrone et al., 2018), answer questions about algorithms such as hyperparameter importance (van Rijn and Hutter, 2018) and facilitate large-scale comparisons of algorithms (Strang et al., 2018).

3. High-level Design of OpenML-Python

The OpenML platform is organized around several entity types which describe different aspects of a machine learning study. It hosts *datasets*, *tasks* that define how models should be evaluated on them, *flows* that record the structure and other details of ML pipelines, and

1. https://github.blog/2019-01-24-the-state-of-the-octoverse-machine-learning/
2. Other clients already exist for R (Casalicchio et al., 2017) and Java (van Rijn, 2016).

OpenML-Python: an extensible Python API for OpenML

```python
import openml; import numpy as np
import matplotlib.pyplot as plt
df = openml.evaluations.list_evaluations_setups(
    'predictive_accuracy', flows=[8353], tasks=[6],
    output_format='dataframe', parameters_in_separate_columns=True,
) # Choose an SVM flow (e.g. 8353), and the dataset 'letter' (task 6).
hp_names = ['sklearn.svm.classes.SVC(16)_C','sklearn.svm.classes.SVC(16)_gamma']
df[hp_names] = df[hp_names].astype(float).apply(np.log)
C, gamma, score = df[hp_names[0]], df[hp_names[1]], df['value']
cntr = plt.tricontourf(C, gamma, score, levels=12, cmap='RdBu_r')
plt.colorbar(cntr, label='accuracy')
plt.xlim((min(C), max(C))); plt.ylim((min(gamma), max(gamma)))
plt.xlabel('C (log10)', size=16); plt.ylabel('gamma (log10)', size=16)
plt.title('SVM performance landscape', size=20)
```

Figure 1: Code for retrieving the predictive accuracy of an SVM classifier on the 'letter' dataset and creating a contour plot with the results.

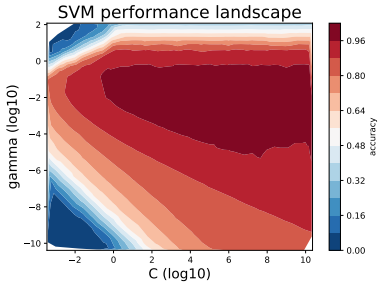
runs that record the experiments evaluating specific *flows* on certain *tasks*. For instance, an experiment (*run*) shared on OpenML can show how a random forest (*flow*) performs on 'Iris' (*dataset*) if evaluated with 10-fold cross-validation (*task*), and how to reproduce that result. In OpenML-Python, all these entities are represented by classes, each defined in their own submodule. This implements a natural mapping from OpenML concepts to Python objects. While OpenML is an online platform, we facilitate offline usage as well.

Extensions. To allow users to automatically run and share machine learning experiments with different libraries through the same OpenML-Python interface, we designed an extension interface that standardizes the interaction between machine learning library code and OpenML-Python. We also created an extension for *scikit-learn* (Pedregosa et al., 2011), as it is one of the most popular Python machine learning libraries. This extension can be used for any library which follows the *scikit-learn* API (Buitinck et al., 2013).

An extension's responsibility is to convert between the libraries' models and OpenML flows, interact with its training interface and format predictions. For example, the *scikit-learn* extension can convert an *OpenMLFlow* to an Estimator (including hyperparameter settings), train models and produce predictions for a task, and create an *OpenMLRun* object to upload the predictions to the OpenML server. The extension also handles advanced procedures, such as *scikit-learn*'s random search or grid search and uploading its traces (hyperparameters and scores of each model evaluated during search). We are working on more extensions, and anyone can contribute their own using the *scikit-learn* extension implementation as a reference.

4. Examples

We show two example uses of OpenML-Python to demonstrate its API's simplicity. First, we show how to retrieve results and evaluations from the OpenML server in Figure 1 (generating the plot on the right).



SVM hyperparameter contour plot generated by the code in Figure 1.

```
from openml import study, tasks, runs, extensions
from sklearn import compose, impute, pipeline, preprocessing, tree
cont, cat = extensions.sklearn.cont, extensions.sklearn.cat # feature types
clf = pipeline.make_pipeline( compose.make_column_transformer(
    (impute.SimpleImputer(), cont),
    (preprocessing.OneHotEncoder(handle_unknown='ignore'), cat)),
    tree.DecisionTreeClassifier())  # build a classification pipeline
benchmark_suite = study.get_suite('OpenML-CC18')  # task collection
for task_id in benchmark_suite.tasks:  # iterate over all tasks
  task = tasks.get_task(task_id)  # download the OpenML task
  run = runs.run_model_on_task(clf, task)  # run classifier on splits
  # run.publish()  # upload the run to the server; optional, requires API key
```

Figure 2: Training and evaluating a classification pipeline from scikit-learn on each task of the OpenML-CC18 benchmark suite (Bischl et al., 2019).

Second, in Figure 2 we show how to conduct experiments on a benchmark suite (Bischl et al., 2019). Further examples, including how to create datasets and tasks and how OpenML-Python was used in previous publications, can be found in the online documentation.[3]

5. Project development

The project has been set up for development through community effort from different research groups, and has received contributions from numerous individuals. The package is developed publicly through Github which also provides an issue tracker for bug reports, feature requests and usage questions. To ensure a coherent and robust code base we use continuous integration for Windows and Linux as well as automated type and style checking. Documentation is also rendered on continuous integration servers and consists of a mix of tutorials, examples and API documentation.

For ease of use and stability, we use well-known and established 3rd-party packages where needed. For instance, we build documentation using the popular *sphinx Python documentation generator*,[4] use an extension to automatically compile examples into documentation and Jupyter notebooks,[5] and employ standard open-source packages for scientific computing such as *numpy* (Harris et al., 2020), *scipy* (Virtanen et al., 2020), and *pandas* (McKinney, 2010). The package is written in Python3 and open-sourced with a 3-Clause BSD License.[3]

6. Conclusion

OpenML-Python allows easy interaction with OpenML from within Python. It makes it easy for people to share and reuse the data, meta-data, and empirical results which are generated as part of an ML study. This allows for better reproducibility, simpler benchmarking and easier collaboration on ML projects. Our software is shipped with a scikit-learn extension and has an extension mechanism to easily integrate other ML libraries written in Python.

3. We provide documentation, a list of extensions and code examples on http://openml.github.io/openml-python and host the project on http://github.com/openml/openml-python.
4. http://www.sphinx-doc.org [5]https://sphinx-gallery.github.io/

Acknowledgments

MF, **NM** and **FH** acknowledge funding by the Robert Bosch GmbH. **AK**, **JvR** and **FH** acknowledge funding by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme under grant no. 716721. **JV** and **PG** acknowledge funding by the Data Driven Discovery of Models (D3M) program run by DARPA and the Air Force Research Laboratory. **The authors** also thank Bilge Celik, Victor Gal and everyone listed at https://github.com/openml/openml-python/graphs/contributors for their contributions.

References

M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, et al. TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems. In *Proc. of OSDI'16*, 2016.

B. Bischl, G. Casalicchio, M. Feurer, F. Hutter, M. Lang, R. G. Mantovani, J. N. van Rijn, and J. Vanschoren. OpenML Benchmarking Suites. *arXiv:1708.03731v2 [cs.LG]*, 2019.

L. Buitinck, G. Louppe, M. Blondel, F. Pedregosa, A. Müller, O. Grisel, V. Niculae, P. Prettenhofer, A. Gramfort, J. Grobler, et al. API design for machine learning software: experiences from the scikit-learn project. In *ECML PKDD LML Workshop*, 2013.

G. Casalicchio, J. Bossek, M. Lang, D. Kirchhoff, P. Kerschke, B. Hofner, H. Seibold, J. Vanschoren, and B. Bischl. OpenML: An R package to connect to the machine learning platform OpenML. *Computational Statistics*, 32(3), 2017.

D. Dua and C. Graff. UCI machine learning repository, 2019. URL http://archive.ics.uci.edu/ml.

M. Feurer, A. Klein, K. Eggensperger, J. T. Springenberg, M. Blum, and F. Hutter. Efficient and Robust Automated Machine Learning. In *Proc. of NeurIPS'15*, 2015.

N. Fusi, R. Sheth, and M. Elibol. Probabilistic Matrix Factorization for Automated Machine Learning. In *Proc. of NeurIPS'18*. 2018.

C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, et al. Array programming with NumPy. *Nature*, 585, 2020.

W. McKinney. Data Structures for Statistical Computing in Python. In *Proc. of SciPy*, 2010.

R. S. Olson, W. La Cava, P. Orzechowski, R. J. Urbanowicz, and J. H. Moore. PMLB: a large benchmark suite for machine learning evaluation and comparison. *BioData Mining*, 10(36), 2017.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, et al. Scikit-learn: Machine Learning in Python. *JMLR*, 12, 2011.

V. Perrone, R. Jenatton, M. Seeger, and C. Archambeau. Scalable Hyperparameter Transfer Learning. In *Proc. of NeurIPS'18*. 2018.

B. Strang, P. van der Putten, J. N. van Rijn, and F. Hutter. Don't Rule Out Simple Models Prematurely: A Large Scale Benchmark Comparing Linear and Non-linear Classifiers in OpenML. In *Proc. of IDA XVII*, 2018.

J. N. van Rijn. *Massively Collaborative Machine Learning*. PhD thesis, Leiden University, 2016.

J. N. van Rijn and F. Hutter. Hyperparameter Importance Across Datasets. In *Proc. of KDD'18*, 2018.

J. Vanschoren, J. N. van Rijn, B. Bischl, and L. Torgo. OpenML: Networked science in machine learning. *SIGKDD*, 15(2):49–60, 2014.

P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, et al. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17, 2020.

8

OPENML BENCHMARKING SUITES

Bischl, Bernd, Giuseppe Casalicchio, *Matthias Feurer*, Pieter Gijsbers, Frank Hutter, Michel Lang, Rafael G. Mantovani, Jan N. van Rijn, and Joaquin Vanschoren (2021). "OpenML Benchmarking Suites." In: *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*. Ed. by J. Vanschoren and S. Yeung. Vol. 1.

PAPER SUMMARY. This paper introduces benchmarking suites, collections of clearly defined machine learning tasks, for reproducible and simplified machine learning research. Moreover, this paper describes their implementation into `OpenML.org`, introduces an exemplary benchmarking suite (OpenML-CC18), and discusses other existing suites.

PROJECT IDEA. This project idea was developed by Bernd Bischl, Giuseppe Casalicchio, Matthias Feurer, Frank Hutter, Michel Lang, Rafael G. Mantovani, Jan N. van Rijn, and Joaquin Vanschoren.

IMPLEMENTATION AND EXPERIMENTATION. Most of the work was put in creating the OpenML-CC18: defining selection criteria for datasets and curating datasets according to these criteria, and this work was conducted by Bernd Bischl, Giuseppe Casalicchio, Matthias Feurer, Frank Hutter, Michel Lang, Rafael G. Mantovani, Jan N. van Rijn and Joaquin Vanschoren. Jan N. van Rijn and Joaquin Vanschoren were responsible for the REST API and several server extensions to provide an interface to OpenML for creating and retrieving benchmarking suites. Matthias Feurer, Jan N. van Rijn and Pieter Gijsbers were responsible for extending and adapting the Python interface. Giuseppe Casalicchio was responsible for extending and adapting the R interface. Jan N. van Rijn was responsible for extending the WEKA interface.

PAPER WRITING. The paper was drafted by all co-authors jointly except for the section on the AutoML benchmark, which was drafted by Pieter Gijsbers. The paper was then revised by all co-authors. The final version was, to a large extent, written by Pieter Gijsbers, Matthias Feurer, Giuseppe Casalicchio, and Joaquin Vanschoren.

OpenML Benchmarking Suites

Bernd Bischl[1]*, **Giuseppe Casalicchio**[1], **Matthias Feurer**[2], **Pieter Gijsbers**[3], **Frank Hutter**[2,4], **Michel Lang**[5], **Rafael G. Mantovani**[6], **Jan N. van Rijn**[7], **Joaquin Vanschoren**[3]
[1] Department of Statistics, LMU Munich, Germany
[2] Department of Computer Science, University of Freiburg, Germany
[3] Department of Computer Science, Eindhoven University of Technology, the Netherlands
[4] Bosch Center for Artificial Intelligence
[5] Department of Statistics, TU Dortmund University, Germany
[6] Federal Technology University Paraná (UTFPR), Brazil
[7] Leiden Institute of Advanced Computer Science (LIACS), Leiden University, the Netherlands

Abstract

Machine learning research depends on objectively interpretable, comparable, and reproducible algorithm benchmarks. We advocate the use of curated, comprehensive *suites* of machine learning tasks to standardize the setup, execution, and reporting of benchmarks. We enable this through software tools that help to create and leverage these benchmarking suites. These are seamlessly integrated into the OpenML platform, and accessible through interfaces in Python, Java, and R. OpenML benchmarking suites (a) are easy to use through standardized data formats, APIs, and client libraries; (b) come with extensive meta-information on the included datasets; and (c) allow benchmarks to be shared and reused in future studies. We then present a first, carefully curated and practical benchmarking suite for classification: the **OpenML C**urated **C**lassification benchmarking suite 20**18** (OpenML-CC18). Finally, we discuss use cases and applications which demonstrate the usefulness of OpenML benchmarking suites and the OpenML-CC18 in particular.

1 Introduction

Algorithm benchmarks shine a beacon for machine learning research. They allow us, as a community, to track progress over time, identify challenging issues, to raise the bar and learn how to do better. To learn as much as possible from them, they must include well-designed, challenging sets of tasks, be easily accessible and practical to use. Evaluations of algorithms on these tasks should be performed in standardized ways to support a rigorous analysis and clear conclusions. And above all, these evaluations must be easy to find, easily interpretable, reproducible, and directly comparable to evaluations run by other scientists.

The OpenML platform [Vanschoren et al., 2013] already serves thousands of datasets together with tasks in a machine-readable way. Tasks define the evaluation procedure for a specific dataset. Concretely, a task contains a reference to a dataset, information on the task type (e.g., classification or regression), the target feature (in the case of supervised problems), the evaluation procedure (e.g., k-fold CV, hold-out), the specific splits for that procedure, and the target performance metric, which together allow for reproducible evaluation schemes. OpenML is also integrated into many machine learning libraries, so that fine details about machine learning models (or pipelines) and their performance evaluations can be automatically collected. This integration allows experiments to be automatically shared and organized on the platform, linked to the underlying datasets and tasks.

*Authors are ordered alphabetically. Correspondence to {bernd.bischl | giuseppe.casalicchio}@lmu.de.

35th Conference on Neural Information Processing Systems (NeurIPS 2021) Track on Datasets and Benchmarks.

However, OpenML did not yet facilitate the simple creation and sharing of well-designed benchmark suites and results of experiments ran on them.

We introduce a novel benchmarking layer on top of OpenML, fully integrated into the platform and its APIs, that streamlines the creation of *benchmarking suites*, i.e., collections of tasks designed to thoroughly evaluate algorithms. These suites can then be easily imported, used in systematic benchmarking experiments, and the results can be automatically shared and organized on the OpenML platform, where they can be easily searched, reused and compared to the results of others. We develop tools that allow for creating a well-defined benchmark suite, and propose a new benchmark suite designed with these tools: the **C**urated **C**lassification benchmarking suite 20**18** (OpenML-CC18).

In short, the contributions of this paper are as follows: (1) we advocate the use of curated, comprehensive *suites* of machine learning tasks (i.e., a dataset with meta-information about the evaluation procedure) to standardize benchmarking, (2) we provide software tools to easily create and use these benchmarking suites, (3) we propose a new benchmark suite (OpenML-CC18), (4) have a closer look at an existing AutoML benchmark suite, and (5) discuss their impact on machine learning research. [1]

We will first discuss related work. Next, we explain how OpenML benchmarking suites work and how to use them in practice. We then present the OpenML-CC18 and review other benchmarking suites, including the AutoML benchmark. Finally, we discuss the impact of benchmarking suites on machine learning research and present our conclusions.

2 A Brief History of Benchmarking Suites

The machine learning field has long recognized the importance of dataset repositories. The UCI repository [Dheeru and Taniskidou, 2017] and LIBSVM [Chang and Lin, 2011] offer a wide range of datasets. Many more focused repositories also exist, such as UCR [Chen et al., 2015] for time series data and Mulan [Tsoumakas et al., 2011] for multilabel datasets. Some repositories also provide programmatic access. `Kaggle.com` and PMLB [Olson et al., 2017] offer a Python API for downloading datasets, skdata [Bergstra et al., 2015] offers a Python API for downloading computer vision and natural language processing datasets, and KEEL [Alcala et al., 2010] offers a Java and R API for imbalanced classification and datasets with missing values.

Several platforms can also link datasets to reproducible experiments (similar to OpenML tasks). Reinforcement learning environments such as the OpenAI Gym [Brockman et al., 2016] run and evaluate reinforcement learning experiments, the COCO suite standardizes benchmarking for black-box optimization [Hansen et al., 2020] and ASLib provides a benchmarking protocol for algorithm selection [Bischl et al., 2016a]. The Ludwig Benchmarking Toolkit orchestrates the use of datasets, tasks and models for personalized benchmarking and so far integrates the Ludwig deep learning toolbox [Narayan et al., 2021]. `PapersWithCode` maintains a manually updated overview of model evaluations linked to datasets.

Although for many years machine learning researchers have benchmarked their algorithms on some subset of these datasets, this has not yet led to standardized benchmarks that can be easily compared between individual studies. This often results in suboptimal shortcuts in study design, producing rather small-scale experiments that should be interpreted with caution [Aha, 1992], are hard to reproduce [Pedersen, 2008, Hutson, 2018], and even lead to contradictory results [Keogh and Kasetty, 2003]. An often criticized aspect is the competitive mindset in benchmarking which focuses too much on dominating the state-of-art on a few datasets, instead of a rigorous and informative analysis of large-scale studies, including negative results where popular algorithms fail [Sculley et al., 2018].

3 OpenML

OpenML is a collaborative platform that allows anyone to share new datasets, and enables anyone to easily import these datasets and subsequently share their own models and experiments run on them. It organizes everything based on four fundamental, machine-readable building blocks: (1) the *data*, (2) the machine learning *task* to be solved, specifying the dataset, the task type (e.g., classification or

[1] We previously published a preprint on arXiv, which has already been used in new research. This is the reason we can both introduce OpenML-CC18 and benchmark suites technology, but also review their use. For example, the AutoML benchmark suite was created with the technology described in this paper (and the preprint).

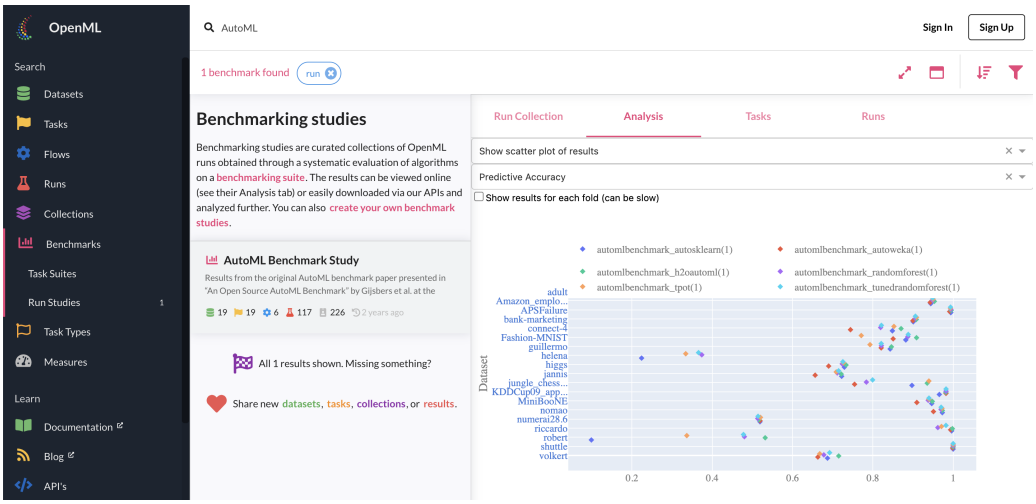


Figure 1: OpenML website showing a list of benchmark studies on the left, and interactive exploration of the results of the AutoML Benchmark (see Section 7.1) on the right. Can be viewed online at `https://www.openml.org/s/226`.

regression), the target feature (in the case of supervised problems), the evaluation procedure (e.g., k-fold CV, hold-out), the specific splits for that procedure, and the target performance metric (3) the *flow* which specifies a machine learning pipeline that solves the *task*, and (4) the *run* that contains experiment results (e.g., predictions and performance evaluations) when a *flow* is executed on a *task* (see Vanschoren et al. [2013] for more details). OpenML goes beyond the platforms mentioned in Section 2, as it includes extensive programmatic access to all datasets, tasks, flows, and runs, comprehensive logging of experiments, and automated sharing of results, which have enabled the collection of millions of publicly shared and reproducible experiments, linked to the exact datasets, machine learning pipelines and hyperparameter settings. OpenML offers bindings with the Java, Python and R ecosystems [van Rijn, 2016, Feurer et al., 2021b, Casalicchio et al., 2017] to provide easy integration in common machine learning tools, workflows, and environments. An introduction and detailed information can be found on `https://docs.openml.org`.

4 OpenML Benchmarking Suites

As with any platform where people can upload new datasets, an overwhelming amount and variety of datasets is available, and it can be unclear how well they are curated. We designed OpenML benchmarking suites as a remedy to allow researchers to compile and publish well-defined collections of curated tasks and datasets, and collect benchmarking results from many scientists in a single place. More precisely, we define:

An OpenML benchmarking suite is a set of OpenML tasks carefully selected to evaluate algorithms under a precise set of conditions.

Using a set of *tasks* instead of a set of *datasets* makes experiments performed on them comparable and reproducible. Compared to other (static) collections of datasets, the use of OpenML benchmarking suites has the following advantages:

- Easy creation of benchmarks (see Section 5.1): OpenML hosts thousands of datasets, and scientists can easily filter them down to those needed for their benchmarks (see Sections 6 and 7 for examples).
- Convenient access and sharing of suites: Each suite receives a unique ID, which can be used to retrieve the suite via APIs, and via its own webpage. Figure 1 illustrates how results collected on these suites can be explored online.
- Permanence and provenance: Because benchmarking suites are its own entity on OpenML, it is clear who created them (provenance). It also guarantees no one but the original creator can edit or remove the suite (permanence), this is an advantage over the previously used community tagging mechanism which allowed any user to add tasks to a suite.

- Community of practice: Curated benchmark suites allow scientists to thoroughly benchmark their machine learning methods without having to worry about finding and selecting datasets for their benchmarks.
- Building on existing suites: Scientists can extend, subset, or adapt existing benchmarking suites to correct issues, raise the bar, or run personalized benchmarks.
- Reproducibility of benchmarks: Based on machine-readable OpenML *tasks*, with detailed instructions for evaluation procedures and train-test splits, shared results are comparable and reproducible.
- Conducting benchmark studies: After creating an OpenML benchmarking suite, existing and new experiments (*runs*) on the underlying *tasks* can be associated with the suite. This is also illustrated in Figure 3. Such data reuse bootstraps the creation of new benchmark studies that can analyze existing machine learning algorithms in new ways, or to design new challenging benchmark suites.
- Collaborative work: OpenML benchmarking suites benefit from the OpenML community, where users can help to identify and report bugs and errors in the contained datasets.
- Dynamic benchmarks: Benchmarks are never perfect, and when used for a long time, scientists may overfit on specific sets of tasks. However, benchmarking suites can be easily corrected and extended over time (e.g., on a yearly basis), leading to dynamic benchmarks that respond to novel concerns, and evaluate methods on new and ever more challenging tasks. More than providing a snapshot, this allows longitudinal studies that truly track progress over time.

5 How to Use OpenML Benchmarking Suites

To realize all these benefits, we have developed a series of extensions to the OpenML platform:[2]

- We added the concepts of a 'benchmark suite' as a collection of *tasks*, and a 'benchmark study' as a collection of benchmark results (*runs*) obtained on them.
- We added data filtering procedures to the APIs and website that allow researchers to exactly specify the constraints for tasks to be included in a benchmark suite.
- We provide scripts and notebooks that facilitate the creation and quality assessment of benchmark suites. For instance, they filter out datasets that are modeled too easily, and hence cannot be used to differentiate between most algorithms (see Section 5.1).
- Certain types of datasets, such as multilabel, time series, or artificial datasets, may require additional care. We added collaborative and automated annotation (tagging) to filter such datasets accordingly.

In the following, we discuss the three main use cases for benchmarking suites, i.e., creating new suites, retrieving existing suites, and running benchmarks. We provide code examples on how to retrieve, iterate the contents of a benchmark suite and run machine learning algorithms on it in Figure 2.[3]

5.1 Creating New Suites

To collect data sets for a new suite, one usually starts by determining a list of constraints that datasets or tasks should adhere to (e.g., have a minimal size, a limited amount of class imbalance, and not be a time series). This is often an iterative refinement process, during which the distribution of currently selected tasks can be visualized, and any existing benchmarking results on these tasks can be retrieved. An example of this workflow is illustrated in the provided notebook.[4] The final selection of tasks can then be used to create a new benchmark suite. Each benchmark suite is assigned a unique id and an overview webpage with a description and an analysis dashboard (e.g., `https://www.openml.org/s/99`). The description text can be used to describe the goals and design criteria, provide links to external resources, and address any ethical concerns that should be taken into consideration when using the benchmark suite. We give an exemplary curation protocol in Appendix C.

5.2 Retrieving Existing Suites

Existing benchmark suites can be easily downloaded via any of the OpenML client libraries using its unique id or alias (see Figure 2). The tasks and datasets are all uniformly formatted, and come

[2] All code is open, BSD-3 licenced, and available on `https://github.com/openml`

[3] More detailed and up-to-date instructions can be found on: `https://docs.openml.org/benchmark`

[4] Notebooks can be found at `https://github.com/openml/benchmark-suites`

with extensive meta-data to streamline the execution of benchmarks on them. For instance, if a dataset contains missing values, this is indicated in a machine-readable way so that researchers

```python
from openml import config, study, tasks, runs, extensions
from sklearn import compose, impute, metrics, pipeline, preprocessing, tree

clf = pipeline.make_pipeline(
    compose.make_column_transformer(
        (impute.SimpleImputer(), extensions.sklearn.cont),
        (preprocessing.OneHotEncoder(handle_unknown='ignore'), extensions.sklearn.cat),
    ),
    tree.DecisionTreeClassifier(max_depth=1)
)  # build a fast and simple classification pipeline

benchmark_suite = study.get_suite('OpenML-CC18')  # obtain the benchmark suite
# config.apikey = 'FILL_IN_OPENML_API_KEY'        # uploading to OpenML requires an API key

run_ids = []
for task_id in benchmark_suite.tasks:             # iterate over all tasks
    task = tasks.get_task(task_id)                # download the OpenML task
    X, y = task.get_X_and_y()                     # get the data (not used in this example)
    run = runs.run_model_on_task(clf, task)       # run classifier on splits given by the task
    score = run.get_metric_fn(metrics.accuracy_score) # compute and print the accuracy score
    print(f'Data set: {task.get_dataset().name}; Accuracy: {score.mean():.2}')
    run.publish()
    run_ids.append(run.id)

benchmark_study = study.create_study(             # create a study to share the set of results
    name="CC18-Example",
    description="An example study reporting results of a decision stump.",
    run_ids=run_ids,
    benchmark_suite=benchmark_suite.id
)
benchmark_study.publish()
print(f"Results are stored at {benchmark_study.openml_url}")
```

(a) Python, available as pypi package OpenML

```java
public static void runTasksAndUpload() throws Exception {
  OpenmlConnector openml = new OpenmlConnector("FILL_IN_OPENML_API_KEY");
  Study benchmarksuite = openml.studyGet("OpenML-CC18", "tasks");    // obtain the benchmark suite
  Classifier tree = new REPTree();                                    // build a Weka classifier
  for (Integer taskId : benchmarksuite.getTasks()) {                 // iterate over all tasks
    Task t = openml.taskGet(taskId);                                  // download the OpenML task
    Instances d = InstancesHelper.getDatasetFromTask(openml, t);      // obtain the dataset
    Pair<Integer, Run> result = RunOpenmlJob.executeTask(openml, new WekaConfig(), taskId, tree);
    Run run = openml.runGet(result.getLeft());
  }
}
```

(b) Java, available on Maven Central with artifact id org.openml.openmlweka

```r
library(OpenML)                                   # requires at least package version 1.8
library(mlr)
lrn = makeLearner('classif.rpart')                # construct a simple CART classifier
bsuite = getOMLStudy('OpenML-CC18')               # obtain the benchmark suite
task.ids = extractOMLStudyIds(bsuite, 'task.id')  # obtain the list of suggested tasks
for (task.id in task.ids) {                       # iterate over all tasks
  task = getOMLTask(task.id)                      # download single OML task
  data = as.data.frame(task)                      # obtain raw data set
  run = runTaskMlr(task, learner = lrn)           # run constructed learner
  setOMLConfig(apikey = 'FILL_IN_OPENML_API_KEY')
  upload = uploadOMLRun(run)                      # upload and tag the run
}
```

(c) R, available on CRAN via package OpenML

Figure 2: Complete code examples, in different programming languages, of how any benchmarking suite (here the 'OpenML-CC18' suite) can be downloaded and used to evaluate a given algorithm. The Python code also creates a new benchmark study and shares all results. Uploading requires a (free) API key.

can automatically adjust for this when running their algorithms. Datasets can be investigated using exploratory data analysis tools, and existing runs on these tasks can be downloaded and analyzed.

5.3 Running Benchmarks

After retrieving the tasks from a suite, new experiments can be conducted locally. As illustrated in Figure 2, this is easiest with the readily integrated machine learning libraries, such as scikit-learn [Pedregosa et al., 2011], mlr [Bischl et al., 2016b] or its successor mlr3 [Lang et al., 2019], and Weka [Hall et al., 2009]. Integrations for deep learning libraries are under development, and we welcome further open source integrations.[5] Custom code can often be wrapped, e.g., using the scikit-learn interface.

The results of these experiments (runs) can also (optionally) be bundled in a benchmark study and published on OpenML, as illustrated for Python in Figure 2. Runs include all experiment details, including hyperparameter configurations, in a structured way. This allows entire communities of scientists to bring together benchmarks of a wide range of algorithms, all evaluated uniformly on the same tasks, in a single place where they can be directly compared on predictive performance and analysed in novel ways. Figure 3 visualizes the results of 3.8 million runs collected on a single benchmarking suite, which we will discuss next.

6 OpenML-CC18

To demonstrate the functionality of OpenML benchmarking suites, we created a first standard of 72 classification tasks built on a carefully curated selection of datasets from the many thousands available on OpenML: the OpenML-CC18. It can be used as a drop-in replacement for many typical benchmarking setups. These datasets are deliberately medium-sized for practical reasons. An overview of the benchmark suite can be found at `https://www.openml.org/s/99` and in Table 1. We first describe the design criteria of the OpenML-CC18 before discussing uses of the benchmark and success stories.[1,6]

6.1 Design Criteria

The OpenML-CC18 contains all verified and publicly licenced OpenML datasets until mid-2018 that satisfy a large set of clear requirements for thorough yet practical benchmarking:

(a) The number of observations is between 500 and $100\,000$ to focus on medium-sized datasets that can be used to train models on almost any computing hardware.
(b) The dataset has less than 5000 features, counted after one-hot-encoding categorical features (which is the most frequent way to deal with categorical variables), to avoid most memory issues.
(c) The target attribute has at least two classes, with no class of less than 20 observations. This ensures sufficient samples per class per fold when running 10-fold cross-validation experiments.
(d) The ratio of the minority and majority class is above 0.05 (to eliminate highly imbalanced datasets which require special treatment for both algorithms and evaluation measures).
(e) The dataset is not sparse because not all machine learning models can handle them gracefully, this constraint facilitates our goal of wide applicability.
(f) The dataset does not require taking time dependency between samples into account, e.g., time series or data streams, as this is often not implemented in standard machine learning libraries. As a precaution, we also removed datasets where each sample constitutes a single data stream.
(g) The dataset does not require grouped sampling. Such datasets would contain multiple data points for one subject and require that all data points for a subject are put into the same data split for evaluation. We introduce this constraint and the one above to simplify usage of the datasets, as one does not have to use specialized cross-validation procedures.

We also applied several more opinionated criteria to avoid issues with problematic datasets:

[5] Development is carried out on GitHub. Contributor guidance can be found at `https://docs.openml.org`.

[6] The OpenML-CC18 is the successor of a preliminary benchmarking study called OpenML100, containing 100 classification datasets, and fixes several issues we encountered when working with the OpenML100.

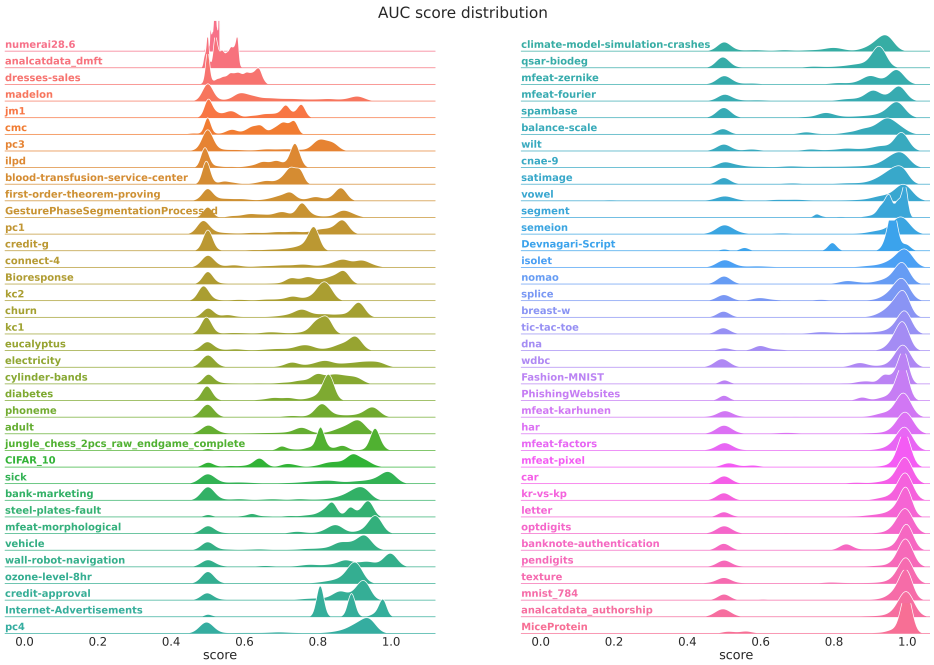


Figure 3: Distribution of the scores (average area under ROC curve, weighted by class support) of 3.8 million experiments with thousands of machine learning pipelines, shared on the CC18 benchmark tasks. Some tasks prove harder than others, some have wide score ranges, and for all there exist models that perform poorly (0.5 AUC). Code to reproduce this figure (for any metric) is available on GitHub.[4]

(a) We strived to remove artificial datasets, as it is hard to reliably assess their difficulty. Admittedly, there is no perfect distinction between artificial and simulated datasets (for example, a lot of phenomena can be simulated that can be as simple as an artificial dataset). Therefore, we removed datasets if we were in doubt of whether they are simulated or artificial.
(b) We removed datasets which are a subset of larger datasets. Allowing subsets would be very subjective, as there is no objective choice of a dataset subset size or a subset of the variables or classes.
(c) We excluded tasks for which the original target feature has been transformed or changed, e.g., when classes of a categorical target feature were merged or when a continuous target feature (for original regression tasks) was discretized to create a classification task.
(d) We removed datasets without any source or reference. We want to be able to learn more about their intended use and how to interpret learned models, and avoid black box datasets.

