

Systems for AutoML Research

Citation for published version (APA):

Gijsbers, P. (2022). *Systems for AutoML Research*. [Phd Thesis 1 (Research TU/e / Graduation TU/e), Mathematics and Computer Science]. Eindhoven University of Technology.

Document status and date:

Published: 19/05/2022

Document Version:

Publisher's PDF, also known as Version of Record (includes final page, issue and volume numbers)

Please check the document version of this publication:

- A submitted manuscript is the version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher's website.
- The final author version and the galley proof are versions of the publication after peer review.
- The final published version features the final layout of the paper including the volume, issue and page numbers.

[Link to publication](#)

General rights

Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal.

If the publication is distributed under the terms of Article 25fa of the Dutch Copyright Act, indicated by the "Taverne" license above, please follow below link for the End User Agreement:

www.tue.nl/taverne

Take down policy

If you believe that this document breaches copyright please contact us at:

openaccess@tue.nl

providing details and we will investigate your claim.

Systems for AutoML Research

PROEFSCHRIFT

ter verkrijging van de graad van doctor aan de
Technische Universiteit Eindhoven, op gezag van de
rector magnificus prof.dr.ir. F.P.T. Baaijens, voor een
commissie aangewezen door het College voor
Promoties, in het openbaar te verdedigen op
donderdag 19 mei 2022 om 13:30 uur

door

Pieter Gijsbers

geboren te Eindhoven

Dit proefschrift is goedgekeurd door de promotoren en de samenstelling van de promotiecommissie is als volgt:

voorzitter:	prof. dr. J.J. Lukkien
1e promotor:	prof. dr. M. Pechenizkiy
copromotor:	dr. ir. J. Vanschoren
leden:	prof. dr. I. Tsamardinos (University of Crete) prof. dr. T.H.W. Bäck (Universiteit Leiden) dr. Y. Zhang
adviseurs:	dr. M. Sebag (Centre national de la recherche scientifique) dr. S.C. Hess

Het onderzoek of ontwerp dat in dit proefschrift wordt beschreven is uitgevoerd in overeenstemming met de TU/e Gedragscode Wetenschapsbeoefening.

Systems for AutoML Research by Pieter Gijsbers.
Eindhoven: Technische Universiteit Eindhoven, 2022. Proefschrift.

A catalogue record is available from the Eindhoven University of Technology
Library.

ISBN: 978-90-386-5510-9.



SIKS Dissertation Series No. 2022-16

The research reported in this thesis has been carried out under the auspices of
SIKS, the Dutch Research School for Information and Knowledge Systems.

Acknowledgements

I have been very fortunate to have had the opportunity to spend the past four years working with many brilliant and kind people, without whom I surely would not have been able to produce the work presented in this thesis.

First, I would like to thank Joaquin Vanschoren, not just for providing excellent scientific guidance and equally good barbecues, but also for convincing me to start this journey in the first place. He also assembled a terrific group of researchers and engineers which has been a pleasure to work with. Joaquin Vanschoren and everyone in his team, many of whom I have had the pleasure to share an office with, have provided insight, fun conversations, and support, for which I am grateful. Thank you, Joaquin, Bilge, Sahithya, Prabhant, Marcos, Andrei, Israel, Juan, Onur, Ceren, Fangqin, and Jiarong.

I have had an amazing experience working as part of the DAI cluster, for which I want to thank all of my colleagues. In particular, I would like to thank Wouter Duivesteijn and Simon van der Zon for organizing lunch gatherings and other social activities, the coffee people for providing an excellent environment in which to have my tea breaks, Riet, Ine, and most of all José for helping me find a way through university bureaucracy, and Mykola for his leadership in the DM group and his support as my promotor.

I am grateful to Vlado, Anil, and Bilge for introducing me to bouldering and our many bouldering sessions. The exercise as well as the discussions have helped me stay healthy in both body and mind - and above all, it has just been a lot of fun.

In the summer of 2018, I spent one month on a research visit to work on what would ultimately become one of the main projects I have worked on for the past four years. I would like to thank *H2O.ai*, in particular Erin LeDell, for this opportunity and Janek Thomas for joining me on that venture.

It goes without saying that the OpenML community has been most influ-

ential in my work. The many workshops and discussions have been inspiring, thought-provoking, and exhilarating. I extend my gratitude to the steering committee past and present: Bernd, Giuseppe, Heidi, Jan, Joaquin, and Matthias, most of whom I have had the pleasure to get to know in person.

I would also like to thank my friends, family, and girlfriend who have been understanding and provided support even if at times I got lost in my work. My parents, for giving me every opportunity in my upbringing and always providing a carefree environment to escape to. My friends from Gemert, Eindhoven, and Otaniemi for all the fun times we shared. And my girlfriend, for always being by my side.

Special thanks go out to my committee members, Ioannis Tsamardinos, Thomas Bäck, and Yingqian Zhang and the committee advisors, Sibylle Hess and Michèle Sebag for taking their time to partake in the defense ceremony and providing valuable feedback on this manuscript.

I realize that many people remain unnamed that nevertheless have helped my last four years be enjoyable and fruitful ones, people I have met at conferences, workshops, the university, or other venues. Please know that I am grateful to all of you.

Finally, I would like to acknowledge funding for my employment by AFRL and DARPA (under contract FA8750-17-C-0141) and EU's Horizon 2020 research and innovation program (under grant agreement No. 952215 (TAILOR)).

Pieter Gijsbers
Eindhoven, April 2022

Summary

Machine learning (ML) is used in many applications but creating a useful model from data is a knowledge-intensive and laborious task. Automated machine learning (AutoML) aims to automate the construction of machine learning pipelines in a data-driven way, which allows novice users to create ML models and expert users to focus on other tasks. A diverse set of approaches for AutoML have been proposed, however previous work largely compares frameworks or techniques in an ad-hoc fashion. There is little consistency in the choice of datasets, performance metrics, or hardware constraints. This makes it hard to track the progress of the field or to compare new ideas published in separate papers. Moreover, AutoML methods are often compared as a whole, as opposed to evaluating the contribution of each component through ablation studies. This stems from the difficulty of integrating new ideas in existing frameworks. Often, novel methods are instead presented in new AutoML frameworks. This greatly increases the amount of work required to develop and evaluate a novel method and obfuscates its contributions.

In this thesis, we present the research and development of tools that facilitate novel, correct, and reproducible AutoML research. Our hope is that this accelerates both the rate and quality of future research.

To address the difficulty of exploring novel ideas in AutoML, we present the modular AutoML tool **GAMA** (a General Automated ML Assistant). It features a modular design that allows for evaluating the contributions of individual components in the AutoML pipelines by systematic ablation studies. **GAMA** features several asynchronous optimization methods out-of-the-box to make efficient use of compute resources during search, and new components may easily be developed independently of the rest of the AutoML pipeline. Additionally, **GAMA** automatically tracks experiments and compiles data that researchers can use to better understand the workings of individual components, e.g., by visualizing

their optimization trace. The fact that **GAMA** has already been used in AutoML research for online learning, clustering, and comparing optimization strategies are early signs that the modular AutoML tool is valuable for research.

To allow for reproducible and comparable evaluations, we recognize the need for curated and standardized benchmarks. To this end, we extend the OpenML platform to enable creating, sharing and re-using *benchmark suites*. A benchmark suite is a collection of precise and machine-readable definitions of machine learning experiments, including information about the dataset, evaluation strategy, and performance metric. A good benchmark suite, if used by the community, allows not only for a thorough evaluation but also for the comparison of results across papers. We propose a practical benchmarking suite for ML algorithms, which has been used in several studies. Its use indicates that benchmarking suites are useful but also that a continuous conversation with the research community is essential to evolve the benchmarks over time to make them better and more useful.

We propose the open source AutoML benchmark, and use it to conduct a large-scale evaluation of AutoML frameworks. The benchmarking tool prepares the experimental setup, and scripts developed together with authors of AutoML frameworks ensure that the training and evaluation of each framework are done correctly. The tool greatly reduces the effort required to produce reproducible results and at the same time avoids issues one may encounter when using (and installing) AutoML tools for experimental evaluation. The AutoML benchmark has grown to be an accepted benchmark, as many AutoML researchers and developers have proceeded to integrate their frameworks into the benchmark and use it for their empirical evaluations in scientific studies.

Finally, we propose a meta-learning method to find symbolic hyperparameter defaults, which may allow AutoML methods to find good models faster. The usefulness of hyperparameter optimization on each separate dataset motivates that there is a relationship between the dataset properties and the optimal hyperparameter configuration, yet most hyperparameter defaults currently employed are independent of dataset properties. We propose a method based on symbolic regression to automatically find such relationships, which we call symbolic hyperparameter defaults, in a data-driven way. We show that our method is capable of finding symbolic hyperparameter defaults which are as good as hand-crafted ones, at least as good as constant hyperparameter defaults, and in almost all cases better than current implementation defaults.

Contents

Acknowledgements	iv
Summary	vii
List of Figures	xii
List of Tables	xiv
1 Introduction	1
1.1 Automated Machine Learning	3
1.2 Meta-learning	5
1.3 Challenges and Research Questions	7
1.4 Thesis Outline and Contributions	9
2 Automated Machine Learning	13
2.1 Problem Definition	14
2.2 Search Space Design	16
2.3 Search Strategies	17
2.3.1 Grid- and Random Search	18
2.3.2 Evolutionary Algorithms	19
2.3.3 Bayesian Optimization	22
2.3.4 Successive Halving and Hyperband	24
2.3.5 Other Methods	27
2.4 Post-Processing	29
2.4.1 Weighted Voting	29
2.4.2 Stacking	29
2.4.3 Model Information	30

2.5	AutoML in Other Settings	31
2.5.1	Online Learning	31
2.5.2	Unsupervised AutoML	32
2.5.3	Multi-Label Classification	32
2.5.4	Remaining Useful Life Estimation	33
3	GAMA - Modular AutoML	35
3.1	Related Work	36
3.2	The Modular AutoML Pipeline	37
3.2.1	Search	37
3.2.2	Post-processing	43
3.2.3	Configuring an AutoML Pipeline	44
3.3	Accelerating Research	45
3.3.1	Interface	46
3.3.2	Artifacts	46
3.4	Use in Research	50
3.4.1	Online AutoML	50
3.4.2	Multi-fidelity Evolution	50
3.4.3	Clustering	51
3.5	Conclusion, Limitations, and Future Work	51
4	Reproducible Benchmarks	53
4.1	OpenML	54
4.2	OpenML-Python	55
4.2.1	Design and Development	56
4.2.2	Related Work	57
4.2.3	Use Cases	57
4.3	Benchmarking Suites	62
4.3.1	OpenML Benchmarking Suites	62
4.3.2	How to Use OpenML Benchmarking Suites	64
4.3.3	OpenML-CC18	67
4.4	Conclusion and Future Work	71
5	The AutoML Benchmark	73
5.1	Related Work	74
5.2	AutoML Tools	77
5.2.1	Integrated Frameworks	77
5.2.2	Baselines	81
5.3	Software	81

5.3.1	Extensible Framework Structure	82
5.3.2	Extensible Benchmarks	83
5.3.3	Running the tool	83
5.4	Benchmark Design	84
5.4.1	Benchmark Suites	85
5.4.2	Experimental Setup	88
5.4.3	Limitations	89
5.4.4	Overfitting the Benchmark	91
5.5	Results	94
5.5.1	Performance	94
5.5.2	BT-Trees	96
5.5.3	Model Accuracy vs. Inference Time Trade-offs	99
5.5.4	Observed AutoML Failures	100
5.6	Conclusion and Future Work	103
6	Meta-Learning for Symbolic Hyperparameter Defaults	105
6.1	A Motivating Example	106
6.2	Related Work	107
6.3	Problem Definition	108
6.3.1	Supervised Learning and Risk of a Configuration	108
6.3.2	Learning an Optimal Configuration	109
6.3.3	Learning a Symbolic Configuration	109
6.3.4	Metadata and Surrogates	110
6.4	Finding Symbolic Defaults	114
6.4.1	Grammar	115
6.4.2	Algorithm	115
6.5	Experimental Setup	116
6.5.1	General setup	116
6.5.2	Experiments for RQ1 & RQ2	118
6.6	Results	120
6.6.1	Surrogates and Surrogate Quality	120
6.6.2	Experiment 1 - Benchmark on surrogates	121
6.6.3	Experiment 2 - Benchmark on real data	124
6.7	Conclusion and Future Work	126
7	Conclusion and Future Work	129
7.1	Conclusions	129
7.2	Limitations	132
7.3	Future Work	133

7.3.1	Meta-learning for AutoML	133
7.3.2	Benchmark Design	134
7.3.3	Trust in AutoML	135
Bibliography		137
Appendices		167
List of Publications		195
SIKS Dissertations		197

List of Figures

1.1	Visualization of models generated by different ML pipelines. . . .	3
1.2	An overview of the thesis.	10
2.1	Typical building blocks of AutoML approaches.	14
2.2	An illustration of grid search and random search.	18
2.3	An illustration of the $(\mu + \lambda)$ -algorithm.	20
2.4	Cross-over for two genetic programming trees.	21
2.5	Two steps of Bayesian optimization on a 1D function.	23
2.6	Illustration of successive halving.	25
3.1	The benefit of asynchronous evaluations.	38
3.2	Performance comparison of GAMA and TPOT	42
3.3	Critical difference plot of AutoML benchmark results.	46
3.4	Visualization of logs	47
3.5	Evolutionary optimization on Higgs on a one hour time budget. .	49
3.6	ASHA on a one hour time budget with reduction factor 3.	49
3.7	Comparison of convergence for ASHA and EA on Higgs.	49
4.1	Schematic overview of OpenML building blocks.	55
4.2	Contour plot of SVM performance based on hyperparameter configurations.	60
4.3	Website interface for OpenML benchmarking suites.	63
4.4	Distribution of scores of millions of experiments on OpenML-CC18. .	68
5.1	Properties of the tasks in both benchmarking suites.	93
5.2	Critical difference plots for all experiments.	95
5.3	Aggregated scaled performance for all experiments.	97