Finally, to ensure that datasets are sufficiently challenging, we applied the following restrictions:

(a) We removed datasets which can be perfectly classified by a single attribute or a decision stump, as they do not allow us to meaningfully compare machine learning algorithms.
(b) We removed datasets where a decision tree could achieve 100% accuracy on a 10-fold cross-validation task, to remove datasets which can be solved by a simple algorithm which is prone to overfitting training data. We found that this is a good indicator of too easy datasets. Obviously, other datasets will appear easy for several algorithms, and we aim to learn more about the characteristics of such datasets in future studies.

We created the OpenML-CC18 as a first, practical benchmark suite. In hindsight, we acknowledge that our initial selection still contains several mistakes. Concretely, *sick* is a newer version of the *hypothyroid* dataset with several classes merged, *electricity* has time-related features, *balance_scale* is an artificial dataset and *mnist_784* requires grouping samples by writers. We will correct these mistakes in new versions of this suite and also screen the more than 900 new datasets that were uploaded to OpenML since the creation of the OpenML-CC18. Moreover, to avoid the risk of

overfitting on a specific benchmark, and to include feedback from the community, we plan to create a dynamic benchmark with regular release updates that evolve with the machine learning field. We want to clarify that while we include some datasets which may have ethical concerns, we do not expect this to have an impact if the suite is used responsibly (i.e., the benchmark suite is used for its intended purpose of benchmarking algorithms, and not to construct models to be used in real-world applications).

6.2 Usage of the OpenML-CC18

The OpenML-CC18 has been acknowledged and used in various studies.[1] For instance, Van Wolputte and Blockeel [2020] used it to study iterative imputation algorithms for imputing missing values, König et al. [2020] used it to develop methods to improve upon uncertainty quantification of machine learning classifiers and De Bie et al. [2020] introduced deep networks for learning meta-features, which they computed for all OpenML-CC18 datasets. In some cases, the authors needed a filtered subset of the OpenML-CC18, which is natively supported in most OpenML clients. Other uses of the OpenML-CC18 include interpreting its multiclass datasets as multi-arm contextual bandit problems [Bibaut et al., 2021a,b] and using the individual columns to test quantile sketch algorithms [Mitchell et al., 2021].

Cardoso et al. [2021] claim that the machine learning community has a strong focus on algorithmic development, and advocate a more data-centric approach. To this end, they studied the OpenML-CC18 utilizing methods from Item Response Theory to determine which datasets are hard for many classifiers. After analyzing 60 of its datasets (excluding the largest), they find that the OpenML-CC18 consists of both easy and hard datasets. They conclude that the suite is not very challenging as a whole, but that it includes many appropriate datasets to distinguish good classifiers from bad classifiers, and then propose two subsets: one that can be considered challenging, and one subset to replicate the behavior of the full suite. The careful analysis and subsequent proposed updates are a nice example of the natural evolution of benchmarking suites.

For completeness, we also briefly mention uses of OpenML100, a predecessor of the OpenML-CC18 that includes 100 datasets and less strict constraints. Fabra-Boluda et al. [2020] use this suite to build a taxonomy of classifiers. They argue that the taxonomies provided by the community can be misleading, and therefore learn taxonomies to cluster classifiers based on predictive behavior. van Rijn and Hutter [2018] and Probst et al. [2019a] used it to quantify the hyperparameter importance of machine learning algorithms, while Probst et al. [2019b] used it to learn the best strategy for tuning random forest based on large-scale experiments (although Probst et al. [2019a] and Probst et al. [2019b] use only the binary datasets without missing values).

Based upon these works, we conclude that the OpenML-CC18 is being used to facilitate very diverse directions of machine learning research.

7 Further OpenML Benchmarking Suites

We now review other OpenML benchmarking suites. For this, we focus on AutoML benchmarking suites, but also provide examples of others.

7.1 The AutoML Benchmark Suite

The AutoML benchmark [Gijsbers et al., 2019] also makes use of an OpenML benchmark suite to evaluate AutoML tools in a reproducible manner. Combined with code to automatically run experiments, any of the integrated AutoML tools can be evaluated on any suitable OpenML task or suite directly from the command line.

7.1.1 Benchmark Suite Design

The AutoML benchmark explicitly sources part of their datasets from the OpenML-CC18, but also includes datasets used in AutoML competitions (primarily Guyon et al. [2019]) or previous comparisons of AutoML systems. A step-by-step list of recreating the benchmark suite does not exist, but general guidelines are provided. Since the original release in 2019, the AutoML benchmark has

been extending their selection of datasets.[7,8] In the discussion below, aspects which are specific to the newer selection are indicated with an asterisk (*).

The suite shares some of its design criteria with OpenML-CC18, such as the minimum number of instances, as well as the exclusion of artificial datasets and those which require grouped sampling. However, it loosens some other restrictions specifically because of the assumption that AutoML tools should be able to deal with additional complexities:

(a) There is no limit to 100 000 instances or 5000 features, tools can restrict themselves to learners which scale well or use, e.g., low-fidelity estimates.
(b) There is no limit for class imbalance, tools can use their preferred techniques to deal with imbalanced data (e.g., SMOTE [Chawla et al., 2002]).
(c) It includes sparse data, though it is currently converted to dense format for tools that don't support sparse data.*
(d) It includes a suite of regression problems.*

Some other restrictions are instead stricter because of the tabular AutoML context:

(a) The "easy dataset" filter also takes into account results from OpenML across various learners, to try to avoid datasets which need little search beyond algorithm selection.
(b) The number of image classification problems is explicitly restricted, as they are typically better solved with Deep Learning and the benchmark's focus is tabular AutoML tools.

Similar to OpenML-CC18, the AutoML benchmark suite is intended to be regularly updated to reflect modern day challenges and to avoid overfitting.

7.1.2 Usage of the AutoML Benchmark Suite

Before the introduction of the AutoML benchmark suite, the closest to an accepted standard for tabular AutoML benchmarking was the set of datasets on which Auto-WEKA was originally evaluated [Thornton et al., 2013]. This selection of tasks was still used in, e.g., Mohr et al. [2018] and consisted of 21 problems, a third of which are image classification tasks which are typically not the intended use-case for the AutoML tools. However, it was by no means a standard. For example, Drori et al. [2018], Rakotoarison et al. [2019] and Gil et al. [2018], all published at the same workshop, each used different selections of datasets.

The original AutoML benchmark suite has been used in multiple AutoML publications, either as is [LeDell and Poirier, 2020, Wang et al., 2021, Feurer et al., 2021a] or with modifications. Sometimes more datasets are used, as Zöller and Huber [2021] combine it with OpenML-CC18 and OpenML100 and Kadra et al. [2021] add datasets from UCI and Kaggle. For the latter, hold-out evaluation is used instead of the suite-defined 10-fold cross-validation. Erickson et al. [2020] use additional datasets from Kaggle competitions to compare directly to solutions proposed by human competitors.

Other times not all datasets in the benchmark suite are used, e.g., Zimmer et al. [2021] uses all but four big datasets for computational reasons, while Parmentier et al. [2019] limit themselves to only four of the big datasets in the suite to assess their method designed for big datasets. Mohr and Wever [2021] omitted some datasets because of technical issues.

7.2 Further Existing OpenML Benchmarking Suites

OpenML contains other benchmark suites as well, such as the OpenML100-friendly that only contains the subset of the OpenML100 without missing values and with only numerical features, or Foreign Exchange data for machine learning research [Schut et al., 2019].

We invite the community to create additional benchmarks suites for other tasks besides classification, for larger datasets or more high-dimensional ones, for imbalanced or extremely noisy datasets, as well as for text, time series, and many other types of data. We are confident that benchmarking suites will help standardize evaluation and track progress in many subfields of machine learning, and also intend to create new suites and make it ever easier for others to do so.

[7] Announcement of the new suites: `https://github.com/openml/automlbenchmark/issues/187`

[8] `https://www.openml.org/s/{218,269,271}` are the original, regression, and expanded suite, respectively

8 Limitations and Future Work

As benchmarking suites are increasingly being picked up by the machine learning community, we also observed several limitations that should be tackled in future work.

Overfitting. While it has not yet been demonstrated, we assume that as more methods are being evaluated on benchmarking suites, overfitting on fixed suites is increasingly likely. We therefore aim to periodically update existing suites with new datasets that follow the specifications laid out by the benchmark designers (e.g., as done for computer vision research [Recht et al., 2019]) and invite the community to extend existing suites with harder tasks, as done in NLP research [Kiela et al., 2021].

Credit Assignment. Curating a benchmark is a lot of work, and we have manually inspected and corrected datasets for the OpenML-CC18 over the course of multiple months. It is therefore important to give proper credit to everyone involved in creating benchmarking suites, for example by somehow making benchmarking suites citable.

Automating the curation of useful suites. We are not aware of any related work that describes how to curate machine learning benchmark suites. In this paper we have defined benchmarking suites by formalizing objective, but also more subjective constraints. Providing automated ways to create high quality, diverse and realistic benchmarking suites is thus an important, open research question. Additionally, post-hoc research, such as the one conducted by Cardoso et al. [2021], is important to check the validity of benchmarking suites, and we hope for more such techniques to be developed and also to become applicable during the suite design process.

Computational issues. While studying applications of the OpenML-CC18 in Section 6.2 we realized that even though we consciously focused on mid-size datasets, some larger ones still incurred too high computational load, so some researchers have used subsets of the OpenML-CC18 in their work. Future suites could more carefully trade off the completeness of benchmarking suites and computational issues, for example by choosing representative subsets [Cardoso et al., 2021].

Breadth of current benchmarking suites. On the other hand, many researchers are interested in benchmarking larger (deep learning) models on larger datasets from many domains (including language and vision). We are working on ways to enable the creation of such benchmarking suites as well, and welcome further involvement from the community.

Specification of resource constraints. The task and suite specifications do not yet allow for constraints on resources, e.g., memory or time limits. Specific benchmark studies could impose identical hardware requirements, e.g., to compare running times. Where requiring identical hardware is impractical, general constraints would ensure results are more comparable when multiple people run their experiments on a suite. Explicit constraints also help interpret earlier results.

Disclosure of ethical issues We currently encourage creators to disclose any ethical concerns with datasets in their benchmark suite in its description. In the future we want to support this natively on a dataset level (e.g., by integrating datasheets [Gebru et al., 2018]) and benchmark suite level (by providing a dedicated information field).

9 Conclusion

Our goal is to simplify the creation of well-designed benchmarks to push machine learning research forward. More than just creating and sharing benchmarks, we want to allow anyone to effortlessly run and publish their own benchmarking results and organize them online in a single place where they can be easily explored, downloaded, shared, compared, and analyzed. We created a new benchmarking layer on the OpenML platform that allows scientists to do all the above with just a few lines of code. We then introduced the OpenML-CC18, a benchmark suite created with these tools for general classification benchmarking.

The use of suites is further motivated by a closer look at the AutoML benchmark suite. We also reviewed how other scientists have adopted these benchmarking suites in their own work, from which it becomes clear that a continuous conversation with the research community is essential to evolve benchmarks and make them better and more useful over time. We hope that this work will unleash a rapid evolution of benchmarks suites and large-scale studies that teach us more about machine learning than any single study could.

Acknowledgements This work has partly been funded by the German Federal Ministry of Education and Research (BMBF) under grant no. 01IS18036A, by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – 460135501 (NFDI project MaRDI), by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme under grant no. 716721 (Beyond BlackBox) and 952215 (TAILOR), through grant #2015/03986-0 from the São Paulo Research Foundation (FAPESP), by AFRL and DARPA under contract FA8750-17-C-0141, as well as through the Priority Programme Autonomous Learning (SPP 1527, grant HU 1900/3-1) and Collaborative Research Center SFB 876/A3 from the German Research Foundation (DFG).

References

D. W. Aha. Generalizing from case studies: A case study. *Proceedings of the International Conference on Machine Learning (ICML)*, pages 1–10, 1992.

J. Alcala, A. Fernandez, J. Luengo, J. Derrac, S. Garcia, L. Sanchez, and F. Herrera. Keel datamining software tool: Data set repository, integration of algorithms and experimental analysis framework. *Journal of Multiple-Valued Logic and Soft Computing*, 17(2-3):255–287, 2010.

J. Bergstra, N. Pinto, and D. Cox. Skdata: data sets and algorithm evaluation protocols in python. *Computational Science & Discovery*, 8(1), 2015.

A. Bibaut, A. Chambaz, M. Dimakopoulou, N. Kallus, and M. van der Laan. Risk minimization from adaptively collected data: Guarantees for supervised and policy learning. *arXiv:2106.01723 [stat.ML]*, 2021a.

A. Bibaut, A. Chambaz, M. Dimakopoulou, N. Kallus, and M. van der Laan. Post-contextual-bandit inference. *arXiv:2106.00418 [stat.ML]*, 2021b.

B. Bischl, P. Kerschke, L. Kotthoff, M. Lindauer, Y. Malitsky, A. Frechétte, H. Hoos, F. Hutter, K. Leyton-Brown, K. Tierney, and J. Vanschoren. ASlib: A benchmark library for algorithm selection. *Artificial Intelligence*, 237:41–58, 2016a.

B. Bischl, M. Lang, L. Kotthoff, J. Schiffner, J. Richter, E. Studerus, G. Casalicchio, and Z. M. Jones. mlr: Machine learning in R. *Journal of Machine Learning Research*, 17(170), 2016b.

G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. OpenAI Gym. *arXiv:1606.01540 [cs.LG]*, 2016.

L. F. Cardoso, V. C. Santos, R. S. K. Francês, R. B. Prudêncio, and R. C. Alves. Data vs classifiers, who wins? *arXiv:2107.07451 [cs.LG]*, 2021.

G. Casalicchio, J. Bossek, M. Lang, D. Kirchhoff, P. Kerschke, B. Hofner, H. Seibold, J. Vanschoren, and B. Bischl. OpenML: An R package to connect to the machine learning platform OpenML. *Computational Statistics*, 34(3):977–991, 2017.

C. C. Chang and C. J. Lin. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 2(3):27, 2011.

N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer. Smote: Synthetic minority over-sampling technique. *Journal of Artificial Intelligence Research*, 16(1):321–357, 2002.

Y. Chen, E. Keogh, B. Hu, N. Begum, A. Bagnall, A. Mueen, and G. Batista. The UCR time series classification archive, July 2015. `www.cs.ucr.edu/~eamonn/time_series_data/`.

G. De Bie, H. Rakotoarison, G. Peyré, and M. Sebag. Distribution-based invariant deep networks for learning meta-features. *arXiv:2006.13708 [stat.ML]*, 2020.

D. Dheeru and E. K. Taniskidou. UCI machine learning repository, 2017. URL `http://archive.ics.uci.edu/ml`.

I. Drori, Y. Krishnamurthy, R. Rampin, R. Lourenço, J. One, K. Cho, C. Silva, and J. Freire. Alphad3m: Machine learning pipeline synthesis. In *5th ICML Workshop on Automated Machine Learning (AutoML)*, 2018.

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

R. Fabra-Boluda, C. Ferri, F. Martínez-Plumed, J. Hernández-Orallo, and M. J. Ramírez-Quintana. Family and prejudice: A behavioural taxonomy of machine learning techniques. In *ECAI 2020 - 24th European Conference on Artificial Intelligence*, pages 1135–1142. IOS Press, 2020.

M. Feurer, K. Eggensperger, S. Falkner, M. Lindauer, and F. Hutter. Auto-sklearn 2.0: Hands-free automl via meta-learning. *arXiv:2007.04074 [cs.LG]*, 2021a.

M. Feurer, J. N. van Rijn, A. Kadra, P. Gijsbers, N. Mallik, S. Ravi, A. Müller, J. Vanschoren, and F. Hutter. Openml-python: an extensible python api for openml. *Journal of Machine Learning Research*, 22(100):1–5, 2021b.

T. Gebru, J. Morgenstern, B. Vecchione, J. W. Vaughan, H. Wallach, H. Daumé III, and K. Crawford. Datasheets for datasets. *arXiv:1803.09010 [cs.DB]*, 2018.

P. Gijsbers, E. LeDell, J. Thomas, S. Poirier, B. Bischl, and J. Vanschoren. An open source automl benchmark. In *6th ICML Workshop on Automated Machine Learning (AutoML)*, 2019.

Y. Gil, K.-T. Yao, V. Ratnakar, D. Garijo, G. Ver Steeg, P. Szekely, R. Brekelmans, M. Kejriwal, F. Luo, and I.-H. Huang. P4ml: A phased performance-based pipeline planner for automated machine learning. In *5th ICML Workshop on Automated Machine Learning (AutoML)*, 2018.

I. Guyon, L. Sun-Hosoya, M. Boullé, H. J. Escalante, S. Escalera, Z. Liu, D. Jajetic, B. Ray, M. Saeed, M. Sebag, A. Statnikov, W.-W. Tu, and E. Viegas. Analysis of the automl challenge series 2015–2018. In F. Hutter, L. Kotthoff, and J. Vanschoren, editors, *Automated Machine Learning: Methods, Systems, Challenges*, pages 177–219. Springer International Publishing, 2019.

M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. The WEKA data mining software: An update. *ACM SIGKDD Explorations Newsletter*, 11(1):10–18, 2009.

N. Hansen, A. Auger, R. Ros, O. Mersman, T. Tušar, and D. Brockhoff. COCO: A platform for comparing continuous optimizers in a black-box setting. *Optimization Methods and Software*, 2020.

M. Hutson. Missing data hinder replication of artificial intelligence studies. *Science News*, 2018. URL `https://www.science.org/content/article/missing-data-hinder-replication-artificial-intelligence-studies`.

A. Kadra, M. Lindauer, F. Hutter, and J. Grabocka. Regularization is all you need: Simple neural nets can excel on tabular data. *arXiv:2106.11189 [cs.LG]*, 2021.

E. Keogh and S. Kasetty. On the need for time series data mining benchmarks: A survey and empirical demonstration. *Data Mining and Knowledge Discovery*, 7(4):349–371, 2003.

D. Kiela, M. Bartolo, Y. Nie, D. Kaushik, A. Geiger, Z. Wu, B. Vidgen, G. Prasad, A. Singh, P. Ringshia, Z. Ma, T. Thrush, S. Riedel, Z. Waseem, P. Stenetorp, R. Jia, M. Bansal, C. Potts, and A. Williams. Dynabench: Rethinking benchmarking in NLP. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 4110–4124. Association for Computational Linguistics, 2021.

M. König, H. H. Hoos, and J. N. van Rijn. Towards algorithm-agnostic uncertainty estimation: Predicting classification error in an automated machine learning setting. In *7th ICML Workshop on Automated Machine Learning (AutoML)*, 2020.

M. Lang, M. Binder, J. Richter, P. Schratz, F. Pfisterer, S. Coors, Q. Au, G. Casalicchio, L. Kotthoff, and B. Bischl. mlr3: A modern object-oriented machine learning framework in r. *Journal of Open Source Software*, 4(44):1903, 2019.

E. LeDell and S. Poirier. H2o automl: Scalable automatic machine learning. In *7th ICML Workshop on Automated Machine Learning (AutoML)*, 2020.

R. Mitchell, E. Frank, and G. Holmes. An empirical study of moment estimators for quantile approximation. *ACM Transactions on Database Systems*, 46(1), 2021.

F. Mohr and M. Wever. Replacing the ex-def baseline in automl by naive automl. In *8th ICML Workshop on Automated Machine Learning (AutoML)*, 2021.

F. Mohr, M. Wever, and E. Hüllermeier. Ml-plan: Automated machine learning via hierarchical planning. *Machine Learning*, 107(8-10):1495–1515, 2018.

A. Narayan, P. Molino, K. Goel, W. Neiswanger, and C. Re. Personalized benchmarking with the ludwig benchmarking toolkit. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, 2021.

R. S. Olson, W. La Cava, P. Orzechowski, R. J. Urbanowicz, and J. H. Moore. PMLB: A large benchmark suite for machine learning evaluation and comparison. *BioData Mining*, 10(36), 2017.

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

T. Pedersen. Empiricism is not a matter of faith. *Computational Linguistics*, 34:465–470, 2008.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

P. Probst, A.-L. Boulesteix, and B. Bischl. Tunability: Importance of hyperparameters of machine learning algorithms. *Journal of Machine Learning Research*, 20(53):1–32, 2019a.

P. Probst, M. N. Wright, and A.-L. Boulesteix. Hyperparameters and tuning strategies for random forest. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 9(3):e1301, 2019b.

H. Rakotoarison, M. Schoenauer, and M. Sebag. Automated machine learning with monte-carlo tree search. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence*, pages 3296–3303, 2019.

B. Recht, R. Roelofs, L. Schmidt, and V. Shankar. Do ImageNet classifiers generalize to ImageNet? In K. Chaudhuri and R. Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97, pages 5389–5400, 2019.

F. Schut, J. N. van Rijn, and H. Hoos. Towards automated technical analysis for foreign exchange data. In *Workshop on Automating Data Science @ ECML/PKDD*, 2019.

D. Sculley, J. Snoek, A. Wiltschko, and A. Rahimi. Winner's curse? on pace, progress, and empirical rigor. In *Workshop of the International Conference on Representation Learning (ICLR)*, 2018.

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

G. Tsoumakas, E. Spyromitros-Xioufis, J. Vilcek, and I. Vlahavas. Mulan: A java library for multi-label learning. *Journal of Machine Learning Research*, pages 2411–2414, Jul 2011.

J. N. van Rijn. *Massively collaborative machine learning*. PhD thesis, Leiden University, 2016.

J. N. van Rijn and F. Hutter. Hyperparameter importance across datasets. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 2367–2376. ACM, 2018.

E. Van Wolputte and H. Blockeel. Missing value imputation with mercs: A faster alternative to missforest. In *Discovery Science - 23rd International Conference*, volume 12323 of *Lecture Notes in Computer Science*, pages 502–516. Springer, 2020.

J. Vanschoren, J. N. van Rijn, B. Bischl, and L. Torgo. OpenML: Networked science in machine learning. *SIGKDD Explorations*, 15(2):49–60, 2013.

C. Wang, Q. Wu, M. Weimer, and E. Zhu. FLAML: A fast and lightweight automl library. *Proceedings of Machine Learning and Systems*, 3, 2021.

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

M.-A. Zöller and M. F. Huber. Benchmark and survey of automated machine learning frameworks. *Journal of Artificial Intelligence Research*, 70:409–472, 2021.

Part V

CONCLUSION

9 CONCLUSION

In this final part of the thesis, we discuss how far we have come to make machine learning more accessible by efficient and robust Automated Machine Learning. First, we summarize what we have achieved and discuss the impact of our work. Second, and finally, we state what we deem important future work.

9.1 SUMMARY AND DISCUSSION

Our goal with this thesis was to find answers to the question *Can we make machine learning more accessible by efficient and robust automated machine learning*? In order to do so we have conducted work and research in the three directions outlined in Section 1.2:

1. *Hyperparameter Optimization (HPO)*: Can we improve the applicability of HPO to costly problems?
2. *Automated Machine Learning (AutoML)*: Can we improve the efficiency of Combined Algorithm Selection and Hyperparameter Optimization (CASH) for supervised learning?
3. *Benchmarking and the OpenML platform*: Can we facilitate AutoML research and development by providing better data to the researcher?

We now summarize and discuss each direction in turn in the following three subsections.

9.1.1 *Hyperparameter optimization*

In Part ii, we have advanced the state of the art for efficient HPO by surveying HPO methods with a focus on efficient multi-fidelity methods and by developing a new HPO method that can take into account prior knowledge.

First, we have summarized the state-of-the-art in HPO in Chapter 3. We have put a strong focus on multi-fidelity HPO and described the three overarching approaches: learning curves, bandits, and adaptive methods. Multi-fidelity HPO continues to receive a lot of attention, and there are now multi-fidelity versions of population-based algorithms (Parmentier et al., 2019; Awad, Mallik, and Hutter, 2021) and new multi-fidelity extensions of popular acquisition functions in Bayesian optimization (Wu et al., 2019; Takeno et al., 2020).

Multi-fidelity is becoming even more important as models continue to increase in size (Kaplan et al., 2020). The most important follow-up on this work is *HPOBench* (Eggensperger, Müller, et al., 2021, also co-authored by the author of this thesis), which empirically verifies the benefits of multi-fidelity optimization compared with their single-fidelity counterparts.

We have also posed open questions, and we are excited about the tremendous progress in solving some of the open problems we have identified. There are now multiple benchmark collections available (Eggensperger, Müller, et al., 2021; Pineda et al., 2021; Turner et al., 2021), and gradient-based HPO has been extended to tackle architectural hyperparameters of deep neural networks (H. Liu, Simonyan, and Y. Yang, 2019). Moreover, evolutionary algorithms have become more efficient (Awad, Mallik, and Hutter, 2021), and researchers tackle the problem of optimizing larger pipelines, which we discuss further in Section 9.2.2.

It is hard to quantify the impact of our survey on the development and usage of more efficient HPO. However, the paper is being cited widely as a standard reference for HPO, and we hope that it helps future researchers to get into HPO.

Second, in Chapter 4, we proposed the first transfer HPO method to fulfill four desirable properties for applying transfer HPO without human intervention. An early version of RGPE, that we have published in an arXiv preprint (Feurer, Letham, and Bakshy, 2018), has been picked up as a baseline for transfer HPO (Pineda et al., 2021; Xiao, Xing, and Neiswanger, 2021), implemented as transfer HPO method in a Bayesian optimization package (Y. Li et al., 2021), and used for tuning databases (X. Zhang et al., 2021). Even as more complex and potentially more powerful transfer learning methods based on deep networks are developed (Volpp et al., 2020; Wei, Zhao, and J. Huang, 2021; Wistuba and Grabocka, 2021), we believe our hyperparameter-free, linear ensemble of standard Gaussian process models has its raison d'être because it is a very transparent method, which allows interpreting and understanding its behavior and potential failure cases.

9.1.2 *Automated Machine Learning*

In part iii, we have researched whether we can further increase the accessibility of machine learning by improving the performance of AutoML systems. More specifically, we have researched whether we can increase the efficiency of CASH solvers.

To this end, we introduced Auto-sklearn in Chapter 5, which extended Auto-WEKA by including meta-learning and post-hoc ensembling. We demonstrated that Auto-sklearn improves over the previous CASH-based AutoML systems Auto-WEKA and hyperopt-sklearn.

Furthermore, we have conducted the most extensive evaluation of AutoML systems to that point in time using 140 datasets. We demonstrated that both of our methodological additions improve performance over the pure CASH setting by themselves and combined. Together with our success in the 1st ChaLearn AutoML competition, and especially the Tweakathon track in which participants were allowed to tackle datasets on their local computers without budget restrictions, this clearly demonstrates that efficient and robust automated machine learning can increase the accessibility of machine learning.

Auto-sklearn has been a popular baseline since its inception (Y. Zhang et al., 2016; de Sa et al., 2017; Mohr, Wever, and Hüllermeier, 2018; B. Chen et al., 2018; Fusi, Sheth, and Elibol, 2018; Drori, Krishnamurthy, Rampin, et al., 2018; Drori, Krishnamurthy, Lourenco, et al., 2019; C. Yang, Akimoto, et al., 2019; Rakotoarison, Schoenauer, and Sebag, 2019; Gijsbers et al., 2019; C. Yang, Fan, et al., 2020; Zöller, Nguyen, and Huber, 2021). While researchers have claimed improved performance by using better CASH optimizers, this is not yet universally agreed upon (Mohr and Wever, 2021). The datasets collection we introduced to build meta-data for Auto-sklearn and run the experiments also presents an early incarnation of a benchmarking suite from OpenML; others, later on, used this to benchmark their AutoML systems (L. Li et al., 2018; Rakotoarison, Schoenauer, and Sebag, 2019). Due to its popularity, the PipelineProfiler (Ono et al., 2021) implements a connection to Auto-sklearn to visualize the found pipelines.

Moreover, others have used the open-source implementation of Auto-sklearn in their applications and we briefly mention a small sample of them. One application domain is medicine, where it has been used to classify EEG recordings (Gemein et al., 2020), predict the 9-year course of mood and anxiety disorders (van Eeden et al., 2021) and predict the short- and long-term treatment response in schizophrenia (Ambrosen et al., 2020). Auto-sklearn has also been used for software analysis (Wattanakriengkrai et al., 2018; Maipradit, B. Lin, et al., 2020; Maipradit, Treude, et al., 2020; Chinthanet et al., 2021), graph analysis and combinatorial optimization (Lauri and Dutta, 2019; Lauri, Dutta, et al., 2020) and opinion quantification (Gurukar et al., 2020). Denkena et al. (2020) applied Auto-sklearn in machining, Huo et al. (2019) in smart-grid monitoring and G. Liu et al. (2021) to climate modeling, receiving a best paper award at the NeurIPS 2021 AI for climate change workshop. All these applications have a common story: Auto-sklearn matches or outperforms the manually designed machine learning pipelines, and most importantly, it does so without human intervention. The fact that researchers who are not affiliated with us successfully implemented their projects with Auto-sklearn demonstrates the potential of capabilities of AutoML systems to broaden access to machine learning and is a strong case to

apply both AutoML and Auto-sklearn for machine learning problems with manually extracted features.

We presented our follow-up Auto-sklearn 2.0 in Chapter 6. We first introduced new building blocks that helped win the 2nd ChaLearn AutoML competition: successive halving for faster evaluation of pipelines and static portfolios for warmstarting Bayesian optimization. In addition, we used a manually designed fallback to cross-validation. Therefore, we generalized the assignment of budget and the way we measure generalization to *budget allocation strategies* and *model selection strategies*. The portfolios are of particular interest as they are a meta-feature-free and simple initialization of Auto-sklearn with performance guarantees. In an experimental study, we found that the best one differs per dataset and time horizon, but our choice for the 2nd ChaLearn AutoML competition was the best overall choice. Most importantly, we were the first to exploit the fact that different model selection strategies, i.e., holdout or cross-validation, behave differently depending on the time budget and dataset size.

However, we realized that this system was very much hand-designed which is against the main idea of AutoML. A practitioner applying Auto-sklearn would have to manually make such decisions as we did for the 2nd ChaLearn AutoML competition. Therefore, we defined the meta-problem of AutoML, which is a generalization of the expected risk minimization paradigm over datasets (instead of over individual data points). We could use meta-learning to tune the AutoML system itself based on this. Concretely, we used algorithm selection to choose which of multiple instantiations of Auto-sklearn to apply for a new dataset.

In contrast to when we introduced Auto-sklearn 1.0, there was now a standardized benchmark: the OpenML AutoML benchmark. It defines the full evaluation setting and includes 39 datasets to evaluate AutoML systems (Gijsbers et al., 2019). We constructed a complementary set of 209 datasets for meta-learning. Using both the benchmark and the additional datasets for meta-learning, we demonstrated improved performance over Auto-sklearn 1.0 and the AutoML systems Auto-WEKA (Thornton et al., 2013; Kotthoff et al., 2019), H2O AutoML (LeDell and Poirier, 2020), and TPOT (Olson, Urbanowicz, et al., 2016; Olson and Moore, 2019). This improved performance further supports our finding that we can boost machine learning accessibility by going beyond the standard CASH formalism.

9.1.3 *Benchmarking and the OpenML Platform*

All improvements to the realm of automated machine learning from Parts ii and iii hinge on the availability of extensive meta-data. We introduced the tools to increase their availability in Part iv by intro-

ducing *OpenML-Python* (Chapter 7) and *OpenML benchmarking suites* (Chapter 8).

OpenML-Python allows for full access to OpenML and contains a scikit-learn integration. We furthermore demonstrated research adoption by giving examples of how other researchers used OpenML-Python. As of today, OpenML-Python is the most successful OpenML API by the number of contributors, open issues, and GitHub stars (OpenML.org, 2022).

While the availability of a large amount of datasets is the first step, these datasets need to be curated to form realistic benchmarks. For this, we have introduced *benchmarking suites* in Chapter 8, which can be used to organize the datasets on OpenML and provide fully-defined benchmarks for further use. Furthermore, we introduced the *OpenML-CC18* benchmarking suite as the first practical example, which should be used to compare standard machine learning algorithms. Because we introduced the first benchmarking suites in 2017 we were able to discuss their adoption in research and show that researchers appreciate having access to pre-selected datasets.[1]

We omit further discussions of our contribution to the OpenML benchmarking platform because both Chapters 7 and 8 themselves already include an extensive and up-to-date discussion of the work.

9.2 FUTURE WORK

The field of Automated Machine Learning (AutoML) has made great strides over the last years. Our work provides important contributions to the field of AutoML. We conclude this thesis by discussing possible extensions to our work and additional future research that we expect to be important over the next few years.

9.2.1 *Future Work To Improve Auto-sklearn*

Despite our continuous effort to improve the AutoML system Auto-sklearn, there are numerous possible extensions and potential open research questions.