5.4	Bradley-Terry tree for the one hour classification benchmark. . .	98
5.5	Prediction duration aggregated across all runs.	100
5.6	Pareto fronts of framework performance to prediction speed. . .	101
5.7	An overview of framework errors in the benchmark.	102
5.8	Time spent during AutoML search by each framework.	103
6.1	SVM hyperparameter response before and after symbolic scaling.	107
6.2	Correlation between SVM surrogate predictions and real data. .	121
6.3	Comparison of found SVM defaults to static and implementation defaults.	122
6.4	Comparison of defaults across learner algorithms.	123
6.5	Comparison of found symbolic hyperparameter defaults to con- stants and random search.	124
6.6	Comparison of symbolic and implementation defaults with eval- uations on datasets.	125
B.1	A BT tree generated with only ‘features’ and ‘instances’ for split criteria, based on all results for one hour experiments.	180
B.2	A BT tree generated with only ‘features’ and ‘instances’ for split criteria, based on all results for four hour experiments.	181
C.1	Results for the elastic net algorithm on surrogate data.	187
C.2	Results for the decision tree algorithm on surrogate data. . . .	188
C.3	Results for the approximate k-nearest neighbours algorithm on surrogate data.	189
C.4	Results for the random forest algorithm on surrogate data. . .	190
C.5	Results for the XGBoost algorithm on surrogate data.	191
C.6	Results for the decision tree algorithm on real data.	192
C.7	Results for the Elastic Net algorithm on real data.	192

List of Tables

3.1	Comparison of most closely related AutoML work.	36
5.1	Used AutoML frameworks in the experiments.	78
6.1	BNF Grammar for symbolic defaults search.	112
6.2	Available meta-features with corresponding symbols	113
6.3	Fixed and optimizable hyperparameters for different algorithms.	117
6.4	Mean normalized log-loss (standard deviation) across all tasks with baselines.	125
A.1	Tasks OpenML-CC18.	171
A.2	Tasks in the AutoML regression suite.	173
A.3	Tasks in the AutoML classification suite.	175
B.1	An overview of errors for framework (A-H).	183
B.2	An overview of errors for framework (I-Z).	184
C.1	Existing defaults for algorithm implementations.	186

Chapter 1

Introduction

Data is used every day to make informed decisions or discover new insight. The digitalization of our world has contributed greatly to the amount of data that can be collected, which in turn greatly increased the demand to make sense of that data. *Machine Learning* (ML) algorithms have been used to great effect to rise to this demand since with them computers can identify patterns in the data automatically. More formally, an often used definition of ML is given by Mitchell [167]:

A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P , if its performance at tasks in T , as measured by P , improves with experience E .

For example, in disease diagnosis, the task T is determining whether or not the patient has a certain disease, the performance P is the percentage of correct diagnoses, and experience E is the experience with past patients. Provided with enough high quality and relevant data, ML can find useful models across a wide range of domains automatically and is now used to make or assist in decisions in various applications including medicine [137, 241], self-driving cars [8], and reading recommendations [31], while new applications continue to be explored [251].

Unfortunately, you can't use any ML algorithm for any problem and expect a useful outcome. Creating a good ML model requires many interdependent steps, such as data cleaning (e.g., encoding of categorical variables or imputation of missing values), feature extraction (e.g., PCA), feature engineering (e.g., lag features in temporal problems), and choosing the learning algorithm (e.g.,

SVM [32]). These steps are combined into an *ML pipeline*, a series of steps that build an ML model from the original data. Each of those steps have hyperparameters to be tuned, and the effectiveness of a hyperparameter configuration or algorithm choice depends both on the data and the other design choices in the ML pipeline. All of these decisions affect not only model performance but also other aspects like the model’s interpretability or the time it takes to make predictions on new data.

Figure 1.1 demonstrates the importance of tuning hyperparameters and using appropriate preprocessing on a synthetic dataset, through visualizing the models created by several ML pipelines for a binary classification problem. The dataset is visualized through dots, whose color represents their class. The model’s decision boundary is drawn, and the predicted class is indicated by the background color. The accuracy of each model is computed through 5-fold cross-validation (CV) [202]. Logistic regression [41] (top-middle) outperforms a badly tuned decision tree [36] (bottom-left) but not a well tuned one (bottom-middle), demonstrating the importance of both algorithm selection and hyperparameter optimization. In this scenario, encoding the discrete feature (on the vertical axis) with target encoding [164] (top-right) is detrimental to the performance of the decision tree (bottom-right), but in general target based encoding is highly effective for high cardinality features [180].

In conclusion, many algorithms can be used as components in ML pipelines which all have their own hyperparameters to tune. Creating a useful model requires expertise about the data, the algorithms, and how to tune them.

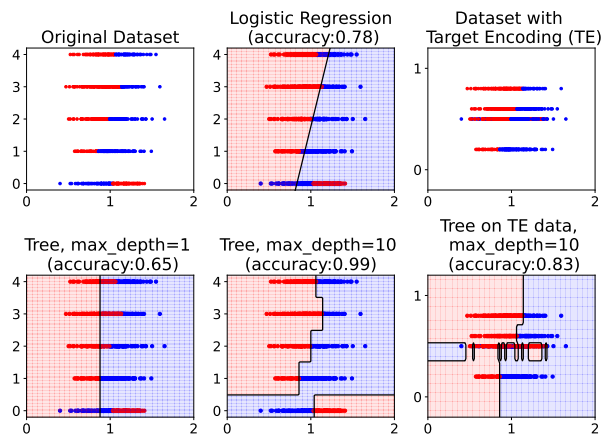


Figure 1.1: A visualization of models generated by different ML pipelines, which shows the importance of algorithm selection and hyperparameter optimization. Each dot is a data point and the color represent their class. The background color denotes the model’s class prediction.

1.1 Automated Machine Learning

The previous section highlighted some of the difficulties of creating an effective ML pipeline. The complexity of creating good ML models is identified as a hurdle for its application [264]. Automated Machine Learning (AutoML) aims to take away this hurdle by automating the design decisions for creating an ML pipeline in a data-driven way [120]. The first formal definition of AutoML was, to the best of our knowledge, in 2009 by Escalante, Montes, and Sucar [75] under the name *full model selection*. It was later re-introduced as *combined algorithm selection and hyperparameter optimization* (CASH) [238], and finally as AutoML for the first AutoML workshop at ICML in 2014¹.

Automating ML pipeline construction democratizes ML by providing an easy-to-use interface with which novice users can create ML models without needing an expert understanding of ML algorithms. For experts, AutoML frees up time for other tasks e.g., it allows them to spend more time understanding

¹<https://sites.google.com/site/automlwsicml14/>

the data and models, or to scale up and develop more ML solutions [249]. AutoML has many characteristics which make it a difficult problem from both an optimization and engineering perspective. Here follows a concise overview, but a more in-depth review will be given in Chapter 2.

The difficulty starts with the data which may come from various domains (semantic differences), from various sources (for example, human data-entry results in different types of errors than sensor data), span orders of magnitude in size, and use different data types such as numerical or categorical data. Moreover, depending on the application of the model, concerns for fairness or interpretability may be even greater when the AutoML user may also lack the knowledge to assess the model adequately. Because of these diverse requirements, some tools opt to profile themselves for specific domains e.g., for medical data [3, 245] or finance applications [249], though they can still be used in other contexts. Even when datasets used to develop the AutoML tool are similar to those they are tested on, it can be difficult to create robust tools. In a Lifelong Learning AutoML challenge [76] roughly 40% of submissions failed to produce results on test datasets even though they were similar to train datasets and evaluated under similar hardware constraints.

The optimization of ML pipelines is also difficult. To perform pipeline optimization, AutoML draws on a rich literature for algorithm selection and hyperparameter optimization [27]. Many optimization algorithms have been used to optimize ML pipelines, for example particle swarm optimization [70] by Escalante, Montes, and Sucar [75], sequential model-based algorithm configuration (SMAC) [119] in **Auto-WEKA** [238], genetic programming [10] in **TPOT** [179], hierarchical planning [74] in **ML-Plan** [169], and random search in **H2O AutoML** [148]. Given a *search space*, which denotes the space of all allowed ML pipelines, and a way to evaluate an ML pipeline, such as measuring model accuracy through cross-validation, the optimization algorithm aims to find the best pipeline. The optimization algorithm repeatedly selects one or more pipelines to evaluate based on the evaluations that came before. Nonetheless, optimization remains difficult because it is a black-box optimization problem, pipeline evaluations are typically expensive, and the number of different hyperparameters leads to a combinatorial explosion. To lower the cost of evaluations, multi-fidelity approaches have been explored through using less data [99] or using iterative algorithms which may only fit a few iterations at a time [82]. Considerably smaller search spaces from which to design ML pipelines are also considered e.g., containing only iterative learners [82] or even a single learner [237]. After optimization of ML pipelines, several post-processing techniques are available to combine those pipelines into a combined model through e.g., weighted voting [48, 49]

and stacking [253].

A considerable amount of recent work in AutoML focuses on Neural Architecture Search (NAS) [72], the automated design of neural networks. This has mostly been a separate endeavor from automated ML pipeline construction, both in approach and the tasks they currently aim to solve. Whereas AutoML for ML pipelines is typically used to solve tabular data problems, neural networks tackle less structured problems such as computer vision or natural language processing. NAS can design their search procedures around the properties of neural networks e.g., the hierarchical structure by designing subcomponents (cells) [283, 286] or the sharing of network weights [189]. In the remainder of this thesis, we focus on automated ML pipeline construction, and ‘AutoML’ will refer to that particular task.

1.2 Meta-learning

So far, we only considered finding good pipelines by performing only evaluations on the dataset at hand. Human experts don’t work this way, but leverage experience they have from creating models on other tasks and in this way learn to optimize ML pipelines on new tasks faster. This is called *meta-learning* [257]:

The challenge in meta-learning is to learn from prior experience in a systematic, data- driven way [...] to extract and transfer knowledge that guides the search for optimal models for new tasks.

For example, Brazdil, Gama, and Henery [33] train a model that recommends classification algorithms based on tabular dataset characteristics. They achieve this through training a decision tree model on a *meta-dataset*, which contains data about the performance of algorithms across datasets. In the meta-dataset, each dataset is described through *meta-features*, such as the number of rows or classes, and each algorithm’s performance as either applicable, if it is within three standard deviations of the performance of the best classifier on that dataset, or non-applicable otherwise. More generally, meta-datasets can include hyperparameter configurations or entire ML pipelines definitions, and their performance can be metric scores (e.g., accuracy) or other meta-data such as training time.

Similar setups are explored where the produced predictions are generalization estimates [14, 61, 108, 204], or rankings [34, 35, 226]. Sometimes the time to train a model is taken into account [35, 226], or the meta-model discerns not just algorithms, but also specific hyperparameter configurations [61, 108].

Initially, meta-features were simple statistical and information-theoretic metrics such as described by Michie, Spiegelhalter, and Taylor [165], but later other features were introduced, such as landmarking features which record the performance of simple learners [187] and model-based features which describe properties of models induced by simple learners [185], and modern packages can calculate hundreds of meta-features [5]. Recent work explores learning meta-features automatically [63, 125, 132, 198]. One important consideration for meta-features, in addition to their usefulness, is how efficiently they can be computed. The time saved from using the meta-model should exceed the time spent to compute the meta-features. How this translates to constraints depends on the application.

Meta-learning can be used to speed-up AutoML through *warm-starting*, where instead of starting optimization by sampling configurations at random, a meta-model is employed to recommend pipelines based on the dataset. This is employed in, for example, **auto-sklearn** through k-NN [85] and through collaborative filtering in OBOE [279].

In **auto-sklearn 2** [82], optimization is warm-started with a portfolio instead. This portfolio is a static collection of pipelines that performed well on previous tasks. Despite not taking into account dataset specific meta-features, it still provides the same performance benefit while being simpler [82]. Additionally, **auto-sklearn 2** uses meta-learning to automatically configure AutoML hyperparameters, such as the evaluation procedure used (e.g., hold-out or cross-validation), by learning over a meta-dataset which contains evaluation data of **auto-sklearn 2** itself [82]. Meta-models can also be used during the optimization procedure, for example, to prohibit a pipeline candidate from being evaluated. Mohr et al. [170] and Laadan et al. [144] use meta-models to prohibit pipelines evaluations that are expected to take too long or provide bad results, respectively.

Instead of implicitly learning the importance of hyperparameter values for the final model performance, it can also be made explicit through the variance they induce [118, 255] or the performance that can be gained by tuning them [196, 267]. These types of studies can be used to inform the search space design of AutoML systems.

Meta-learning can also be used to transfer information about parameters instead of hyperparameters. While this is possible for several model classes, much of the research focuses on transfer learning for neural networks which can share weights or architectures [257]. Examples include using features extracted by networks trained on one task to solve other tasks [221], learning weights that generalize well to allow quick learning of other tasks [89], or using a neural

network to train other neural networks [115].

1.3 Challenges and Research Questions

AutoML is a very active area of research, but exploring novel AutoML ideas is very time-intensive and evaluating those ideas is error-prone. This thesis focuses on the research and development of tools that facilitate novel, correct, and reproducible AutoML research. We hope that this accelerates both the rate and quality of future research. Finding the answers to the research questions asked in this section contributes to this overarching goal.

Q1: How can we make implementing novel AutoML ideas easier?

To explore a novel AutoML idea, a researcher has to decide whether to develop a new AutoML tool or use an existing one as a springboard. When developing a new tool, the evaluation of the novel idea requires other aspects of the AutoML pipeline, such as interpreting a search space, to be implemented as well. Even then, implementing an AutoML tool from scratch to evaluate a new search algorithm diminishes the capability to compare with other implementations, as it now also differs in implementation and potentially other design decisions.

On the other hand, using an existing tool as a springboard to explore a novel idea is hard too, as existing tools are not generally designed to be open to integrating new algorithms. Including a novel optimization algorithm (or another component) often involves a steep learning curve and may require considerable alterations to the original AutoML tool. Additionally, if the original authors of the AutoML tool are not convinced of the added value, the modifications may never be absorbed into the tool.

Q2: How can we enable the use of common benchmark suites?