To improve performance, Auto-sklearn could use ideas from *green AutoML* (T. Tornede et al., 2021) that predict whether configurations will be able to finish within the given budget to not waste time evaluating configurations unnecessarily (Nguyen, Musial, and Gabrys, 2021; Mohr, Wever, A. Tornede, et al., 2021).

In Chapter 4, we have demonstrated strong performance of our new transfer HPO method RGPE(TAF). Like the portfolios currently used in Auto-sklearn, it comes with theoretical guarantees but can dynamically adapt to the dataset at hand. In addition, it is hyperparameter-free

1 We have introduced the first benchmarking suites in 2017, but only published our work in Chapter 8 in 2021.

and does not require setting the size of the portfolio. Therefore, we suspect that it would be a fruitful addition to Auto-sklearn.

Next, the search space of Auto-sklearn is currently hand-designed. The reader of this thesis should immediately understand that this is sub-optimal and that by creating an automated procedure, we will be able to use better search spaces. We could also add alternative primitives that do not reside within scikit-learn, such as XGBoost (T. Chen and Guestrin, 2016) or LightGBM (Ke et al., 2017), and let an automated method decide which ones to use in the final search space. Similar to the meta-learning ideas we developed in Chapter 6, the search space could be targeted at a specific time horizon, or extending on that, be constructed with a specific application in mind. To realize such automated search space construction, ideas from search space pruning could be used (Wistuba, Schilling, and Schmidt-Thieme, 2015; Perrone and Shen, 2019). However, we expect a combination of prior-guided Bayesian optimization (Hvarfner et al., 2022) and learned priors (Rijn and Hutter, 2018) to be an even better option. Such a method would not prune the search space but instead focus on the most promising parts of the search space in the beginning. In the long run, it would fade out the prior and consider the full search space.

To improve the efficiency of Auto-sklearn 2.0 we also suggest using the budget as a context in the selector to not have to rebuild the AutoML system for each budget setting. Yet, this will be very challenging as it would at least require us to change the way we generate the training data to train the selector to include the performance of Auto-sklearn for multiple budgets.

We designed Auto-sklearn to mimic the API of scikit-learn (Pedregosa, Varoquaux, et al., 2011; Buitinck et al., 2013) and users can add or exclude components that are searched over. However, our system does not allow for changing the overall pipeline structure. Other AutoML systems such as Auto-KERAS (Jin, Song, and Hu, 2019) and LALE (Baudart et al., 2021) provide *expert* interfaces, in which the user can pass in the pipeline structure to optimize. We consider this important future work because it would allow Auto-sklearn to cover more use cases, for example, Gemein et al., 2020 could have optimized the hyperparameters of the feature extraction from EEG readings by adding it as an additional pipeline step in Auto-sklearn.

9.2.2 *What makes a well-performing AutoML system?*

AutoML systems are complex pieces of engineering that consist of many individual parts. So far we only know very little about how to design efficient AutoML systems, and which parts are responsible for the performance under which circumstances.

Machine Learning Primitives Used in AutoML Research

Auto-sklearn is based solely on scikit-learn (Pedregosa, Varoquaux, et al., 2011). Its development started before more powerful machine learning packages such as XGBoost (T. Chen and Guestrin, 2016), LightGBM (Ke et al., 2017), CatBoost (Prokhorenkova et al., 2018) and PyTorch (Paszke et al., 2019) were available and contained implementations of state-of-the-art algorithms at the time. Using these packages can result both in improved predictive performance and lower training times.

Recent AutoML systems, such as FLAML (C. Wang et al., 2021), AutoGluon (Erickson et al., 2020), and LightAutoML (Vakhrushev et al., 2021) make heavy use of these and therefore have access to potentially stronger base models to optimize and combine. Therefore, it is important to not conflate research on AutoML methods with research on AutoML systems. On one hand, we acknowledge that using different sets of primitives is valid when comparing AutoML systems. While it can be interesting which method can be implemented best to maximally boost performance, one must ensure to not just conduct a *bake-off* that provides us with little scientific knowledge (Langley, 1996; Drummond, 2006), and keep in mind that different implementations of the same algorithm can have large runtime deviations (Kriegel, Schubert, and Zimek, 2017). On the other hand, a fair comparison of AutoML methods requires them to use the same set of primitives. This means they would have to use the same set of machine learning algorithms, with the same implementation and the same hyperparameter ranges to choose from. Such an approach is for example proposed by the D3M project (DARPA, 2022; Milutinovic, 2019). To the best of our knowledge, this has not been adopted by the research community in general, and it is therefore an open question how to approach fair comparisons.

Search Space Design

Unfortunately, there is a plethora of different pipeline and search space representations, and their expressiveness and searchability are not immediately clear. To make things worse, papers that introduce a new pipeline representation often intertwine it with the search method and vice versa.

In Chapters 5 and 6 we follow the CASH (Thornton et al., 2013; Kotthoff et al., 2019) paradigm. In the realm of evolutionary algorithms, some researchers describe the search problem using strongly typed genetic programming (Křen, Pilát, and Neruda, 2015; Pilát, Křen, and Neruda, 2016; Křen, Pilát, and Neruda, 2017, GP-ML), tree-structured genetic programming (Olson, Bartley, et al., 2016; Olson and Moore, 2019, TPOT) and grammar-based genetic programming (de Sa et al., 2017, RECIPE) and mutation and crossover rules in a ge-

netic algorithm (Qi et al., 2019, DarwinML). There are also works that employ heuristic search algorithms to find pipelines (Mohr, Wever, and Hüllermeier, 2018; Rakotoarison, Schoenauer, and Sebag, 2019; Katz et al., 2020; Zöller, Nguyen, and Huber, 2021; Marinescu et al., 2021), however, they use different formalisms to describe the search graph. Moreover, the pipeline and search space representation can also follow the methodology and be tailored to a reinforcement learning algorithm. Reinforcement learning has been applied to pipeline edit operations (Drori, Krishnamurthy, Rampin, et al., 2018; Drori, Krishnamurthy, Lourenco, et al., 2019), linear pipelines (X. Sun, J. Lin, and Bischl, 2019; Khurana and Samulowitz, 2020) and pipelines in a grid world (Heffetz et al., 2020).[2]

To overcome the representation issues we propose to use a general pipeline description language, that incorporates the pipeline structure. It must also allow describing operations, their hyperparameters and search spaces, as well as the pre- and post-conditions for applying them. Such a language would allow describing the problem to be solved independently of the search algorithm. We could then better describe AutoML by breaking it down into the *search space*, *search method* and *performance estimation* as it was proposed for Neural Architecture Search (NAS) algorithms (Elsken, Metzen, and Hutter, 2019b). Based on this we could for example, compare the different Monte Carlo Tree Search (MCTS) approaches described in the literature (Rakotoarison, Schoenauer, and Sebag, 2019; Zöller, Nguyen, and Huber, 2021) with respect to different pipeline sizes, operators, and maybe even with respect to the applicability of a method to construct pipelines of unlimited size.

Leveraging Heterogeneous Ensembles in AutoML

Currently, the majority of AutoML systems, including our own Auto-sklearn 1.0 and Auto-sklearn 2.0 (see Chapters 5 and 6), follow the CASH paradigm (Escalante, Montes, and Sucar, 2009; Q. Sun, Pfahringer, and Mayo, 2013; Thornton et al., 2013; Komer, Bergstra, and Eliasmith, 2014; Bürger and Pauli, 2015; Y. Zhang et al., 2016; Swearingen et al., 2017; Alaa and Schaar, 2018; Fusi, Sheth, and Elibol, 2018; C. Yang, Akimoto, et al., 2019; Zimmer, Lindauer, and Hutter, 2021; C. Wang et al., 2021; Vakhrushev et al., 2021). Some of them use additional post-hoc ensembling similar to the post-hoc ensembling we introduced in Chapter 5 (Escalante, Montes, and Sucar, 2009; Bürger

2 We have chosen the above mentioned works because they represent unique contributions in the description of the search space or the search strategy. However, there are several other works, also earlier ones, that would already qualify as AutoML systems but did not focus on the search for pipelines and can therefore for example be subsumed by the CASH formalism (Caruana et al., 2004; Statnikov et al., 2005). We would like to point the interested reader to a great survey on intelligent assistants for data analysis (Serban et al., 2013).

and Pauli, 2015; Alaa and Schaar, 2018; C. Yang, Akimoto, et al., 2019; Zimmer, Lindauer, and Hutter, 2021).

Systems based on genetic algorithms go a different route. They allow classifiers as intermediate pipeline steps, that either allow implicit stacking (Olson, Bartley, et al., 2016; Olson and Moore, 2019; Qi et al., 2019) or explicitly incorporate stacking into their search space (Pilát, Křen, and Neruda, 2016; Křen, Pilát, and Neruda, 2017). Instead of allowing for stacking architectures, Automatic Frankensteining (Wistuba, Schilling, and Schmidt-Thieme, 2017) explicitly builds a stacking model in a 2-stage process. Auto-Stacker goes one step further and directly searches for a two-level stacking ensemble (B. Chen et al., 2018).

All these methods include an HPO component. Contrarily, AutoGluon completely discards HPO, and instead invests all available time budget into robustly stacking models, potentially in multiple layers (Erickson et al., 2020).[3] This resulted in superior performance compared to Auto-sklearn 1.0 (see Chapter 5), TPOT (Olson, Urbanowicz, et al., 2016; Olson and Moore, 2019) and H2O AutoML (LeDell and Poirier, 2020).

These differences raise several obvious questions that need to be examined further:

1. Is stacking in an AutoML system in general preferable to solving a CASH problem? Or can the improved performance of AutoGluon be explained by other factors, for example the other set of primitives used by AutoGluon (see also Section 9.2.2)?

2. Can we find a middle ground that combines efficient HPO and stacking in one single-stage approach (compared to multiple stages used by Wistuba, Schilling, and Schmidt-Thieme (2017))?

3. Do we require different models on the stacking level, or can we employ the base models found by a CASH procedure in a post-hoc phase to realize the gains of stacking?

4. Are the post-hoc ensembles, such as those employed by Auto-sklearn, a good idea, and could other methods that do not use them, for example, TPOT, be improved by using such post-hoc ensembles, too?

As many of the above-mentioned tools are based on CASH solvers, it is a natural question whether we can gain further performance improvements in this area. Recent publications (Y. Zhang et al., 2016; Mohr, Wever, and Hüllermeier, 2018; Rakotoarison, Schoenauer, and Sebag, 2019; S. Liu et al., 2020) suggest that the optimization algorithm used in Auto-sklearn could be improved. We deem further improvements in this area important, too. Such improved CASH solvers can

3 Multi-layer stacking was originally introduced by the StackNet model which was successfully applied to win multiple Kaggle competitions (Michailidis, 2017).

lead to better AutoML systems for short time horizons, or better ways to generate meta-data for AutoML systems such as Auto-sklearn 1.0 and Auto-sklearn 2.0. However, except for MOSAIC (Rakotoarison, Schoenauer, and Sebag, 2019), none of these publications evaluate whether such improvements can also be realized together with meta-learning. On the other side, there were numerous improvements in transfer HPO that could be leveraged here as well, as for example, the RGPE method we introduced in Chapter 4, that could lead to further improvements.

9.2.3 *Extending the Scope of AutoML and Bringing the Human Back into the Loop*

Until now we have interpreted AutoML as an instance of the CASH problem, and have only discussed extensions that optimize the machine learning pipeline itself in Section 9.2.2. Moreover, we have only discussed optimizing the predictive performance of AutoML systems. However, in real-world applications we are often interested in secondary objectives or have to abide constraints, for example the final machine learning pipeline must have a low enough latency that allows the system to be deployed. Therefore, researchers have started to build multi-objective AutoML systems (Pfisterer, Thomas, and Bischl, 2019; Pfisterer, Coors, et al., 2019; S. Liu et al., 2020; D. Wang, Ram, et al., 2020).[4] It will be an interesting question whether such secondary objectives can be taken into account when designing meta-learning approaches, too.

Next, the full machine learning workflow consists of more steps than constructing the machine learning pipeline. Caruana (2015) suggested a sequential workflow of *Problem Definition, Data Collection, Data Cleaning, Data Coding, Metric Selection, Algorithm Selection, Parameter Optimization, Post-Processing, Deployment, Online Evaluation and Debug*. D. Wang, Liao, et al. (2021) created an even more detailed breakdown of steps and describes 10 stages and 43 sub-tasks of the data science and machine learning lifecycle. Many of these steps have the potential to be automated, but we would like to note that there are steps like *Problem Definition* or *Metric Selection* that must be left to a human expert. The ease.ML (Aguilar Melgar et al., 2021) system is an example of a fully integrated lifecycle management system for machine learning, and it integrates a total of eight steps. The system was constructed based on interaction with human practitioners and contains automated data ingestion, cleaning, and a traditional

4 We would like to note that other subfields of AutoML suggested secondary objectives earlier, for example HPO research suggested fairness and model size as a secondary objectives (Gardner et al., 2014; Gelbart, Snoek, and Adams, 2014; Chakraborty et al., 2019) and both neuro-evolution and NAS researchers suggested optimizing for model size (Kim et al., 2017; Hsu et al., 2018; Zhou et al., 2018; Dong et al., 2018; Elsken, Metzen, and Hutter, 2019a; Lu et al., 2019).

AutoML system to construct machine learning pipelines. However, such a front-to-end approach is not suitable for every user and use case (Drozdal et al., 2020; D. Wang, Liao, et al., 2021) and there need to be solutions that automate only individual steps. While the state of HPO and AutoML can be well-observed these days using common benchmarking tools (Turner et al., 2021; Eggensperger, Müller, et al., 2021; Gijsbers et al., 2019), other steps such as data ingestion from CSVs, data cleaning and data coding need to be integrated in the benchmarking process. As a community, we need to agree on the steps we are interested in and then define useful benchmarks to measure progress and analyze proposed solutions. In the long run we also need benchmark problems for the full end-to-end machine learning process in case this is even possible at all.

Besides blindly increasing automation there are also other dimensions to which the scope of AutoML could be extended. Several works argue for various ways to include humans into the loop and 1) give them more control over the AutoML process (Q. Wang et al., 2019; Pfisterer, Thomas, and Bischl, 2019), 2) study whether they trust the AutoML systems in place (Drozdal et al., 2020), 3) provide them with more analysis capabilities of the AutoML systems' output (Q. Wang et al., 2019; Weidele et al., 2020; Ono et al., 2021)[5], or 4) study the accessibility of commercial AutoML services by their functionality (Xanthopoulos et al., 2020).

Similarly, AutoML could be approached from a user perspective and improved w.r.t. the outcome of user studies (Q. Wang et al., 2019; Crisan and Fiore-Gartland, 2021) instead of solely targeting performance benchmarks.

9.2.4 *Relation to Deep Learning and Neural Architecture Search*

In this work, we focus on featurized, i.e., tabular, data. Furthermore, we only used "traditional" machine learning methods such as tree-based methods, support vector machines and multi-layer perceptrons. Given the recent progress in *deep learning* research, this begs two questions:

1. Will such "traditional" methods remain relevant?
2. Will the methodology we developed in this thesis be applicable to *deep learning*-based AutoML systems?

Answering the first question is, of course, very speculative. First works claim that deep learning can be state-of-the-art for tabular

5 There exists a related stream of works that analyze hyperparameter importance which could be used here, too (Jones, Schonlau, and Welch, 1998; Hutter, Hoos, and Leyton-Brown, 2014; Fawcett and Hoos, 2016; Golovin et al., 2017; Biedenkapp, Lindauer, et al., 2017; Biedenkapp, Marben, et al., 2018; Rijn and Hutter, 2018; Probst, Boulesteix, and Bischl, 2019; Moosbauer et al., 2021).

data (Kadra et al., 2021) and mixed tabular and text data (Shi et al., 2021). However, these methods are novel and not yet battle-tested, for example in Kaggle competitions. Still, we expect deep learning methods to catch up to states where they will be used at least as frequently as tree-based methods. Nonetheless, we expect that at least a subset of the "traditional" machine learning methods will remain relevant as they can match the performance of *deep learning*, but bring other benefits, such as better interpretability and better scalability. Furthermore, they have a different inductive biases that enables boosting predictive performance by constructing hybrid ensembles (Mendoza et al., 2019; Zimmer, Lindauer, and Hutter, 2021).

Luckily, the second question can be answered more scientifically. It has already been shown that the techniques of Auto-sklearn 2.0 can be applied to deep learning models on both tabular and image data (Zimmer, Lindauer, and Hutter, 2021). Interestingly, at the moment, there is a gap between CASH and NAS, although both can be tackled with similar methods (Kandasamy et al., 2018; Mendoza et al., 2019; White, Neiswanger, and Savani, 2021). Efficient methods for NAS, such as DARTS (H. Liu, Simonyan, and Y. Yang, 2019), use gradients, and it would be unclear how to integrate general-purpose machine learning algorithms into such a framework. Future work should explore combining the different paradigms of CASH and NAS into powerful, hybrid algorithms.

While the methods that will be useful in the future are unclear yet, we think that one must take into account that we will be able to conduct search and optimization on a larger scale in the future, and that all methods we develop must therefore scale to such enhanced compute capabilities (Sutton, 2019).

9.2.5 *OpenML and Benchmarking*

The future work for OpenML-Python is very clearly laid-out by the roadmap of the OpenML platform to which OpenML-Python provides an interface; it is therefore more interesting to discuss how we can extend on the benchmark suites and improve upon them. In Chapter 8 we have already discussed the issues of overfitting to benchmarking suites, credit assignment for contributions to a benchmarking suite, automating the curation of useful suites, computational issues, breadth of current benchmarking suites, specification of resource constraints and disclosure of ethical issues. We have further identified several new research questions.

Integrating the Latest Datasets

Another issue the author observed is that there is no automated process to upload new datasets to OpenML. As an example, at the NeurIPS 2021 datasets and benchmarks track (Vanschoren, Yeung,

and Xenochristou, 2021), several papers introduced tabular datasets for supervised classification, for example the *folktables datasets* (Ding et al., 2021). They are reproductions of the famous *UCI Adult* dataset. Because the data was gathered with the goal of understanding bias in the original *UCI Adult* dataset, it was converted into one dataset per year (2014-2018) and state (i.e., the 50 states of the United States of America) and contains additional prediction targets to allow more fine-grained analysis. However, at the time of writing, they were not available on OpenML. This could be due to three issues: no incentive to upload the data to OpenML, no knowledge about the OpenML platform, or too much hassle to upload to OpenML. Instead, we propose that future work considers a bot that parses conference submissions for dataset artifacts so that an editor can upload them to OpenML in a semi-automated fashion.

Furthermore, we must also find ways to automatically flag datasets as superseded. The *folktables datasets*, for example, are reproductions of the famous *UCI Adult* dataset and aim to fix several issues and make the dataset more realistic. Ideally, we would also like to learn about such updates in an automated fashion to then retire outdated datasets.

Can We Improve The Curation Protocols For Benchmarking Suites?

To the best of our knowledge, the OpenML-CC18is the first benchmarking suite that was created following strict curation standards. In the future, it would be good to study the field of dataset design (Aroyo and Welty, 2015; Recht et al., 2019) to check whether we can apply their findings on the meta-level. One such example would be a dataset distribution shift, meaning that datasets could be labeled differently at different time steps.

Consider a digit classification dataset for which the dataset creator computed HOG (Dalal and Triggs, 2005), SIFT (Lowe, 1999), or other manually defined features from the raw features. These datasets would have been of interest until the early 2010s; however, current approaches to digit classification solely rely on deep neural networks, which only consume raw data (Clune, 2019; Sutton, 2019), rendering these preprocessed datasets uninteresting. In retrospect, both benchmarking suites we introduced and described in Chapter 8 contain image datasets with manually extracted features that we consider to be no longer relevant. Future dataset collections should at least consider checking whether the concept a certain dataset represents is still of interest.

Can We Support Meta-Learning Research by Providing Training Suites?

For meta-learning research, it is important to not only have access to benchmarking suites that can be used as test datasets, such as the OpenML AutoML benchmark (Gijsbers et al., 2019). To generate meta-

data, we also need access to training datasets, such as the ones we created in Chapters 5 and 6.

Such training and testing suites could be designed following different aspects or different goals in meta-learning research:

- in-distribution meta-learning: the test datasets come from the same distribution or data generator, for example, the *folktables datasets* (Ding et al., 2021) contain a dataset per US state and we could split them into training and test,
- concept drift meta-learning: the training and test datasets come from the same dataset distribution at different time steps, for example the *folktables datasets* (Ding et al., 2021) are given for the years 2014-2018, and one could use 2014-2017 as training datasets and 2018 as the test dataset,
- mixed-distribution meta-learning: the test datasets share some overlap with the distribution of training datasets, which is what we implicitly do by using all data available on OpenML, and
- out-of-distribution meta-learning: e.g., training the AutoML system on manufacturing tasks and then applying them to biology tasks.

So far only HPO-B (Pineda et al., 2021) recognizes the necessity of separating an available benchmark collection into a training and a testing set. However, HPO-B assigns the datasets at random and it is therefore unclear whether it follows any of the aspects mentioned above.

In this last section, we have laid out current research challenges for AutoML systems. We believe that tackling these will allow us to better understand the factors that enable efficient and robust AutoML. With this knowledge, we will be capable of designing AutoML systems that support human experts in all steps of the machine learning and data science workflow and thereby fully democratize machine learning.

Part VI

APPENDIX

A

APPENDIX FOR PRACTICAL TRANSFER LEARNING FOR BAYESIAN OPTIMIZATION

A. Relation between TAF, SFMO and Active Testing

Leite et al. (2012) introduced *active testing* which uses relative landmarking (pairwise ranking of all observations so far) to decide which configuration $\mathbf{x}$, from a finite set of choices, to run next. For each candidate configuration $\mathbf{x}$ that was not yet run on the target dataset, they do a table lookup to determine whether and by how much that algorithm has improved over the current best observed algorithm on each meta-dataset, and weight its improvement by a task similarity measure computed using the pairwise ranking of all observations so far. More specifically, active testing selects the next algorithm according to

$$\arg\max_{\mathbf{x}} \sum_{i=1}^{t-1} RL(\mathbf{x}^i, \mathbf{x}^i_{best}) Sim(t,i) \tag{15}$$

with RL being a *relative landmark* defined as

$$\begin{aligned} & RL(\mathbf{x}^i_k, \mathbf{x}^i_{best}) \\ & = \mathbb{1}(y^i_k < y^i_{best})(y^i_{best} - y^i_k)) \\ & = \max(0, y^i_{best} - y^i_k))) \\ & = I(\mathbf{x}^i_k), \end{aligned} \tag{16}$$

$\mathbf{x}_{best}$ being the best algorithm observed on the target task t and $Sim(t, \cdot)$ is a similarity function defined as the fraction ranking pairs after a Laplace correction. We briefly clarify that $\mathbf{x}^i$ means the configuration $\mathbf{x}$ on dataset i, similarly $\mathbf{x}^i_k$ is the configuration indexed by k on dataset i and y^i_k is the observed performance value.

Replacing the similarity $Sim(\cdot, \cdot)$ in Equation 15 with the similarity divided by the sum of all similarities to the target model, $w_i = \frac{Sim(t,i)}{\sum_{j=1}^{t-1} Sim(t,j)}$, and also replacing the relative landmark in Equation 15 with the improvement from Equation 16 shows us that *active testing* is similar to the *transfer acquisition function* (TAF) with the following differences:

1. Active Testing works only on a finite set of algorithms a, while TAF can be applied to problems with continuous hyperparameters.

2. TAF models function values with a surrogate model, while active testing relies only on previously conducted function evaluations.

3. TAF includes a model and an expected improvement term for the target task as well.

4. The weighting function is slightly different to TAF, as it only takes the proportion of correct pairwise comparisons into account, but performs Laplace smoothing to compute $Sim(\cdot, \cdot)$. Most importantly, active testing does not include a bandwidth factor which determines whether a base model should be dropped.

5. Active Testing models the improvement over the current best known design point, while TAF leaves the choice of $\mathbf{x}_{best}$ to the individual base models.

This finding has two implications: First, we can use the weighting scheme we developed in the main paper in this setting as well, and would benefit from all improvements over

TST-R(TAF) described in the text. Second, we have shown how to extend active testing to work on a continuous design space[10] and incorporate a model for the target tasks.

If we instead drop the similarity term from Equation 15 and replace y^i_{best} by $\min_{k \in (1,2,\ldots,n)} y^i_k$, which means we consider improving over the performance of everything evaluated so far on dataset i, we obtain sequential model-free optimization described in Section 7.

B. Scaling Study

In this section we compare the online training time of the RGPE weighting scheme we proposed in Section 4 of the main paper with two competitor methods:

1. **TST-R** (Wistuba et al., 2016, also see Section 3.1), which for each model in the ensemble requires a prediction of the current observation and the computation of the loss function.

2. **Multi-task Gaussian processes (MTGP)** (Swersky et al., 2013, also see Section 7), which defines a covariance function $K((\mathbf{x}_k, i), (\mathbf{x}_l, j))$ between both the input location and the tasks, and put all observations on all tasks into a single Gaussian process. Task descriptions are then learned jointly with the kernel hyperparameters. The multi-task Gaussian process has a fitting complexity of $O(t^3 n^3)$ and requires fitting and requires expensive hyperparameter tuning (see Appendix 7). We use the implementation from BoTorch (Balandat et al., 2020), as it is, to the best of our knowledge, the only actively maintained implementation of a multi-task Gaussian process for Bayesian optimization.

We give the fitting times on the $\text{AdaBoost}_{g,2D}$ benchmark in Figure 3. Each bar represents the average over five repetitions, and we used $5, 10, \ldots, 50$ tasks for fitting. Furthermore, we used 50 randomly sampled observations per task.

We can observe that TST-R and RGPE are substantially faster than the multi-task Gaussian process, with TST-R always requiring less than a second to fit the whole ensemble. In contrast, RGPE required at most 2 seconds to fit the whole ensemble for 50 tasks. In practice RGPE requires less overhead as the fitting time is strongly influenced by the number of observations for the target task; we used 50 observations for the target task, while in practice we only observe these many observations at the very end of an optimization run. This overhead is a small premium one has to pay to obtain a hyperparameter-free transfer learning algorithm in Bayesian optimization. Lastly, we can observe why using a single Gaussian process to fit all observations from all tasks does not scale to the size of benchmarks we consider here. The MTGP model requires more time to fit five tasks than the RPGE model for the highest number of tasks. Fitting the model takes roughly one hour for 25 tasks. The maximum time it required to fit the model was roughly 21000 seconds, which is almost six hours. Such an overhead rules out the usage of MTGP for the kind of problems we discuss in this paper.

10. In order to make this extension of active testing work on a design space which only contains discrete choices one would simply have to use a different model, such as matrix factorization models (Fusi et al., 2018; Yang et al., 2019, 2020)

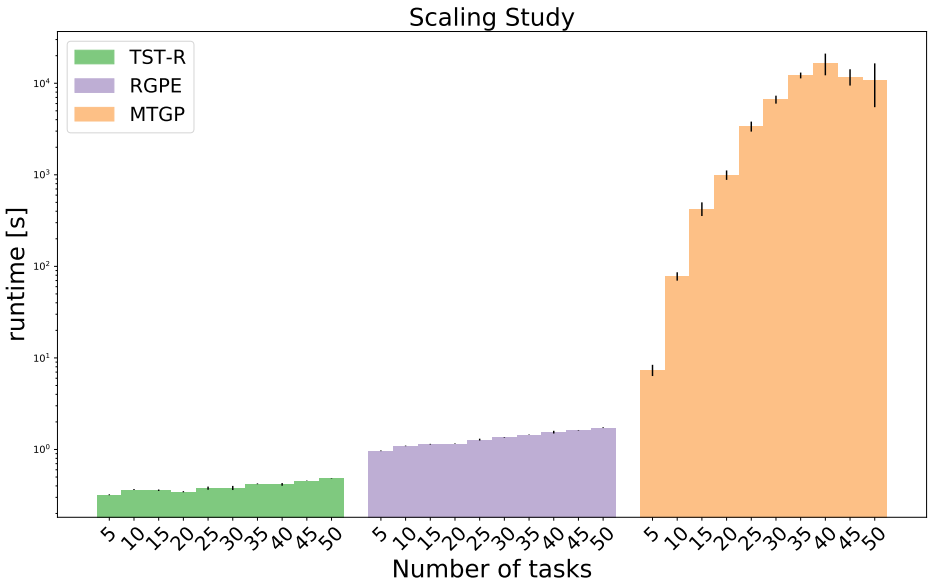


Figure 3: Scaling study comparing the proposed RGPE to TST-R and multi-task GPs.

C. Full Proof of Theorem 4.1

In this section, we show that the Bayesian optimization methods we introduced in the main paper which are built on our sampling-based weighting scheme are only a multiplicative factor slower than regular Bayesian optimization in the worst case. Specifically, we show that the convergence rates given by Bull (2011) still hold and are slowed down by only a multiplicative factor. We will first restate the theoretical statement from the main paper. Second, we will discuss a statement for the specific case of functions with smoothness $\nu < 1$. Third, we will discuss a more general statement for functions with smoothness $\nu < \infty$.

C.1 General Statement

In each iteration, the proposed weighting mechanism has a positive chance of performing vanilla BO, in which only the target model is used. In these iterations, regular BO is conducted, and standard proofs for the convergence of BO therefore apply (Bull, 2011) with a slowdown factor.[11]

Theorem 1 (From the main paper) *Bayesian optimization using a linear combination of Gaussian processes with weights learned according to Section 4.2 of the main paper is at most a factor of*

$$1/\left(\frac{1}{H}\sum_{h=1}^{H}\left(\frac{h}{H}\right)^{t-1}\right)$$

11. We note that this only applies to results which provide guarantees in terms of the simple regret and not the cumulative regret.

slower than Bayesian optimization in the worst case. As before, H is the optimization horizon, while we change the number of observed data points n^t to h, i.e. the current iteration.

Proof sketch: In order for the proposed method to fall back to vanilla BO, we need the weights of all models except that of the base model to be zero. Given the definition of p_{drop} in Equation 9, and setting n_t to h in $p_{drop}(i,h)$, we can calculate the probability of dropping all base models at step h as $\prod_{i=0}^{t-1} p_{drop}(i,h)$, and so the expected proportion of iterations that proceed as vanilla BO is

$$\frac{1}{H}\sum_{h=1}^{H}\prod_{i=1}^{t-1} p_{drop}(i,h) \tag{17}$$

$$= \frac{1}{H}\sum_{h=1}^{H}\prod_{i=1}^{t-1}\left(1-\left(\left(1-\frac{h}{H}\right)\frac{\sum_{s=1}^{S}\mathbb{1}(l_{i,s}<l_{t,s})}{S}\right)\right) \tag{18}$$

$$\geq \frac{1}{H}\sum_{h=1}^{H}\left(1-\left(\left(1-\frac{h}{H}\right)\frac{S}{S}\right)\right)^{t-1} \tag{19}$$

$$= \frac{1}{H}\sum_{h=1}^{H}\left(\frac{h}{H}\right)^{t-1} \tag{20}$$

$$> 0 \tag{21}$$

The observations gathered in iterations when not all base models are dropped do not impose any issues on the convergence proof by Bull (2011). We will show this in the remainder of the section for fixed hyperparameters of a Gaussian process.

C.2 Functions with Smoothness $\nu < 1$

We briefly restate Theorem 2 from Bull (2011), which gives near-optimal convergence rates for GP-based Bayesian optimization with prior smoothness $\nu \leq 1$:

Theorem 2 (From Bull (2011)), *Let π be a prior with length-scale $\theta \in \mathcal{R}_+^d$. For any $R > 0$,*

$$L_n(EI(\pi), \mathcal{H}_\theta(X), R) = \begin{cases} O(n^{-\nu/d}(\log n)^\alpha), & \nu \leq 1, \\ O(n^{-1/d}), & \nu > 1. \end{cases} \tag{22}$$

The smoothness ν can be seen as a kernel (prior) hyperparameter to the Matérn kernel, and the kernel (prior) approaches the squared exponential covariance function for $\nu \to \infty$ (Rasmussen and Williams, 2006). L_n is the loss suffered over the ball B_R in the reproducing Hilbert space $\mathcal{H}_\theta(X)$ after n steps when applying an expected improvement strategy with given prior (kernel with hyperparameter values set to θ):

$$\begin{aligned} &L_n(EI(\pi), \mathcal{H}_\theta(X), R) \\ &= \sup_{||f||_{H_\theta(X)} \leq R} \mathbb{E}_f^{EI(\pi)}[f(x_n^*) - \min f]. \end{aligned}$$

Practical Transfer Learning for Bayesian Optimization

In contrast to the main paper, we follow the notation from Bull (2011) and use f to denote the function we aim to minimize and x to denote a single observation (in contrast to $\mathbf{x}$ in the main paper). We will describe symbols which are different from the notation in the main paper when introducing them. α is either zero for $\nu \notin \mathbb{N}$ or 0.5 otherwise, and R is an upper bound on the function norm, i.e. the maximum absolute value of the function f. X is a subset of $\mathbb{R}^d$, and x_n^* is the estimated optimizer of f after n steps.

Proof of Bull's Theorem 2 still valid The proof for Theorem 2 bounds the simple regret at some time step n via several inequalities depending on Lemma 8 (which bounds the improvement over the current best observation based on a fitted GP), but the regret bound itself and its proof do not have a dependence on the Gaussian process conditioned on the observed data. Furthermore, the proof shows that there is a time step n_k, where the expected improvement is low and therefore BO must be close to the minimum. The existence of such a time step depends on Lemma 7, which states that there are at most k time steps in which the posterior variance of the Gaussian process exceeds a certain value (which converges linearly to zero as we observe more data points). Together with an argument that BO can only improve k times in total, Bull concludes that there must be a time step $k \leq k_n < 3k$ where the posterior variance at a future observation is small and some improvement occurs. We will therefore continue to discuss Lemma 7 and Lemma 8. We first start by restating Lemma 8:

Lemma 8 (From Bull (2011)) *Let* $||f||_{\mathcal{H}_\theta(X)} \leq R$. *For* $x \in X, n \in \mathbb{N}$, *set* $I = \max(0, f(x_n^*) - f(x))$, *and* $s = s_n(x, \theta)$. *Then for*

$$\tau(x) := x\Phi(x) + \phi(x),$$

we have

$$\max\left(I - Rs, \frac{\tau(-R/\sigma)}{\tau(R/\sigma)} I\right) \leq EI_n(x; \pi) \leq I + (R + \sigma)s,$$

where s_n is the predictive standard deviation of the Gaussian process at iteration n without the global scale of variation (Bull, 2011, Equation 2), which is given by σ in the above Lemma.

This bounds the expected improvement at a given point $x \in X$ for the case of the Gaussian process model predicting a variance larger than zero for x. The actual predictive distribution of the underlying Gaussian process changes with the observed data, but the Lemma is stated with respect to a Gaussian process conditioned on data, and therefore, Bull's proof holds for any sequence of previous observations $x_1, \ldots, x_n$.

In contrast, Lemma 7 has a dependence on the sequence of data points observed.

Lemma 7 (From Bull (2011)) *Set*

$$\beta := \begin{cases} \alpha, & \nu \leq 1, \\ 0, & \nu > 1. \end{cases}$$

Given $\theta \in \mathbb{R}_+^d$, *there is a constant* $C' > 0$ *depending only on* X, K *and* θ *which satisfies the following. For any* $k \in \mathbb{N}$, *and sequences* $x_n \in X, \theta_n \geq \theta$, *the inequality*

$$s_n(x_{n+1}; \theta_n) \geq C' k^{-(\nu \wedge 1)/d} (\log k)^\beta$$

holds for at most k *distinct* n.

The proof of Lemma 7 consists of two parts. The first part shows that the posterior variance is bounded by the distance to the nearest point observed so far. As this describes the general behavior of a Gaussian process, it is a bound on the posterior variance given the observed data. The second part shows that most design points x_{n+1} are close to a previous x_i, and that therefore the posterior variance of the Gaussian process model is effectively bounded. As we alter the strategy with which we choose the "previous" x_i, we need to ensure that this property still holds for points not chosen by the standard EI strategy. We fully restate this part of the proof:

We next show that most future evaluations x_{n+1} are close to a previous x_i. X is bounded, so can be covered by k balls of radius $O(k^{-1/d})$. If x_{n+1} lies in a ball containing some earlier point x_i, $i \leq n$, then we may conclude

$$s_n^2(x_{n+1}; \theta_n) \leq C'^2 k^{-2(\nu \wedge 1)/d} (\log k)^{2\beta},$$

for a constant $C' > 0$ depending only on X, K and θ. Hence as there are k balls, at most k points x_{n+1} can satisfy

$$s_n(x_{n+1}; \theta_n) \leq C' k^{-(\nu \wedge 1)/d} (\log k)^{\beta}.$$

Having stated this part of the original proof, we can see that the location of previously queried points is used to bound the predictive variance of the Gaussian process. However, whether a previously queried point in one of the k balls was found by the initial design, EI with only the target GP, or BO with meta-learning does not matter for the posterior variance of the Gaussian process. We therefore conclude that BO methods based on the weighting mechanism introduced in the main paper take at most a multiplicative factor given by Theorem 4.1 longer to reach the time where most future evaluations x_{n+1} are close to a previous x_i compared to running only vanilla BO. This slowdown is exactly what we stated in Theorem 4.1.

As both Lemma 7 and Lemma 8 still hold, we conclude that Theorem 2 does not suffer from additional points observed between the iterations performing Bayesian optimization. As methods using the sampling-based weighting mechanism perform vanilla Bayesian optimization with probability $\frac{1}{H} \sum_{h=1}^{H} \left(\frac{h}{H}\right)^{t-1}$, these methods are at most a factor of

$$\frac{1}{\left(\frac{1}{H} \sum_{h=1}^{H} \left(\frac{h}{H}\right)^{t-1}\right)}$$

slower than vanilla Bayesian optimization. ■

C.3 Functions with Smoothness $\nu < \infty$

We briefly restate Theorem 5 from Bull (2011), which gives near-optimal convergence rates for GP-based Bayesian optimization in the more general case of prior smoothness $\nu < \infty$.

Theorem 5 (Adapted from Bull (2011) to only describe the case of a known prior π.), *If $\nu < \infty$, then for any $R > 0$,*

$$L_n(EI(\pi, \epsilon), \mathcal{H}_{\theta^U}(X), R) = O((n/\log n)^{-\nu/d} (\log n)^{\alpha}),$$

while if $\nu = \infty$, the statement holds for all $\nu < \infty$.

Proof of Bull's Theorem 5 still valid Bull's proof of Theorem 5 works slightly differently than the proof of Theorem 2. Instead of relying on previously sampled points to bound the variance of the Gaussian process (and thereby also bound expected improvement and actual improvement), this proof relies on the observations being quasi-uniform in the search space. For this, the presented technique relies on falling back to a random sampling strategy with probability $0 < \epsilon < 1$.

The proof of Theorem 5 depends on Lemma 12, the proof for which preludes the proof of Theorem 5. It basically gives a probabilistic bound on the mesh norm of points chosen at random. More specifically, it gives the probability for $\sup_{x \in X} \min_{i=1}^{n} ||x - x_i||$ exceeding a certain value, which converges to zero with rate $\log n/n$. Conversely, the probability of having a low mesh norm increases with the number of randomly drawn samples, and having a small mesh norm is important for the remainder of Bull's proof. The proof of Lemma 12 does not take into account points selected by Bayesian optimization, but only points selected by random sampling, and therefore will not depend on points suggested by a meta-learning strategy.

Interestingly, the proof of Theorem 5 also does not depend on the points selected by Bayesian optimization, but rather states that posterior variances are small due to the quasi-uniform coverage of the search space, and therefore there exist times n_k where the observed function values are close to $\min f$. The resulting bound is

$$\mathbb{E}_f^{EI(\pi,\epsilon)}[z^*_{2n+1} - z^*] \leq (2C'R + C''(2R + \sigma))r_n.$$

As stated in Equation 2.1 and Sections 2.0 and 2.1 by Bull (2011), the left-hand side is the average-case performance at some future time $2n + 1$, given by the expected loss under an expected improvement strategy with known hyperparameters of the Gaussian process. The quantities on the right-hand side are:

- r_n: a constant depending on the current iteration n: $r_n = (n/\log n)^{-\nu/d}(\log n)^{\alpha}$.
- R: an upper bound on the function norm: $||f||_{\mathcal{H}_\theta(X)} \leq R$.
- C': a constant > 0, which multiplied with r_n gives the upper bound on the probability of having chosen enough points at random, one point by expected improvement, and the mesh norm being small enough. Bull (2011) states that this constant does not depend on f. As only a fraction of points are gathered by a combination of vanilla BO with every $\frac{1}{\epsilon}$th step sampled at random, it can take up to a factor given in Theorem 4.1 longer to observe the necessary points.
- C'': a constant > 0, which multiplied with r_n gives the upper bound on the predictive variance $s_n(x)$ of the Gaussian process for a given time step n, which in turn is bounded by the mesh norm. Bull (2011) states that this constant depends only on X, K and C (a constant introduced in the proof of Lemma 12, which explains the random sampling behavior), therefore, it follows that it does not depend on f.

We conclude that Theorem 5 does not suffer from additional points observed in iterations where meta-learning is used and that the presented methods are at most a factor of

$$\frac{1}{\left(\frac{1}{H}\sum_{h=1}^{H}\left(\frac{h}{H}\right)^{t-1}\right)}$$

slower than vanilla Bayesian optimization. ■

D. Experiments

In this section we give further details on the software used for this paper and details on the experiments to help reproducibility and replicability of our results. Furthermore, we give additional results on what scaling to use within the presented ensemble models, provide a sensitivity analysis of the hyperparameter of both the TST-R(EI) and TST-R(TAF) method and finish with a comparison to our earlier work published in arXiv preprints.

D.1 Search Spaces

The search spaces for the surrogate benchmarks can be found in Tables 9, 10 and 11 and the search space for the grid benchmarks can be found in Tables 12, 13 and 14.

Name	Range	Log	Cond
α	$[0, 1]$	No	No
λ	$[2^{-10}, 2^{10}]$	Yes	No

Table 9: Search space for the GLMNET$_{s,2D}$ surrogate benchmark.

Name	Range	Log	Cond
Kernel	{Linear, Polynomial, Radial}	No	No
C	$[2^{-10}, 2^{10}]$	Yes	Yes
γ	$[2^{-10}, 2^{10}]$	Yes	Yes
Degree	$[2, 5]$	Yes	Yes

Table 10: Search space for the SVM$_{s,4D}$ surrogate benchmark.

Name	Range	Log	Cond
#Rounds	$[1, 5000]$	No	No
η	$[2^{-10}, 2^{0}]$	Yes	No
Subsample	$[0, 1]$	No	No
Booster	{GBLinear, GBTree}	No	No
Max depth	$\{1, 15\}$	No	Yes
Min child weight	$[2^{0}, 2^{7}]$	Yes	Yes
Colsample by tree	$[0, 1]$	No	Yes
Colsample by level	$[0, 1]$	No	Yes
λ	$[2^{-10}, 2^{10}]$	Yes	No
α	$[2^{-10}, 2^{10}]$	Yes	No

Table 11: Search space for the XGBoost$_{s,10D}$ surrogate benchmark.

Name	Range	Log	Cond
I	$[2, 10000]$	Yes	No
M	$[2, 30]$	Yes	No

Table 12: Search space for the AdaBoost$_{g,2D}$ grid benchmark.

Name	Range	Log	Cond
Use linear kernel	{No, Yes}	No	No
Use polynomial kernel	{No, Yes}	No	No
Use RBF kernel	{No, Yes}	No	No
C	$[2^{-5}, 2^{6}]$	No	No
d	$[2, 10]$	No	No
γ	$[0.0001, 1000]$	Yes	No

Table 13: Search space for the SVM$_{g,6D}$ grid benchmark.

Name	Range	Log	Cond
Batch size	$[16, 256]$	Yes	No
Learning rate	$[1e^{-4}, 1e^{-1}]$	Yes	No
Momentum	$[0.1, 0.99]$	No	No
Weight decay	$[1e^{-5}, 1e^{5}]$	No	No
Number of layers	$[0.1, 0.99]$	No	No
Maximum number of units per layer	$[16, 1024]$	Yes	No
Dropout	$[0.0, 1.0]$	No	No

Table 14: Search space for the Neural Network grid benchmark.

D.2 Further Implementation Details

We now further discuss how we verified that our reimplementations of TST-R and TAF are correct. The results reported in Figure 9 of Wistuba et al. (2018) were obtained by measuring the performance of Bayesian optimization with respect to multiple random seeds while keeping the random sample of configurations for training the base models the same across repetitions. This experimental design measures the performance with respect to one way the available meta-data is sampled. In contrast, we aim to measure the robustness of our method with respect to different ways the meta-data is obtained (i.e. every repetition having a different sample of configurations to train the base models on). To ensure that our reimplementation is correct we strived to compare it to the original Java implementation in the same experimental setting. To allow a fair comparison we modified the implementation of TST-R Wistuba (2016) to sample the configurations for each repetition based on the random seed. Then, we ran both our reimplementation and the original implementation with 25 different random seeds on the SVM$_{g,6D}$ benchmark and observed that both methods perform highly similar and conclude that our reimplementation is correct.

D.3 Standardization, Copula Transform or Raw Values?

As the results for different tasks can live on different scales, the different methods need to take this into account. We discussed in Section 4.2 that we standardize the observations $\{y_k^i\}_{k=1}^{n_i}$ separately per task $i \in (1, \ldots, t)$ so that for each task the mean of the observations is zero and the variance is one. However, it has so far not been studied whether this is necessary or one can also use the raw values. Moreover, we are interested in whether the Copula transformation that was recently proposed as an alternative scaling for meta-learning in hyperparameter optimization is a generally applicable option (Salinas et al., 2020).

We apply the three different transformations to the proposed methods as well as to baseline methods: Our proposed *RGPE(NEI)* and *RGPE(mean)* as well as the accompanying *TST-R* baseline (including HPO); our proposed *RGPE(MoGP)* and *RGPE(TAF)* as well as the *TST-R(TAF)* baseline (including HPO); and also the ABLR (Perrone et al., 2018) and SMFO (Wistuba et al., 2015a; Pfisterer et al., 2021) (see Section 7 for more information on these). We give these results in Table 15.

Methods that combine on a model level (*RGPE(NEI)*, *RGPE(mean)*, *TST-R*, *ABLR* and *SMFO*) use the zero mean unit variance scaling as their default. In contrast, the methods that combine the individual models on the acquisition function level do not scale by default. Looking at the former, using no scaling on average results in worse performance than using the default scaling. However, we find that using the Copula transform can lead to further improvements, and except for two methods on the $\text{AdaBoost}_{g,2D}$ benchmark, always performs best or not significantly worse than the best scaling for the respective method. Looking at the latter, not scaling the data at all appears to be a good default. The Copula appears to perform en par for the two TAF versions, and appears to be slightly better for RGPE(MoGP). Finally, the zero mean unit variance scaling performs rather well, too, but the other two methods have a slight edge.

Interestingly, the Copula transform appears to work very well on average, but leads to substantially worse results on the $\text{SVM}_{s,4D}$ benchmark and can lead to slightly worse results for the TAF methods on the $\text{NN}_{t,7D}$ benchmark. We checked what caused these performance drops and found that for $\text{SVM}_{s,4D}$ this was caused by a single dataset. Consequently, we checked that the target model was picked as the sole model for sufficient amount of time, and found that this is the case. Therefore, we attribute this failure to our underlying GP which was not able to escape this local minimum (most likely due to the conditional nature of the hyperparameter space). For the $\text{NN}_{t,7D}$ benchmark we cannot fully attribute this failure to a single dataset.

D.4 Sensitivity Analysis for TST-R(EI) and TST-R(TAF) Methods

In this subsection we study certain aspects of the baseline methods TST-R(EI) and TST-R(TAF). Specifically, we are interested in effect of weight dilution prevention on both methods and in how to set the hyperparameter ρ.

We compare both methods with and without weight dilution prevention in Table 16.

Lastly, we study the effect of tuning the hyperparameter ρ for TST-R(EI) and TST-R(TAF) and show results in Table 17. In total, we compare three different strategies to set ρ:

Practical Transfer Learning for Bayesian Optimization

	$\text{GLMNET}_{s,2D}$	$\text{SVM}_{s,4D}$	$\text{NN}_{t,7D}$	$\text{XGBoost}_{s,10D}$	$\text{AdaBoost}_{g,2D}$	$\text{SVM}_{g,6D}$
RGPE(NEI,mean/var)*	**2.20**	**0.37**	2.71	0.64	0.62	**0.39**
RGPE(NEI,Copula)	**0.13**	**1.41**	**2.13**	**0.60**	**0.57**	**0.32**
RGPE(NEI,unscaled)	**2.70**	**0.34**	3.05	0.94	**0.49**	**0.34**
RGPE(mean,mean/var)*	0.23	**0.26**	2.55	0.65	**0.55**	**0.47**
RGPE(mean,Copula)	**0.20**	**1.55**	**2.22**	**0.62**	0.66	**0.42**
RGPE(mean,unscaled)	**0.15**	0.62	2.94	0.64	**0.44**	**0.43**
TST-R(EI,HPO,mean/var)*	**0.21**	**0.27**	3.19	**0.61**	**0.57**	**0.55**
TST-R(EI,HPO,Copula)	**0.29**	2.17	**2.69**	**0.57**	**0.49**	**0.53**
TST-R(EI,HPO,unscaled)	**0.18**	**0.60**	4.73	0.64	**0.64**	**0.59**
TST-R(TAF,HPO,mean/var)	**0.10**	**0.58**	**1.67**	0.66	0.68	**0.49**
TST-R(TAF,HPO,Copula)	**0.08**	**1.13**	**1.85**	**0.53**	**0.60**	**0.47**
TST-R(TAF,HPO,unscaled)*	**0.17**	**0.50**	**1.71**	0.62	**0.57**	0.50
RGPE(TAF,mean/var)	**0.14**	**1.30**	2.06	**0.57**	**0.65**	**0.35**
RGPE(TAF,Copula)	**0.14**	**1.30**	2.06	**0.57**	**0.65**	**0.35**
RGPE(TAF,unscaled)*	**0.16**	**0.38**	**1.55**	0.67	**0.63**	**0.45**
RGPE(MoGP,mean/var)	**0.17**	**0.38**	**1.99**	0.71	**0.73**	**0.53**
RGPE(MoGP,Copula)	**0.16**	**1.28**	**1.89**	**0.56**	**0.76**	**0.37**
RGPE(MoGP,unscaled)*	**0.16**	**0.30**	**1.93**	0.80	**0.76**	**0.58**
ABLR(mean/var)*	2.95	0.86	**5.03**	1.58	**0.52**	**1.86**
ABLR(Copula)	**0.19**	**0.61**	**5.18**	**1.19**	0.72	**1.71**
SMFO(mean/var)*	**0.25**	**1.74**	**1.75**	**0.87**	**1.11**	**1.21**
SMFO(Copula)	**0.30**	**1.67**	**1.85**	**0.90**	**0.71**	**1.08**

Table 15: Results comparing standardization (Yogatama and Mann, 2014, mean/var), the Copula transform (Salinas et al., 2020, Copula) and no transformation (unscaled) for RGPE(NEI), RGPE(mean), TST-R(EI), TST-R(TAF), RGPE(TAF), RGPE(MoGP), ABLR and SMFO. We mark the method's default by an asterisk. The numbers reported are the average normalized regret (Wistuba et al., 2018). We boldface the best value per benchmark and underline methods that are not significantly different according to a Wilcoxon signed-rank test with $\alpha = 0.05$ (Demšar, 2006).

	$\text{GLMNET}_{s,2D}$	$\text{SVM}_{s,4D}$	$\text{NN}_{t,7D}$	$\text{XGBoost}_{s,10D}$	$\text{AdaBoost}_{g,2D}$	$\text{SVM}_{g,6D}$
TST-R(EI)	**0.20**	0.49	**3.00**	**0.78**	**0.52**	**0.64**
TST-R(EI,wd)	**0.14**	**0.43**	**3.01**	**0.72**	0.59	**0.76**
TST-R(TAF)	**0.17**	**0.48**	**2.95**	**0.64**	**0.73**	**0.46**
TST-R(TAF,wd)	**0.17**	**0.60**	**2.84**	**0.67**	**0.89**	**0.40**

Table 16: Ablation study on the effect of weight dilution for TST-R(EI) and TST-R(TAF). The numbers reported are the average normalized regret (Wistuba et al., 2018). We boldface the best value per benchmark and number of function evaluations and underline methods that are not significantly different according to a Wilcoxon signed-rank test with $\alpha = 0.05$ (Demšar, 2006); separately for TST-R(EI) and TST-R(TAF)

TST-R(EI): The hyperparameter value that has the best average result on the remaining five benchmarks.

TST-R(EI,HPO): Leave-one-task-out hyperparameter tuning. For this, we conducted a grid search on the all but $t-1$ tasks of a benchmark and applied the best found hyperparameter setting to the target task.

TST-R(EI,benchmark name): The hyperparameter value that is best on all tasks of the given benchmark. This should be overly optimistic when applied to the same benchmark.

For TST-R(EI) we observe that tuning the hyperparameters using the leave-one-task-out procedure directly on the benchmark does not necessarily give an advantage over using a fixed setting. Tuning directly on the remaining tasks of the benchmark improves in only three out of six benchmarks ($\text{SVM}_{s,4D}$, $\text{XGBoost}_{s,10D}$ and $\text{SVM}_{g,6D}$) while it results in roughly the same performance for another two ($\text{GLMNET}_{s,2D}$ and $\text{AdaBoost}_{g,2D}$) and a performance degradation for the $\text{NN}_{t,7D}$ benchmark. Interestingly, it never arrives at a value as good as choosing the hyperparameter on all t tasks of a benchmark, and in two cases it is significantly worse. When looking at the performance of transferring the hyperparameter setting from a single benchmark, we find quite different results for the different benchmarks. For $\text{GLMNET}_{s,2D}$ the transfer works from all other benchmarks, while for $\text{SVM}_{s,4D}$ only the transfer from $\text{SVM}_{g,6D}$ gives significantly similar results as the best setting. For the $\text{NN}_{t,7D}$ benchmark three out of five settings give good results, while two give results that are significantly worse than the best.

Conversely, we can observe a strong influence of hyperparameter optimization on TST-R(TAF). Tuning hyperparameters using the leave-one-task-out procedure directly on the benchmark improves two out of six benchmarks and ties on the remaining six, and tuning hyperparameters directly on the benchmark is not significantly worse picking the hyperparameter value on the remaining tasks. The most drastic change is for the neural network

	$\text{GLMNET}_{s,2D}$	$\text{SVM}_{s,4D}$	$\text{NN}_{t,7D}$	$\text{XGBoost}_{s,10D}$	$\text{AdaBoost}_{g,2D}$	$\text{SVM}_{g,6D}$
TST-R(EI)	**0.20**	0.49	**3.00**	0.78	**0.52**	**0.64**
TST-R(EI,HPO)	**0.21**	**0.27**	3.24	**0.61**	**0.58**	0.55
TST-R(EI,HPO,$\text{GLMNET}_{s,2D}$)	**0.14**	0.49	**2.97**	**0.77**	**0.46**	**0.58**
TST-R(EI,HPO,$\text{SVM}_{s,4D}$)	**0.20**	**0.27**	**3.00**	0.78	**0.52**	**0.64**
TST-R(EI,HPO,$\text{XGBoost}_{s,10D}$)	**0.15**	0.51	3.21	**0.61**	**0.51**	0.63
TST-R(EI,HPO,$\text{AdaBoost}_{g,2D}$)	**0.14**	0.49	**2.97**	**0.77**	**0.46**	**0.58**
TST-R(EI,HPO,$\text{SVM}_{g,6D}$)	**0.17**	**1.06**	3.57	0.81	0.61	**0.51**
TST-R(EI,HPO,$\text{NN}_{t,7D}$)	**0.14**	0.49	**2.97**	**0.77**	**0.46**	**0.58**
TST-R(TAF)	0.17	0.48	2.95	**0.64**	**0.73**	**0.46**
TST-R(TAF,HPO)	0.17	**0.50**	1.72	**0.62**	**0.56**	**0.49**
TST-R(TAF,HPO,$\text{GLMNET}_{s,2D}$)	**0.12**	0.50	**1.73**	0.68	**0.71**	**0.44**
TST-R(TAF,HPO,$\text{SVM}_{s,4D}$)	**0.16**	**0.37**	3.08	0.87	**0.51**	**0.53**
TST-R(TAF,HPO,$\text{XGBoost}_{s,10D}$)	**0.18**	0.53	**1.63**	**0.62**	**0.78**	**0.40**
TST-R(TAF,HPO,$\text{AdaBoost}_{g,2D}$)	**0.16**	**0.45**	2.84	**0.67**	**0.44**	**0.50**
TST-R(TAF,HPO,$\text{SVM}_{g,6D}$)	**0.18**	**0.43**	**1.63**	**0.67**	**0.89**	**0.34**
TST-R(TAF,HPO,$\text{NN}_{t,7D}$)	0.16	0.60	**1.58**	**0.64**	**0.75**	**0.40**

Table 17: Ablation study for how to set the hyperparameter ρ of TST-R(EI) and TST-R(TAF). The numbers reported are the average normalized regret (Wistuba et al., 2018). We boldface the best value per benchmark and number of function evaluations and underline methods that are not significantly different according to a Wilcoxon signed-rank test with $\alpha = 0.05$ (Demšar, 2006); separately for TST-R(EI) and TST-R(TAF)

benchmark, where TST-R(TAF) improves substantially and becomes very competitive with the proposed RGPE-based methods (see Tables 4, 5 and 6). Furthermore, we find that for TST-R(TAF) all methods of choosing ρ work for the grid-based benchmarks $\text{AdaBoost}_{g,2D}$ and $\text{SVM}_{g,6D}$, while success is not guaranteed for the remaining four benchmarks.

Based on these results, we find that TST-R(TAF) can be a competitive method, but is very sensitive to the setting of its hyperparameters ρ. If tuned correctly it drastically outperforms its competitor TST-R(EI) in terms of raw performance.

We would like to highlight that none of these hyperparameter optimization would be practically feasible, as they require tremendous amount of computation. For the hyperparameter value that was best on all other benchmarks we required access to five auxiliary benchmarks and needed to do meta-level hyperparameter optimization on them. For the leave-one-task-out hyperparameter tuning we would only require access to the remaining $t-1$ tasks and run Bayesian optimization with all meta-level hyperparameters on it. The hyperparameter that is tuned based on a single benchmark has similar costs in that it requires to solve t tasks for each hyperparameter value. While the overhead of the BO method is negligible, real function calls to the algorithm to be optimized make such procedure

extremely expensive. We were only able to conduct this hyperparameter optimization of the hyperparameter optimization algorithm because the calls to the target algorithms were either pre-recorded or simulated by a surrogate benchmark. In some settings it might also not be possible to access the base tasks any more, for example due to data retention rules.

D.5 Comparison to Earlier Versions

For the sake of completeness, in this section we compare against previous implementations of the methods in this paper that were laid out in earlier, publicly available preprints. Concretely, we compare the RGPE version from this paper (RGPE(NEI)) against the closed-form noisy EI (Feurer et al., 2021, RGPE(CFNEI)) and noisy EI, using posterior samples from the Gaussian processes to compute the weightings and a 95-quantile weight-dilution strategy (Feurer et al., 2018, RGPE(NEI,95)), and give results in Table 18.[12] We can observe that the latest implementation of our method is either the best or not significantly worse than the best on all six benchmark problems. The older competitors are not significantly worse, either, except on the neural network benchmark. Nevertheless, since the RGPE(NEI) from this paper is substantially worse than RGPE(TAF) we also introduced in this work, we conclude that we have made significant improvements over these earlier versions.

	$\text{GLMNET}_{s,2D}$	$\text{SVM}_{s,4D}$	$\text{NN}_{t,7D}$	$\text{XGBoost}_{s,10D}$	$\text{AdaBoost}_{g,2D}$	$\text{SVM}_{g,6D}$
RGPE(CFNEI)	**<u>0.16</u>**	**0.54**	2.94	**0.68**	**0.67**	**0.44**
RGPE(NEI,95)	**0.21**	**0.45**	3.28	**0.78**	**0.90**	**0.41**
RGPE(NEI)	**2.20**	**<u>0.37</u>**	**<u>2.71</u>**	**<u>0.64</u>**	**<u>0.62</u>**	**<u>0.39</u>**

Table 18: Comparison of the method proposed in this paper (RGPE(NEI)) to previous versions proposed in earlier preprints (Feurer et al., 2018, 2021). The numbers reported are the average normalized regret (Wistuba et al., 2018). We boldface the best value per benchmark and number of function evaluations and underline methods that are not significantly different according to a Wilcoxon signed-rank test with $\alpha = 0.05$ (Demšar, 2006).

12. We would like to highlight that Equation 2 of Feurer et al. (2018) was used with a noise-free Gaussian process. In the more general case of also tuning the noise hyperparameter of the Gaussian process that we consider in this work we employ Equation 4 from this paper.

B

APPENDIX FOR EFFICIENT AND ROBUST AUTOMATED MACHINE LEARNING

Supplementary Material for Efficient and Robust Automated Machine Learning

Matthias Feurer **Aaron Klein** **Katharina Eggensperger**
Jost Tobias Springenberg **Manuel Blum** **Frank Hutter**
Department of Computer Science
University of Freiburg, Germany
{feurerm,kleinaa,eggenspk,springj,mblum,fh}@cs.uni-freiburg.de

A Description of the classification algorithms and preprocessing methods

In this section we give a more detailed explanation of the classification and preprocessing methods that we used in AUTO-SKLEARN.

A.1 Classification algorithms

Our AUTO-SKLEARN framework contains 15 base classifiers from scikit learn (out of which exactly one is chosen at each point during the optimization process). The 15 algorithms can generally be separated into 7 categories: generalized linear models (2 algorithms), support vector machines (2), discriminant analysis (2), nearest neighbors (1), naïve Bayes (3), decision trees (1) and ensemble methods (4). A complete list of the algorithms is given in Table 1a in the main paper. While an in-depth description of each algorithm is out of the scope of this paper we want to give a brief description of each category and highlight complementary strengths of algorithms within one category.

Generalized linear models. The first class of algorithms we consider are generalized linear models (GLM) for classification. These are linear classification algorithms. Since we are interested in scaling our AutoML system to medium to large datasets we only use online learning algorithms from this category: Linear Classification via online stochastic gradient descent (SGD) either with a negative log likelihood, a hinge or a Huber los, and maximum margin classification via online passive aggressive algorithms [1] – which iteratively solve constrained optimization problems to update the model weights to both guarantee small steps and retain a large margin.

Support vector machines. Closely related to the algorithms from the GLM class described above support vector machines construct a maximal margin separating hyperplane by minimizing the hinge loss on the training data. As is well known, they can also be used for non-linear classification by employing the "kernel trick". The SVM implementations used in scikit-learn are based on online optimization using LibSVM [2] or liblinear [3] as backends.

Discriminant analysis. We also consider two instantiations from the family of discriminant analysis methods: (1) Quadratic discriminant analysis (QDA) assumes that the feature values for each class are normally distributed. Classification is done by applying the likelihood ratio test. (2) Linear discriminant analysis (LDA) makes the additional assumption that the covariance of each of the classes is identical, which leads to a linear decision boundary.

Nearest neighbors classification. k-nearest neighbors is a non-parametric classification algorithm that classifies samples based on the class membership of their nearest neighbors in feature space. Nearest neighbor classifiers often exhibit strong performance in problems where a proper metric in feature space is known, but can be computationally expensive to compute for large datasets (when using a basic implementation as that contained in scikit-learn).

Naïve Bayes. Naïve Bayes methods assume independence between every pair of features allowing the use of Bayes' theorem to find the most probable class given the training data. They are fast to train and very robust due to their simplifying assumptions. We consider three variants of Naïve Bayes: In Gaussian Naïve Bayes the likelihood of the features is assumed to be Gaussian. Multinomial Naïve Bayes is a variant suitable for multinomially distributed data. Bernoulli Naïve Bayes assumes a Bernoulli distribution. However, as shown in our experiments they proved too simplistic of a classifier choice to be effective in the AutoML setting.

Decision trees. Decision trees are one of the most frequently used baseline classifiers in operation. They also constitute the building block of ensemble methods such as random forests which often show strong empirical performance. Basic decision trees (as used in our pipeline) are constructed by recursively splitting the training data into subsets based on the feature values. The criteria for determining the best rule for splitting in scikit-learn are based on a cross-entropy measurement or Gini impurity.

Ensemble methods. The final set of machine learning models we consider are simple, yet powerful, ensemble methods. Concretely we consider AdaBoost, gradient boosting, random forests and extremely randomized trees. Among these, AdaBoost is perhaps the most prototypical ensemble method which combines a sequence of "weak learners" into a weighted majority vote. Successive weak learners are trained with reweighted versions of the training data, where higher weights are assigned to misclassified samples. We use decision trees with a maximum depth of 10 as weak learners. The other ensemble methods we consider also use decision trees as base classifiers: Gradient boosting generalizes the idea of AdaBoost to arbitrarily differentiable loss functions. Random forests and extremely randomized trees are ensembles of decision trees that are trained with a bootstrap sample of the training data. In random forests the best splitting rule is determined by optimizing Gini impurity or information gain among a random subset of the features. Extremely randomized trees use randomly generated splitting rules as candidates and choose the best one.

A.2 Feature preprocessing algorithms

In addition to the classifier choices AUTO-SKLEARN contains a large set of different feature preprocessing algorithms; which can optionally be selected by the Bayesian optimization algorithm. These again can be separated roughly into 8 categories.

Matrix decomposition. The first category of feature preprocessing methods decomposes the given data into maximally descriptive components. Among these we consider Principal component analysis (PCA), a truncated SVD, Kernel PCA and Independent component analysis (ICA). Principal component analysis (PCA) is perhaps the most well known feature preprocessing method and performs a linear mapping of the data onto its principal components. Truncated SVD is an approximation to PCA which also works in a spare data regime. Kernel PCA is performing principal component analysis in a reproducing kernel Hilbert space, allowing for non-linear mappings. Independent component analysis (ICA) finds basis vectors such that data projected onto these basis have maximum statistical independence.

Univariate feature selection. A second category of feature preprocessing methods which, although simple, often performs well is to "simply" select features based on univariate statistical tests on the dataset. From these, scikit-learn includes: (1) feature selection according to a percentile of the highest scores given some scoring function (such as the feature variance) (2) discarding features lower than a given threshold on a scoring function (this is called *select rates* in the main paper).

Classification-based feature selection. Feature selection can also be performed by more elaborate machine learning methods. We include classification-based feature selection which consists of fitting a classifier to the data and choosing features that the classifier deems to be important for correct classification. Concretely, we allow for the use of l_1-regularized linear SVMs for feature selection by fitting the SVM to the data and choosing features corresponding to non-zero model coefficients. Additionally, extremely randomized trees can be used as a preprocessor for feature selection. The relative importance of a feature is calculated as the reduction of the splitting criterion brought by that feature. Then only the most important features are selected.

Feature clustering. Instead of feature selection one can also merge features (i.e. add them together) which highly correlate. For this purpose the feature agglomeration preprocessing, implemented in scikit-learn is included in our AutoML system.

Kernel approximations. Can be used to approximate kernel functions (such as the RBF kernel) over the dataset without the need for actual (costly) computation of the kernel between all data points. From these we consider *random kitchen sinks* and *nystroem sampling*. Random kitchen sinks map the data to a higher dimensional feature space through a randomized feature map that guarantees that inner product between pairs of points in feature space approximates the evaluation of a kernel (in our case the Gaussian kernel). Nystroem sampling is a technique that accomplishes the same goal by projecting examples on a random subset of the data.

Polynomial feature expansion. Simply expands the set of available features by calculating all polynomial combinations (up to a given degree) of the features.

Feature embeddings. Project the set of features into a feature space through a non-linear embedding. While there exists a multitude of such embedding methods we consider only embedding through random forests. More precisely our random trees embedding uses an ensemble of totally random trees for unsupervised transformation of the data to a sparse representation. Points are encoded according to the leaf of each tree they are sorted in.

Sparse representation transformation. For completeness we also include a simple sparse to dense transformation in our preprocessing pipeline which, while costly, allows us to use algorithms on sparse data that cannot natively handle sparsely represented inputs.