Every novel idea needs to be carefully evaluated. A thorough analysis requires evaluations on multiple datasets to adequately assess its generalizability and to identify the strengths and weaknesses of the new approach [219]. However, datasets used in evaluations are typically chosen in an ad-hoc manner. This leads to evaluations across papers being performed on different datasets making comparison impossible. For example, AutoML tools [68, 105, 199] were all published at the same venue and evaluated on different datasets. The lack

of common benchmarks can also lead to ill-suited benchmarks being propagated. Roughly a third of the datasets for evaluation of tabular AutoML tools by Thornton et al. [238], Feurer et al. [85], and Mohr, Wever, and Hüllermeier [169] were image datasets, despite not being representative of the intended use of the respective AutoML tools. Finally, it is not always clear how to reproduce the results of AutoML evaluations, as the used datasets may be scattered across repositories or are without clearly documented validation splits. There are clear benefits to using a shared collection of curated tasks or, in other words, a benchmark suite. It allows for better comparison across papers, both for simultaneous publications and over time. Ideally, it can also lead to fewer resources being required to conduct a study, since previous results can be compared to directly without the need for additional evaluations.

Q3: How to evaluate AutoML tools in a correct and reproducible manner?

Being able to use common benchmark suites makes the experimental setup easier. However, datasets with reproducible train-test splits alone are insufficient to produce a reproducible and correct setup. While AutoML frameworks typically provide a simple interface, we still identify several issues in the evaluations of AutoML frameworks in research [100]. These lead to incorrect conclusions about the comparative performance of the frameworks. Errors are often caused by incorrect installation or configuration, either because the hardware or software stack deviates from the developers' expectations, or because the tools are used outside of their intended use.

In AutoML research the use of benchmark suites can only provide part of the solution. Since AutoML tools often work with time budgets, their output is heavily influenced by the resources they have available during that time (e.g., memory or CPU). For this reason, we still need a way to allow researchers to evaluate the AutoML tools of other researchers on their own hardware, despite the pitfalls mentioned above.

Q4: How can we speed up AutoML by learning from prior experiments?

In Section 1.2 we discussed how meta-learning is used to speed up optimization by generating recommendations for algorithms, pipelines, and hyperparameter configurations. However, in all those settings a trained learner is required. From a practical standpoint, this can be problematic when trying to share the learned information, because the model can be of considerable size or require specific software to generate new recommendations. This limits its use in machine learning packages and across different AutoML tools. We observe

that for each dataset we have to tune the hyperparameters of learners because there is a relationship between the dataset and the hyperparameter configuration that produces the optimal model for that learner. The meta-model is in effect a mapping that aims to transform the dataset characteristics to the ideal hyperparameter configuration for a learner. We postulate that we can also express this relationship explicitly for each hyperparameter by using *symbolic hyperparameter defaults*, defaults that map dataset characteristics to a valid hyperparameter value, and find them in a data-driven way.

Symbolic hyperparameters defaults should then only have to be found once for a specific algorithm, and could ideally come packaged with that algorithm. The symbolic hyperparameters defaults may also provide insight into the relationship between the hyperparameter and the dataset. While implementation differences might influence the ideal symbolic hyperparameter default, it is still likely that the default transfers reasonably well across implementations, e.g., from `mlr3`'s [145] to `scikit-learn`'s [184] decision tree. The model-free approach allows it to be used in all AutoML frameworks for e.g., warm-starting search or transforming the search space, and with additional experiments, symbolic hyperparameter defaults might even be learned for AutoML systems themselves.

1.4 Thesis Outline and Contributions

In this section, we will detail our contributions chapter-by-chapter, illustrated by the high-level overview of our contributing chapters in Figure 1.2. After providing related background information in Chapter 2, we present our contributions to answering research questions 1 through 4 in Chapters 3 through 6, respectively. The first three of those chapters directly contribute to correct and reproducible AutoML research and come with software artifacts that may be used for independent research: a modular AutoML tool, machine readable benchmarking suites, and an AutoML benchmark, respectively. The work in Chapter 6 details a meta-learning approach to finding *symbolic hyperparameter defaults*, which may be used to speed up AutoML in future work.

First, we will provide a more in-depth overview of the AutoML literature in Chapter 2. We first give a formal definition of the AutoML problem, which is followed by a discussion of the different design axes of AutoML systems, such as search space design, optimization algorithms, and post-processing used in AutoML. Then, we briefly discuss some of the work outside of the typical regression and single-label classification setting. The chapter's aim is not only

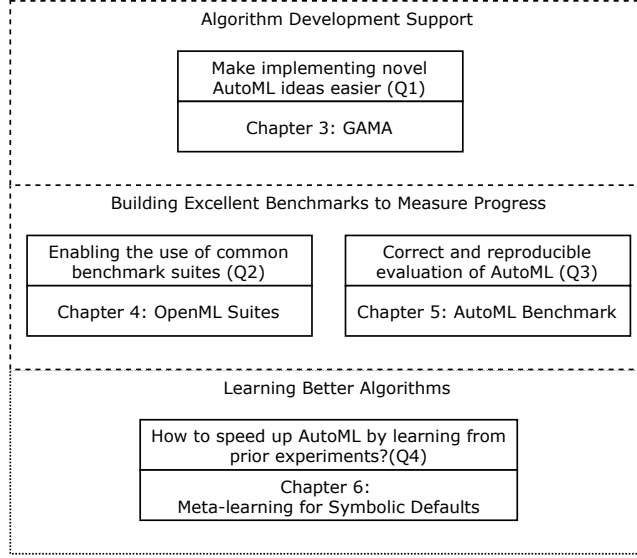


Figure 1.2: An overview of the thesis structure. Chapters 3 through 5 detail our contributions to correct and reproducible AutoML research (Q1-Q3). Chapter 6 presents an approach to learn *symbolic hyperparameter defaults*, which may be used to speed up AutoML in the future (Q4).

to provide a better understanding of the techniques currently employed but also to provide a stronger context for the difficulty of development and research of AutoML systems.

In Chapter 3, we introduce our answer to Q1 in the form of the General Automated Machine learning Assistant (GAMA [103, 104]), a tool to address the difficulty of exploring novel ideas in AutoML. As discussed in the last section, developing a completely new AutoML tool just to evaluate a novel idea adds a lot of overhead and additionally can lead to less informative experimental results. Using an existing tool allows for better comparisons, but comes with a steep learning curve and risks the new idea not being integrated by the original authors for public releases. GAMA features a modular and flexible design, which allows researchers to write or modify individual components of the AutoML pipeline easily. This does not only allow much faster iterations over new ideas but also allows for better comparison. We review other work built with GAMA and see

early signs that the modular AutoML tool is valuable for research.

In Chapter 4, we examine different platforms for sharing data and machine learning experiments, and motivate the choice to build on OpenML [258]. We build a programmatic interface to the platform called `openml-python` [87], which enables further automation of downstream tasks which greatly increases the ease with which reproducible experiments can be conducted. For example, it is possible to automatically download datasets alongside meta-data to conduct reproducible 10-fold cross-validation. By enabling the development of comprehensive benchmarking suites on the platform [28], we allow researchers to identify collections of interesting tasks and to share them. We believe that the ease with which benchmarking suites can now be shared and reproduced greatly contributes to the use of high-quality tasks in evaluations, and show early signs that might confirm this (Q2).

Next, we build on that to address Q3 and create the AutoML benchmark [100] which we present in Chapter 5. The AutoML benchmark introduces a benchmarking tool for completely automated AutoML evaluations. To achieve this, we work together with the authors of AutoML frameworks and integrate with OpenML through `openml-python`. We present two benchmarking suites for benchmarking AutoML frameworks, one classification and one regression suite, and survey the current AutoML landscape through large-scale evaluation of AutoML frameworks. Since its initial presentation in 2019 [100], the AutoML community has used the benchmark extensively, both integrating AutoML frameworks and using the suites for large-scale evaluations.

In Chapter 6, we develop a method for finding symbolic hyperparameter defaults using meta-learning [102]. We use symbolic regression to optimize symbolic hyperparameter values for multiple hyperparameters of a learner jointly and do so for 6 different learners. Because symbolic regression relies on many evaluations, we use surrogate models to make optimization tractable. We compare the performance of the found default values to implementation defaults both on the surrogate models and through experiments on real data. The automatically designed symbolic hyperparameter defaults can match hand-crafted symbolic hyperparameter defaults and outperform the current constant defaults.

We summarize the work and discuss open challenges and future work in Chapter 7.

Chapter 2

Automated Machine Learning

In this chapter we give a more thorough introduction to AutoML for tabular datasets. We will first give a definition of the AutoML problem in Section 2.1. The most common approach to tackle the problem is to iteratively explore the search space and optionally perform a post-processing step, as is visualized in Figure 2.1. For that reason, we structure the three sections following the problem statement in that order. First, we review work on search space design, then we cover search and evaluations strategies together, and finally we discuss ways to use post-processing to create a final model.

In the remainder of the chapter we discuss the various settings in which AutoML has been researched. Subsequent chapters will detail our contributions. Each of those chapters will discuss additional literature that is relevant to that chapter.

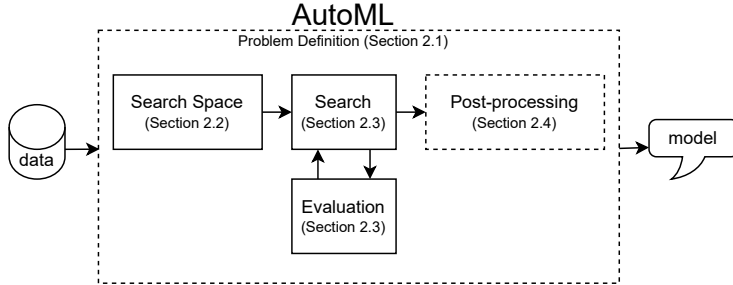


Figure 2.1: Typical building blocks of AutoML approaches.

2.1 Problem Definition

The AutoML problem has been (re)formulated many times. There are many mathematical formulations which broadly have the same meaning as the following definition of *full model selection* [75]:

Given a pool of preprocessing methods, feature selection and learning algorithms, select the combination of these that obtains the lowest error for a given data set.

Mathematical definitions with a similar intent often define the problem as a direct extension of the hyperparameter optimization problem by encoding the choice of algorithms used as additional hyperparameters [27, 238], which is also known as the Combined Algorithm Selection and Hyperparameter optimization (CASH) problem [238]. In some cases authors explicitly make a distinction between preprocessing algorithms, which transform a dataset into another dataset, and learners, which learn to predict labels for a dataset [3, 169]. However, these definitions require a liberal interpretation to generalize across implementations. For example, the paper which introduced **auto-sklearn** [85] adopts the CASH [238] formulation and uses sequential model-based algorithm configuration (SMAC) [119] to tune pipelines. However, after optimizing over the pipeline space the resulting models get combined into an ensemble as described in [48, 49], which only fits the CASH definition by a very liberal interpretation of the notion of an algorithm and indicator hyperparameters. For this reason, as far as mathematical formulations go, we prefer the interpretation of AutoML as optimizing a directed acyclic graph (DAG) of operations as given through

a series of definitions by Zöller and Huber [285], which we will use in adapted form:

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

$$(g, \mathbf{A}, \boldsymbol{\lambda})^* \in \arg \min_{g \in G, \mathbf{A} \in \mathcal{A}^{|g|}, \boldsymbol{\lambda} \in \Lambda} \mathcal{R}(\mathcal{P}_{g, \mathbf{A}, \boldsymbol{\lambda}}, \mathcal{D}). \quad (2.1)$$

within a given resource budget B .

where:

- $g \in G$ is a graph from the set of all valid graphs,
- $\mathbf{A} \in \mathcal{A}^{|g|}$ is a vector which for each node in graph g specifies the algorithm from the set of algorithms \mathcal{A} ,
- $\boldsymbol{\lambda} \in \Lambda$ is a vector specifying the hyperparameter configuration of each algorithm from the set of all possible configurations,
- and B is a resource budget, which may be given as e.g., time or iterations.

\mathcal{R} is the empirical risk of the pipeline $\mathcal{P}_{g, \mathbf{A}, \boldsymbol{\lambda}}$ according to some evaluation procedure. For example, the root mean square error of the predictions of pipeline $\mathcal{P}_{g, \mathbf{A}, \boldsymbol{\lambda}}$ for a validation set $\mathcal{D}_v \subset \mathcal{D}$ after being trained on $\mathcal{D}_{train} = \mathcal{D} \setminus \mathcal{D}_v$. \mathcal{R} may also be defined over multiple objectives in which case a Pareto optimal set of pipelines is to be found.

We purposely do not define specific characteristics for \mathcal{D} , so that the definition generalizes beyond single-label classification and regression to e.g., multi-label classification and clustering. When we refer to the AutoML problem in this work, we refer to the above definition.

Note that this definition is still quite narrow, specifically only formalizing the automated optimization of machine learning pipelines, and geared towards a quantitative assessment of final model performance. In a broader sense, AutoML systems may also be understood to automate other tasks in the ‘machine learning engineering pipeline’ [206, 213, 276], including exploratory data analysis, reports on model quality and interpretability, and model deployment. Santu et al. [213] define multiple levels of AutoML based on which steps are automated

and consequently how much help a domain expert would need from an ML expert in order to produce ML models. They reference current work that automates some of these steps independently, and also provide additional directions for research, such as computer-assisted task formulation (specifying exactly what the ML model has to learn). In a qualitative comparison, Xanthopoulos et al. [276] find that multiple AutoML frameworks automate more than just pipeline design, for example, providing automated interpretability reports or data visualization, but none of the systems cover full end-to-end automation. Additionally, they define several qualities beyond automation, such as the quality of documentation and support, or the ability to integrate with other systems. While we acknowledge that the automation of other parts of the ‘machine learning engineering pipeline’ is interesting and important work, this work focuses primarily on automated pipeline design.

2.2 Search Space Design

The search space is the space of all possible pipelines an AutoML system can create, or in terms of the pipeline creation problem it is the set $\{(g, \mathbf{A}, \boldsymbol{\lambda}) \mid g \in G, \mathbf{A} \in \mathcal{A}^{|\mathcal{g}|}, \boldsymbol{\lambda} \in \Lambda\}$. Search space design is then the act of picking G , \mathcal{A} and Λ . AutoML search spaces are very large and hard to optimize over, as discussed in Section 1.1. A well designed search space should allow for (near)-optimal pipelines on as wide a range of tasks as possible. On the other hand, keeping the search space small makes it easier to explore the search space and perform meaningful optimization. As an example of this, Sá et al. [212] showed that statistically significant different results could be obtained by matching the search space for their method to match that of another system being compared to. Modifying the search space may also be used to enhance other aspects of the final solution, such as inference time or interpretability. For example, it may be desirable to use only linear models and decision trees.

The set of allowed pipelines G is often a subset of DAGs such as a linear pipeline of fixed length, e.g., **Auto-WEKA** [238], or of variable length, e.g., **ML-Plan** [169], or a tree, e.g., **TPOT** [179]. Most tools [73, 85, 103, 148, 249, 265] allow for a multi-phase approach where two subsets of G are explored in succession, e.g., **auto-sklearn** [85] first optimizes fixed-length linear pipelines and then builds an ensemble with a subset of the evaluated pipelines in a post-search step, effectively creating a tree-graph model without considering the full search space of all trees. In the above examples, G is only indirectly modifiable by choosing whether or not to perform a post-search step. In some cases G is

directly modifiable by the end user, for example by providing a template of the desired ML pipeline [147].

The set of algorithms \mathcal{A} and their hyperparameters Λ are the other axes along which the search space can be designed. This is one of the main parts where ML experts can insert prior knowledge into the AutoML system, defining the most useful algorithms and hyperparameter ranges. For example, to allow TPOT to perform well on big biomedical data, a feature selector step was introduced which allows the domain expert to identify meaningful subsets of the data [147], e.g., specific genes in a gene expression analysis, and TPOT will then identify the most appropriate subset in its AutoML process.

Wistuba, Schilling, and Schmidt-Thieme [270] use meta-learning to automatically prune Λ for Bayesian optimization strategies, which are discussed in Section 2.3.3. First, they create surrogate models to predict the performance of hyperparameter configurations on a number of tasks. To prune the search space for a new task a number of related tasks is first identified. Based on the performance estimates of their surrogate models, the regions in the search space which are expected to perform poorly are pruned. This method may even be used to further prune the search space during search based on already evaluated ML pipeline designs.

Hyperparameter defaults are also a part of the search space, and may be used implicitly or explicitly. Implicitly, the hyperparameter defaults for hyperparameters which are not tuned, and thus left at their default value, may affect which configurations are optimal and how good the optima are. Explicitly, the knowledge embedded in the choice of hyperparameter default can be exploited, for example by sampling around the default values [278]. Additionally, Anastacio, Luo, and Hoos [6] show that some hyperparameter optimization strategies are more sensitive to defaults than others, and using the default values to shrink the search space may lead to better results.

2.3 Search Strategies

One of most distinct differences between AutoML systems is the optimization algorithm they employ to perform pipeline search. Here we will briefly discuss a few frequently used optimization algorithms. For a more comprehensive overview on hyperparameter optimization techniques see [27, 84].

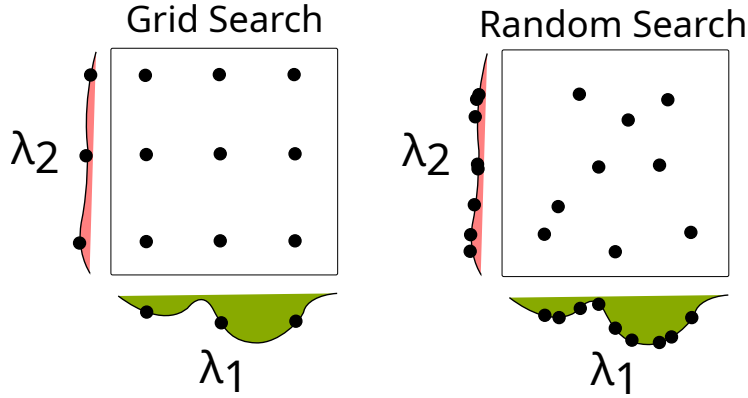


Figure 2.2: An illustration of grid search (left) and random search (right). Random search explores more values in each dimension which means that, unlike grid search, it stays efficient even when the effective dimensionality is low. Figure based on Bergstra and Bengio [17].

2.3.1 Grid- and Random Search

One naive approach to finding the best pipeline is to perform an exhaustive search. Continuous hyperparameters make a true exhaustive search impossible, but after discretizing the search space it is possible to create a grid containing each pipeline and evaluate them all. However, the number of possible pipelines grows exponentially with the number of hyperparameters and algorithms, so this quickly becomes infeasible. Additionally, grid search’s anytime performance is also influenced by the order in which they explore the different hyperparameters.

Bergstra and Bengio [17] showed that random search is better suited than grid search for hyperparameter optimization. An illustrative example is given in Figure 2.2, which shows grid search (on the left) and random search (on the right) optimizing two hyperparameters (λ_1 and λ_2). The curves on the respective axes show the effect the different hyperparameter values have on the performance of the model. In practice, the effective dimensionality of the optimization problem is smaller than its true dimensionality as not every hyperparameter has meaningful influence on the model performance on every dataset (here, λ_2). For these hyperparameters grid search then needlessly optimizes their value, while random search at the same time also samples new values for other hyperparameters, making it more effective in practice. The advantage of

both methods is that they are trivially parallelizable since each evaluation is independent of all others. They are also easily understood and they don't have many design decisions.

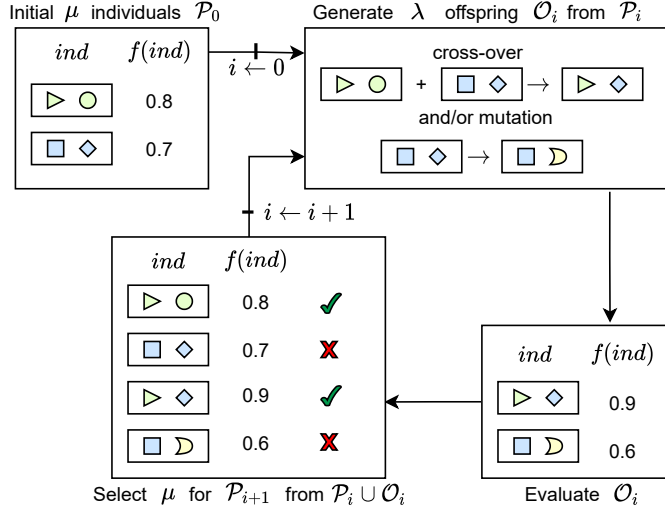
To the best of our knowledge, grid search is seldom used in AutoML tools, and only to optimize parts of the ML pipeline [192, 245]. Random search is used in e.g., H2O AutoML [148], though not as the only means to create pipelines. For example, H2O AutoML evaluates a pipeline portfolio before performing random search, and uses stacking afterwards.

2.3.2 Evolutionary Algorithms

Evolutionary algorithms are inspired by biological evolution, and simulate populations which evolve over time to perform better at a specific objective (or multiple objectives). In the context of AutoML, an evolutionary algorithm maintains a collection of ML pipeline candidates, also called a *population of individuals*. These individuals are assigned a *fitness* score based some evaluation function f (e.g., accuracy from k-fold cross-validation), and through the process of *cross-over*, *mutation* and *selection* the population changes over time. Genetic programming (GP) [140] is often used in AutoML [77, 103, 179, 190, 212], where the individuals are typically GP trees with algorithms as nodes and hyperparameter values as leaves.

Figure 2.3 illustrates the $(\mu + \lambda)$ -algorithm which is used in TPOT [179]. First, an initial population \mathcal{P}_0 of size μ is generated (step 0). This can be done at random but some form of warm-starting can also be used by creating an initial population with ML pipelines that worked well on previous tasks [144]. This initial population is evaluated, after which the following steps take place in a loop (i starts at 0 and increments by 1 every iteration):

- Step 1. λ new individuals, called *offspring* \mathcal{O}_i , are generated by selecting parents from the population \mathcal{P}_i and applying cross-over and/or mutation. Parents can be selected uniformly at random or (partially) based on their fitness.
- Step 2. Offspring \mathcal{O}_i is evaluated on function f , e.g., accuracy from k-fold CV.
- Step 3. μ individuals are selected based on their fitness from the total population of parents and offspring ($\mathcal{P}_i \cup \mathcal{O}_i$) to be the new parent population \mathcal{P}_{i+1} . In the case of the (μ, λ) strategy, individuals are selected only from \mathcal{O}_i .

Figure 2.3: An illustration of the $(\mu + \lambda)$ -algorithm.

Fitness Evaluations

To evaluate the fitness of a candidate, k-fold cross-validation is used where typically $k = 5$ and splits are fixed throughout the optimization procedure (TPOT [179], GAMA [103], GP-ML [190]). One deviation is found in RECIPE [212] where $k = 3$ and the splits are resampled every 5 generations to avoid overfitting. However, it is possible this is not required as Pilát et al. [190] report that even after re-evaluating their best solutions on resampled splits, even with different k , they did not find any performance drop that would indicate overfit solutions. In TPOT and GAMA the pipeline length is also computed as part of the fitness score for their multi-objective optimization. Křen, Pilát, and Neruda [142] report that using time as a secondary objective instead results in much faster pipelines, as expected, but may make optimization more susceptible to local optima, ultimately leading to worse results.

Selection, Mutation, and Cross-over

There are several design choices left open, such as the choice of selection strategies. Here we make a distinction between *survival selection*, which determines

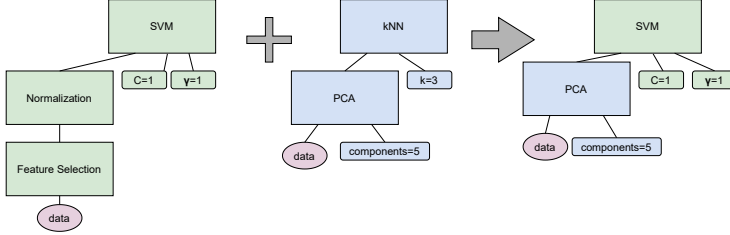


Figure 2.4: Cross-over for two genetic programming trees which represent ML pipelines. Nodes are algorithms, leave are hyperparameters or data.

how individuals from \mathcal{P}_i and \mathcal{O}_i are selected to form \mathcal{P}_{i+1} (step 3), and *parent selection* which determines how individuals are selected to generate offspring (step 1). Survival selection is typically elitist (TPOT, GAMA, RECIPE, GP-ML), carrying over the best solutions from $\mathcal{P}_i \cup \mathcal{O}_i$ in deterministic fashion. TPOT uses multi-objective NSGA-II [64] selection to maximize performance and minimize pipeline length, i.e., the number of algorithms in the ML pipeline which the individual represents. Varying parent selection schemes are used, including tournament selection in RECIPE and GAMA and uniform at random selection in TPOT. GAMA’s tournament selection uses the crowded comparison operator from NSGA-II, taking into account pipeline performance and length.

The *mutation* and *cross-over* operators govern how offspring is created from parents. Cross-over operators exchange subtrees in parents as shown in Figure 2.4. Here, the subtree exchanged includes the entire preprocessing pipeline, but more generally the subtree can be as small as the configuration for a single hyperparameter. Common mutation operators include changing hyperparameter values of one or more hyperparameters, growing or shrinking a subtree or replacing a node, i.e., an algorithm in the pipeline.

Asynchronous Evolution

The algorithm outlined above denotes *synchronous* evolution, where all offspring is evaluated before performing survival selection. In the context of AutoML, where different ML pipelines can have wildly varying runtimes spanning orders of magnitude [170, 279], this can lead to situation were resources are idle when waiting for stragglers when there are resources to parallelize the evaluation of ML pipelines. For this reason, GP-ML [190] and GAMA use an *asynchronous evolutionary algorithm* [218] which generates new offspring from the population

one at a time as resources are available. If the offspring outperforms the worst individual in the population it replaces it, otherwise it is discarded. Chapter 3 will discuss this variant of evolutionary optimization in more detail.

2.3.3 Bayesian Optimization

Bayesian optimization is an iterative optimization algorithm that is sample efficient, which makes it suitable for optimizing expensive functions such as finding the optimal ML pipeline design through empirical evaluations [223]. Bayesian optimization achieves its sample efficiency by building a surrogate model, which models the effect of the pipeline configuration on the model performance and the uncertainty of that estimate, and an acquisition function, which recommends the next configuration to sample based on the posterior distribution. Pseudocode for this procedure is given in Listing 1. Every iteration the acquisition function is used to find the next configuration to sample based on the posterior distribution (line 3). To build a useful surrogate model at least a few evaluated sample points are required, so early on random sampling or configurations recommended through meta-learning may be used instead [82, 88]. After a configuration is evaluated, results are stored and used to update the surrogate model (lines 4-6). This repeats until some stopping criterion is met.

Figure 2.5 illustrates this procedure. The dotted line is the true function we aim to optimize (maximize), the surrogate model response is shown in solid black with blue uncertainty bounds. In the first panel we see the initial surrogate model being fit to the first two observations (shown as black dots), and the subsequent panel displays an iteration of optimization.

Algorithm 1 Bayesian optimization

Require: Search space Λ , surrogate model algorithm \mathcal{A} , acquisition function α

```

1:  $\mathcal{H} \leftarrow \emptyset$ 
2: for  $i = 0, \dots, n$  do
3:    $\lambda_i \leftarrow \arg \min_{\lambda \in \Lambda} \alpha(\mathcal{M}, \lambda)$   $\triangleright$  First iterations sample at random instead
4:    $s_i \leftarrow \text{evaluate}(\lambda_i)$ 
5:    $\mathcal{H} \leftarrow \mathcal{H} \cup \{(\lambda_i, s_i)\}$ 
6:    $\mathcal{M} \leftarrow \mathcal{A}(\mathcal{H})$   $\triangleright$  Update the surrogate model
7: end for
```

In Figure 2.5 we see that the acquisition function determines the trade-off between exploration and exploitation of Bayesian optimization. Here, the acquisition function favors sampling not where the posterior mean is highest, but

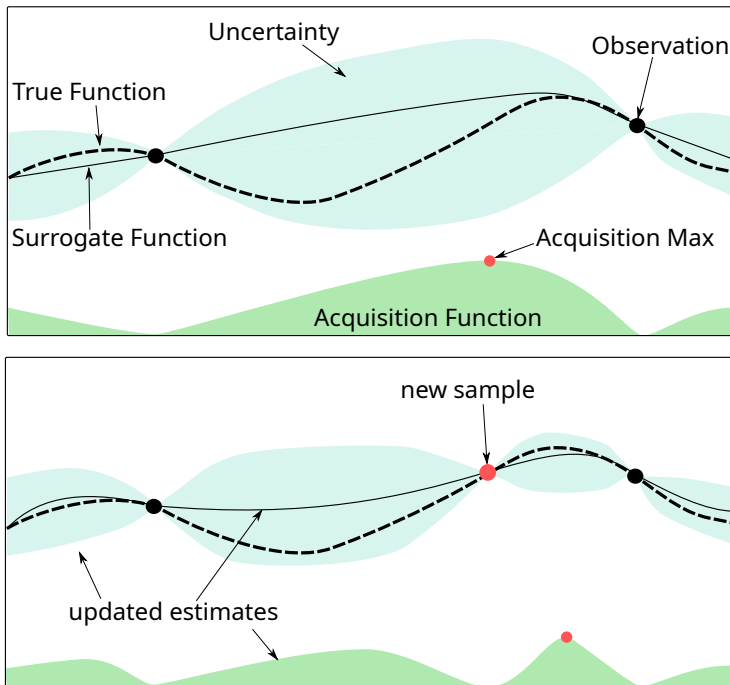


Figure 2.5: An illustration of two steps of Bayesian optimization on a 1D function. The top panel shows the initial surrogate model based on the first two sample points indicated by black dots. The bottom panel shows the updated surrogate model after sampling the point which maximized the acquisition function (in red).

around potentially good solutions which still have a relatively large uncertainty. The choice of acquisition function and its configuration will determine exactly how the posterior mean and uncertainty are used to determine the next sample point. Expected Improvement [127] is the most commonly used acquisition function and it also has an extension which takes into account the evaluation time [223], but many more acquisition functions are available [62, 126].