A.3 Data preprocessing algorithms

Prior to doing feature preprocessing and classification, the data is preprocessed by the following algorithms in the presented order:

1. One Hot Encoding replaces categorical features f with domain $v_1, \ldots, v_k$ by k binary variables, only the i-th of which is set to true for data points where f is set to v_i.
2. Imputation will replace missing values by the mean, median or most frequent value.
3. Rescaling either standardizes the features to have zero mean and unit variance or rescales them into the range $[0, 1]$. Alternatively, it can normalize samples to have unit length or leave features unscaled.
4. Balancing activates a class weight mechanism of the classification algorithm if it supports one.

B Details of AUTO-SKLEARN

As with every robust real-world system, we had to handle many important details in AUTO-SKLEARN. To make the most of our computational power and not get stuck in a very slow run of a certain combination of preprocessing and machine learning algorithm, we implemented measures to prevent such long runs. First, we limited the time for each evaluation of an instantiation of the ML framework, typically to $\frac{1}{10}$ of the overall time limit. We also limited the memory of such evaluations to prevent the operating system from swapping. When an evaluation went over one of those limits, we killed it and returned the worst possible score for the given evaluation metric. For some of the models we employed an iterative training procedure; we instrumented these to still return a performance value when a limit was reached. To further reduce the amount of overly long runs, we forbade several combinations of preprocessors and classification methods: in particular, kernel approximation was forbidden to be active in conjunction with non-linear and tree-based methods as well as the KNN algorithm. (SMAC handles such forbidden combinations natively.) For the same reason we also left out feature learning algorithms, such as dictionary learning.

Another issue in hyperparameter optimization is overfitting and data resampling. Here we had to trade off between running a more robust cross-validation (which comes at little additional overhead in SMAC) and evaluating models on all cross-validation folds to allow for ensemble construction with these models. Thus, for tasks with a rigid time limit of 1h, we used a simple train/test split. In

contrast, we are able to employ ten-fold crossvalidation in our 24h and 30h runs, as well as in our experiments for the human track of the AutoML challenge.

Finally, not every supervised learning task (for example classification with multiple targets), can be solved by all of the algorithms available in AUTO-SKLEARN. Thus, given a new dataset, AUTO-SKLEARN preselects the methods that are suitable for the dataset's properties. Since scikit-learn methods are restricted to numerical input values, we transformed data by applying a one-hot encoding to categorical features. In order to keep the number of dummy features low, we configured a percentage threshold. A value occurring more seldom than this percentage was transformed to a special *other* variable [4].

C Meta-features

Meta-feature	Value			Calculation time (s)		
	Minimum	**Mean**	**Maximum**	**Minimum**	**Mean**	**Maximum**
class-entropy	0.64	1.92	4.70	0.00	0.00	0.00
class-probability-max	0.04	0.43	0.90	0.00	0.00	0.00
class-probability-mean	0.04	0.28	0.50	0.00	0.00	0.00
class-probability-min	0.00	0.19	0.48	0.00	0.00	0.00
class-probability-std	0.00	0.10	0.35	0.00	0.00	0.00
dataset-ratio	0.00	0.06	0.62	0.00	0.00	0.00
inverse-dataset-ratio	1.62	141.90	1620.00	0.00	0.00	0.00
kurtosis-max	-1.30	193.43	4812.49	0.00	0.01	0.05
kurtosis-mean	-1.30	24.32	652.23	0.00	0.01	0.05
kurtosis-min	-3.00	-0.59	5.25	0.00	0.01	0.05
kurtosis-std	0.00	48.83	1402.86	0.00	0.01	0.05
landmark-1NN*	0.20	0.79	1.00	0.01	0.61	8.97
landmark-decision-node-learner*	0.07	0.55	0.96	0.00	0.13	1.34
landmark-decision-tree*	0.20	0.78	1.00	0.00	0.49	5.23
landmark-lda*	0.26	0.79	1.00	0.00	1.39	70.08
landmark-naive-bayes*	0.10	0.68	0.97	0.00	0.06	1.05
landmark-random-node-learner*	0.07	0.47	0.91	0.00	0.02	0.26
log-dataset-ratio	-7.39	-3.80	-0.48	0.00	0.00	0.00
log-inverse-dataset-ratio	0.48	3.80	7.39	0.00	0.00	0.00
log-number-of-features	1.10	2.92	5.63	0.00	0.00	0.00
log-number-of-instances	4.04	6.72	9.90	0.00	0.00	0.00
number-of-Instances-with-missing-values	0.00	96.00	2480.00	0.00	0.00	0.01
number-of-categorical-features	0.00	13.25	240.00	0.00	0.00	0.00
number-of-classes	2.00	6.58	28.00	0.00	0.00	0.00
number-of-features	3.00	33.91	279.00	0.00	0.00	0.00
number-of-features-with-missing-values	0.00	3.54	34.00	0.00	0.00	0.00
number-of-instances	57.00	2126.33	20000.00	0.00	0.00	0.00
number-of-missing-values	0.00	549.49	22175.00	0.00	0.00	0.00
number-of-numeric-features	0.00	20.67	216.00	0.00	0.00	0.00
pca-95percent*	0.02	0.52	1.00	0.00	0.00	0.00
pca-kurtosis-first-pc*	-2.00	13.38	730.92	0.00	0.00	0.01
pca-skewness-first-pc*	-27.07	-0.16	6.46	0.00	0.00	0.04
percentage-of-Instances-with-missing-values	0.00	0.14	1.00	0.00	0.00	0.00
percentage-of-features-with-missing-values	0.00	0.16	1.00	0.00	0.00	0.00
percentage-of-missing-values	0.00	0.03	0.65	0.00	0.00	0.00
ratio-categorical-to-numerical	0.00	1.35	33.00	0.00	0.00	0.00
ratio-numerical-to-categorical	0.00	0.49	7.00	0.00	0.00	0.00
skewness-max	0.00	5.34	67.41	0.00	0.00	0.04
skewness-mean	-0.56	1.27	14.71	0.00	0.00	0.04
skewness-min	-21.19	-0.62	1.59	0.00	0.00	0.04
skewness-std	0.00	1.60	18.89	0.00	0.01	0.05
symbols-max	0.00	13.09	429.00	0.00	0.00	0.00
symbols-mean	0.00	3.01	41.38	0.00	0.00	0.00
symbols-min	0.00	1.44	12.00	0.00	0.00	0.00
symbols-std	0.00	3.06	107.21	0.00	0.00	0.00
symbols-sum	0.00	71.04	1648.00	0.00	0.00	0.00

Table 1: List of implemented meta-features. Meta-features marked with an asterisks were only used to do the dataset clustering in Section 6

D Ensemble selection

Pseudocode explaining our implementation of the ensemble selection algorithm [5].

Procedure 1: EnsembleSelection(M, S)

Input : Models M, Ensemble size S , $n = |M|$
Output : Ensemble E

$E \leftarrow \emptyset$
for $i = 0 \ldots S$ **do**
  $b \leftarrow \text{argmax}_{j=0\ldots n}\, performance(E \cup M[j])$
  $E \leftarrow E \cup M[b]$
return E

E Configuration spaces for different dataset properties

(a) classifiers

name	#λ	cat (cond)	cont (cond)
AdaBoost (AB)	4	1 (-)	3 (-)
Bernoulli naïve Bayes	2	1 (-)	1 (-)
decision tree (DT)	4	1 (-)	3 (-)
extreml. rand. trees	5	2 (-)	3 (-)
Gaussian naïve Bayes	-	-	-
gradient boosting (GB)	6	-	6 (-)
kNN	3	2 (-)	1 (-)
LDA	4	1 (-)	3 (1)
linear SVM	4	2 (-)	2 (-)
kernel SVM	7	2 (-)	5 (2)
multinomial naïve Bayes	2	1 (-)	1 (-)
passive aggressive	3	1 (-)	2 (-)
QDA	2	-	2 (-)
random forest (RF)	5	2 (-)	3 (-)
SGD	10	4 (-)	6 (3)

(b) preprocessing methods

name	#λ	cat (cond)	cont (cond)
densifier	-	-	-
extreml. rand. trees prepr.	5	2 (-)	3 (-)
kernel PCA	5	1 (-)	4 (3)
rand. kitchen sinks	2	-	2 (-)
linear SVM prepr.	3	1 (-)	2 (-)
no preprocessing	-	-	-
nystroem sampler	5	1 (-)	4 (3)
random trees embed.	4	-	4 (-)
select percentile	2	1 (-)	1 (-)
select rates	3	2 (-)	1 (-)
truncated SVD	1	-	1 (-)
one-out-of-k encoding	2	1 (-)	1 (1)
imputation	1	1 (-)	-
balancing	1	1 (-)	-
rescaling	1	1 (-)	-

Table 2: Number of hyperparameters for each possible classifier (left) and feature preprocessing method (right) for a **binary classification** dataset in **sparse** representation.

(a) classifiers

name	#λ	cat (cond)	cont (cond)
AdaBoost (AB)	4	1 (-)	3 (-)
decision tree (DT)	4	1 (-)	3 (-)
extreml. rand. trees	5	2 (-)	3 (-)
Gaussian naïve Bayes	-	-	-
gradient boosting (GB)	6	-	6 (-)
kNN	3	2 (-)	1 (-)
LDA	4	1 (-)	3 (1)
linear SVM	4	2 (-)	2 (-)
kernel SVM	7	2 (-)	5 (2)
multinomial naïve Bayes	2	1 (-)	1 (-)
passive aggressive	3	1 (-)	2 (-)
QDA	2	-	2 (-)
random forest (RF)	5	2 (-)	3 (-)
SGD	10	4 (-)	6 (3)

(b) preprocessing methods

name	#λ	cat (cond)	cont (cond)
extreml. rand. trees prepr.	5	2 (-)	3 (-)
fast ICA	4	3 (-)	1 (1)
feature agglomeration	4	3 (-)	1 (-)
kernel PCA	5	1 (-)	4 (3)
rand. kitchen sinks	2	-	2 (-)
linear SVM prepr.	3	1 (-)	2 (-)
no preprocessing	-	-	-
nystroem sampler	5	1 (-)	4 (3)
PCA	2	1 (-)	1 (-)
polynomial	3	2 (-)	1 (-)
random trees embed.	4	-	4 (-)
select percentile	2	1 (-)	1 (-)
select rates	3	2 (-)	1 (-)
one-out-of-k encoding	2	1 (-)	1 (1)
imputation	1	1 (-)	-
balancing	1	1 (-)	-
rescaling	1	1 (-)	-

Table 3: Number of hyperparameters for each possible classifier (left) and feature preprocessing method (right) for a **multiclass classification** dataset in **dense** representation.

(a) classifiers

name	#λ	cat (cond)	cont (cond)
AdaBoost (AB)	4	1 (-)	3 (-)
decision tree (DT)	4	1 (-)	3 (-)
extreml. rand. trees	5	2 (-)	3 (-)
Gaussian naïve Bayes	-	-	-
gradient boosting (GB)	6	-	6 (-)
kNN	3	2 (-)	1 (-)
LDA	4	1 (-)	3 (1)
linear SVM	4	2 (-)	2 (-)
kernel SVM	7	2 (-)	5 (2)
multinomial naïve Bayes	2	1 (-)	1 (-)
passive aggressive	3	1 (-)	2 (-)
QDA	2	-	2 (-)
random forest (RF)	5	2 (-)	3 (-)
SGD	10	4 (-)	6 (3)

(b) preprocessing methods

name	#λ	cat (cond)	cont (cond)
densifier	-	-	-
extreml. rand. trees prepr.	5	2 (-)	3 (-)
kernel PCA	5	1 (-)	4 (3)
rand. kitchen sinks	2	-	2 (-)
linear SVM prepr.	3	1 (-)	2 (-)
no preprocessing	-	-	-
nystroem sampler	5	1 (-)	4 (3)
random trees embed.	4	-	4 (-)
select percentile	2	1 (-)	1 (-)
select rates	3	2 (-)	1 (-)
truncated SVD	1	-	1 (-)
one-out-of-k encoding	2	1 (-)	1 (1)
imputation	1	1 (-)	-
balancing	1	1 (-)	-
rescaling	1	1 (-)	-

Table 4: Number of hyperparameters for each possible classifier (left) and feature preprocessing method (right) for a **multiclass classification** dataset in **sparse** representation.

F Properties of datasets used in the experiments

Name	#Continuous	#Nominal	#Classes	Sparse?	Missing Values	#Training Samples	#Test Samples
Abalone	7	1	26	-	-	2924	1253
Amazon	10000	0	50	-	-	1050	450
Car	0	6	4	-	-	1210	518
Cifar10	3072	0	10	-	-	50000	10000
Cifar-10-Small	3072	0	10	-	-	10000	10000
Convex	784	0	2	-	-	8000	50000
Dexter	20000	0	2	X	-	420	180
Dorothea	100000	0	2	X	-	805	345
GermanCredit	7	13	2	-	-	700	300
Gisette	5000	0	2	-	-	4900	2100
KDD09-Appetency	192	38	2	-	X	35000	15000
KR-vs-KP	0	36	2	-	-	2238	958
Madelon	500	0	2	-	-	1820	780
MNIST Basic	784	0	10	-	-	12000	50000
Rot. MNIST + BI	784	0	10	-	-	12000	50000
Secom	590	0	2	-	X	1097	470
Semeion	256	0	10	-	-	1116	477
Shuttle	9	0	7	-	-	43500	14500
Waveform	40	0	3	-	-	3500	1500
Wine Quality	11	0	7	-	-	3429	1469
Yeast	8	0	10	-	-	1039	445

Table 5: Auto-WEKA datasets [6].

ID	Name	#Continuous	#Nominal	#Classes	Sparse?	Missing Values	#Training Samples	#Test Samples
38	Sick	7	22	2	-	X	2527	1245
46	Splice	0	60	3	-	-	2137	1053
179	adult	2	12	2	-	X	32724	16118
184	KROPT	0	6	18	-	-	18797	9259
554	MNIST	784	0	10	-	-	46900	23100
772	quake	3	0	2	-	-	1459	719
917	fri_c1_1000_25 (binarized)	25	0	2	-	-	670	330
1049	pc4	37	0	2	-	-	976	482
1111	KDDCup09 Appetency	192	38	2	(X)	X	33500	16500
1120	Magic Telescope	10	0	2	-	-	12743	6277
1128	OVA Breast	10935	0	2	-	-	1035	510
293	Covertype (binarized)	54	0	2	X	-	389278	191734
389	fbis_wc	2000	0	17	X	-	1651	812

Table 6: Representative datasets for the 13 clusters obtained via g-means clustering of the 140 datasets' meta-feature vectors.

ID	Name	#Continuous	#Nominal	#Classes	Sparse?	Missing Values	#Training Samples	#Test Samples
3	kr-vs-kp	0	36	2	-	-	2141	1055
6	letter	16	0	26	-	-	13402	6598
12	mfeat-factors	216	0	10	-	-	1340	660
14	mfeat-fourier	76	0	10	-	-	1340	660
16	mfeat-karhunen	64	0	10	-	-	1340	660
18	mfeat-morphological	6	0	10	-	-	1340	660
21	car	0	6	4	-	-	1157	571
22	mfeat-zernike	47	0	10	-	-	1340	660
23	cmc	2	7	3	-	-	986	487
24	mushroom	0	22	2	-	X	5443	2681
26	nursery	0	8	5	-	-	8682	4278
28	optdigits	64	0	10	-	-	3765	1855
30	page-blocks	10	0	5	-	-	3666	1807
31	credit-g	7	13	2	-	-	670	330
32	pendigits	16	0	10	-	-	7364	3628
36	segment	19	0	7	-	-	1547	763
38	sick	7	22	2	-	X	2527	1245
44	spambase	57	0	2	-	-	3082	1519
46	splice	0	60	3	-	-	2137	1053
57	hypothyroid	7	22	4	-	X	2527	1245
60	waveform-5000	40	0	3	-	-	3351	1649
179	adult	2	12	2	-	X	32724	16118
180	covertype	14	40	7	-	-	73962	36431
181	yeast	8	0	10	-	-	991	493
182	satimage	36	0	6	-	-	4308	2122
184	kropt	0	6	18	-	-	18797	9259
185	baseball	15	1	3	-	X	897	443
273	IMDB.drama	1001	0	2	X	-	81007	39899
293	covertype	54	0	2	X	-	389278	191734
300	isolet	617	0	26	-	-	5224	2573
351	codrna	8	0	2	X	-	327338	161227
354	poker	10	0	2	X	-	686756	338254
357	vehicle_sensIT	100	0	2	X	-	66012	32516
389	fbis.wc	2000	0	17	X	-	1651	812
390	new3s.wc	26832	0	44	X	-	6401	3157
391	re0.wc	2886	0	13	X	-	1007	497
392	oh0.wc	3182	0	10	X	-	672	331
393	la2s.wc	12432	0	6	X	-	2059	1016
395	re1.wc	3758	0	25	X	-	1109	548
396	la1s.wc	13195	0	6	X	-	2146	1058
398	wap.wc	8460	0	20	X	-	1044	516
399	ohscal.wc	11465	0	10	X	-	7478	3684
401	oh10.wc	3238	0	10	X	-	702	348
554	mnist_784	784	0	10	-	-	46900	23100
679	rmftsa_sleepdata	2	0	4	-	-	687	337
715	fri_c3_1000_25	25	0	2	-	-	670	330
718	fri_c4_1000_100	100	0	2	-	-	670	330
720	abalone	7	1	2	-	-	2798	1379
722	pol	48	0	2	-	-	10050	4950
723	fri_c4_1000_25	25	0	2	-	-	670	330
727	2dplanes	10	0	2	-	-	27314	13454
728	analcatdata_supreme	7	0	2	-	-	2714	1338
734	ailerons	40	0	2	-	-	9212	4538
735	cpu_small	12	0	2	-	-	5488	2704
737	space_ga	6	0	2	-	-	2081	1026
740	fri_c3_1000_10	10	0	2	-	-	670	330
741	rmftsa_sleepdata	1	1	2	-	-	686	338
743	fri_c1_1000_5	5	0	2	-	-	670	330
751	fri_c4_1000_10	10	0	2	-	-	670	330
752	puma32H	32	0	2	-	-	5488	2704
761	cpu_act	21	0	2	-	-	5488	2704
772	quake	3	0	2	-	-	1459	719
797	fri_c4_1000_50	50	0	2	-	-	670	330
799	fri_c0_1000_5	5	0	2	-	-	670	330
803	delta_ailerons	5	0	2	-	-	4776	2353
806	fri_c3_1000_50	50	0	2	-	-	670	330
807	kin8nm	8	0	2	-	-	5488	2704
813	fri_c3_1000_5	5	0	2	-	-	670	330
816	puma8NH	8	0	2	-	-	5488	2704
819	delta_elevators	6	0	2	-	-	6376	3141
821	house_16H	16	0	2	-	-	15265	7519
822	cal_housing	8	0	2	-	-	13828	6812
823	houses	8	0	2	-	-	13828	6812
833	bank32nh	32	0	2	-	-	5488	2704
837	fri_c1_1000_50	50	0	2	-	-	670	330
843	house_8L	8	0	2	-	-	15265	7519
845	fri_c0_1000_10	10	0	2	-	-	670	330
846	elevators	18	0	2	-	-	11121	5478
847	wind	14	0	2	-	-	4404	2170

Table 7: All datasets which were used for generating metadata and the experiments in Section 5 of the main paper.

ID	Name	#Continuous	#Nominal	#Classes	Sparse?	Missing Values	#Training Samples	#Test Samples
849	fri_c0_1000_25	25	0	2	-	-	670	330
866	fri_c2_1000_50	50	0	2	-	-	670	330
871	pollen	5	0	2	-	-	2578	1270
881	mv	7	3	2	-	-	27314	13454
897	colleges_aaup	13	2	2	-	X	777	384
901	fried	10	0	2	-	-	27314	13454
903	fri_c2_1000_25	25	0	2	-	-	670	330
904	fri_c0_1000_50	50	0	2	-	-	670	330
910	fri_c1_1000_10	10	0	2	-	-	670	330
912	fri_c2_1000_5	5	0	2	-	-	670	330
913	fri_c2_1000_10	10	0	2	-	-	670	330
914	balloon	1	0	2	-	-	1340	661
917	fri_c1_1000_25	25	0	2	-	-	670	330
923	visualizing_soil	3	1	2	-	-	5789	2852
930	colleges_usnews	32	1	2	-	X	872	430
934	socmob	1	4	2	-	-	774	382
953	splice	0	60	2	-	-	2137	1053
958	segment	19	0	2	-	-	1547	763
959	nursery	0	8	2	-	-	8683	4277
962	mfeat-morphological	6	0	2	-	-	1340	660
966	analcatdata_halloffame	15	1	2	-	X	897	443
971	mfeat-fourier	76	0	2	-	-	1340	660
976	kdd_JapaneseVowels	14	0	2	-	-	6673	3288
977	letter	16	0	2	-	-	13400	6600
978	mfeat-factors	216	0	2	-	-	1340	660
979	waveform-5000	40	0	2	-	-	3350	1650
980	optdigits	64	0	2	-	-	3765	1855
991	car	0	6	2	-	-	1157	571
993	kdd_ipums_la_97-small	33	27	2	-	X	4702	2317
995	mfeat-zernike	47	0	2	-	-	1340	660
100	hypothyroid	7	22	2	-	X	2527	1245
100	kdd_ipums_la_98-small	16	39	2	-	X	5014	2471
101	kdd_ipums_la_99-small	15	41	2	-	X	5925	2919
101	pendigits	16	0	2	-	-	7364	3628
102	mfeat-karhunen	64	0	2	-	-	1340	660
102	page-blocks	10	0	2	-	-	3666	1807
103	sylva_agnostic	216	0	2	-	-	9644	4751
104	sylva_prior	108	0	2	-	-	9644	4751
104	gina_prior2	784	0	10	-	-	2322	1146
104	pc4	37	0	2	-	-	976	482
105	pc3	37	0	2	-	-	1047	516
105	jm1	21	0	2	-	X	7292	3593
105	mc1	38	0	2	-	-	6342	3124
106	kc1	21	0	2	-	-	1413	696
106	pc1	21	0	2	-	-	743	366
106	pc2	36	0	2	-	-	3744	1845
1111	KDDCup09_appetency	192	38	2	-	X	33500	16500
1112	KDDCup09_churn	192	38	2	-	X	33500	16500
1114	KDDCup09_upselling	192	38	2	-	X	33500	16500
1116	musk	166	1	2	-	-	4420	2178
1119	adult-census	6	8	2	-	X	21815	10746
1120	MagicTelescope	10	0	2	-	-	12743	6277
1128	OVA_Breast	10935	0	2	-	-	1035	510
1130	OVA_Lung	10935	0	2	-	-	1035	510
1134	OVA_Kidney	10935	0	2	-	-	1035	510
1138	OVA_Uterus	10935	0	2	-	-	1035	510
1139	OVA_Omentum	10935	0	2	-	-	1035	510
1142	OVA_Endometrium	10935	0	2	-	-	1035	510
1146	OVA_Prostate	10935	0	2	-	-	1035	510
1161	OVA_Colon	10935	0	2	-	-	1035	510
1166	OVA_Ovary	10935	0	2	-	-	1035	510

Table 8: All datasets which were used for generating metadata and the experiments in Section 5 of the main paper (continued).

G Setup section 4

To compare against Auto-WEKA, we used 21 datasets (detailed in Table 5) with their original train/test split [6], a walltime limit of 30 hours, 10-fold cross validation (where the evaluation of each fold was allowed to take 150 minutes, except for hyperopt-sklearn which uses a 80/20 train/test split), and 10 independent optimization runs with SMAC on each dataset. Our results for Auto-WEKA resemble those of Thornton et al. [7], with minor differences caused by our faster machines: All our experiments ran on Intel Xeon E5-2650 v2 eight-core processors with 2.60GHz and 4GiB of RAM. We allowed the machine learning framework to use 3GiB and reserved the rest for SMAC. All experiments used Auto-WEKA 0.5 and scikit-learn 0.16.1.

H Evaluation of our new AutoML methods - additional plot

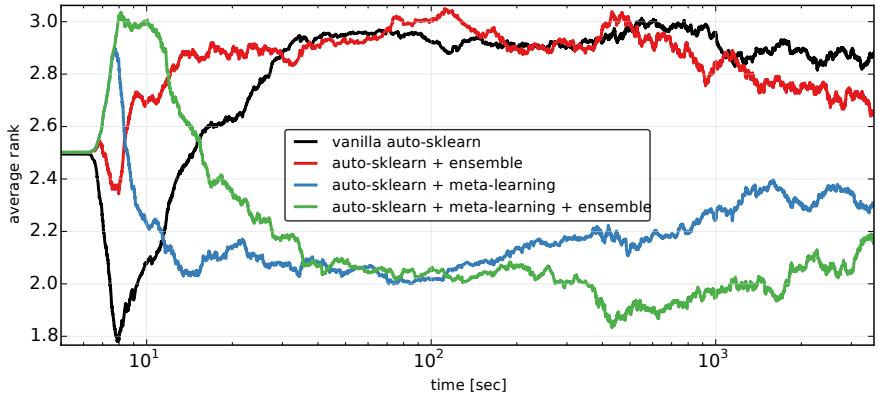


Figure 1: Average rank of all four AUTO-SKLEARN versions ranked by balanced error (BER) across 140 datasets. In contrast to the plot in the main paper, this plot is on a log-scale. Due to the little additional overhead that meta-learning and ensemble selection cause, vanilla AUTO-SKLEARN is able to achieve the best rank within the first 10 seconds as it produces predictions before the other AUTO-SKLEARN variants finish training their first model.

I Average rank over datasets for optimizing single classifiers and preprocessors compared to AUTO-SKLEARN

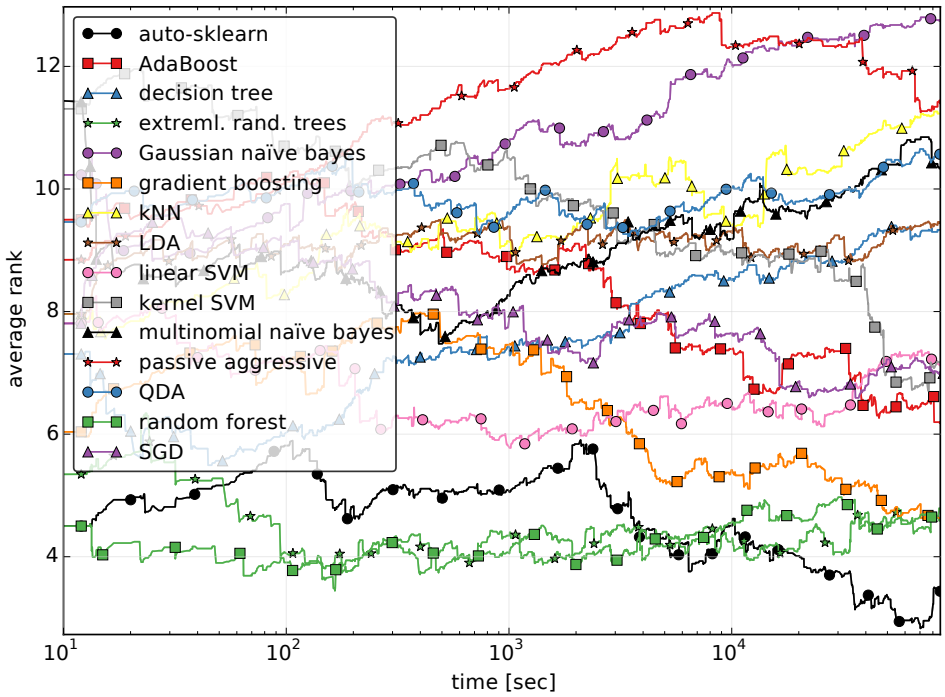


Figure 2: Average rank. We compare test performance over time for optimizing each classifier with all preprocessing methods separately with optimizing the joint space AUTO-SKLEARN. We optimize each method for one day. Each line shows the average across 13 datasets; for each dataset drew a bootstrap sample of 100 joint runs and computed the average rank across these runs.

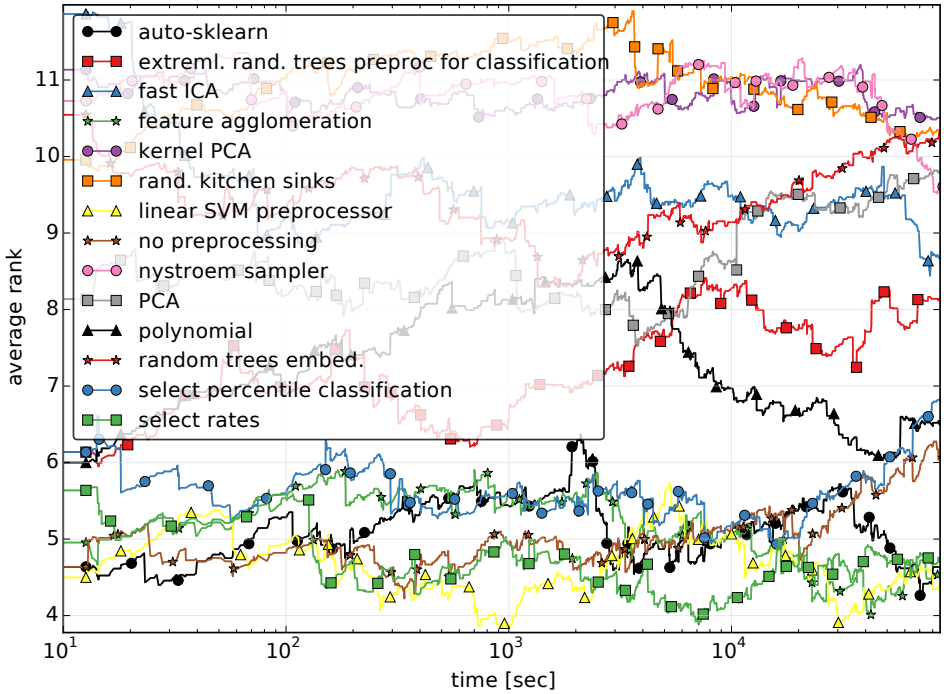
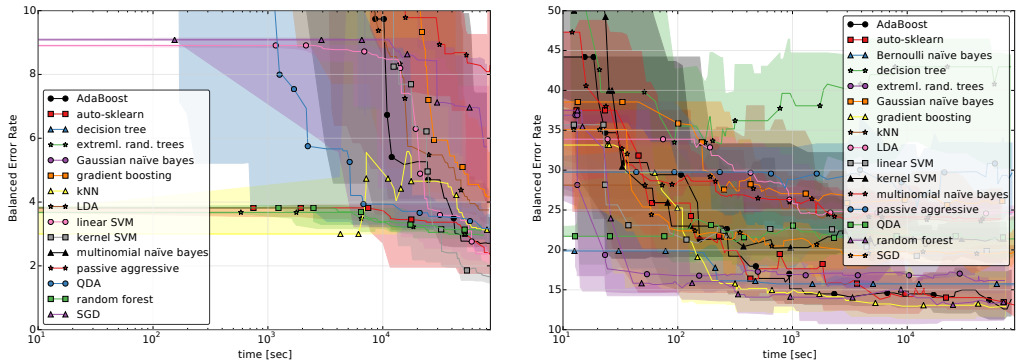


Figure 3: Average rank for different preprocessing methods on a dense dataset. We compare test performance over time for optimizing each preprocessing method with all classifier with optimizing the joint space AUTO-SKLEARN. We optimize each method for one day. Each line shows the average across 13 datasets; for each dataset drew a bootstrap sample of 100 joint runs and computed the average rank across these runs.



(a) Classifiers on dataset 554

(b) Classifiers on dataset 1049

Figure 4: Performance (balanced classification error, BER) of different subspaces compared to AUTO-SKLEARN over time. We show the median test performance over time for all classifiers with all preprocessing methods separately with optimizing the joint space.

References

[1] Koby Crammer, Ofer Dekel, Joseph Keshet, Shai Shalev-Shwartz, and Yoram Singer. Online passive-aggressive algorithms. *Journal of Machine Learning Research*, 7:551–585, 2006.

[2] Chih-Chung Chang and Chih-Jen Lin. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27:1–27:27, 2011. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

[3] Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin. LIBLINEAR: A library for large linear classification. *The Journal of Machine Learning Research*, 9:1871–1874, 2008.

[4] A. Niculescu-Mizil, C. Perlich, G. Swirszcz, V. Sindhwani, Y. Liu, P. Melville, D. Wang, J. Xiao, J. Hu, M. Singh, W. Shang, and Y. Zhu. Winning the KDD cup orange challenge with ensemble selection. *The 2009 Knowledge Discovery in Data Competition*, pages 23–34, 2009.

[5] R. Caruana, A. Niculescu-Mizil, G. Crew, and A. Ksikes. Ensemble selection from libraries of models. In *Proc. of ICML'04*, page 18, 2004.

[6] Auto-WEKA website. `http://www.cs.ubc.ca/labs/beta/Projects/autoweka`.

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

C

APPENDIX FOR AUTO-SKLEARN 2.0: HANDS-FREE AUTOML VIA META-LEARNING

Appendix A. Additional pseudo-code

We give pseudo-code for computing the estimated generalization error of $\mathcal{P}$ across all meta-datasets $\mathbf{D}_{\text{meta}}$ for K-folds cross-validation in Algorithm 2 and successive halving in Algorithm 3.

Algorithm 2: Estimating the generalization error of a portfolio with K-Fold Cross-Validation

Input: Ordered set of ML pipelines $\mathcal{P}$, datasets $\mathbf{D}_{\text{meta}}$, number of folds K,
$L = 0$
for $d \in (1, 2, \ldots, |\mathbf{D}_{\text{meta}}|)$ **do**
  $l_d = \infty$
  for $p \in \mathcal{P}$ **do**
    $l = 0$
    for $k \in (1, 2, \ldots, K)$ **do**
      $l = l + \widehat{GE}(\mathcal{M}_{\boldsymbol{\lambda}}^{\mathcal{D}_{\text{train}}^{(\text{train},k)}}, \mathcal{D}_{\text{train}}^{(\text{val},k)})$
    end for
    $l = l/K$
    if $l < l_d$ **then**
      $l_d = l$
    end if
  end for
  $L = L + l_d$
end for
return $L/|\mathbf{D}_{\text{meta}}|$

Appendix B. Additional results and experiments

In this section we will give additional results backing up our findings. Concretely, we will give further details on the reduced search space and provide further experimental evidence, we will provide the main results from the main paper without post-hoc ensembles, and we will give the raw numbers before averaging.