Recent methods allow human experts to provide a prior which is used to adjust the model estimates [229] or acquisition function [122] to leverage that knowledge. It is also possible to transfer surrogate models from earlier tasks [3, 86]. Both of these techniques may be used to overcome the need to start with random sampling.

Gaussian processes [200] were traditionally used to model the target function because of their expressiveness, smooth and well-calibrated uncertainty estimates, and closed-form computability of the predictive distribution [84]. However they scale poorly which results in considerable overhead when it is possible to sample many configurations. Additionally, Gaussian processes scale poorly to high dimensional search spaces, such as the search space for ML pipelines. Extensions, such as using additive kernels [3] or cylindrical kernels [176], may be used to mitigate this issue.

An alternative is to use a different approach altogether to model the objective function. In AutoML the best known example is SMAC [119] which is used by `auto-sklearn` and `Auto-WEKA`. Random Forests scale much better and natively work with non-continuous objective functions and a hierarchical search space [71], and a slight modification allows for approximating the uncertainty of the prediction [121]. Other ML algorithms to create surrogate models have also been explored, such as neural networks [224] and gradient boosting [116].

2.3.4 Successive Halving and Hyperband

Jamieson and Talwalkar [123] identified the hyperparameter optimization problem as a non-stochastic¹ best arm problem for multi-armed bandits and proposed to use Successive Halving² (SH) to find the best hyperparameter configuration from a set of configurations. The idea is succinctly explained by Jamieson and Talwalkar [123]:

Given an input budget, uniformly allocate the budget to a set of arms [hyperparameter configurations] for a predefined amount of

¹Meaning no assumptions are made about the generation of rewards.

²Originally called Sequential Halving [128].

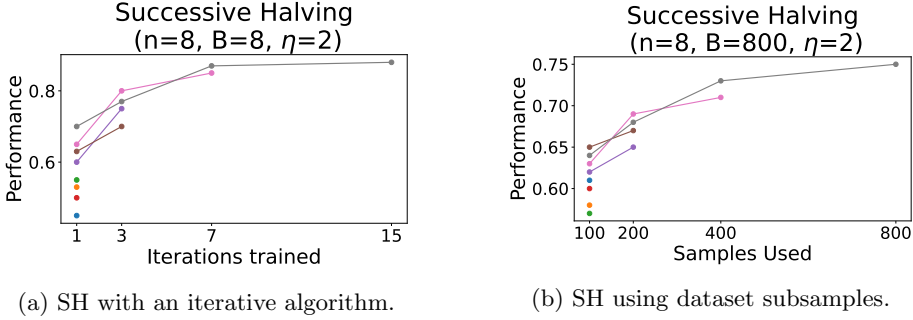


Figure 2.6: Illustrations of successive halving with $n = 8$ configurations. On the left learning iterations are used as a budget, and on the right subsets of the dataset are used instead. Each iteration of SH the same budget is evenly distributed across the considered configurations. Only $\frac{1}{\eta} = \frac{1}{2}$ of the configurations pass to the next iteration.

iterations [or samples], evaluate their performance, throw out the worst half, and repeat until just one arm remains.

More generally, you can extend the definition to keep a fraction of only $\frac{1}{\eta}$ at each iteration³ instead of $\frac{1}{2}$. At iteration i , starting from 0 with n configurations, SH evaluates $\frac{n}{\eta^i}$ configurations and the entire algorithm requires $\lceil \log_{\eta} n \rceil$ iterations. This introduces three hyperparameters to the algorithm: the number of initial configurations n , the resource budget for a single iteration B (e.g., the number of iterations of the learning algorithm), and the reduction factor η .

Figure 2.6a visualizes SH for an iterative algorithm, where at each rung each remaining candidate is trained for a number of iterations. For non-iterative algorithms, the resource budget can be specified as number of samples used during evaluations, which is visualized in Figure 2.6b. Note that with non-iterative algorithms, computation from earlier rungs does not carry over, whereas it is possible to resume training the same models with iterative algorithms.

Efficient parallelization of this optimization procedure is non-trivial, because identifying the final best configuration ultimately requires completing all previous iterations. In the context of hyperparameter optimization, the time required for evaluations of different configurations may differ orders of magnitude, which

³Here and for the remainder of the subsection *iteration* refers to an iteration of SH, not of the learner algorithm (e.g., epoch). This is also called a *rung* in [151].

makes waiting for stragglers very inefficient. Li et al. [151] propose an adaptation for the parallel setting called the Asynchronous Successive Halving Algorithm (ASHA). The adaptation does not wait for all evaluations to be completed, but greedily promotes the top $\lfloor \frac{1}{\eta} \rfloor$ configurations of currently evaluated configurations to the next iteration. Despite potentially transferring some configurations erroneously, they empirically demonstrate that this leads to faster optimization than a synchronous approach.

There is a risk that a good hyperparameter configuration is discarded early. To combat this, one might start with fewer configurations and dedicate a larger budget for them, which has the downside that fewer hyperparameter configurations are considered. A priori it is not generally known whether it is best to consider many hyperparameter configurations and use low-fidelity estimates, or to start with fewer configurations that allow for higher-fidelity estimates with the same resources.

Hyperband [153] addresses this problem by using SH as a subroutine. It starts SH multiple times with different n dividing the total resource budget evenly. Each instantiation of SH within HB is also called a *bracket*. Through brackets with high n a large part of the search space can be explored, but brackets with low n still allow a chance to find configurations which converge slowly yet ultimately have a good performance.

Hyperband and successive halving have also been combined with other optimization methods. To generalize to non-iterative learning algorithms, sometimes a subset of the data is used instead to obtain low-fidelity estimates [99, 181]. BOHB [80] has been integrated in `auto-sklearn` 2 [82] and combines Bayesian optimization with Hyperband by replacing the random selection of pipelines in Hyperband with ones recommended by Bayesian optimization. Layered TPOT (LTPOT) [99]⁴ and TPOT-SH [181] combine TPOT’s evolutionary algorithm with multi-fidelity estimates. In LTPOT evolution takes place in multiple populations across different rungs (resource budgets) and individuals may be promoted to higher fidelity rungs. In TPOT-SH a single population is maintained and the evaluations across generations become of increasingly higher fidelity by training and evaluating the pipelines on more data. DEHB [7] combines hyperband with differential evolution [194] in a similar way to LTPOT, though they decrease the population size for higher fidelity estimates. DEHB has not yet been included in an AutoML tool.

⁴Research carried out during my master’s thesis.

2.3.5 Other Methods

There are many more methods through which pipelines may be constructed automatically, this subsection highlights a few of them.

Particle Swarm Optimization

Escalante, Montes, and Sucar [75] use Particle Swarm Optimization (PSO) [70] to optimize complete pipelines. In this population based approach, each pipeline is encoded as a vector that represents both the hyperparameter configurations of each algorithm and indicator variables that denote which algorithms are included in the pipeline. A random swarm of particles is first generated, these particles have a location in the search space that corresponds to a ML pipeline configuration. After evaluating each particle’s location (i.e., evaluating the model induced by the ML pipeline according to some metric), they each traverse the search space taking into account both the best configuration that the individual particle encountered and the global best known configuration among all particles.

Planning

Hierarchical task network planning [98] is used in **ML-Plan** [169] to search for ML pipelines. A directed acyclic graph is maintained where each edge denotes a design decision for the ML pipeline, these decisions can be either abstract (e.g., add a preprocessor) or concrete (e.g., set k neighbours to 8), and the nodes denote a partially-planned pipeline. There is one source node in the graph, corresponding to the high level concept of building a pipeline, and many sink nodes, which represent concrete pipelines. A two-tiered search is then performed by first expanding all algorithm selection decisions, and then performing random completions for their hyperparameter configurations. Nodes get assigned a score based on all evaluations reachable from that node, and these scores are used to guide the search. A similar strategy is used in **Mosaic** [199], which uses Monte Carlo Tree Search [135] to optimize a pipeline with a fixed ‘plan’: select a preprocessor, configure its hyperparameters, select the learner, and configure the learner’s hyperparameters.

Active Learning

OBOE [279] runs a set of fast but informative algorithms to profile a dataset, and subsequently uses collaborative filtering to recommend pipelines. Afterwards,

an iterative active learning process evaluates pipelines that are most likely to result in better recommendations. Active testing [149] similarly uses a history of algorithm performance on previous tasks. It maintains an incumbent algorithm configuration and use the history to predict a configuration which is most likely to outperform it. The best of the two configurations as evaluated in a tournament is then set to be the new incumbent algorithm and the procedure repeats. In contrast to OBOE, active testing only recommend single algorithm configurations instead of pipelines.

Heuristic Search

FLAML [265] has a portfolio of algorithms and a decision process based on the estimated cost for improvement that decides which algorithm to tune and with which evaluation budget. The hyperparameter tuning step is performed with Cost-Frugal Optimization (CFO) [273] which is based on randomized direct search.

Racing

Maron and Moore [161] proposed *racing*, an algorithm that iteratively evaluates a set of models (for example, one cross-validation fold at a time) and discards bad configurations early as determined by statistical tests to efficiently find the best models in the set. While Maron and Moore [161] proposed to use Hoeffding bounds to discern the quality of models, Birattari et al. [25] instead proposed to use the Friedman test because it allows for a blocking design, comparing the mean performance difference as opposed to their bounds, which reduces the number of evaluations needed to discern between similar models [161]. In iterated racing [156], used in AutoML framework *iSklearn* [261], this procedure is executed multiple times in a row, each time starting with a new set of configurations that are sampled based on the winners of the last races.

Predefined Pipelines

There are also AutoML methods where a specific optimization algorithm is not central to the ML pipeline creation process. *AutoGluon* [73] defines a fixed model architecture that uses stacked pipelines and scales in size depending on the given computational constraints. Mohr and Wever [168] propose a naive AutoML process that simulates a data scientist’s workflow and breaks the pipeline construction down into six steps, only one of which requires a non-exhaustive search.

2.4 Post-Processing

After the search stage additional computation may be performed, which we refer to as post-processing, to enhance model performance or provide the user with additional information. The search stage often evaluates many pipelines which results in a diverse set of trained models, but also makes it prone to overfitting. One common post-processing step is to learn how to map the predictions of individual models to a single combined prediction, which reduces the variance and can exhibit better performance than any individual model. However, for practical applications, model performance is not the only consideration. When a model has been trained by the AutoML system, additional techniques may be used to give additional insight in the model, for example, to provide interpretability or to provide a generalization estimate. Below, we will first discuss different methods for combining model predictions, and then cover some of the methods that may provide the users with more information about the produced model.

2.4.1 Weighted Voting

`auto-sklearn` [85] and `GAMA` [103] implement an ensemble construction procedure from Caruana, Munson, and Niculescu-Mizil [48] and Caruana et al. [49], which describes a hill-climbing algorithm that iteratively constructs an ensemble. Each pipeline in the ensemble has a weight, and the ensemble’s prediction is the weighted sum of the pipeline predictions. The ensemble construction algorithm first creates an initial ensemble with the n best pipelines. It then adds pipelines one at a time, by picking with replacement from all evaluated pipelines. To decide which pipeline to add, the performance of the ensemble is evaluated with that pipeline included, or with its weight increased if it was already included. The change which leads to the best performance is made permanent, and then this procedure repeats until the ensemble has a predetermined size. Instead of finding the weights for pipelines through hill-climbing, `AutoPrognosis` [3] uses Bayesian model averaging to determine the weights. Their weighing scheme automatically favors the use of “diverse” pipelines based on their learnt search space decomposition.

2.4.2 Stacking

H2O `AutoML` [148] instead uses the stacking procedure from [253] where another learner is used to learn how to combine predictions of base-learners into one. The

advantage of this method is that it can also use non-linear models. Note that in some approaches, e.g., TPOT, Auto-WEKA and ML-Plan, stacking ensembles are included in the search space and are directly optimized over in the search step.

A different application of post-processing is that of the *interpreter* module of AutoPrognosis [3]. They learn a model using a Bayesian associative classifier that gives explanations for the predictions of the generated ensemble model. These explanations are expressed as logical rules e.g., $\text{age} \geq 40 \wedge \text{diabetic} \rightarrow \text{high risk}$, which makes them interpretable by clinicians. This makes the models more likely to be used [260], and in some cases may even be a legal requirement [21].

2.4.3 Model Information

After producing a model, the user of the AutoML system has to consider whether they want to deploy it. For this, it is useful to have additional information, such as the generalization estimate or visualizations to aid model interpretability.

Generalization Estimates

Providing a generalization estimate is not straightforward. Reporting the internal estimate that was found during optimization, and used to select the model, will be optimistically biased [259]. Tsamardinos, Rakhshani, and Lagani [247] empirically demonstrate this effect on real-world data, which is predominantly present with small datasets (containing fewer than 1000 rows), and find that nested cross-validation or a method proposed by Tibshirani and Tibshirani [239] may be used to generate less biased generalization estimates. Bootstrap Bias Corrected Cross-Validation (BBC-CV) [246] was later proposed to efficiently obtain unbiased estimates in the model selection setting. It repeats model selection multiple times on bootstrapped matrices with model predictions, computes their performance on the respective bootstrapped predictions, and reports the mean generalization estimate which has less bias than previous methods.