B.1 Early Stopping and Retrieving Intermittent Results

Estimating the generalization error of a pipeline $\mathcal{M}_{\boldsymbol{\lambda}}$ practically requires to restrict the CPU-time per evaluation to prevent that one single, very long algorithm run stalls the optimization procedure (Thornton et al., 2013; Feurer et al., 2015a). If an algorithm does not return a result within the assigned time limit, it is terminated and the worst possible generalization error is assigned. If the time limit is set too low, a majority of the algorithms do not return a result and thus provide very scarce information for the optimization procedure. A too high time limit, however, might as well not return any meaningful results since all time may be spent on long-running, under-performing pipelines. Additionally, for iterative algorithms (e.g., gradient boosting and linear models trained with stochastic

Algorithm 3: Estimating the generalization error of a portfolio with Successive Halving

Input: Ordered set of ML pipelines $\mathcal{P}$, datasets $\mathbf{D}_{\text{meta}}$, minimal budget b_{min}, maximal budget b_{max}, downsampling rate η
$L = \infty$
$R = b_{max}/b_{min}$
$s_{max} = \lfloor \log_\eta(R) \rfloor$
$B = (s_{max} + 1)R$
$n = \lceil \frac{B}{R} \frac{\eta^{s_{max}}}{(s_{max}+1)} \rceil$
$r = R\eta^{-s_{max}}$
for $d \in (1, 2, \ldots, |\mathbf{D}_{\text{meta}}|)$ **do**
  $l_d = \infty$
  $\mathcal{P}_d = \mathcal{P}$
  while True **do**
    $\mathcal{P}' = \mathcal{P}.pop(r)$ # Pop top r machine learning pipelines
    $\mathbf{l} = [\ ]$
    for $i \in (0, \ldots, s_{max})$ **do**
      $n_i = \lfloor n\eta^{-i} \rfloor$
      $r_i = r\eta^i$
      for $p \in \mathcal{P}'$ **do**
        $l = \widehat{GE}(\mathcal{M}_{\boldsymbol{\lambda}}^{\mathcal{D}_{\text{train}}^{\text{train}}}, \mathcal{D}_{\text{train}}^{\text{val}})$
        $\mathbf{l} = \mathbf{l} \cup l$
        if $l < l_d$ **then**
          $l_d = l$
        end if
      end for
      $\mathcal{P}' = top(\mathcal{P}', \mathbf{l}, \lfloor (n_i/eta) \rfloor)$, where $top(\mathcal{P}, \mathbf{l}, k)$ returns the top k performing machine learning pipelines.
    end for
    if $|\mathcal{P}_d| == 0$ **then**
      break
    end if
  end while
  $L = L + l_d$
end for
return $L/|\mathbf{D}_{\text{meta}}|$

AUTO-SKLEARN 2.0

		10	STD 10	60	STD 60
(1)	Auto-sklearn (1.0)	16.21	0.27	7.17	0.30
(2)	Auto-sklearn (1.0) ISS	18.10	0.13	9.57	0.22
(3)	Auto-sklearn (1.0) ISS + IRR	5.29	0.13	3.98	0.21
(4)	Auto-sklearn (1.0) ISS + IRR + Port	**3.70**	0.14	**3.08**	0.13

Table 10: Comparison of *Auto-sklearn 1.0* (1) with using only the iterative search space (2), using the iterative search space and iterative results retrieval (3) and also using a portfolio (4).

gradient descent), it is important to set the number of iterations such that the training converges and does not overfit, but most importantly finishes within this timelimit. Setting this number too high (training exceeds time limit and/or overfit) or too low (training has not yet converged although there is time left) has detrimental effects to the final performance of the AutoML system. To mitigate this risk we implemented two measures for iterative algorithms. Firstly, we use the early stopping mechanisms implemented by scikit-learn. Specifically, training stops if the loss on the training or validation set (depending on the model and the configuration) increases or stalls, which prevents overfitting (i.e. early stopping). Secondly, we make use of intermittent results retrieval, e.g., saving the results at checkpoints spaced at geometrically increasing iteration numbers, thereby ensuring that every evaluation of an iterative algorithm returns a performance and thus yields information for the optimizer. With this, our AutoML tool can robustly tackle large datasets without the necessity to finetune the number of iterations dependent on the time limit.

To study the effect of using the iterative results retrieval we compare standard *Auto-sklearn 1.0* with *Auto-sklearn* with the following changes applied one after the other: 1) move to a configuration space which consists only of iterative algorithms 2) enable intermittent results retrieval and 3) replace the KND by the portfolio. We give results in Table 10 and note that the KND uses meta-data gathered specifically for use with the reduced configuration space. Only restricting the configuration space leads to decreased performance which we attribute to the reduced hypothesis space. Intermittently writing results to disk reduces the amount of failures, and using a portfolio instead of the KND results in the best overall performance.

Once again, we also view the results through the eyes of a ranking plot in Figure 7. These results demonstrate that the iterative search space combined with intermittent results retrieval and a portfolio is especially dominating in the short term, and it takes a total of 50 minutes for *Auto-sklearn 1.0* to catch up. We would like to note that the performance of *Auto-sklearn 2.0* is even better as can be seen in Table 5, but it would be interesting to see how a portfolio of the full configuration space would perform, which we note as a further research question.

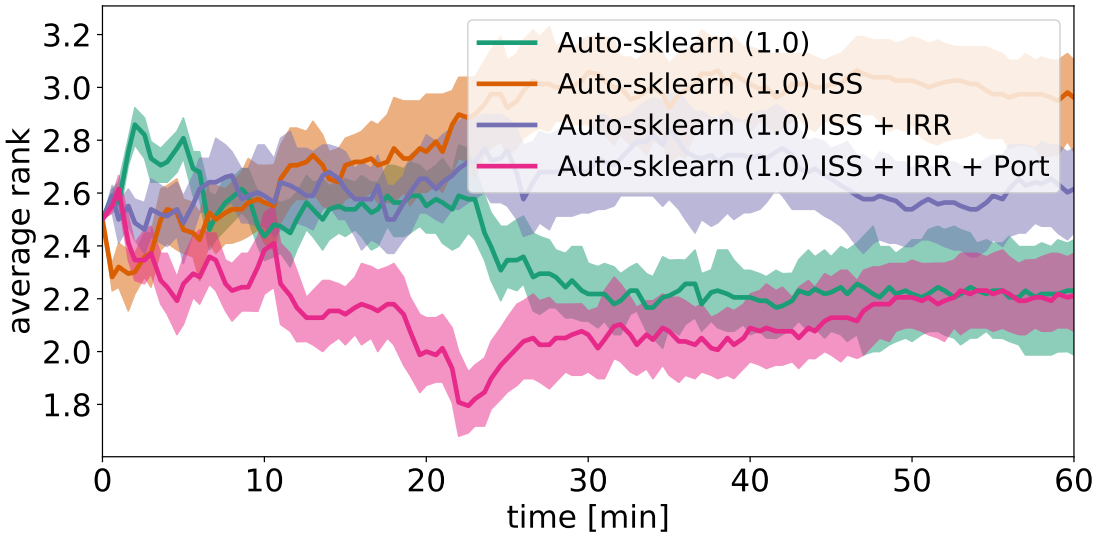


Figure 7: Ranking plot (lower is better) comparing *Auto-sklearn 1.0* (1) with using only the iterative search space (2), using the iterative search space and iterative results retrieval (3) and also using a portfolio (4). Compared to Figure 6 we randomly sample 500 combinations of the 10000 combinations of the 10 seeds of the 4 AutoML systems.

AUTO-SKLEARN 2.0

	10 minutes			60 minutes		
	BO	KND	Port	BO	KND	Port
holdout	7.27	6.43	**4.76**	4.58	4.99	**4.02**
SH; holdout	6.61	6.70	**5.76**	4.70	4.63	**3.97**
3CV	9.58	8.95	**7.88**	7.10	7.12	**5.98**
SH; 3CV	8.88	8.97	**7.20**	6.81	6.47	**6.01**
5CV	**10.48**	15.24	13.77	7.34	7.47	**5.66**
SH; 5CV	11.70	13.29	**8.06**	7.05	6.69	**5.93**
10CV	23.20	27.45	**18.73**	17.59	17.47	**16.17**
SH; 10CV	23.98	27.70	**18.84**	16.94	16.98	**16.07**

Table 11: Results from Table 2 without post-hoc ensembles.

	10MIN		60MIN	
	∅	std	∅	std
Auto-sklearn (2.0)	**5.01**	0.18	**3.18**	0.31
PoSH-Auto-sklearn	5.76	0.12	3.97	0.22
Auto-sklearn (1.0)	23.24	0.29	8.68	0.21

Table 12: Results from Table 5 without post-hoc ensembles.

B.2 Performance Without Post-Hoc Ensembling

We first give numbers comparing only Bayesian optimization, k-nearest datasets (KND) and a greedy portfolio. These results are similar to Table 2, but do not show the results of post-hoc ensembling, but using the single best model. Overall, they are qualitatively very similar, but it can be observed that the ensemble improves the average normalized balanced error rate in every case.

Next, we compare *Auto-sklearn 2.0* with *PoSH Auto-sklearn* and *Auto-sklearn 1.0*, but again only show the performance of the single best model and not of an ensemble as in the main paper. Again, the ensemble result in uniform performance improvements with *Auto-sklearn 2.0* still leading in terms of performance.

B.3 Unaggregated results

To allow the readers to asses the performance of the individual methods on the individual datasets we present the balanced error rates before normalizing and averaging them. We give the raw results for portfolios from Table 2 in Tables 13 and 14. Additionally, we give the raw results for *Auto-sklearn 2.0*, *PoSH Auto-sklearn* and *Auto-sklearn 1.0* in Tables 15 and 16.

Task ID	Name	holdout	SH; holdout	3CV	SH; 3CV	5CV	SH; 5CV	10CV	SH; 10CV
167104	Australian	0.1721	0.1569	0.1622	0.1617	0.1583	0.1602	**0.1556**	0.1559
167184	blood-transfusion	0.3641	**0.3610**	0.3725	0.3666	0.3689	0.3722	0.3674	0.3689
167168	vehicle	0.2211	0.2267	0.2017	0.2093	0.2172	0.2052	0.2310	**0.1870**
167161	credit-g	0.2942	**0.2841**	0.2939	0.2955	0.2942	0.2911	0.2939	0.2934
167185	cnae-9	0.0658	0.0680	0.0651	0.0616	**0.0550**	0.0629	0.0626	0.0553
189905	car	0.0049	0.0049	0.0097	0.0029	0.0047	0.0017	0.0023	**0.0009**
167152	mfeat-factors	0.0152	0.0164	0.0141	**0.0107**	0.0150	0.0117	0.0153	0.0149
167181	kc1	0.2735	0.2688	0.2720	0.2713	0.2547	0.2660	**0.2477**	0.2719
189906	segment	0.0666	0.0687	0.0681	**0.0620**	0.0664	0.0621	0.0643	0.0671
189862	jasmine	0.2044	0.2051	**0.1982**	0.1986	0.2010	0.2027	0.2043	0.2027
167149	kr-vs-kp	**0.0067**	0.0077	0.0093	0.0085	0.0079	0.0078	0.0071	0.0080
189865	sylvine	0.0592	0.0594	0.0600	0.0608	0.0582	0.0582	**0.0560**	0.0578
167190	phoneme	0.1231	0.1245	0.1168	0.1160	0.1152	0.1136	**0.1129**	0.1144
189861	christine	0.2670	0.2621	0.2608	0.2556	**0.2517**	0.2567	0.2587	0.2645
189872	fabert	0.3387	0.3399	0.3140	0.3120	**0.3096**	0.3204	0.3180	0.3172
189871	dilbert	0.0241	0.0248	0.0258	0.0220	**0.0191**	0.0211	0.0303	0.0647
168794	robert	**0.5489**	0.5861	0.5762	0.5583	0.5854	0.5873	0.6230	0.6230
168797	riccardo	0.0035	0.0052	0.0067	0.0054	**0.0027**	**0.0027**	0.5000	0.5000
168796	guillermo	0.2186	**0.2102**	0.2311	0.2228	0.2165	0.2837	0.5000	0.5000
75097	Amazon	**0.2361**	0.2431	0.2526	0.2526	0.2379	0.2385	0.2448	0.2443
126026	nomao	0.0353	0.0381	0.0360	0.0345	**0.0312**	0.0331	0.0403	0.0401
189909	jungle_chess	0.1212	0.1251	0.1232	0.1156	0.1280	0.1180	**0.1134**	0.1141
126029	bank-marketing	0.1397	0.1436	0.1402	0.1407	**0.1352**	0.1435	0.1378	0.1362
126025	adult	0.1579	0.1575	0.1553	**0.1540**	0.1591	0.1562	0.1545	0.1585
75105	KDDCup09	0.2450	0.2495	**0.2449**	0.2512	0.2525	0.2487	0.2497	0.2456
168795	shuttle	0.0093	0.0086	0.0085	**0.0084**	0.0085	0.0087	0.0088	0.0084
168793	volkert	0.3735	0.3724	0.3939	**0.3703**	0.3720	0.3775	0.3867	0.3957
189874	helena	0.7483	0.7624	**0.7475**	0.7478	0.7483	0.7534	0.7476	0.7575
167201	connect-4	0.2629	0.2721	0.2653	0.2630	**0.2537**	0.2565	0.3003	0.2771
189908	Fashion-MNIST	**0.1050**	0.1098	0.1217	0.1195	0.1181	0.1153	0.1437	0.1437
189860	APSFailure	0.0384	0.0402	0.0355	0.0364	0.0355	**0.0354**	0.0410	0.0455
168792	jannis	0.3654	0.3685	0.3648	0.3655	0.3651	**0.3567**	0.3912	0.3881
167083	numerai28.6	0.4776	0.4765	0.4752	0.4749	**0.4747**	0.4775	0.4789	0.4788
167200	higgs	0.2736	0.2764	0.2730	0.2742	**0.2724**	0.2744	0.2832	0.2844
168798	MiniBooNE	**0.0581**	0.0589	0.0691	0.0633	0.0585	0.0644	0.0691	0.0685
189873	dionis	**0.1172**	0.1205	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
189866	albert	**0.3135**	0.3171	0.3469	0.3277	0.5000	0.3354	0.3714	0.3703
75127	airlines	0.3423	0.3424	0.3450	**0.3384**	0.3419	0.3429	0.3408	0.3456
75193	covertype	0.0568	0.0564	0.0683	0.0600	**0.0548**	0.0556	0.2519	0.2527

Table 13: Results from Table 2 for 10 minutes using portfolios. We boldface the lowest error.

Auto-sklearn 2.0

Task ID	Name	holdout	SH; holdout	3CV	SH; 3CV	5CV	SH; 5CV	10CV	SH; 10CV
167104	Australian	0.1742	0.1674	0.1623	0.1626	0.1598	0.1608	0.1625	**0.1557**
167184	blood-transfusion	0.3648	**0.3618**	0.3631	0.3641	0.3689	0.3692	0.3684	0.3692
167168	vehicle	0.2125	0.2344	0.1702	0.1944	**0.1657**	0.1960	0.1959	0.2151
167161	credit-g	0.2922	**0.2895**	0.3035	0.2957	0.3056	0.2978	0.3008	0.2931
167185	cnae-9	0.0733	0.0761	0.0560	0.0616	0.0537	0.0536	0.0675	**0.0518**
189905	car	0.0036	0.0013	0.0037	0.0098	**0.0007**	0.0012	0.0008	0.0010
167152	mfeat-factors	0.0169	0.0186	0.0130	**0.0117**	0.0139	0.0132	0.0151	0.0122
167181	kc1	0.2728	0.2739	0.2680	0.2724	0.2678	0.2804	**0.2546**	0.2576
189906	segment	0.0708	0.0692	0.0647	0.0635	**0.0588**	0.0596	0.0621	0.0613
189862	jasmine	0.2048	0.2049	0.1995	0.1989	0.1995	**0.1976**	0.1995	0.1980
167149	kr-vs-kp	0.0060	0.0080	0.0081	0.0068	0.0064	0.0068	0.0055	**0.0053**
189865	sylvine	0.0590	0.0591	0.0584	0.0587	0.0577	0.0578	**0.0573**	0.0573
167190	phoneme	0.1222	0.1237	0.1152	0.1155	0.1111	0.1130	0.1117	**0.1105**
189861	christine	0.2673	0.2666	0.2575	0.2584	**0.2532**	0.2575	0.2549	0.2588
189872	fabert	0.3381	0.3319	0.3099	0.3097	0.3119	0.3080	**0.3027**	0.3071
189871	dilbert	0.0200	0.0200	**0.0132**	0.0146	0.0185	0.0209	0.0212	0.0149
168794	robert	0.5273	0.5199	0.5238	**0.5183**	0.5456	0.5605	0.5652	0.5407
168797	riccardo	0.0029	**0.0016**	0.0018	0.0019	0.0025	0.0076	0.5000	0.5000
168796	guillermo	**0.2012**	0.2025	0.2057	0.2081	0.2100	0.2039	0.5000	0.5000
75097	Amazon	0.2376	0.2394	0.2338	0.2381	0.2431	0.2384	**0.2312**	0.2324
126026	nomao	0.0352	0.0353	0.0334	0.0331	0.0320	0.0327	**0.0313**	0.0319
189909	jungle_chess	0.1214	0.1221	0.1154	0.1172	0.1171	0.1153	**0.1108**	0.1141
126029	bank-marketing	0.1388	0.1398	0.1380	0.1392	0.1382	0.1382	**0.1370**	0.1380
126025	adult	0.1546	0.1541	0.1550	0.1540	0.1550	0.1550	0.1539	**0.1538**
75105	KDDCup09	0.2492	**0.2461**	0.2477	0.2532	0.2466	0.2488	0.2617	0.2485
168795	shuttle	0.0136	0.0107	0.0125	0.0093	0.0084	**0.0063**	0.0127	0.0087
168793	volkert	0.3600	0.3673	**0.3449**	0.3551	0.3496	0.3487	0.3581	0.3563
189874	helena	0.7449	0.7494	**0.7331**	0.7369	0.7407	0.7404	0.7562	0.7452
167201	connect-4	0.2539	0.2556	0.2382	0.2428	0.2370	0.2373	0.2416	**0.2369**
189908	Fashion-MNIST	0.1010	**0.0971**	0.1046	0.1066	0.1102	0.1105	0.1191	0.1075
189860	APSFailure	0.0362	0.0374	0.0345	0.0364	0.0372	0.0347	0.0343	**0.0334**
168792	jannis	0.3670	0.3638	0.3589	0.3576	0.3584	0.3565	**0.3473**	0.3572
167083	numerai28.6	0.4765	0.4763	0.4774	0.4770	0.4750	0.4767	0.4755	**0.4743**
167200	higgs	0.2712	0.2734	0.2718	**0.2680**	0.2696	0.2680	0.2701	0.2683
168798	MiniBooNE	0.0576	0.0583	0.0560	**0.0536**	0.0571	0.0565	0.0560	0.0608
189873	dionis	**0.0961**	0.1068	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
189866	albert	0.3116	0.3168	0.3170	0.3172	**0.3094**	0.3199	0.3183	0.3186
75127	airlines	0.3403	0.3410	**0.3375**	0.3401	0.3388	0.3390	0.3398	0.3399
75193	covertype	0.0537	0.0519	0.0496	0.0496	**0.0454**	0.0461	0.0458	0.0459

Table 14: Results from Table 2 for 60 minutes using portfolios. We boldface the lowest error.

Task ID	Name	Auto-sklearn (2.0)	PoSH-Auto-sklearn	Auto-sklearn (1.0)
167104	Australian	0.1617	**0.1569**	0.1628
167184	blood-transfusion	0.3694	0.3610	**0.3534**
167168	vehicle	0.2030	0.2267	**0.1654**
167161	credit-g	0.2903	**0.2841**	0.2951
167185	cnae-9	**0.0635**	0.0680	0.0674
189905	car	**0.0015**	0.0049	0.0057
167152	mfeat-factors	**0.0123**	0.0164	0.0185
167181	kc1	0.2707	0.2688	**0.2301**
189906	segment	0.0646	0.0687	**0.0624**
189862	jasmine	0.2020	0.2051	**0.1989**
167149	kr-vs-kp	0.0086	**0.0077**	0.0089
189865	sylvine	0.0597	0.0594	**0.0583**
167190	phoneme	**0.1158**	0.1245	0.1257
189861	christine	**0.2562**	0.2621	0.2666
189872	fabert	**0.3250**	0.3399	0.3323
189871	dilbert	0.0240	0.0248	**0.0066**
168794	robert	**0.5861**	**0.5861**	0.6545
168797	riccardo	**0.0052**	**0.0052**	0.5000
168796	guillermo	**0.2102**	**0.2102**	0.5000
75097	Amazon	0.2435	**0.2431**	0.2610
126026	nomao	**0.0336**	0.0381	0.0383
189909	jungle_chess	**0.1205**	0.1251	0.1231
126029	bank-marketing	**0.1402**	0.1436	0.1412
126025	adult	**0.1547**	0.1575	0.1608
75105	KDDCup09	**0.2460**	0.2495	0.2863
168795	shuttle	**0.0084**	0.0086	0.0111
168793	volkert	**0.3717**	0.3724	0.4233
189874	helena	**0.7493**	0.7624	0.9157
167201	connect-4	**0.2642**	0.2721	0.2809
189908	Fashion-MNIST	**0.1070**	0.1098	0.1383
189860	APSFailure	0.0372	0.0402	**0.0370**
168792	jannis	0.3654	0.3685	**0.3637**
167083	numerai28.6	**0.4753**	0.4765	0.4774
167200	higgs	**0.2746**	0.2764	0.2777
168798	MiniBooNE	0.0603	**0.0589**	0.0622
189873	dionis	**0.1205**	**0.1205**	0.6731
189866	albert	**0.3171**	**0.3171**	0.4407
75127	airlines	**0.3404**	0.3424	0.3536
75193	covertype	**0.0564**	**0.0564**	0.8571

Table 15: Results from Table 5 for 10 minutes. We boldface the lowest error.

AUTO-SKLEARN 2.0

Task ID	Name	Auto-sklearn (2.0)	PoSH-Auto-sklearn	Auto-sklearn (1.0)
167104	Australian	**0.1562**	0.1674	0.1658
167184	blood-transfusion	0.3669	0.3618	**0.3572**
167168	vehicle	0.2187	0.2344	**0.1822**
167161	credit-g	0.2980	**0.2895**	0.3004
167185	cnae-9	**0.0566**	0.0761	0.0620
189905	car	0.0038	**0.0013**	0.0043
167152	mfeat-factors	**0.0126**	0.0186	0.0136
167181	kc1	0.2600	0.2739	**0.2250**
189906	segment	**0.0609**	0.0692	0.0697
189862	jasmine	**0.1971**	0.2049	0.1985
167149	kr-vs-kp	**0.0060**	0.0080	0.0085
189865	sylvine	0.0572	0.0591	**0.0555**
167190	phoneme	**0.1140**	0.1237	0.1235
189861	christine	**0.2592**	0.2666	0.2619
189872	fabert	**0.3120**	0.3319	0.3185
189871	dilbert	0.0163	0.0200	**0.0090**
168794	robert	**0.5199**	**0.5199**	0.5327
168797	riccardo	0.0016	0.0016	**0.0016**
168796	guillermo	0.2025	0.2025	**0.1964**
75097	Amazon	**0.2371**	0.2394	0.2481
126026	nomao	**0.0323**	0.0353	0.0361
189909	jungle_chess	0.1145	0.1221	**0.1136**
126029	bank-marketing	**0.1387**	0.1398	0.1428
126025	adult	0.1544	**0.1541**	0.1574
75105	KDDCup09	0.2504	**0.2461**	0.2549
168795	shuttle	**0.0093**	0.0107	0.0109
168793	volkert	0.3563	0.3673	**0.3440**
189874	helena	**0.7399**	0.7494	0.7693
167201	connect-4	**0.2408**	0.2556	0.2709
189908	Fashion-MNIST	0.1023	**0.0971**	0.0984
189860	APSFailure	**0.0343**	0.0374	0.0375
168792	jannis	**0.3591**	0.3638	0.3641
167083	numerai28.6	**0.4759**	0.4763	0.4760
167200	higgs	**0.2690**	0.2734	0.2738
168798	MiniBooNE	**0.0561**	0.0583	0.0620
189873	dionis	**0.1068**	**0.1068**	0.6731
189866	albert	0.3168	0.3168	**0.3143**
75127	airlines	**0.3394**	0.3410	0.3449
75193	covertype	**0.0519**	**0.0519**	0.8571

Table 16: Results from Table 5 for 60 minutes. We boldface the lowest error.

Appendix C. Theoretical properties of the greedy algorithm

C.1 Definitions

Definition 1 *(Discrete derivative, from Krause and Golovin, 2014) For a set function $f : 2^{\mathcal{V}} \to \mathbb{R}, \mathcal{S} \subseteq \mathcal{V}$ and $e \in \mathcal{V}$ let $\Delta_f(e|\mathcal{S}) = f(\mathcal{S} \cup \{e\}) - f(\mathcal{S})$ be the* discrete derivative *of f at S with respect to e.*

Definition 2 *(Submodularity, from Krause and Golovin, 2014): A function $f : 2^{\mathcal{V}} \to \mathbb{R}$ is* submodular *if for every $\mathcal{A} \subseteq \mathcal{B} \subseteq \mathcal{V}$ and $e \in \mathcal{V} \setminus \mathcal{B}$ it holds that $\Delta_f(e|\mathcal{A}) \geq \Delta_f(e|\mathcal{B})$.*

Definition 3 *(Monotonicity, from Krause and Golovin, 2014): A function $f : 2^{\mathcal{V}} \to \mathbb{R}$ is* monotone *if for every $\mathcal{A} \subseteq \mathcal{B} \subseteq V, f(\mathcal{A}) \leq f(\mathcal{B})$.*

C.2 Choosing on the test set

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

Proposition 2 *Minimizing the test loss of a portfolio $\mathcal{P}$ on a set of datasets $\mathcal{D}_1, \ldots, \mathcal{D}_{|\mathbf{D}_{meta}|}$, when choosing a ML pipeline from $\mathcal{P}$ for $\mathcal{D}_d$ based on performance on $\mathcal{D}_{d,test}$, is equivalent to the* sensor placement problem *for minimizing detection time (Krause et al., 2008).*

Following Krause et al. (Krause et al., 2008), sensor set placement aims at maximizing a so-called *penalty reduction* $R(\mathcal{A}) = \sum_{i \in \mathcal{I}} P(i) R(\mathcal{A}, i)$, where $\mathcal{I}$ are intrusion scenarios following a probability distribution P with i being a specific intrusion. $\mathcal{A} \subset \mathcal{C}$ is a *sensor placement*, a subset of all possible locations $\mathcal{C}$ where sensors are actually placed. Penalty reduction R is defined as the reduction of the penalty when choosing $\mathcal{A}$ compared to the maximum penalty possible on scenario i: $R(\mathcal{A}, i) = \text{penalty}_i(\infty) - \text{penalty}_i(T(\mathcal{A}, i))$. In the simplest case where action is taken upon intrusion detection, the penalty is equal to the detection time ($\text{penalty}_i(t) = t$). The detection time of a sensor placement $T(\mathcal{A}, i)$ is simply defined as the minimum of the detection times of its individual members: $\min_{s \in \mathcal{A}} T(s, i)$.

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

1. Intrusion scenarios $\mathcal{I}$: datasets $\{\mathcal{D}_1, \ldots, \mathcal{D}_{|\mathbf{D}_{\text{meta}}|}\}$,
2. Possible sensor locations $\mathcal{C}$: set of candidate ML pipelines of our algorithm $\mathcal{C}$, Detection time $T(s \in \mathcal{A}, i)$ on intrusion scenario i: test performance $\mathcal{L}(\mathcal{M}_C, \mathcal{D}_{d,\text{test}})$ on dataset $\mathcal{D}_d$,
3. Detection time of a sensor placement $T(\mathcal{A}, i)$: test loss of applying portfolio $\mathcal{P}$ on dataset $\mathcal{D}_d$: $\min_{p \in \mathcal{P}} \mathcal{L}(p, \mathcal{D}_{d,\text{test}})$
4. Penalty function $\text{penalty}_i(t)$: loss function $\mathcal{L}$, in our case, the penalty is equal to the loss.
5. Penalty reduction for an intrusion scenario $R(\mathcal{A}, i)$: the penalty reduction for successfully applying a portfolio $\mathcal{P}$ to dataset d: $R(\mathcal{P}, d) = \text{penalty}_d(\infty) - \min_{p \in \mathcal{P}} \mathcal{L}(p, \mathcal{D}_{d,\text{test}})$.[10]

■

10. This would be the general case for a metric with no upper bound. In case of metrics such as the misclassification error, the maximal penalty would be 1.

C.3 Choosing on the validation set

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

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

C.4 Successive Halving

As in the previous subsection, we use a simple example to demonstrate that selecting an algorithm via the successive halving model selection strategy is neither submodular nor monotone. To simplify notation we argue in terms of performance instead of penalty reduction, which is equivalent.

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

C.5 Further equalities

In addition, our problem can also be phrased as a *facility location* problem (Krarup and Pruzan, 1983) and statements about the facility location problem can be applied to our problem setup as well.

Appendix D. Implementation Details

D.1 Software

We implemented the AutoML systems and experiments in the Python3 programming language, using *numpy* (Harris et al., 2020), *scipy* (Virtanen et al., 2020), *scikit-learn* (Pedregosa et al., 2011), *pandas* (Wes McKinney, 2010; Reback et al., 2021), and *matplotlib* (Hunter, 2007). We used version 0.12.6 of the *Auto-sklearn* Python package for the experiments and added *Auto-sklearn 2.0* functionality in version 0.12.7 which we then used for the AutoML benchmark. We give the exact version numbers used for the AutoML benchmark in Table 17.

D.2 Configuration Space

We give the configuration space we use in *Auto-sklearn 2.0* in Table 18.

Package	Version
Auto-sklearn 2.0	0.12.7
Auto-sklearn 1.0	0.12.6
Auto-WEKA	2.6.3
TPOT	0.11.7
H2O AutoML	3.32.1.4
Tuned Random Forest	0.24.2
AutoML benchmark	973de79617e68a881dcc640842ea1d21dfd4b36c

Table 17: Package versions used for the AutoML benchmark.

D.3 Successive Halving hyperparameters

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

Appendix E. Datasets

We give the name, OpenML task ID and the size of all datasets we used in Table 19 and 20.

AUTO-SKLEARN 2.0

Name	Domain	Default	Log
Classifier	(Extra Trees, Gradient Boosting, MLP, Passive Aggressive, Random Forest, SGD)	Random Forest	-
Extra Trees: Bootstrap	(True, False)	False	-
Extra Trees: Criterion	(gini, entropy)	gini	-
Extra Trees: Max Features	$[0.0, 1.0]$	0.5	No
Extra Trees: Min Samples Leaf	$[1, 20]$	1	No
Extra Trees: Min Samples Split	$[2, 20]$	2	No
Gradient Boosting: Early Stopping	(off, valid, train)	off	-
Gradient Boosting: L2 Regularization	$[1e-10, 1.0]$	0.0	Yes
Gradient Boosting: Learning Rate	$[0.01, 1.0]$	0.1	Yes
Gradient Boosting: Max Leaf Nodes	$[3, 2047]$	31	Yes
Gradient Boosting: Min Samples Leaf	$[1, 200]$	20	Yes
Gradient Boosting: N Iter No Change	$[1, 20]$	10	No
Gradient Boosting: Validation Fraction	$[0.01, 0.4]$	0.1	No
MLP: Activation	(tanh, relu)	relu	-
MLP: Alpha	$[1e-07, 0.1]$	0.0001	Yes
MLP: Early Stopping	(valid, train)	valid	-
MLP: Hidden Layer Depth	$[1, 3]$	1	No
MLP: Learning Rate Init	$[0.0001, 0.5]$	0.001	Yes
MLP: Num Nodes Per Layer	$[16, 264]$	32	Yes
Passive Aggressive: C	$[1e-05, 10.0]$	1.0	Yes
Passive Aggressive: Average	(False, True)	False	-
Passive Aggressive: Loss	(hinge, squared_hinge)	hinge	-
Passive Aggressive: Tol	$[1e-05, 0.1]$	0.0001	Yes
Random Forest: Bootstrap	(True, False)	True	-
Random Forest: Criterion	(gini, entropy)	gini	-
Random Forest: Max Features	$[0.0, 1.0]$	0.5	No
Random Forest: Min Samples Leaf	$[1, 20]$	1	No
Random Forest: Min Samples Split	$[2, 20]$	2	No
Sgd: Alpha	$[1e-07, 0.1]$	0.0001	Yes
Sgd: Average	(False, True)	False	-
Sgd: Epsilon	$[1e-05, 0.1]$	0.0001	Yes
Sgd: Eta0	$[1e-07, 0.1]$	0.01	Yes
Sgd: L1 Ratio	$[1e-09, 1.0]$	0.15	Yes
Sgd: Learning Rate	(optimal, invscaling, constant)	invscaling	-
Sgd: Loss	(hinge, log, modified Huber, squared hinge, perceptron)	log	-
Sgd: Penalty	(l1, l2, elasticnet)	l2	-
Sgd: Power T	$[1e-05, 1.0]$	0.5	No
Sgd: Tol	$[1e-05, 0.1]$	0.0001	Yes
Balancing: Strategy	(none, weighting)	none	-
Categorical Encoding: Choice	(no encoding, one hot encoding)	one hot encoding	-
Category Coalescence: Choice	(minority coalescer, no coalescense)	minority coalescer	-
Category Coalescence: Minimum Fraction	$[0.0001, 0.5]$	0.01	Yes
Imputation of missing values	(mean, median, most frequent)	mean	-
Rescaling: Choice	(Min/Max, none, normalize, Power, Quantile, Robust, standardize)	standardize	-
Quantile Transformer: N Quantiles	$[10, 2000]$	1000	No
Quantile Transformer: Output Distribution	(uniform, normal)	uniform	-
Robust Scaler: Q Max	$[0.7, 0.999]$	0.75	No
Robust Scaler: Q Min	$[0.001, 0.3]$	0.25	No

Table 18: Configuration space for *Auto-sklearn 2.0* using only iterative models and only preprocessing to transform data into a format that can be usefully employed by the different classification algorithms. The final column (log) states whether we actually search $log_{10}(\lambda)$.