Most AutoML frameworks do not provide unbiased performance estimates out-of-the-box, but require that the user perform an evaluation procedure to obtain a generalization estimate. However, JADBio [245] uses BBC-CV for less biased generalization estimates, which is especially relevant since it is developed for biomedical datasets which are typically small.

Interpretability

Several AutoML frameworks provide reports which help the user interpret what the model learned and how different features affect it. Many techniques for general interpretability and explainability in ML may be applied directly as they are designed to work with black-box models. This includes training interpretable models to mimic complex ones found by AutoML [3, 133] or post-hoc model-agnostic explanation methods such as LIME [205], partial dependence plots [93], or individual conditional expectation plots [106]. Interpretability reports are currently available with most commercial AutoML frameworks (for example, `mljar` [192] and `JADbio` [245]).

2.5 AutoML in Other Settings

Most work in AutoML research is focused around finding ML pipelines that maximize the performance for offline classification and regression problems. Recently, we see work starting to focus on applying AutoML techniques in other settings, e.g., semi-supervised learning [150], time series [277]. In this section we briefly discuss some of this relatively unexplored work.

2.5.1 Online Learning

In the offline setting, a single batch of data is used to train a model and it is evaluated on a test set. By contrast, in the online setting data is provided in batches (possibly of size 1) and the models need to be maintained over time. This can be further complicated by the presence of concept drift [95], i.e., changes in the underlying data distributions, which can make old knowledge either temporarily or permanently obsolete. In this setting, prequential evaluation is used to evaluate models i.e., when a new data batch comes in, it is first used to evaluate the model and is only then used to update the model.

The simple idea to train a few different models and use the best performing one with respect to a recent window w , dubbed BLAST (for Best Last), proved very effective [208]. Champion-Challengers (ChaCha) [274] performs online algorithm configuration by maintaining a champion and a set of challengers. It repeats the process of identifying a new champion by evaluating the challengers on increasingly larger budgets until one is statistically significantly better. That challenger then replaces the champion and a new set of challengers is generated based on the new champion, and the process repeats. In [52] several adaptation strategies are evaluated on a variety of AutoML approaches. The proposed

adaptation strategies successfully allow AutoML tools to handle concept drift, though the best type of adaptation changes both depends on type of concept drift and the optimization methods. While those adapted systems still used offline learners, [51] introduces an AutoML framework which, in contrast to ChaCha, uses *multiple* online learning algorithms such as Hoeffding Adaptive Trees [24] and Adaptive Random Forests [107], and designs pipelines including preprocessing algorithms. Additionally they evaluate multiple strategies to keep models up-to-date, e.g., storing a set of models which worked well in the past or creating an ensemble and updating its weights, and trigger additional pipeline searches after detecting concept drift.

2.5.2 Unsupervised AutoML

AutoML for clustering is heavily focused around algorithm selection through meta-learning [191, 228]. Recently, new work has explored AutoML beyond just algorithm selection. Both **AutoClust** [193] and **AutoCluster** [155] still use meta-learning for algorithm selection. **AutoClust** subsequently uses Bayesian optimization for hyperparameter configuration, whereas **AutoCluster** uses grid search and an ensembling post-processing step.

It is the very nature of clustering, the lack of ground truth labels, which makes automated clustering especially hard. Automated approaches often consider multiple internal metrics, i.e., metrics that don't require ground truth, when optimizing pipelines. Which metric is most appropriate is subjective, and for this reason we see that they are either combined into one objective function [193] or separately optimized and the resulting pipelines combined in an ensemble [155]. By contrast, Ditton et al. instead proposed a semi-automated approach where bad configurations are discarded automatically and present the remaining cluster sets, with rich meta-data, to a domain expert [67].

2.5.3 Multi-Label Classification

In a multi-label classification setting each sample can have multiple target labels. This setting can be modeled as multiple single-label classification tasks, where a single model is trained for each label independently, or as a joint modeling problem. To tackle multi-label classification, ML^2 -Plan [268] restricts search to algorithm selection only, though several algorithms are included for which they also consider the selection of base learner algorithms. They argue that in this setting it is common for multiple models to be trained in a single evaluation, making even just algorithm selection a hard problem. De Sá et al. [211]

use genetic programming for algorithm selection and hyperparameter optimization, but they find that their method often selects base level learners which are cheaper to evaluate.

2.5.4 Remaining Useful Life Estimation

In remaining useful life (RUL) estimation, the task is to predict how long an asset is still useful for (e.g., safe to use), based on historic data. The historical data contains variable length time series data and thus can not be directly used in existing AutoML tools. For this reason, the data is transformed into a regression problem either by a predefined pipeline [129], by including the transformation as part of the ML pipeline design [243], or by coevolving a data transformation pipeline and a regression pipeline [242].

Chapter 3

GAMA - Modular AutoML

In this chapter, we introduce an AutoML framework developed for AutoML research called the General Automated Machine Learning Assistant (**GAMA** [103, 104]). As described in more detail in Chapter 2, there are a myriad of design decisions when building AutoML tools. This includes the type of ML pipeline (e.g., fixed or variable length), the optimization algorithm (e.g., evolutionary or Bayesian optimization), and whether or how to employ meta-learning (e.g., warm-starting) or post-processing (e.g., ensembling or stacking).

After coming up with a novel addition or improvement on any one of those AutoML components, a researcher has to make a decision: integrate it with an existing tool or develop an entirely new tool. Developing the new method within the framework of an existing tool can incur a lot of overhead because the frameworks are typically not designed to accommodate new ideas, which requires considerable time spent understanding the existing code base and possibly refactoring it. Additionally, there is the risk that even after the idea is implemented, the original authors will not integrate it with the tool, which hinders future work and adoption. On the other hand, developing an entirely new tool also brings considerable overhead, and any comparison to previous approaches is now obfuscated by other design decisions and even implementation details, which makes it impossible to attribute measured improvements to the novel idea.

This chapter is derived from: Pieter Gijsbers and Joaquin Vanschoren. “GAMA: A General Automated Machine Learning Assistant”. In: *Machine Learning and Knowledge Discovery in Databases. Applied Data Science and Demo Track*. Ed. by Yuxiao Dong et al. Cham: Springer International Publishing, 2021, pp. 560–564. ISBN: 978-3-030-67670-4

GAMA is an open-source AutoML framework¹ which distinguishes itself by abstracting the AutoML process through its *modularity*, allowing users to compose AutoML systems from components, *extensibility*, allowing new components to be added, and *transparency*, tracking and visualizing the search process to better understand what the AutoML framework is doing. These properties make it an ideal tool for researchers to perform systematic AutoML research, especially when evaluating novel ideas.

3.1 Related Work

Chapter 2 discussed many different AutoML frameworks, many of which optimize `scikit-learn` [184] pipelines. Table 3.1 provides a small summary of the frameworks most similar to **GAMA**’s default configuration. While **GP-ML** [141] is most similar as they both perform multi-objective optimization using an asynchronous ($\mu + 1$)-algorithm [218] with NSGA-II [64] selection², there are two big differences. First, **GAMA** uses models found during search in a post-processing ensembling step to improve performance and reduce the chance to overfit, similar to **auto-sklearn** [85]. Second, **GAMA** optimizes linear ML pipelines, whereas **GP-ML** structures pipelines as directed acyclic graphs (DAGs) [141], and **TPOT** structures pipelines as trees [179].

Framework	Algorithm	Multi-objective	Post-processing
GAMA [103, 104]	$\mu + 1$ [218]	NSGA-II [64]	Ensemble [48, 49]
GP-ML [141, 142, 190]	$\mu + 1$ [218]	NSGA-II [64]	No
TPOT [179]	$\mu + \lambda$ [20]	NSGA-II [64]	No
RECIPE [212]	$\mu + \lambda$ [20]	No	No
auto-sklearn 1 [85]	SMBO [119]	No	Ensemble [48, 49]

Table 3.1: Comparison of most closely related AutoML work.

While asynchronous evolution is **GAMA**’s default search algorithm, it can also be configured to use the asynchronous successive halving algorithm (ASHA) [151], and others may be easily added. We are not aware of any other AutoML framework using ASHA, however, multiple approaches have been proposed with some

¹Code and documentation can be found at <https://github.com/openml-labs/gama/>

²This work was developed independently of **GP-ML**.

form of multi-fidelity optimization [58, 82, 99, 181].

While the other methods allow modifications to their search space, they do have a fixed AutoML pipeline. To the best of our knowledge, **GAMA** is the only AutoML framework that offers a modular and extensible composition of AutoML systems, and extensive support for AutoML research.

3.2 The Modular AutoML Pipeline

Rather than prescribing a specific combination of AutoML techniques, **GAMA** allows users to combine different search and post-processing algorithms into a flexible AutoML ‘pipeline’. The types of AutoML pipeline that **GAMA** allows to be designed matches those of the prototype AutoML pipeline shown in Figure 2.1. This design fits, e.g., **TPOT** (evolutionary optimization), **H2O AutoML** (random search followed by stacking), and **auto-sklearn** (Bayesian optimization followed by ensemble construction). However, designs that do not fit this prototypical pipeline already exist, e.g., **AutoGluon** does not employ search, and many more designs are conceivable.

The configurability of the prototypical AutoML pipeline allows for easy ablation studies by changing one specific step in the pipeline, but may also be used to tune the AutoML framework to the problem at hand. This section gives an overview of the currently implemented methods and shows how to configure a custom AutoML pipeline with **GAMA**.

3.2.1 Search

There are three types of optimization algorithms currently implemented in **GAMA** to search for optimal machine learning pipelines: random search, the bandit-based asynchronous successive halving algorithm, and an asynchronous evolutionary algorithm. We first give a brief motivation for using asynchronous algorithms and then discuss the different implemented methods in more detail below.

Asynchronous Optimization

In **GAMA**, we chose to incorporate asynchronous algorithms because they parallelize more efficiently than their synchronous counterparts. This is illustrated in Figure 3.1, where the two methods are compared and jobs, visualized as bars,

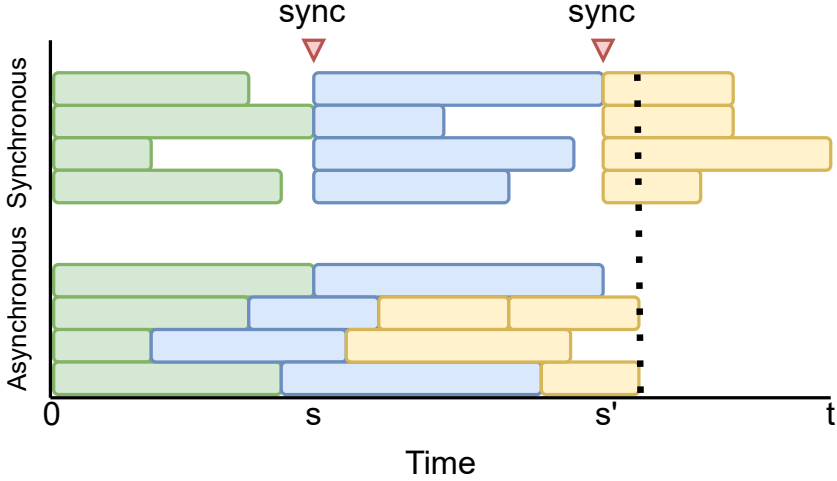


Figure 3.1: A visual example of sync points (e.g., generations in evolution) causing idle workers in synchronous methods. Bars represent jobs distributed over 4 workers for each method. For comparison purposes, their color represents the batch and the same total compute time is used in both methods.

are distributed over 4 workers for each method. The figure shows that synchronous algorithms need to wait until all jobs in a batch are finished, e.g., all individuals of a generation or in a rung are evaluated, which leaves time gaps where workers are idle. By contrast, asynchronous methods define new jobs whenever resources are available, allowing them to parallelize more effectively. ML pipelines can vary dramatically in running time [170, 279], which means synchronous approaches may spend a lot of time waiting for stragglers to finish.

In the example, each job is given a color to represent the batch and the same total compute time is used in both methods. In reality, the asynchronous method will need to generate jobs with different information than the synchronous method, so the results would differ. Evaluating the effect of these differences on convergence time and final model quality would be interesting future work.

Interestingly, there hasn't been much work evaluating asynchronous optimization for AutoML. While the resource utilization is higher for asynchronous

algorithms, new candidate solutions are generated with different information which means it might alter the end result. This has been studied outside of the AutoML context, but very little within [151, 190]. Using asynchronous evolution has been proposed before independently by Pilát, Křen, and Neruda [190] though their evaluation is small-scale and also introduces caching of machine learning pipelines. To the best of our knowledge, none of the systems that use bandit-based optimization include their asynchronous version.

Random Search

Random search is more effective than grid search for hyperparameter optimization [17] and may prove to be a strong baseline given a well-designed search space (as it is for certain types of Neural Architecture Search [152]). **GAMA**'s random search creates pipelines in three steps. First, the pipeline length is chosen uniformly at random (containing a maximum of 3 steps, by default). Then, for each step, an algorithm is chosen uniformly at random. Finally, for each algorithm, the hyperparameter configuration is chosen uniformly at random.³

Asynchronous Successive Halving Algorithm

ASHA [151] uses multi-fidelity estimates to filter out bad pipelines early as shown in Algorithm 2. In short, given a reduction factor η and budget parameters (b, B, s) , configurations are first evaluated on $b\eta^s$ budget. The top $\frac{1}{\eta}$ configurations in rung k , corresponding to resource budget $b \cdot \eta^{s+k}$, get promoted to the next rung with a larger resource budget per pipeline $b \cdot \eta^{s+k+1}$. In ASHA, new configurations are added to the lowest rung anytime no evaluations are scheduled for higher rungs, and all pipelines in the top $\frac{1}{\eta}$ of their rungs have already been promoted. The minimum early stopping rate s can be used to increase the budget of the bottom rung. Because **GAMA** includes non-iterative algorithms in the search space, these multi-fidelity estimates are obtained by subsampling the dataset. For example, on a dataset with 1 million rows, pipelines would first be evaluated with cross-validation on 10,000 rows, the top configurations are subsequently evaluated on 100,000 rows, and the best of those pipelines are evaluated on the full dataset. Pipeline candidates are generated at random, similar to random search.

³Continuous hyperparameters are currently discretized in **GAMA**'s search space.

Algorithm 2 Asynchronous Successive Halving Algorithm [151]

Require: minimum resource b , maximum resource B , reduction factor η , minimum early stopping rate s

```

1: while not stop do                                     ▷ e.g., time, iterations
2:   for each free worker do
3:      $(\theta, k) \leftarrow \text{get\_job}()$                      ▷ In AutoML,  $\theta$  is a ML pipeline
4:      $\text{queue\_evaluation}(\theta, b\eta^{s+k})$ 
5:   end for
6:   for each completed job  $(\theta, k)$  with loss  $l$  do
7:     Update configuration  $\theta$  in rung  $k$  with loss  $l$ .
8:   end for
9: end while
10:
11: function GET_JOB()
12:   for  $k = \lfloor \log_\eta(B/b) \rfloor - s, \dots, 1, 0$  do           ▷ Promote in high rungs first
13:      $\text{candidates} \leftarrow \text{top\_k}(\text{rung } k, \frac{|\text{rung } k|}{\eta})$ 
14:      $\text{promotable} \leftarrow \{t \text{ for } t \in \text{candidates} \text{ if } t \text{ not already promoted}\}$ 
15:     if  $|\text{promotable}| > 0$  then
16:       return  $\text{promotable}[0], k + 1$                        ▷ Always promote if possible
17:     end if
18:   end for
19:   Draw random configuration  $\theta$                            ▷ But grow bottom rung otherwise
20:   return  $\theta, 0$ 
21: end function

```

Asynchronous Multi-Objective Evolutionary Algorithm

The evolutionary algorithm in **GAMA** is identical to the one described in [218] for which pseudo-code is presented in Algorithm 3. The `queue_evaluation(p)` function submits pipeline p to a queue to be evaluated on one of the worker nodes, and the `get_next_evaluation()` function returns whichever evaluation is done first. The algorithm maintains a single population and generates offspring from the population whenever a worker is available.

Algorithm 3 Asynchronous Evolution

Require: P_{start} initial pipeline designs, $N_{max} > 0$

```

1: for all  $p \in P_{start}$  do
2:   queue_evaluation(p)                                ▷ To be evaluated on a worker
3: end for
4:
5:  $P \leftarrow \emptyset$ 
6: while not stop do                                     ▷ E.g., time, iterations
7:    $P \leftarrow P \cup \{ \text{get\_next\_evaluation}() \}$       ▷ Whichever is done first
8:   if  $|P| > N_{max}$  then
9:      $P \leftarrow P \setminus \{ \text{eliminate}(P) \}$           ▷ Remove the worst fitness
10:  end if
11:  if worker is available then
12:    queue_evaluation(create_one(P))                    ▷ Create new pipeline
13:  end if
14: end while

```

While the pseudo-code presented here only differs in form from [218], there are differences in the selection, mutation, cross-over and representation of individuals. **GAMA** uses genetic programming trees to represent linear ML pipelines (see Section 2.3.2), and uses the following operators to optimize them:

Elimination (line 9): Remove an individual from the worst rank pareto front.

Pipeline Creation (line 12):

- **Parent Selection** Binary tournament selection based on pareto rank and crowding distance as in NSGA-II [64].
- **Cross-over** Exchange subtrees (e.g., a preprocessing pipeline).
- **Mutation** One of the following mutations with equal probability⁴:

⁴Considering only valid mutations, e.g., you can't shrink a tree with only a root node.

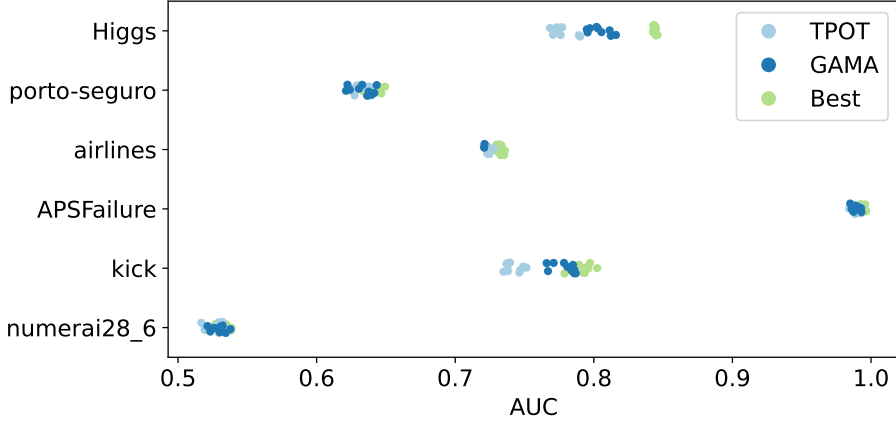


Figure 3.2: A comparison of TPOT and GAMA without ensembling on six binary classification tasks from the benchmark. The best observed score per fold across all frameworks in the benchmark (see Chapter 5) is also shown for reference.

Point Replace a terminal

Point Replace a primitive and its connected terminals

Insert Extend a ‘data’ terminal with a preprocessing subtree.

Shrink Remove (part of) a subprocessing subtree.

Figure 3.2 shows a small scale comparison on six tasks from the AutoML benchmark (see Chapter 5) between TPOT, which uses a synchronous ($\mu + \lambda$) algorithm, and GAMA, using asynchronous evolution. The top and bottom three tasks are the biggest three binary classification tasks under one million and under 100 thousand rows, respectively. The best observed score for each fold across all frameworks is also shown for reference. GAMA’s search space is very similar to that of TPOT, but TPOT allows stacking in the pipeline design by using any learner as a preprocessing step and appending its predictions to the data. While we can’t draw any conclusions because of these multiple design differences, we think it is a promising indication that asynchronous methods also lead to better pipelines being discovered in the same time budget, as GAMA improves over TPOT on tasks where a substantial improvement was shown to be possible. In the future we hope to do a principled comparison by adding synchronous evolution to GAMA.

Algorithm 4 Ensemble selection from libraries of models [49]

Require: \mathcal{P} a set of pipelines with given loss l_p and out-of-fold predicted probabilities \hat{y}_p , initial ensemble size k , final ensemble size K

- 1: $\mathbf{w} \leftarrow [0 \mid p \in \mathcal{P}]$ \triangleright Initialize weight of each pipeline
- 2: **for all** $p \in \text{top.k}(\mathcal{P}, k)$ **do** \triangleright Add k pipelines with the least loss
- 3: $\mathbf{w}_p \leftarrow 1$
- 4: **end for**
- 5:
- 6: **for** $1, \dots, K - k$ **do**
- 7: $\mathbf{L} \leftarrow [\text{Evaluate}(\mathcal{P}, \mathbf{w}'), \mathbf{w}' \text{ is } \mathbf{w} \text{ but with } \mathbf{w}_p \text{ increased by } 1 \mid p \in \mathcal{P}]$
- 8: Increase \mathbf{w}_p by 1 where $p := \arg \min_{p \in \mathcal{P}} \mathbf{L}_p$
- 9: **end for**
- 10:
- 11: **function** EVALUATE(\mathcal{P}, \mathbf{w}')
- 12: **return** Loss incurred by prediction $\frac{1}{|\mathbf{w}'|_1} \sum_{p \in \mathcal{P}} \mathbf{w}'_p \cdot \hat{y}_p$
- 13: **end function**

3.2.2 Post-processing

After the pipeline search has been completed, a post-processing technique may be executed to construct the final model. It is currently possible to either train the single best pipeline on all training data or to create an ensemble out of pipelines evaluated during search. The latter is done using the hillclimbing ensemble algorithm described in [49] and shown in Algorithm 4, including some refinements proposed in [48]. First, an initial ensemble is constructed with the best k pipelines found during search. Then, pipelines are added one by one to the ensemble based on the ensemble performance with that pipeline included until the desired ensemble size is reached. Using an ensemble of pipelines increases the performance and is less prone to overfitting than selecting the best single pipeline.

Later work described a few modifications which are particularly interesting for practical AutoML. First, Caruana, Munson, and Niculescu-Mizil [48] observed that using the all models trained during cross-validation directly in an ensemble led to increased performance over retraining a model on all available data. This is particularly convenient from an engineering perspective, as AutoML systems typically need to adhere to time constraints and the lack of additional (unpredictable) fit procedures makes the ensembling procedure much faster and at the same time more predictable.

Second, they find that pruning the model library (\mathcal{P}) by removing its worst models decreases the risk for ensemble overfitting and subsequently improves the performance. In practice, this means we do not need to store all trained pipelines and generated predictions, but only a subset of them. In their work, they define a fraction to keep relative to the total amount of evaluated models, in **GAMA** we use a set amount of pipelines since the amount of evaluated models will vary greatly depending on the optimization problem. This might introduce the risk of only keeping too similar models since generated pipelines are derived from well performing pipelines (when using evolutionary optimization), but we leave studying whether or not this effect occurs in practice for future work.

Finally, while the original publication recommends repeating the procedure several times with subsets of the model library (\mathcal{P}) to create *bagged* ensembles, later work showed that the benefits of bagged ensembles vanish when the evaluation dataset is sufficiently large. In their experiments, hillclimb evaluation datasets as small as a few hundred samples saw only marginal benefits and no benefits were observed with a hillclimb set of 10.000 samples. Especially when using hillclimbing sets generated with cross-validation, almost all modern datasets meet this size criterion so we forgo bagging.

3.2.3 Configuring an AutoML Pipeline

Listing 3.1 shows how to configure **GAMA** with non-default search and postprocessing methods and use it as a drop-in replacement for **scikit-learn** estimators.⁵ New AutoML algorithms or variations to existing ones can be included and tested with relative ease. For instance, each of the search algorithms described above has been implemented and integrated into **GAMA** with less than 170 lines of code, and they can all make use of shared functions for logging, parallel pipeline evaluation, and adhering to runtime constraints.

While this flexibility does raise the question of how to best configure **GAMA** to obtain the best model, the summarized benchmark results shown in the next section, and in more detail in Chapter 5, show that the out-of-the-box performance is similar to that of fixed AutoML frameworks.

⁵An always up-to-date version of this listing can be found at <https://openml-labs.github.io/gama/master/citing.html>

Listing 3.1: Configuring an AutoML pipeline with GAMA

```
1 from GAMA import GAMAClassifier
2 from GAMA.search_methods import AsynchronousSuccessiveHalving
3 from GAMA.postprocessing import EnsemblePostProcessing
4
5 automl = GAMAClassifier(
6     search=AsynchronousSuccessiveHalving(),
7     post_processing=EnsemblePostProcessing()
8 )
9 automl.fit(X_train, y_train)
10 automl.predict(X_test)
11 automl.score(X_test, y_test)
```

3.3 Accelerating Research

GAMA is integrated in the AutoML Benchmark to be introduced in Chapter 5. That chapter will also provide a detailed report of the experimental evaluation of **GAMA** and many other AutoML frameworks, but some results can be seen in Figure 3.3. The results in this figure are computed by aggregating over all four-hour results of both classification and regression tasks, with **GAMA** using its evolutionary search and ensemble post-processing. Figure 3.3a shows the critical difference diagram [65] of the average ranks, where missing values are first imputed with the worst observed performance for the same task and fold across all frameworks. In Figure 3.3b we see the trade-off between the median prediction speed and performance. To commensurate the different scales of the different metrics and tasks, results are first scaled relative to random forest performance (0) and the best observed performance out of any framework (1). We see that **auto-sklearn** performs very similar to **GAMA**, which is likely because they use approximately the same search space and the same ensemble post-processing algorithm. It should be noted, however, that the focus of **GAMA** is not to be the best performing AutoML framework or provide the fastest inference times, but to allow for easy but principled AutoML research. To this end, **GAMA** provides the researcher with artifacts and a visualization tool which are presented below.

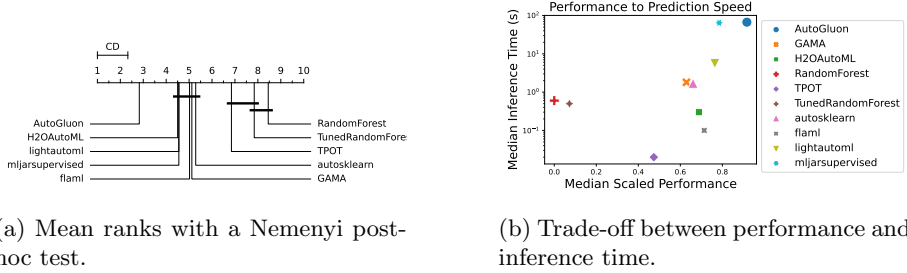


Figure 3.3: Benchmark results aggregated across all tasks of both suites with a four-hour time budget. Here, **GAMA** uses asynchronous evolutionary search and ensemble post-processing.

3.3.1 Interface

GAMA comes with a graphical web interface which allows novice users to start and configure **GAMA**. Moreover, it visualizes the AutoML process to enable researchers to easily monitor and analyze the behavior of specific AutoML configurations.

One can also compare multiple logs at once, creating figures such as Figure 3.4 that shows the convergence rate of five different **GAMA** runs over time on the airline dataset⁶.

3.3.2 Artifacts

GAMA automatically creates logs with information about the pipeline optimization for analysis. It's easy to extend this logging with optimizer specific information. For example, pipelines created through evolution will also keep a reference to their parent(s), and pipelines evaluated in ASHA come with information about their resource budgets. Additionally, other artifacts may be stored as well, such as memory usage logs or evaluated pipelines and their predictions. In this section, we will show a few examples of visualizations created from these logs of a 10-fold cross-validation experiment with a one hour time budget on the Higgs task⁷. This Higgs task is a fairly balanced binary classification problem with 28 numeric features and is subsampled down to one million rows. We evaluated both ASHA and EA.