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

name	tid	# obs	# feat	# class
satim...	2120	6430	37	6
Satel...	189844	5100	37	2
soybe...	271	683	36	19
cardi...	75217	2126	36	10
cjs	146601	2796	35	6
colle...	75212	1302	35	2
puma3...	75153	8192	33	2
Gestu...	75109	9873	33	5
kick	189870	72983	33	2
bank3...	75179	8192	33	2
wdbc	146596	569	31	2
Phish...	75215	11055	31	2
fars	189840	100968	30	8
hypot...	3044	3772	30	4
steel...	168785	1941	28	7
eye_m...	189779	10936	28	3
fri_c...	75136	1000	26	2
fri_c...	75199	1000	26	2
wall-...	75235	5456	25	4
led24...	189841	3200	25	10
colli...	189845	1000	24	30
rl	189869	31406	23	2
mushr...	254	8124	23	2
meta	166875	528	22	2
jm1	75093	10885	22	2
pc1	75159	1109	22	2
kc2	146583	522	22	2
cpu_a...	75233	8192	22	2
autoU...	75089	1000	21	2
GAMET...	167086	1600	21	2
GAMET...	167087	1600	21	2
bosto...	166905	506	21	2
GAMET...	167088	1600	21	2
GAMET...	167089	1600	21	2
churn...	167097	5000	21	2
clima...	167106	540	21	2
micro...	189875	20000	21	5
GAMET...	167090	1600	21	2
Traff...	211724	70340	21	3
ringn...	75234	7400	21	2
twono...	75187	7400	21	2
eucal...	2125	736	20	5
eleva...	75184	16599	19	2
pbcse...	166897	1945	19	2
baseb...	2123	1340	18	3
house...	75174	22784	17	2
colle...	75196	1161	17	2
BachC...	189829	5665	17	102
pendi...	262	10992	17	10
lette...	236	20000	17	26
spoke...	75178	263256	15	10
eeg-e...	75219	14980	15	2
wind	75185	6574	15	2
Japan...	126021	9961	15	9
compa...	211721	5278	14	2
vowel...	3047	990	13	11
cpu_s...	75147	8192	13	2
autoU...	189900	700	13	3
autoU...	75118	1100	13	5
dress...	146602	500	13	2
senso...	166906	576	12	2
wine-...	189836	4898	12	7
wine-...	189843	1599	12	6
Magic...	75112	19020	12	2
mv	75195	40768	11	2
parit...	167101	1124	11	2
mofn-...	167094	1324	11	2
fri_c...	75149	1000	11	2
poker...	340	829201	11	10

name	tid	#obs	#feat	#cls
fri_c...	166950	500	11	2
page-...	260	5473	11	5
ilpd	146593	583	11	2
2dpla...	75142	40768	11	2
fried...	75161	40768	11	2
rmfts...	166859	508	11	2
stock...	166915	950	10	2
tic-t...	279	958	10	2
breas...	245	699	10	2
xd6	167096	973	10	2
cmc	253	1473	10	3
profb...	146578	672	10	2
diabe...	267	768	9	2
abalo...	2121	4177	9	28
bank8...	75141	8192	9	2
elect...	336	45312	9	2
kdd_e...	166913	782	9	2
house...	75176	20640	9	2
nurse...	256	12960	9	5
kin8n...	75166	8192	9	2
yeast...	2119	1484	9	10
puma8...	75171	8192	9	2
analc...	75143	4052	8	2
ldpa	75134	164860	8	11
pm10	166872	500	8	2
no2	166932	500	8	2
LED-d...	146603	500	8	10
artif...	126028	10218	8	10
monks...	3055	554	7	2
space...	75148	3107	7	2
kr-vs...	75223	28056	7	18
monks...	3054	601	7	2
Run_o...	167103	88588	7	2
delta...	75173	9517	7	2
strik...	166882	625	7	2
mammo...	3048	11183	7	2
monks...	3053	556	7	2
kropt...	2122	28056	7	18
delta...	75163	7129	6	2
wilt	167105	4839	6	2
fri_c...	75131	1000	6	2
mozil...	126024	15545	6	2
polle...	75192	3848	6	2
socmo...	75213	1156	6	2
irish...	146575	500	6	2
fri_c...	166931	500	6	2
arsen...	166957	559	5	2
arsen...	166956	559	5	2
walki...	75250	149332	5	22
analc...	146577	797	5	6
bankn...	146586	1372	5	2
arsen...	166959	559	5	2
visua...	75210	8641	5	2
balan...	241	625	5	3
arsen...	166958	559	5	2
volca...	189902	10130	4	5
skin-...	75237	245057	4	2
tamil...	189846	45781	4	20
quake...	75157	2178	4	2
volca...	189893	8654	4	5
volca...	189890	8753	4	5
volca...	189887	9989	4	5
volca...	189884	10668	4	5
volca...	189883	10176	4	5
volca...	189882	1515	4	5
volca...	189881	1521	4	5
volca...	189880	1623	4	5
Titan...	167099	2201	4	2
volca...	189894	1183	4	5

Table 19: Characteristics of the 208 datasets in $\mathbf{D}_{\text{meta}}$ (first part) sorted by number of features. We report for each dataset the name and the task id (as a link) as used on OpenML.org, and furthermore the number of observations, the number of features and the number of classes.

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

Table 20: Characteristics of the 39 datasets in $\mathbf{D}_{\text{test}}$ sorted by number of features. We report for each dataset the name and the task id (as a link) as used on OpenML.org, and furthermore the number of observations, the number of features and the number of classes.

D

APPENDIX FOR OPENML BENCHMARKING SUITES

A OpenML-CC18 dataset list

Table 1: Datasets included in the OpenML-CC18 benchmarking suite. For each dataset, we show: the OpenML task id, dataset id and name, the number of classes (cl), features (p) and observations (n), as well as the ratio of the minority and majority class sizes (MinMaj).

Data id	Task id	Name	cl	p	n	MinMaj
3	3	kr-vs-kp	2	37	3196	0.91
6	6	letter	26	17	20000	0.90
11	11	balance-scale	3	5	625	0.17
12	12	mfeat-factors	10	217	2000	1.00
14	14	mfeat-fourier	10	77	2000	1.00
15	15	breast-w	2	10	699	0.53
16	16	mfeat-karhunen	10	65	2000	1.00
18	18	mfeat-morphological	10	7	2000	1.00
22	22	mfeat-zernike	10	48	2000	1.00
23	23	cmc	3	10	1473	0.53
28	28	optdigits	10	65	5620	0.97
29	29	credit-approval	2	16	690	0.80
31	31	credit-g	2	21	1000	0.43
32	32	pendigits	10	17	10992	0.92
37	37	diabetes	2	9	768	0.54
38	3021	sick	2	30	3772	0.07
44	43	spambase	2	58	4601	0.65
46	45	splice	3	62	3190	0.46
50	49	tic-tac-toe	2	10	958	0.53
54	53	vehicle	4	19	846	0.91
151	219	electricity	2	9	45312	0.74
182	2074	satimage	6	37	6430	0.41
188	2079	eucalyptus	5	20	736	0.49
300	3481	isolet	26	618	7797	0.99
307	3022	vowel	11	13	990	1.00
458	3549	analcatdata_authorship	4	71	841	0.17
469	3560	analcatdata_dmft	6	5	797	0.79
554	3573	mnist_784	10	785	70000	0.80
1049	3902	pc4	2	38	1458	0.14
1050	3903	pc3	2	38	1563	0.11
1053	3904	jm1	2	22	10885	0.24
1063	3913	kc2	2	22	522	0.26
1067	3917	kc1	2	22	2109	0.18
1068	3918	pc1	2	22	1109	0.07
1461	14965	bank-marketing	2	17	45211	0.13
1462	10093	banknote-authentication	2	5	1372	0.80
1464	10101	blood-transfusion-service-center	2	5	748	0.31
1468	9981	cnae-9	9	857	1080	1.00
1475	9985	first-order-theorem-proving	6	52	6118	0.19
1478	14970	har	6	562	10299	0.72
1480	9971	ilpd	2	11	583	0.40
1485	9976	madelon	2	501	2600	1.00
1486	9977	nomao	2	119	34465	0.40
1487	9978	ozone-level-8hr	2	73	2534	0.07
1489	9952	phoneme	2	6	5404	0.42
1494	9957	qsar-biodeg	2	42	1055	0.51
1497	9960	wall-robot-navigation	4	25	5456	0.15
1501	9964	semeion	10	257	1593	0.96
1510	9946	wdbc	2	31	569	0.59
1590	7592	adult	2	15	48842	0.31
4134	9910	Bioresponse	2	1777	3751	0.84
4534	14952	PhishingWebsites	2	31	11055	0.80
4538	14969	GesturePhaseSegmentationProcessed	5	33	9873	0.34
6332	14954	cylinder-bands	2	40	540	0.73
23381	125920	dresses-sales	2	13	500	0.72
23517	167120	numerai28.6	2	22	96320	0.98
40499	125922	texture	11	41	5500	1.00
40668	146195	connect-4	3	43	67557	0.15
40670	167140	dna	3	181	3186	0.46
40701	167141	churn	2	21	5000	0.16
40923	167121	Devnagari-Script	46	1025	92000	1.00
40927	167124	CIFAR_10	10	3073	60000	1.00
40966	146800	MiceProtein	8	82	1080	0.70
40975	146821	car	4	7	1728	0.05
40978	167125	Internet-Advertisements	2	1559	3279	0.16
40979	146824	mfeat-pixel	10	241	2000	1.00
40982	146817	steel-plates-fault	7	28	1941	0.08
40983	146820	wilt	2	6	4839	0.06
40984	146822	segment	7	20	2310	1.00
40994	146819	climate-model-simulation-crashes	2	21	540	0.09
40996	146825	Fashion-MNIST	10	785	70000	1.00
41027	167119	jungle_chess_2pcs_raw_endgame_complete	3	7	44819	0.19

B Useful links

We now collect all relevant links in a single place to simplify access to online material on OpenML benchmarking studies:

- General online documentation: `https://docs.openml.org`
- Online documentation on benchmarking suites: `https://docs.openml.org/benchmark`
- Github repository with additional material, including a notebook to create updated suites: `https://github.com/openml/benchmark-suites`
- Github organization for OpenML.org: `https://github.com/openml`
- Python package: *OpenML* (PyPI)
- R package: *OpenML* (CRAN)
- Java package: *org.openml.openmlweka* (Maven Central)

C Suggested curation protocol

In this section we give an exemplary curation protocol for constructing new benchmarking suites. It is based on our experience constructing the OpenML-CC18 and its predecessor, the OpenML100. Steps can be removed or added depending on the desired benchmark purpose, the steps below serve as a guideline.

1. Steps that can be automated:

(a) Specify the OpenML task type, for example supervised classification or supervised regression.
(b) Specify criteria on dataset properties, such as the size of the dataset, the number of features or the number of classes.
(c) Specify criteria on the data modalities that are supposed to be in the data. Currently, OpenML supports numerical, categorical, date and string.
(d) Specify whether the data should be sparse or not.
(e) Specify whether the data should contain missing values or not.
(f) Check whether tasks are too easy, either by querying for existing results on OpenML or by running machine learning algorithms locally.

2. Steps that cannot be automated and should be performed on the outcome of the previous, automated steps. For our benchmark the following manual steps were added:
   (a) Check for artificial datasets.
   (b) Check for dataset that require grouped or time-aware splitting.
   (c) Check for datasets that are subsets of larger datasets (or binarized datasets in case of classification).
   (d) Check for other forms of derived datasets, for example versions that do no longer contain feature names or only a subset of features.
   (e) Check that all remaining datasets feature a reference.

BIBLIOGRAPHY

Aguilar Melgar, L., D. Dao, S. Gan, N. Gürel, N. Hollenstein, J. Jiang, B. Karlaš, T. Lemmin, T. Li, Y. Li, S. Rao, J. Rausch, C. Renggli, L. Rimanic, M. Weber, S. Zhang, Z. Zhao, K. Schawinski, W. Wu, and C. Zhang (2021). "Ease. ML: A Lifecycle Management System for Machine Learning." In: *Proceedings of the Annual Conference on Innovative Data Systems Research (CIDR), 2021*. CIDR.

Alaa, A. and M. van der Schaar (2018). "AutoPrognosis: Automated Clinical Prognostic Modeling via Bayesian Optimization with Structured Kernel Learning." In: *Proceedings of the 35th International Conference on Machine Learning (ICML'18)*. Ed. by J. Dy and A. Krause. Vol. 80. Proceedings of Machine Learning Research, pp. 139–148.

Alpaydin, E. (2010). *Introduction to machine learning*. 2nd ed. MIT press.

Ambrosen, K., M. Skjerbæk, J Foldager, M. Axelsen, N Bak, L. Arvastson, S. Christensen, L. Johansen, J. Raghava, B. Oranje, E Rostrup, M. Nielsen, M. Osler, B. Fagerlund, C. Pantelis, B. Kinon, B. Glenthøj, L. Hansen, and B Ebdrup (2020). "A machine-learning framework for robust and reliable prediction of short- and long-term treatment response in initially antipsychotic-naïve schizophrenia patients based on multimodal neuropsychiatric data." In: *Translational Psychiatry* 10.276.

Aroyo, L. and C. Welty (2015). "Truth Is a Lie: Crowd Truth and the Seven Myths of Human Annotation." In: *AI Magazine* 36.1, pp. 15–24.

Awad, N., N. Mallik, and F. Hutter (2021). "DEHB: Evolutionary Hyberband for Scalable, Robust and Efficient Hyperparameter Optimization." In: *Proceedings of the 30th International Joint Conference on Artificial Intelligence, IJCAI'21*. Ed. by Z. Zhou. ijcai.org, pp. 2147–2153.

Baudart, G., M. Hirzel, K. Kate, P. Ram, A. Shinnar, and J. Tsay (2021). "Pipeline Combinators for Gradual AutoML." In: *Proceedings of the 34th International Conference on Advances in Neural Information Processing Systems (NeurIPS'21)*. Ed. by M. Ranzato, A. Beygelzimer, K. Nguyen, P. Liang, J. Vaughan, and Y. Dauphin. Curran Associates.

Bello, I., W. Fedus, X. Du, E. Cubuk, A. Srinivas, T.-Y. Lin, J. Shlens, and B. Zoph (2021). "Revisiting ResNets: Improved Training and Scaling Strategies." In: *Proceedings of the 34th International Conference on Advances in Neural Information Processing Systems (NeurIPS'21)*. Ed. by M. Ranzato, A. Beygelzimer, K. Nguyen, P. Liang, J. Vaughan, and Y. Dauphin. Curran Associates.

Bergstra, J., R. Bardenet, Y. Bengio, and B. Kégl (2011). "Algorithms for Hyper-Parameter Optimization." In: *Proceedings of the 24th Interna-*

tional Conference on Advances in Neural Information Processing Systems (NeurIPS'11). Ed. by J. Shawe-Taylor, R. Zemel, P. Bartlett, F. Pereira, and K. Weinberger. Curran Associates, pp. 2546–2554.

Bergstra, J. and Y. Bengio (2012). "Random Search for Hyper-Parameter Optimization." In: *Journal of Machine Learning Research* 13, pp. 281–305.

Bergstra, J., N. Pinto, and D. Cox (2015). "SkData: data sets and algorithm evaluation protocols in Python." In: *Computational Science & Discovery* 8.1.

Biedenkapp, A., M. Lindauer, K. Eggensperger, C. Fawcett, H. Hoos, and F. Hutter (2017). "Efficient Parameter Importance Analysis via Ablation with Surrogates." In: *Proceedings of the Thirty-First Conference on Artificial Intelligence (AAAI'17)*. Ed. by S.Singh and S. Markovitch. AAAI Press, pp. 773–779.

Biedenkapp, A., J. Marben, M. Lindauer, and F. Hutter (2018). "CAVE: Configuration Assessment, Visualization and Evaluation." In: *Proceedings of the International Conference on Learning and Intelligent Optimization (LION)*. Ed. by R. Battiti, M. Brunato, I. Kotsireas, and P. Pardalos. Lecture Notes in Computer Science. Springer.

Bishop, C. (2006). *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer.

Brazdil, P., J. Gama, and B. Henery (1994). "Characterizing the Applicability of Classification Algorithms Using Meta-Level Learning." In: *Machine Learning: ECML-94*. Ed. by F. Bergadano and L. De Raedt. Lecture Notes in Artificial Intelligence. Springer, pp. 83–102.

Brazdil, P., C. Giraud-Carrier, C. Soares, and R. Vilalta (2008). *Metalearning: Applications to Data Mining*. 1st ed. Springer.

Brochu, E., V. Cora, and N. de Freitas (2010). "A Tutorial on Bayesian Optimization of Expensive Cost Functions, with Application to Active User Modeling and Hierarchical Reinforcement Learning." In: *arXiv:1012.2599v1 [cs.LG]*.

Buitinck, L., G. Louppe, M. Blondel, F. Pedregosa, A. Müller, O. Grisel, V. Niculae, P. Prettenhofer, A. Gramfort, J. Grobler, R. Layton, J. Vanderplas, A. Joly, B. Holt, and G. Varoquaux (2013). "API design for machine learning software: experiences from the scikit-learn project." In: *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, pp. 108–122.

Bürger, F. and J. Pauli (2015). "A Holistic Classification Optimization Framework with Feature Selection, Preprocessing, Manifold Learning and Classifiers." In: *Procceedings of 4th International Conference on Pattern Recognition: Applications and Methods (ICPRAM'15)*. Ed. by A. Fred, M. De Marsico, and M. Figueiredo. Vol. 9493. Lecture Notes in Computer Science. Springer, pp. 52–68.

Caruana, R. (2015). *Research Opportunities in AutoML*. Presentation at the 2nd AutoML workshop @ IMCL. URL: https://indico.ijclab.

in2p3.fr/event/2914/contributions/6481/attachments/6048/7173/CaruanaAutoMLWorkshopICML2015rev4.pdf.

Caruana, R., A. Niculescu-Mizil, G. Crew, and A. Ksikes (2004). "Ensemble Selection from Libraries of Models." In: *Proceedings of the 21st International Conference on Machine Learning (ICML'04)*. Ed. by R. Greiner. Omnipress.

Casalicchio, G., J. Bossek, M. Lang, D. Kirchhoff, P. Kerschke, B. Hofner, H. Seibold, J. Vanschoren, and B. Bischl (2017). "OpenML: An R package to connect to the machine learning platform OpenML." In: *Computational Statistics* 32.3.

Chakraborty, J., T. Xia, F. Fahid, and T. Menzies (2019). "Software Engineering for Fairness: A Case Study with Hyperparameter Optimization." In: *Late Breaking Results Track of the 34th IEEE/ACM International Conference on Automated Software Engineering*.

Chang, Chih-Chung and Chih-Jen Lin (2011). "LIBSVM: A Library for Support Vector Machines." In: *ACM Transactions on Intelligent Systems and Technology* 2.3, pp. 1–27.

Chen, B., H. Wu, W. Mo, I. Chattopadhyay, and H. Lipson (2018). "Autostacker: A Compositional Evolutionary Learning System." In: *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'18)*. Ed. by K. Takadama H. Aguirre. ACM, pp. 402–409.

Chen, T. and C. Guestrin (2016). "XGBoost: A Scalable Tree Boosting System." In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*. Ed. by B. Krishnapuram, M. Shah, A. Smola, C. Aggarwal, D. Shen, and R. Rastogi. ACM Press, pp. 785–794.

Chen, Y., A. Huang, Z. Wang, I. Antonoglou, J. Schrittwieser, D. Silver, and N. de Freitas (2018). "Bayesian Optimization in AlphaGo." In: *arXiv:1812.06855 [cs.LG]*.

Chinthanet, B., B. Reid, C. Treude, M. Wagner, R. Kula, T. Ishio, and K. Matsumoto (2021). "What makes a good Node.js package? Investigating Users, Contributors, and Runnability." In: *arXiv:2106.12239 [cs.SE]*.

Clune, J. (2019). "AI-GAs: AI-generating algorithms, an alternate paradigm for producing general artificial intelligence." In: *arXiv: 1905.10985 [cs.AI]*.

Crisan, A. and B. Fiore-Gartland (2021). "Fits and Starts: Enterprise Use of AutoML and the Role of Humans in the Loop." In: *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*. CHI '21 601. Association for Computing Machinery.

Dalal, N. and B. Triggs (2005). "Histograms of oriented gradients for human detection." In: *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*. Vol. 1, pp. 886–893.

DARPA (2022). *Data-Driven Discovery of Models (D3M)*. Retrieved on January 28, 2022. URL: https://datadrivendiscovery.org/.

de Sa, A., W. Pinto, L. Oliveira, and G. Pappa (2017). "RECIPE: A Grammar-based Framework for Automatically Evolving Classification Pipelines." In: *Proceedings of the 20th European Conference on Genetic Programming (EuroGP'17)*. Ed. by M. Castelli, J. McDermott, and L. Sekanina. Vol. 10196. Lecture Notes in Computer Science. Springer, pp. 246–261.

Denkena, B., M.-A. Dittrich, M. Lindauer, J. Mainka, and L. Stürenburg (2020). "Using AutoML to Optimize Shape Error Prediction in Milling Processes." In: *Proceedings of the Machining Innovations Conference (MIC) 2020*, pp. 160–165.

Ding, F., M. Hardt, J. Miller, and L. Schmidt (2021). "Retiring Adult: New Datasets for Fair Machine Learning." In: *Proceedings of the 34th International Conference on Advances in Neural Information Processing Systems (NeurIPS'21)*. Ed. by M. Ranzato, A. Beygelzimer, K. Nguyen, P. Liang, J. Vaughan, and Y. Dauphin. Curran Associates.

Domhan, T., J. Springenberg, and F. Hutter (2015). "Speeding Up Automatic Hyperparameter Optimization of Deep Neural Networks by Extrapolation of Learning Curves." In: *Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI'15)*. Ed. by Q. Yang and M. Wooldridge, pp. 3460–3468.

Dong, J.-D., A.-C. Cheng, D.-C. Juan, W. Wei, and M. Sun (2018). "PPP-Net: Platform-aware Progressive Search for Pareto-optimal Neural Architectures." In: *International Conference on Learning Representations Workshop track*. Published online: iclr.cc.

Drori, I., Y. Krishnamurthy, R. Lourenco, R. Rampin, K. Cho, C. Silva, and J. Freire (2019). "Automatic Machine Learning by Pipeline Synthesis using Model-Based Reinforcement Learning and a Grammar." In: *ICML workshop on Automated Machine Learning (AutoML workshop 2019)*. Ed. by K. Eggensperger, M. Feurer, F. Hutter, and J. Vanschoren.

Drori, I., Y. Krishnamurthy, R. Rampin, R. Lourenco, J. One, K. Cho, C. Silva, and J. Freire (2018). "AlphaD3M: Machine Learning Pipeline Synthesis." In: *ICML workshop on Automated Machine Learning (AutoML workshop 2018)*. Ed. by R. Garnett, F. Hutter J. Vanschoren, P. Brazdil, R. Caruana, C. Giraud-Carrier, I. Guyon, and B. Kégl.

Drozdal, J., J. Weisz, D. Wang, G. Dass, B. Yao, C. Zhao, M. Muller, L. Ju, and H. Su (2020). "Trust in AutoML: Exploring Information Needs for Establishing Trust in Automated Machine Learning Systems." In: *Proceedings of the 25th International Conference on Intelligent User Interfaces*. IUI '20. Association for Computing Machinery, pp. 297–307.

Drummond, C. (2006). "Machine learning as an experimental science (revisited)." In: *AAAI workshop on evaluation methods for machine learning*.

Dua, D. and C. Graff (2019). *UCI Machine Learning Repository*. URL: http://archive.ics.uci.edu/ml.

Eggensperger, K., F. Hutter, H. Hoos, and K. Leyton-Brown (2015). "Efficient Benchmarking of Hyperparameter Optimizers via Surrogates." In: *Proceedings of the Twenty-ninth National Conference on Artificial Intelligence (AAAI'15)*. Ed. by B. Bonet and S. Koenig. AAAI Press, pp. 1114–1120.

Eggensperger, K., P. Müller, N. Mallik, M. Feurer, R. Sass, A. Klein, N. Awad, M. Lindauer, and F. Hutter (2021). "HPOBench: A Collection of Reproducible Multi-Fidelity Benchmark Problems for HPO." In: *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*. Ed. by J. Vanschoren, S. Yeung, and M. Xenochristou. Curran Associates.

Elliott, T. (2019). *The State of the Octoverse: machine learning*. Retrieved: January 14, 2022. URL: https://github.blog/2019-01-24-the-state-of-the-octoverse-machine-learning/.

Elsken, T., J. Metzen, and F. Hutter (2019a). "Efficient Multi-Objective Neural Architecture Search via Lamarckian Evolution." In: *Proceedings of the International Conference on Learning Representations (ICLR'19)*. Published online: iclr.cc.

– (2019b). "Neural Architecture Search." In: *Automated Machine Learning: Methods, Systems, Challenges*. Ed. by F. Hutter, L. Kotthoff, and J. Vanschoren. Vol. 5. The Springer Series on Challenges in Machine Learning. Available for free at http://automl.org/book. Springer, pp. 63–77.

Erickson, N., J. Mueller, A. Shirkov, H. Zhang, P. Larroy, M. Li, and A. Smola (2020). "AutoGluon-Tabular: Robust and Accurate AutoML for Structured Data." In: *arXiv:2003.06505 [stat.ML]*.

Escalante, H. (2021). "Automated Machine Learning—A Brief Review at the End of the Early Years." In: *Automated Design of Machine Learning and Search Algorithms*. Ed. by N. Pillay and R. Qu. Springer, pp. 11–28.

Escalante, H., M. Montes, and E. Sucar (2009). "Particle Swarm Model Selection." In: *Journal of Machine Learning Research* 10, pp. 405–440.

– (2010). "Ensemble particle swarm model selection." In: *Proceedings of the 2010 IEEE International Joint Conference on Neural Networks (IJCNN)*. IEEE Computer Society Press, pp. 1–8.

Fawcett, C. and H. Hoos (2016). "Analysing differences between algorithm configurations through ablation." In: *Journal of Heuristics* 22.4, pp. 431–458.

Fernández-Delgado, M., E. Cernadas, S. Barro, and D. Amorim (2014). "Do we Need Hundreds of Classifiers to Solve Real World Classification Problems?" In: *Journal of Machine Learning Research* 15.90, pp. 3133–3181.

Feurer, M., B. Letham, and E. Bakshy (2018). "Scalable Meta-Learning for Bayesian Optimization." In: *arXiv:1802.02219v1 [stat.ML]*.

Feurer, M., J. Springenberg, and F. Hutter (2015). "Initializing Bayesian Hyperparameter Optimization via Meta-Learning." In: *Proceedings of*

the Twenty-ninth National Conference on Artificial Intelligence (AAAI'15). Ed. by B. Bonet and S. Koenig. AAAI Press, pp. 1128–1135.

Frazier, P. (2018). "A Tutorial on Bayesian Optimization." In: *arXiv: 1807.02811 [stat.ML]*.

Fusi, N., R. Sheth, and M. Elibol (2018). "Probabilistic Matrix Factorization for Automated Machine Learning." In: *Proceedings of the 31st International Conference on Advances in Neural Information Processing Systems (NeurIPS'18)*. Ed. by S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett. Curran Associates, pp. 3348–3357.

Gama, J. and P. Brazdil (1995). "Characterization of Classification Algorithms." In: *Proceedings of the 7th Portuguese Conference on Artificial Intelligence (EPAI'95)*. Ed. by C. Pinto-Ferreira and N. Mamede. Vol. 990. Lecture Notes in Computer Science. Springer, pp. 189–200.

Gardner, J., M. Kusner, Z. Xu, K. Weinberger, and J. Cunningham (2014). "Bayesian Optimization with Inequality Constraints." In: *Proceedings of the 31th International Conference on Machine Learning, (ICML'14)*. Ed. by E. Xing and T. Jebara. Omnipress, pp. 937–945.

Garnett, R. (2022). *Bayesian Optimization*. in preparation. Cambridge University Press.

Gelbart, M., J. Snoek, and R. Adams (2014). "Bayesian Optimization with Unknown Constraints." In: *Proceedings of the 30th conference on Uncertainty in Artificial Intelligence (UAI'14)*. Ed. by N. Zhang and J. Tian. AUAI Press, pp. 250–258.

Gemein, L., R. Schirrmeister, P. Chrabąszcz, D. Wilson, J. Boedecker, A. Schulze-Bonhage, F. Hutter, and T. Ball (2020). "Machine-learning-based diagnostics of EEG pathology." In: *NeuroImage* 220.117021.

Gijsbers, P., E. LeDell, S. Poirier, J. Thomas, B. Bischl, and J. Vanschoren (2019). "An Open Source AutoML Benchmark." In: *ICML workshop on Automated Machine Learning (AutoML workshop 2019)*. Ed. by K. Eggensperger, M. Feurer, F. Hutter, and J. Vanschoren.

Golovin, D., B. Solnik, S. Moitra, G. Kochanski, J. Karro, and D. Sculley (2017). "Google Vizier: A Service for Black-Box Optimization." In: *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*. Ed. by S. Matwin, S. Yu, and F. Farooq. ACM Press, pp. 1487–1495.

Gomes, T., R. Prudêncio, C. Soares, A. Rossi, and A. Carvalho (2012). "Combining meta-learning and search techniques to select parameters for support vector machines." In: *Neurocomputing* 75.1, pp. 3–12.

Goodfellow, I., Y. Bengio, and A. Courville (2016). *Deep Learning*. MIT Press.

Gurukar, S., D. Ajwani, S. Dutta, J. Lauri, S. Parthasarathy, and A. Sala (2020). "Towards Quantifying the Distance between Opinions." In: *Proceedings of the International AAAI Conference on Web and Social Media* 14.1, pp. 229–239.

Guyon, I., L. Sun-Hosoya, M. Boullé, H. Escalante, S. Escalera, Z. Liu, D. Jajetic, B. Ray, M. Saeed, M. Sebag, A. Statnikov, W. Tu, and E. Viegas (2019). "Analysis of the AutoML Challenge Series 2015-2018." In: *Automated Machine Learning: Methods, Systems, Challenges*. Ed. by F. Hutter, L. Kotthoff, and J. Vanschoren. Vol. 5. The Springer Series on Challenges in Machine Learning. Available for free at http://automl.org/book. Springer. Chap. 10, pp. 177–219.

Hansen, N. (2006). "The CMA evolution strategy: a comparing review." In: *Towards a new evolutionary computation. Advances on estimation of distribution algorithms*. Ed. by J. Lozano, P. Larranaga, I. Inza, and E. Bengoetxea. Springer, pp. 75–102.

Hastie, T., R. Tibshirani, and J. Friedman (2001). *The Elements of Statistical Learning*. Springer.

He, K., X. Zhang, S. Ren, and J. Sun (2016). "Deep Residual Learning for Image Recognition." In: *Proceedings of the International Conference on Computer Vision and Pattern Recognition (CVPR'16)*. IEEE Computer Society Press.

Heffetz, Y., R. Vainshtein, G. Katz, and L. Rokach (2020). "DeepLine: AutoML Tool for Pipelines Generation Using Deep Reinforcement Learning and Hierarchical Actions Filtering." In: *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'20)*. Ed. by J. Tang and B. Prakash. ACM Press, pp. 2103–2113.

Hoffman, M., B. Shahriari, and N. de Freitas (2014). "On correlation and budget constraints in model-based bandit optimization with application to automatic machine learning." In: *Proceedings of the Seventeenth International Conference on Artificial Intelligence and Statistics (AISTATS)*. Ed. by S. Kaski and J. Corander. Vol. 33. Proceedings of Machine Learning Research, pp. 365–374.

Hoos, H. (2012). "Programming by optimization." In: *Communications of the ACM* 55.2, pp. 70–80.

Hsu, C.-H., S.-H. Chang, J.-H. Liang, H.-P. Chou, C.-H. Liu, S.-C. Chang, J.-Y. Pan, Y.-T. Chen, and D.-C. Juan W. Wei (2018). "MONAS: Multi-Objective Neural Architecture Search using Reinforcement Learning." In: *arXiv:1806.10332*.

Huo, Y., G. Prasad, L. Lampe, and C. Victor Leung (2019). "Smart-Grid Monitoring: Enhanced Machine Learning for Cable Diagnostics." In: *2019 IEEE International Symposium on Power Line Communications and its Applications (ISPLC)*, pp. 1–6.

Hutter, F., H. Hoos, and K. Leyton-Brown (2014). "An Efficient Approach for Assessing Hyperparameter Importance." In: *Proceedings of the 31th International Conference on Machine Learning, (ICML'14)*. Ed. by E. Xing and T. Jebara. Omnipress, pp. 754–762.

Hutter, F., L. Kotthoff, and J. Vanschoren, eds. (2019). *Automated Machine Learning: Methods, Systems, Challenges*. Vol. 5. The Springer

Series on Challenges in Machine Learning. Available for free at http://automl.org/book. Springer.

Hvarfner, C., D. Stoll, A. Souza, L. Nardi, M. Lindauer, and F. Hutter (2022). "πBO: Augmenting Acquisition Functions with User Beliefs for Bayesian Optimization." In: *International Conference on Learning Representations*.

Jamieson, K. and A. Talwalkar (2016). "Non-stochastic Best Arm Identification and Hyperparameter Optimization." In: *Proceedings of the Seventeenth International Conference on Artificial Intelligence and Statistics (AISTATS)*. Ed. by A. Gretton and C. Robert. Vol. 51. Proceedings of Machine Learning Research.

Jin, H., Q. Song, and X. Hu (2019). "Auto-Keras: An Efficient Neural Architecture Search System." In: *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD'19*. Ed. by A. Teredesai, V. Kumar, Y. Li, R. Rosales, E. Terzi, and G. Karypis. ACM Press, pp. 1946–1956.

Jones, D., M. Schonlau, and W. Welch (1998). "Efficient Global Optimization of Expensive Black Box Functions." In: *Journal of Global Optimization* 13, pp. 455–492.

Jumper, J. et al. (2021). "Highly accurate protein structure prediction with AlphaFold." In: *Nature* 596.7873, pp. 583–589.

Jurafsky, D. and J. Martin (2022). *Speech and Language Processing*. 3rd ed. Draft of January 12, 2022.

Kadra, A., M. Lindauer, F. Hutter, and J. Grabocka (2021). "Well-tuned Simple Nets Excel on Tabular Datasets." In: *Proceedings of the 34th International Conference on Advances in Neural Information Processing Systems (NeurIPS'21)*. Ed. by M. Ranzato, A. Beygelzimer, K. Nguyen, P. Liang, J. Vaughan, and Y. Dauphin. Curran Associates.

Kandasamy, K., W. Neiswanger, J. Schneider, B. Poczos, and E. Xing (2018). "Neural Architecture Search with Bayesian Optimisation and Optimal Transport." In: *Proceedings of the 31st International Conference on Advances in Neural Information Processing Systems (NeurIPS'18)*. Ed. by S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett. Curran Associates, pp. 2016–2025.

Kaplan, J., S. McCandlish, T. Henighan, T. Brown, B. Chess, R. Child, S. Gray, A. Radford, J. Wu, and D. Amodei (2020). "Scaling laws for neural language models." In: *arXiv:2001.08361 [cs.LG]*.

Katz, M., P. Ram, S. Sohrabi, and O. Udrea (2020). "Exploring Context-Free Languages via Planning: The Case for Automating Machine Learning." In: *Proceedings of the 30th International Conference on Automated Planning and Scheduling (ICAPS)*. AAAI Press, pp. 403–411.

Ke, G., Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu (2017). "LightGBM: A Highly Efficient Gradient Boosting Decision Tree." In: *Proceedings of the 30th International Conference on Advances in Neural Information Processing Systems (NeurIPS'17)*.

Ed. by I. Guyon, U. von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett. Curran Associates, pp. 3149–3157.

Kerschke, P., H. Hoos, F. Neumann, and H. Trautmann (2019). "Automated Algorithm Selection: Survey and Perspectives." In: *Evolutionary Computation* 27.1, pp. 3–45.

Khurana, U. and H. Samulowitz (2020). "Autonomous Predictive Modeling via Reinforcement Learning." In: *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*. CIKM '20. Association for Computing Machinery, pp. 3285–3288.

Kim, Y.-H., B. Reddy, S. Yun, and H Seo (2017). "NEMO: Neuro-Evolution with Multiobjective Optimization of Deep Neural Network for Speed and Accuracy." In: *ICML workshop on Automated Machine Learning (AutoML workshop 2017)*. Ed. by J. Vanschoren and R. Garnett.

King, R., C. Feng, and A. Sutherland (1995). "Statlog: comparison of classification algorithms on large real-world problems." In: *Applied Artificial Intelligence* 9.3, pp. 289–333.

King, R., J. Hirst, and M. Sternberg (1993). "New approaches to QSAR: Neural networks and machine learning." In: *Perspectives in Drug Discovery and Design* 1.2, pp. 279–290.

Komer, B., J. Bergstra, and C. Eliasmith (2014). "Hyperopt-Sklearn: Automatic Hyperparameter Configuration for Scikit-Learn." In: *ICML workshop on Automated Machine Learning (AutoML workshop 2014)*. Ed. by F. Hutter, R. Caruana, R. Bardenet, M. Bilenko, I. Guyon, B. Kégl, and H. Larochelle.

– (2019). "Hyperopt-Sklearn." In: *Automated Machine Learning: Methods, Systems, Challenges*. Ed. by F. Hutter, L. Kotthoff, and J. Vanschoren. Vol. 5. The Springer Series on Challenges in Machine Learning. Available for free at http://automl.org/book. Springer. Chap. 5, pp. 97–111.

Kotthoff, L., C. Thornton, H. Hoos, F. Hutter, and K. Leyton-Brown (2019). "Auto-WEKA: Automatic Model Selection and Hyperparameter Optimization in WEKA." In: *Automated Machine Learning: Methods, Systems, Challenges*. Ed. by F. Hutter, L. Kotthoff, and J. Vanschoren. Vol. 5. The Springer Series on Challenges in Machine Learning. Available for free at http://automl.org/book. Springer. Chap. 4, pp. 81–95.

Krause, A. and D. Golovin (2014). "Submodular Function Maximization." In: *Tractability: Practical Approaches to Hard Problems*. Ed. by L. Bordeaux, Y. Hamadi, and P. Kohli. Cambridge University Press. Chap. 3, pp. 71–104.

Krause, A., J. Leskovec, C. Guestrin, J. VanBriesen, and C. Faloutsos (2008). "Efficient Sensor Placement Optimization for Securing Large Water Distribution Networks." In: *Journal of Water Resources Planning and Management* 134 (6), pp. 516–526.

Křen, T., M. Pilát, and R. Neruda (2015). "Evolving Workflow Graphs Using Typed Genetic Programming." In: *2015 IEEE Symposium Series on Computational Intelligence*, pp. 1407–1414.

– (2017). "Automatic Creation of Machine Learning Workflows with Strongly Typed Genetic Programming." In: *International Journal on Artificial Intelligence Tools* 26.05, pp. 1760020-1–1760020-24.

Kriegel, H., E. Schubert, and A. Zimek (2017). "The (Black) Art of Runtime Evaluation: Are We Comparing Algorithms or Implementations?" In: *Knowledge Information System* 52.2, pp. 341–378.

Krizhevsky, A., I. Sutskever, and G. Hinton (2012). "ImageNet Classification with Deep Convolutional Neural Networks." In: *Proceedings of the 25th International Conference on Advances in Neural Information Processing Systems (NeurIPS'12)*. Ed. by P. Bartlett, F. Pereira, C. Burges, L. Bottou, and K. Weinberger. Curran Associates, pp. 1097–1105.

Langley, P. (1996). "Relevance and Insight in Experimental Studies." In: *IEEE Expert Online*. Ed. by D. Price, pp. 11–12.

Lauri, J. and S. Dutta (2019). "Fine-Grained Search Space Classification for Hard Enumeration Variants of Subset Problems." In: *Proceedings of the Thirty-Third Conference on Artificial Intelligence (AAAI'19)*. Ed. by P. Van Hentenryck and Z. Zhou. AAAI Press, pp. 2314–2321.

Lauri, J., S. Dutta, M. Grassia, and D. Ajwani (2020). "Learning fine-grained search space pruning and heuristics for combinatorial optimization." In: *arXiv:2001.01230 [cs.DS]*.

LeCun, Y., B. Boser, J. Denker, D. Henderson, R. Howard, W. Hubbard, and L. Jackel (1989). "Backpropagation Applied to Handwritten Zip Code Recognition." In: *Neural Comput.* 1.4, pp. 541–551.

LeDell, E. (2019). *Panel discussion of the 6th ICML Workshop on Automated Machine Learning*. Ed. by K. Eggensperger, M. Feurer, F. Hutter, and J Vanschoren.

LeDell, E. and S. Poirier (2020). "H2O AutoML: Scalable Automatic Machine Learning." In: *ICML workshop on Automated Machine Learning (AutoML workshop 2020)*. Ed. by K. Eggensperger, M. Feurer, C. Weill, M.Lindauer, F. Hutter, and J. Vanschoren.

Leite, R., P. Brazdil, and J. Vanschoren (2012). "Selecting classification algorithms with active testing." In: *Machine Learning and Data Mining in Pattern Recognition*. Ed. by P. Perner. Vol. 7376. Lecture Notes in Computer Science. Springer, pp. 117–131.

Li, C. (2020). *OpenAI's GPT-3 Language Model: A Technical Overview*. Retrieved on Mar 7, 2022. URL: https://lambdalabs.com/blog/demystifying-gpt-3/.

Li, L., K. Jamieson, G. DeSalvo, A. Rostamizadeh, and A. Talwalkar (2018). "Hyperband: A Novel Bandit-Based Approach to Hyperparameter Optimization." In: *Journal of Machine Learning Research* 18.185, pp. 1–52.

Li, Y., Y. Shen, W. Zhang, Y. Chen, H. Jiang, M. Liu, J. Jiang, J. Gao, W. Wu, Z. Yang, C. Zhang, and B. Cui (2021). "OpenBox: A Gener-

alized Black-Box Optimization Service." In: *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*. Association for Computing Machinery, pp. 3209–3219.

Liu, G., P. Wang, M. Beveridge, Y.-O. Kwon, and I. Drori (2021). "Predicting Atlantic Multidecadal Variability." In: *NeurIPS 2021 Workshop on Tackling Climate Change with Machine Learning*.

Liu, H., K. Simonyan, and Y. Yang (2019). "DARTS: Differentiable Architecture Search." In: *Proceedings of the International Conference on Learning Representations (ICLR'19)*. Published online: iclr.cc.

Liu, S., P. Ram, D. Vijaykeerthy, D. Bouneffouf, G. Bramble, H. Samulowitz, D. Wang, A. Conn, and A. Gray (2020). "An ADMM Based Framework for AutoML Pipeline Configuration." In: *Proceedings of the Thirty-Fourth Conference on Artificial Intelligence (AAAI'20)*. Ed. by F. Rossi, V. Conitzer, and F. Sha. AAAI Press, pp. 4892–4899.

Lowe, D. (1999). "Object recognition from local scale-invariant features." In: *Proceedings of the Seventh IEEE International Conference on Computer Vision*. Vol. 2, pp. 1150–1157.

Lu, Z., I. Whalen, V. Boddeti, Y. Dhebar, K. Deb, E. Goodman, and W. Banzhaf (2019). "NSGA-Net: Neural Architecture Search Using Multi-Objective Genetic Algorithm." In: *Proceedings of the Genetic and Evolutionary Computation Conference*. GECCO '19. Association for Computing Machinery, pp. 419–427.

Macià, N. and E. Bernadó-Mansilla (2014). "Towards UCI+: A mindful repository design." In: *Information Sciences* 261, pp. 237–262.

Maipradit, R., B. Lin, C. Nagy, G. Bavota, M. Lanza, H. Hata, and K. Matsumoto (2020). "Automated Identification of On-hold Self-admitted Technical Debt." In: *2020 IEEE 20th International Working Conference on Source Code Analysis and Manipulation (SCAM)*, pp. 54–64.

Maipradit, R., C. Treude, H. Hata, and K. Matsumoto (2020). "Wait for it: identifying "On-Hold" self-admitted technical debt." In: *Empirical Software Engineering* 25.5, pp. 3770–3798.

Marinescu, R., A. Kishimoto, P. Ram, A. Rawat, M. Wistuba, P. Palmes, and A. Botea (2021). "Searching for Machine Learning Pipelines Using a Context-Free Grammar." In: pp. 8902–8911.

Melis, G., C. Dyer, and P. Blunsom (2018). "On the State of the Art of Evaluation in Neural Language Models." In: *Proceedings of the International Conference on Learning Representations (ICLR'18)*. Published online: iclr.cc.

Mendoza, H., A. Klein, M. Feurer, J. Springenberg, M. Urban, M. Burkart, M. Dippel, M. Lindauer, and F. Hutter (2019). "Towards Automatically-Tuned Deep Neural Networks." In: *Automated Machine Learning: Methods, Systems, Challenges*. Ed. by F. Hutter, L. Kotthoff, and J. Vanschoren. Vol. 5. The Springer Series on Challenges in Machine Learning. Available for free at http://automl.org/book. Springer, pp. 135–149.

Michailidis, M. (2017). "Investigating machine learning methods in recommender systems - Improving prediction for the top K items." PhD thesis. University College London.

Milutinovic, M. (2019). "Towards Automatic Machine Learning Pipeline Design." PhD thesis. University of California, Berkeley.

Mohr, F. and M. Wever (2021). "Replacing the Ex-Def Baseline in AutoML by Naive AutoML." In: *ICML workshop on Automated Machine Learning (AutoML workshop 2021)*. Ed. by K. Eggensperger, M.Lindauer, F. Hutter, and J. Vanschoren.

Mohr, F., M. Wever, and E. Hüllermeier (2018). "ML-Plan: Automated machine learning via hierarchical planning." In: *Machine Learning* 107.8-10, pp. 1495–1515.

Mohr, F., M. Wever, A. Tornede, and E. Hüllermeier (2021). "Predicting Machine Learning Pipeline Runtimes in the Context of Automated Machine Learning." In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 43.9, pp. 3055–3066.

Moosbauer, J., J. Herbinger, G. Casalicchio, M. Lindauer, and B. Bischl (2021). "Explaining Hyperparameter Optimization via Partial Dependence Plots." In: *Proceedings of the 34th International Conference on Advances in Neural Information Processing Systems (NeurIPS'21)*. Ed. by M. Ranzato, A. Beygelzimer, K. Nguyen, P. Liang, J. Vaughan, and Y. Dauphin. Curran Associates.

Murphy, K. (2012). *Machine learning: a probabilistic perspective*. Cambridge, MA: The MIT Press.

Nguyen, T.-D., K. Musial, and B. Gabrys (2021). "AutoWeka4MCPS-AVATAR: Accelerating automated machine learning pipeline composition and optimisation." In: *Expert Systems with Applications* 185.115643.

Nielsen, M. (2011). *Reinventing Discovery: The New Era of Networked Science*. Princeton University Press.

Olson, R., N. Bartley, R. Urbanowicz, and J. Moore (2016). "Evaluation of a Tree-based Pipeline Optimization Tool for Automating Data Science." In: *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'16)*. Ed. by T. Friedrich. ACM, pp. 485–492.

Olson, R. and J. Moore (2019). "TPOT: A Tree-Based Pipeline Optimization Tool for Automating Machine Learning." In: *Automated Machine Learning: Methods, Systems, Challenges*. Ed. by F. Hutter, L. Kotthoff, and J. Vanschoren. Vol. 5. The Springer Series on Challenges in Machine Learning. Available for free at http://automl.org/book. Springer. Chap. 8, pp. 151–160.

Olson, R., R. Urbanowicz, P. Andrews, N. Lavender, L. Kidd, and J. Moore (2016). "Automating Biomedical Data Science Through Tree-Based Pipeline Optimization." In: *Proceedings of the 19th European Conference on Applications of Evolutionary Computation (EvoApplications'16)*. Ed. by G. Squillero and P. Burelli. Springer, pp. 123–137.

Ono, J., S. Castelo, R. Lopez, E. Bertini, J. Freire, and C. Silva (2021). "PipelineProfiler: A Visual Analytics Tool for the Exploration of AutoML Pipelines." In: *IEEE Transactions on Visualization and Computer Graphics* 27.2, pp. 390–400.

OpenML.org (2022). *Repositories of the OpenML organization on github.com*. Retrieved on Feb 23, 2022. URL: https://github.com/orgs/openml/repositories?q=&type=all&language=&sort=stargazers.

Parmentier, L., O. Nicol, L. Jourdan, and M. Kessaci (2019). "TPOT-SH: A Faster Optimization Algorithm to Solve the AutoML Problem on Large Datasets." In: *Proceedings of the 31st IEEE International Conference on Tools with Artificial Intelligence (ICTAI'19)*, pp. 471–478.

Paszke, A. et al. (2019). "PyTorch: An Imperative Style, High-Performance Deep Learning Library." In: *Proceedings of the 32nd International Conference on Advances in Neural Information Processing Systems (NeurIPS'19)*. Ed. by H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett. Curran Associates, pp. 8024–8035.

Pedregosa, F., T. Maharaj, A. Kucukelbir, R. Das, V. Borghesani, F. Bach, D. Blei, and B. Schölkopf (2022). *Retrospectives from 20 Years of JMLR*. URL: https://jmlr.org/news/2022/retrospectives.html.

Pedregosa, F., G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay (2011). "Scikit-learn: Machine Learning in Python." In: *Journal of Machine Learning Research* 12, pp. 2825–2830.

Perrone, V., R. Jenatton, M. Seeger, and C. Archambeau (2018). "Scalable hyperparameter transfer learning." In: *Proceedings of the 31st International Conference on Advances in Neural Information Processing Systems (NeurIPS'18)*. Ed. by S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett. Curran Associates, pp. 12751–12761.

Perrone, V. and H. Shen (2019). "Learning search spaces for Bayesian optimization: Another view of hyperparameter transfer learning." In: *Proceedings of the 32nd International Conference on Advances in Neural Information Processing Systems (NeurIPS'19)*. Ed. by H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett. Curran Associates, pp. 12751–12761.

Pfisterer, F., S. Coors, J. Thomas, and B. Bischl (2019). "Multi-Objective Automatic Machine Learning with AutoxgboostMC." In: *arXiv: 1908.10796 [stat.ML]*.

Pfisterer, F., J. Thomas, and B. Bischl (2019). "Towards Human Centered AutoML." In: *arXiv:1911.02391 [cs.HC]*.

Pilát, M., T. Křen, and R. Neruda (2016). "Asynchronous Evolution of Data Mining Workflow Schemes by Strongly Typed Genetic Programming." In: *2016 IEEE 28th International Conference on Tools with Artificial Intelligence (ICTAI)*, pp. 577–584.

Pineda, S., H. Jomaa, M. Wistuba, and J. Grabocka (2021). "HPO-B: A Large-Scale Reproducible Benchmark for Black-Box HPO based on OpenML." In: *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*. Ed. by J. Vanschoren, S. Yeung, and M. Xenochristou. Curran Associates.

Probst, P., A. Boulesteix, and B. Bischl (2019). "Tunability: Importance of Hyperparameters of Machine Learning Algorithms." In: *Journal of Machine Learning Research* 20.53, pp. 1–32.

Prokhorenkova, L., G. Gusev, A. Vorobev, A. Dorogush, and A. Gulin (2018). "CatBoost: unbiased boosting with categorical features." In: *Proceedings of the 31st International Conference on Advances in Neural Information Processing Systems (NeurIPS'18)*. Ed. by S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett. Curran Associates.

Provost, F., D. Jensen, and T. Oates (1999). "Efficient progressive sampling." In: *The 5th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'99)*. Ed. by U. Fayyad, S. Chaudhuri, and D. Madigan. ACM Press, pp. 23–32.

Qi, F., Z. Xia, G. Tang, H. Yang, Y. Song G. Qian, X. An, C. Lin, and G. Shi (2019). "DarwinML: A Graph-based Evolutionary Algorithm for Automated Machine Learning." In: *arXiv:1901.08013 [cs.NE]*.

Rahimi, A. and B. Recht (2007). "Random Features for Large-Scale Kernel Machines." In: *Proceedings of the 20th International Conference on Advances in Neural Information Processing Systems (NeurIPS'07)*. Ed. by J. Platt, D. Koller, Y. Singer, and S. Roweis. Curran Associates.

Rakotoarison, H., M. Schoenauer, and M. Sebag (2019). "Automated Machine Learning with Monte-Carlo Tree Search." In: *Proceedings of the 28th International Joint Conference on Artificial Intelligence (IJCAI'19)*. Ed. by S. Kraus. ijcai.org, pp. 3296–3303.

Recht, B., R. Roelofs, L. Schmidt, and V. Shankar (2019). "Do ImageNet Classifiers Generalize to ImageNet?" In: *Proceedings of the 36th International Conference on Machine Learning (ICML'19)*. Ed. by K. Chaudhuri and R. Salakhutdinov. Vol. 97. Proceedings of Machine Learning Research, pp. 5389–5400.

Reif, M., F. Shafait, and A. Dengel (2012). "Meta-learning for evolutionary parameter optimization of classifiers." In: *Machine Learning* 87, pp. 357–380.

Reif, M., F. Shafait, M. Goldstein, T. Breuel, and A. Dengel (2014). "Automatic classifier selection for non-experts." In: *Pattern Analysis and Applications* 17.1, pp. 83–96. ISSN: 1433-7541.

Rice, J. (1976). "The Algorithm Selection Problem." In: *Advances in Computers* 15, pp. 65–118.

Rijn, J. van (2016). "Massively Collaborative Machine Learning." PhD thesis. Leiden University.

Rijn, J. van and F. Hutter (2018). "Hyperparameter Importance Across Datasets." In: *Proceedings of the 24th ACM SIGKDD International*

Conference on Knowledge Discovery and Data Mining (KDD). Ed. by Y. Guo and F. Farooq. ACM Press, pp. 2367–2376.

Schrittwieser, J., I. Antonoglou, T. Hubert, K. Simonyan, L. Sifre, S. Schmitt, A. Guez, E. Lockhart, D. Hassabis, T. Graepel, T. Lillicrap, and D. Silver (2020). "Mastering Atari, Go, chess and shogi by planning with a learned model." In: *Nature* 588.7839, pp. 604–609.

Serban, F., J. Vanschoren, J.-U. Kietz, and A. Bernstein (2013). "A Survey of Intelligent Assistants for Data Analysis." In: *ACM Computing Surveys* 45.3.

Shahriari, B., K. Swersky, Z. Wang, R. Adams, and N. de Freitas (2016). "Taking the Human Out of the Loop: A Review of Bayesian Optimization." In: *Proceedings of the IEEE* 104.1, pp. 148–175.

Shi, X., J. Mueller, N. Erickson, M. Li, and A. Smola (2021). "Benchmarking Multimodal AutoML for Tabular Data with Text Fields." In: *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*. Ed. by J. Vanschoren, S. Yeung, and M. Xenochristou. Curran Associates.

Silver, D., A. Huang, C. Maddison, A. Guez, L. Sifre, G. Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis (2016). "Mastering the game of Go with deep neural networks and tree search." In: *Nature* 529.7587, pp. 484–489.

Silver, D., T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. Lillicrap, K. Simonyan, and D. Hassabis (2018). "A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play." In: *Science* 362.6419, pp. 1140–1144.

Smith-Miles, K. (2008). "Cross-disciplinary perspectives on meta-learning for algorithm selection." In: *ACM Computing Surveys* 41.1.

Snoek, J., H. Larochelle, and R. Adams (2012). "Practical Bayesian Optimization of Machine Learning Algorithms." In: *Proceedings of the 25th International Conference on Advances in Neural Information Processing Systems (NeurIPS'12)*. Ed. by P. Bartlett, F. Pereira, C. Burges, L. Bottou, and K. Weinberger. Curran Associates, pp. 2960–2968.

Statnikov, A., I. Tsamardinos, Y. Dosbayev, and C. Aliferis (2005). "GEMS: a system for automated cancer diagnosis and biomarker discovery from microarray gene expression data." In: *International Journal of Medical Informatics* 74, pp. 491–503.

Sun, Q., B. Pfahringer, and M. Mayo (2013). "Towards a Framework for Designing Full Model Selection and Optimization Systems." In: *Multiple Classifier Systems*. Vol. 7872. Springer, pp. 259–270.

Sun, Q. and B. Pfahringer (2013). "Pairwise meta-rules for better meta-learning-based algorithm ranking." In: *Machine Learning* 93.1, pp. 141–161.

Sun, X., J. Lin, and B. Bischl (2019). "ReinBo: Machine Learning pipeline search and configuration with Bayesian Optimization embedded Reinforcement Learning." In: *arXiv:1904.05381 [cs.LG]*.

Sutton, R. (2019). *The Bitter Lesson*. URL: http://www.incompleteideas.net/IncIdeas/BitterLesson.html.

Swearingen, T., W. Drevo, B. Cyphers, A. Cuesta-Infante, A. Ross, and K. Veeramachaneni (2017). "ATM: A distributed, collaborative, scalable system for automated machine learning." In: *2017 IEEE International Conference on Big Data*. IEEE Computer Society Press, pp. 151–162.

Takeno, S., H. Fukuoka, Y. Tsukada, T. Koyama, M. Shiga, I. Takeuchi, and M. Karasuyama (2020). "Multi-fidelity Bayesian Optimization with Max-value Entropy Search and its Parallelization." In: *Proceedings of the 37th International Conference on Machine Learning (ICML'20)*. Ed. by H. Daume III and A. Singh. Vol. 98. Proceedings of Machine Learning Research, pp. 9334–9345.

Teichmann, M. (2021). *Machine Learning 2021*. Published by IDG Business Media GbmH. Available at https://www.lufthansa-industry-solutions.com/de-de/studien/mailing/idg-studie-machine-learning-2021.

Thornton, C., F. Hutter, H. Hoos, and K. Leyton-Brown (2013). "Auto-WEKA: combined selection and hyperparameter optimization of classification algorithms." In: *The 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'13)*. Ed. by I. Dhillon, Y. Koren, R. Ghani, T. Senator, P. Bradley, R. Parekh, J. He, R. Grossman, and R. Uthurusamy. ACM Press, pp. 847–855.

Tornede, T., A. Tornede, J. Hanselle, M. Wever, F. Mohr, and E. Hüllermeier (2021). "Towards Green Automated Machine Learning: Status Quo and Future Directions." In: *arXiv:2111.05850 [cs.LG]*.

Tromp, J. (2016). "The Number of Legal Go Positions." In: *Computers and Games*. Ed. by A. Plaat, W. Kosters, and J. van den Herik. Vol. 10068. Lecture Notes in Computer Science. Springer, pp. 183–190.

Turner, R., D. Eriksson, M. McCourt, J. Kiili, E. Laaksonen, Z. Xu, and I. Guyon (2021). "Bayesian Optimization is Superior to Random Search for Machine Learning Hyperparameter Tuning: Analysis of the Black-Box Optimization Challenge 2020." In: *Proceedings of the NeurIPS 2020 Competition and Demonstration Track*. Ed. by H. Escalante and K. Hofmann. PMLR, pp. 3–26.

Vakhrushev, A., A. Ryzhkov, M. Savchenko, D. Simakov, R. Damdinov, and A. Tuzhilin (2021). "LightAutoML: AutoML Solution for a Large Financial Services Ecosystem." In: *arXiv:2109.01528 [cs.LG]*.

van Eeden, W., C. Luo, A. van Hemert, I. Carlier, B. Penninx, K. Wardenaar, H. Hoos, and E. Giltay (2021). "Predicting the 9-year course of mood and anxiety disorders with automated machine learning:

A comparison between auto-sklearn, naïve Bayes classifier, and traditional logistic regression." In: *Psychiatry Research* 299.113823.

Vanschoren, J. (2019). "Meta-Learning." In: *Automated Machine Learning: Methods, Systems, Challenges*. Ed. by F. Hutter, L. Kotthoff, and J. Vanschoren. Vol. 5. The Springer Series on Challenges in Machine Learning. Available for free at http://automl.org/book. Springer. Chap. 2, pp. 35–61.

Vanschoren, J., M. Braun, and C. Ong (2014). "Open science in machine learning." In: *arXiv:1402.6013 [cs.LG]*.

Vanschoren, J., J. van Rijn, B. Bischl, and L. Torgo (2014). "OpenML: Networked Science in Machine Learning." In: *SIGKDD Explor. Newsl.* 15.2, pp. 49–60.

Vanschoren, J., S. Yeung, and M. Xenochristou, eds. (2021). *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*. Curran Associates.

Vapnik, V. (1991). "Principles of Risk Minimization for Learning Theory." In: *Proceedings of the 4th International Conference on Advances in Neural Information Processing Systems (NeurIPS'91)*. Ed. by J. Moody, S. Hanson, and R. Lippmann. Morgan Kaufmann Publishers.

Volpp, M., L. Fröhlich, K. Fischer, A. Doerr, S. Falkner, F. Hutter, and C. Daniel (2020). "Meta-Learning Acquisition Functions for Transfer Learning in Bayesian Optimization." In: *Proceedings of the International Conference on Learning Representations (ICLR'20)*. Published online: iclr.cc.

Wang, C., Q. Wu, M. Weimer, and E. Zhu (2021). "FLAML: A Fast and Lightweight AutoML Library." In: *Proceedings of Machine Learning and Systems 3*. Ed. by A. Smola, A. Dimakis, and I. Stoica. Vol. 3, pp. 434–447.

Wang, D., Q. Liao, Y. Zhang, U. Khurana, H. Samulowitz, S. Park, M. Muller, and L. Amini (2021). "How Much Automation Does a Data Scientist Want?" In: *arXiv:2101.03970 [cs.LG]*.

Wang, D., P. Ram, D. Weidele, S. Liu, M. Muller, J. Weisz, A. Valente, A. Chaudhary, D. Torres, H. Samulowitz, and L. Amini (2020). "AutoAI: Automating the End-to-End AI Lifecycle with Humans-in-the-Loop." In: *Proceedings of the 25th International Conference on Intelligent User Interfaces Companion*. IUI '20. Association for Computing Machinery, pp. 77–78.

Wang, Q., Y. Ming, Z. Jin, Q. Shen, D. Liu, M. Smith, K. Veeramachaneni, and H. Qu (2019). "ATMSeer: Increasing Transparency and Controllability in Automated Machine Learning." In: *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, pp. 1–12.

Wattanakriengkrai, S., R. Maipradit, H. Hata, M. Choetkiertikul, T. Sunetnanta, and K. Matsumoto (2018). "Identifying Design and Requirement Self-Admitted Technical Debt Using N-gram IDF." In:

2018 9th International Workshop on Empirical Software Engineering in Practice (IWESEP), pp. 7–12.

Wei, Y., P. Zhao, and J. Huang (2021). "Meta-learning Hyperparameter Performance Prediction with Neural Processes." In: *Proceedings of the 38th International Conference on Machine Learning (ICML'21)*. Ed. by M. Meila and T. Zhang. Vol. 139. Proceedings of Machine Learning Research. PMLR, pp. 11058–11067.

Weidele, D., J. Weisz, E. Oduor, M. Muller, J. Andres, A. Gray, and D. Wang (2020). "AutoAIViz: Opening the Blackbox of Automated Artificial Intelligence with Conditional Parallel Coordinates." In: *Proceedings of the 25th International Conference on Intelligent User Interfaces*. IUI '20. Association for Computing Machinery, pp. 308–312.

White, C., W. Neiswanger, and Y. Savani (2021). "BANANAS: Bayesian Optimization with Neural Architectures for Neural Architecture Search." In: *Proceedings of the Thirty-Fifth Conference on Artificial Intelligence (AAAI'21)*. AAAI Press, pp. 10293–10301.

Wistuba, M. and J. Grabocka (2021). "Few-Shot Bayesian Optimization with deep kernel surrogates." In: *Proceedings of the International Conference on Learning Representations (ICLR'21)*. Published online: iclr.cc.

Wistuba, M., N. Schilling, and L. Schmidt-Thieme (2015). "Hyperparameter Search Space Pruning - A New Component for Sequential Model-Based Hyperparameter Optimization." In: *Machine Learning and Knowledge Discovery in Databases (ECML/PKDD'15)*. Ed. by A. Appice, P. Rodrigues, V. Costa, J. Gama, A. Jorge, and C. Soares. Vol. 9285. Lecture Notes in Computer Science. Springer, pp. 104–119.

– (2017). "Automatic Frankensteining: Creating Complex Ensembles Autonomously." In: *Proceedings of the 2017 SIAM International Conference on Data Mining (SDM'17)*. Ed. by N. Chawla and W. Wang. Society for Industrial and Applied Mathematics, pp. 741–749.

Wu, J., S. Toscano-Palmerin, P. Frazier, and A. Wilson (2019). "Practical Multi-fidelity Bayesian Optimization for Hyperparameter Tuning." In: *Proceedings of The 35th Uncertainty in Artificial Intelligence Conference (UAI'19)*. Ed. by R. Adams and V. Gogate. AUAI Press, pp. 788–798.

Xanthopoulos, I., I. Tsamardinos, V. Christophides, E. Simon, and A. Salinger (2020). "Putting the Human Back in the AutoML Loop." In: *ETLMP 2020: Explainability for Trustworthy ML Pipelines*.

Xiao, Y., E. Xing, and W. Neiswanger (2021). "Amortized Auto-Tuning: Cost-Efficient Transfer Optimization for Hyperparameter Recommendation." In: *arXiv:2106.09179 [cs.LG]*.

Xu, L., H. Hoos, and K. Leyton-Brown (2010). "Hydra: Automatically Configuring Algorithms for Portfolio-Based Selection." In: *Proceedings of the Twenty-fourth National Conference on Artificial Intelligence (AAAI'10)*. Ed. by M. Fox and D. Poole. AAAI Press, pp. 210–216.

Xu, L., F. Hutter, H. Hoos, and K. Leyton-Brown (2011). "Hydra-MIP: Automated Algorithm Configuration and Selection for Mixed Integer Programming." In: *Proc. of RCRA workshop at IJCAI*.

Yang, C., J. Akimoto, D. Kim, and M. Udell (2019). "OBOE: Collaborative Filtering for AutoML Model Selection." In: *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD'19*. Ed. by A. Teredesai, V. Kumar, Y. Li, R. Rosales, E. Terzi, and G. Karypis. ACM Press, pp. 1173–1183.

Yang, C., J. Fan, Z. Wu, and M. Udell (2020). "AutoML Pipeline Selection: Efficiently Navigating the Combinatorial Space." In: *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'20)*. Ed. by J. Tang and B. Prakash. ACM Press, pp. 1446–1456.

Zhang, X., Z. Chang, Y. Li, H. Wu, J. Tan, F. Li, and B Cui (2021). "Facilitating Database Tuning with Hyper-Parameter Optimization: A Comprehensive Experimental Evaluation." In: *arXiv:2110.12654 [cs.DB]*.

Zhang, Y., M. Bahadori, H. Su, and J. Sun (2016). "FLASH: Fast Bayesian Optimization for Data Analytic Pipelines." In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*. Ed. by B. Krishnapuram, M. Shah, A. Smola, C. Aggarwal, D. Shen, and R. Rastogi. ACM Press, pp. 2065–2074.

Zhou, Y., S. Ebrahimi, S. Arık, H. Yu, H. Liu, and G. Diamos (2018). "Resource-Efficient Neural Architect." In: *arXiv:1806.07912 [cs.NE]*.

Zimmer, L., M. Lindauer, and F. Hutter (2021). "Auto-Pytorch: Multi-Fidelity MetaLearning for Efficient and Robust AutoDL." In: *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pp. 3079–3090.

Zöller, M.-A., T.-D. Nguyen, and M. Huber (2021). "Incremental Search Space Construction for Machine Learning Pipeline Synthesis." In: *Advances in Intelligent Data Analysis XIX*. Ed. by P. Abreu, P. Rodrigues, A. Fernández, and J. Gama. Springer, pp. 103–115.