⁶For more information, see: <https://www.openml.org/d/1169>

⁷For more information, see: <https://www.openml.org/t/360114>

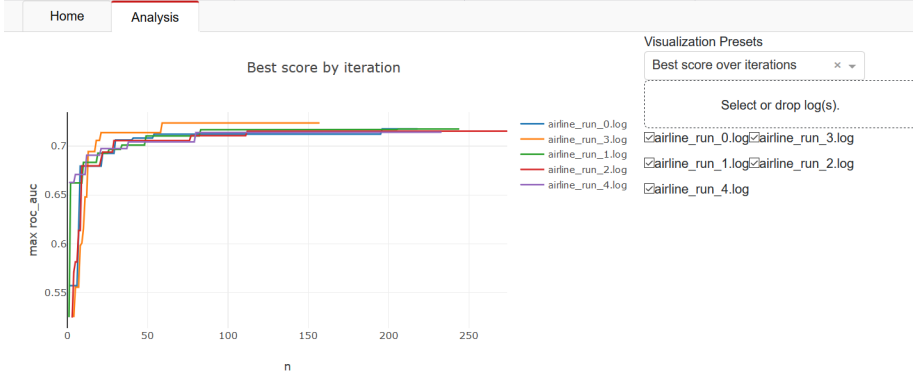


Figure 3.4: Visualization of logs

Figure 3.5 visualizes the evolution history for one fold. Each marker represents an ML pipeline evaluated during search. The x-axis denotes the creation order, though for legibility the initial population of 50 individuals is plotted as though they were generated in sequence (denoted with circles). The y-axis denotes its rank (here, higher is better). As expected, we see that over time we find better pipelines and gradually generate fewer pipelines which fail to evaluate (e.g., due to the time constraints).

The shape of the marker indicates how the pipeline was generated (e.g., cross-over or mutation) and its color shows how much offspring it has. Some individuals generate much offspring despite their lower performance rank, which can be explained by the multi-objective selection that also takes into account the number of steps in the pipeline. The best found pipeline is denoted with an orange marker, and its lineage is visualized through dashed lines that connect to its ancestors. This tells us the pipeline was generated from the initial population and a total of three mutations and one cross-over step.

Figure 3.6 shows an optimization trace for a single fold for ASHA. The resources used at each rung are shown on the x-axis, and the y-axis denotes the pipeline performance. Despite only evaluating a fraction of the pipelines on a full budget compared to the evolutionary approach, more than twice the pipelines were evaluated on the lowest rung.

Finally, the convergence of both methods is compared in Figure 3.7 and we see in this scenario that ASHA finds good solutions more quickly. While these traces are based on internal evaluation scores, the out-of-fold scores are similar

with a mean AUC of 0.791 and 0.776 for ASHA and evolution, respectively. This is to be expected as the Higgs dataset is rather large for a one hour budget, so the benefit of multi-fidelity optimization is emphasized.

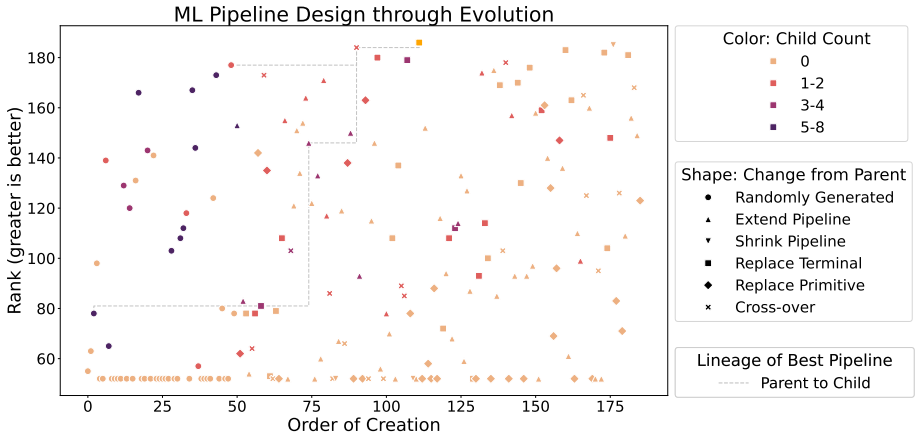


Figure 3.5: Evolutionary optimization on Higgs on a one hour time budget.

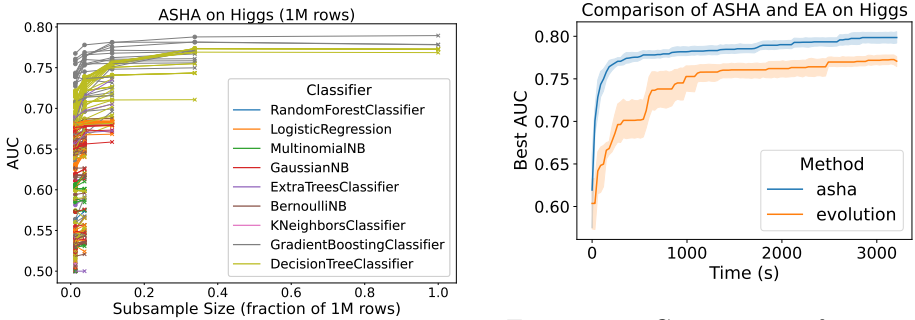


Figure 3.6: ASHA on a one hour time budget with reduction factor 3.

Figure 3.7: Comparison of convergence for ASHA and EA on a one hour time budget across 10 folds.

3.4 Use in Research

In this section, we have a closer look at work that uses **GAMA** in novel AutoML research.⁸

3.4.1 Online AutoML

Celik, Singh, and Vanschoren [51] adapted **GAMA** for online AutoML, because of earlier findings that evolutionary optimization adapts well to concept drift [52]. Because the online setting differs significantly from the offline setting, the proposed approach only uses AutoML pipeline design after detecting concept drift. They experiment with three different methods to keep models up-to-date: keeping a single best pipeline, maintaining and reweighing ensembles, or using a model store with recent good pipelines. Additionally, they defined a search space with online learners from **River** [172] and used prequential evaluation to take into account the temporal relationship in the data. Despite those differences, they still maintained the flexible AutoML procedure and were able to compare different search methods that were provided out-of-the-box.

3.4.2 Multi-fidelity Evolution

Campero Jurado and Vanschoren [45] explored a novel application of the generalized island model in AutoML and combined it with successive halving. The proposed solution evolves a population on each of N islands, and periodically exchanges individuals in the population through migration. This optimization procedure is repeated several times with different resource budgets, using the best individuals from the previous rung as starting populations. The islands each have distinct optimization algorithms, such as differential evolution [194], particle swarm optimization [70] and a $(\mu + 1)$ evolutionary strategy [20]. Three island topologies are explored: a fully disconnected topology, where no migration takes place, a fully connected topology, where individuals migrate to and

⁸Additionally, **GAMA** was used as an out-of-the-box AutoML tool to perform protein abundance prediction [81] and general biomedical applications [245]. Unfortunately, the comparison by Ferreira et al. [81] assigned very different budgets to **TPOT** [179] and **H2O AutoML** [148] which makes it hard to draw any meaningful conclusions about their relative performance. Tsamardinos et al. [245] showed that **GAMA**'s performance did not statistically significantly differ from **JADBio** (the introduced framework) on a large set of biomedical datasets. However, the work mainly focused on AutoML beyond pipeline design, for example, by providing accurate model generalization estimates and identifying interesting feature subsets.

from all islands, and a ring topology, where the islands form a ring and individuals only migrate to and from two adjacent islands. They find that a ring topology provides the best results, and postulate that this is because it allows for a good balance between evolution on each island (exploration) and sharing information through migration (exploitation).

3.4.3 Clustering

In the unsupervised setting, optimization is more subjective as different metrics characterize different properties of clusters. Nevertheless, multiple AutoML for clustering approaches have been proposed [155, 193]. Yildirim et al. [280] adapted **GAMA** to work in the unsupervised setting by defining a search space with `scikit-learn`'s [184] clustering algorithms. In clustering, labeled datasets are typically used to make evaluation more objective. Clusters are generated without knowledge of the class labels but the final evaluation does use the class labels for evaluating the generated clusters. For this reason, the Caliński-Harabasz index [43] is optimized during optimization because it does not require class labels, but the final results are evaluated on the adjusted rand index [232] and adjusted mutual information [262] which take into account the true class labels. They compared **GAMA**'s out-of-the-box search methods of asynchronous evolution and random search and found that evolution outperforms random search, especially for higher resource budgets.

3.5 Conclusion, Limitations, and Future Work

In this chapter, we presented **GAMA**, an open-source AutoML tool that facilitates AutoML research and skillful use through its modular design and built-in logging and visualization. Novice users can make use of the graphical interface to start **GAMA**, or simply use the default configuration which is shown to generate models of similar performance to other AutoML frameworks. Researchers can leverage **GAMA**'s modularity to integrate and test new AutoML search procedures in combination with other readily available building blocks, and then log, visualize, and analyze their behavior, or run extensive benchmarks.

GAMA allows for a more principled evaluation of novel AutoML ideas through ablation studies, comparing design decisions across only one axis of change. It should be noted that when comparing two different optimization methods, any found performance difference is only valid under the other fixed design decisions, e.g., results may differ when considering a different search space. However, this

limitation is not inherent to **GAMA**'s design but holds for any experiment with sufficiently many design decisions.

The modular AutoML pipeline in **GAMA** currently only allows the design of the prototypical pipeline shown in Figure 2.1. However, many other designs are conceivable, e.g., using multiple search algorithms with their own separate search spaces. In general, the AutoML pipeline could be expressed as a directed acyclic graph and contain additional types of steps, e.g., search space design.

In the future, we aim to integrate additional search techniques and additional steps, such as warm-starting the pipeline search with meta-data, so that more pipeline designs are available out-of-the-box. Additionally, we plan to allow for more flexibility in the design of the AutoML pipeline itself. Finally, we aim to greatly increase the tools available to researchers to analyze their AutoML idea. Beyond providing more visualizations and artifacts, we want to provide programmatic hooks to allow researchers for easier real-time interaction and visualizations.

Chapter 4

Reproducible Benchmarks

In this chapter we present work that extends the OpenML platform [258] to enable to use of common benchmarking suites. In this introduction we will first provide a brief overview of other dataset repositories. We subsequently provide a short but comprehensive description of the OpenML platform in Section 4.1. The two sections thereafter detail our contributions, the programmatic interface to the platform called `openml-python` (section 4.2), and the addition of reproducible OpenML benchmarking suites (section 4.3).

Related Work

Evaluating novel (automated) machine learning ideas requires experimental evaluation on datasets. For this purpose, the machine learning field has long recognized the importance of dataset repositories. The UCI repository [66] and LIBSVM [53] offer a wide range of datasets. Many more focused repositories also exist, such as UCR [56] for time series data and Mulan [248] for multilabel datasets. Some repositories also provide programmatic access. `Kaggle.com` and PMLB [178] offer a Python API for downloading datasets, `skdata` [15] and

The work described in this chapter was largely carried out concurrently through an iterative development process.

Section 4.2 is derived from Matthias Feurer et al. “Openml-python: an extensible python api for openml”. In: *Journal of Machine Learning Research* 22.100 (2021), pp. 1–5.

Section 4.3 is derived from Bernd Bischl et al. “OpenML Benchmarking Suites”. In: *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*. 2021.

Both works were used in making this chapter introduction and Section 4.1.

tensorflow [1] offer a Python API for downloading computer vision and natural language processing datasets, and KEEL [4] offers a Java and R API for imbalanced classification and datasets with missing values.

Several platforms can also link datasets to reproducible experiments. Reinforcement learning environments such as the OpenAI Gym [39] run and evaluate reinforcement learning experiments, the COCO suite standardizes benchmarking for blackbox optimization [112] and ASLib provides a benchmarking protocol for algorithm selection [30]. The Ludwig Benchmarking Toolkit orchestrates the use of datasets, tasks and models for personalized benchmarking and so far integrates the Ludwig deep learning toolbox [174]. **PapersWithCode** maintains a manually updated overview of model evaluations linked to datasets.

4.1 OpenML

OpenML is a collaborative online machine learning platform [258]. More than just linking datasets to reproducible experiments, it is meant for sharing results and building on prior empirical machine learning research. OpenML goes beyond the platforms mentioned above, as it includes extensive programmatic access to all experiment data and automated analyses of datasets and experiments, 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 organizes everything based on four fundamental, machine-readable building blocks. These four blocks are shown in Figure 4.1 together with the new blocks we introduce in this chapter. The four blocks on which we built are:

- The *dataset*, tabular datasets that are annotated with rich meta-data such as automatically computed *meta-features*.
- The machine learning *task* to be solved, specifying the dataset, 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.
- The *flow* which specifies a machine learning pipeline that solves the *task*, e.g., an ML pipeline that first performs imputation of missing values and encoding of categorical features, followed by training a Random Forest model.

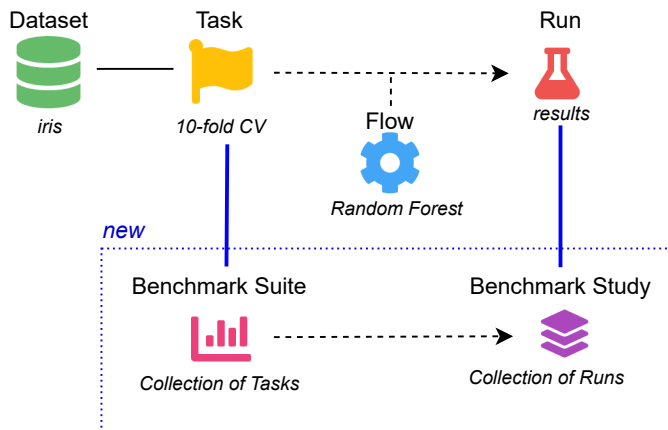


Figure 4.1: Schematic overview of OpenML building blocks, highlighting the new contributions.

- The *run* that contains experiment results (i.e., predictions and performance evaluations) when a *flow* is executed on a *task*.

Each of these are accessible in machine-readable formats, with packages in the Java, Python and R ecosystems [50, 87, 207] to provide easy integration in common machine learning tools, workflows, and environments¹.

OpenML also features a web interface² which allows access and exploration of all the artifacts stored on the platform. It allows finding datasets through a direct search or by filtering based on dataset qualities, and each dataset page features interactive plots and automated exploratory data analysis. For each flow or task, an overview of the stored runs is provided and for each run an analysis of the produced predictions is provided.

4.2 OpenML-Python

`openml-python` is a seamless integration of OpenML into the popular Python ML ecosystem³, that provides easy programmatic access to all OpenML data

¹See <https://docs.openml.org> for more information.

²<https://new.openml.org>

³<https://github.blog/2019-01-24-the-state-of-the-octoverse-machine-learning/>

and automates the sharing of new experiments. In this section, we introduce `openml-python`'s core design, showcase its extensibility to new ML libraries, and give code examples for several common research tasks.

4.2.1 Design and Development

The OpenML platform is organized around several entity types which describe different aspects of a machine learning study. 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. OpenML makes this information available through a REST API, and `openml-python` wraps the complexity of communication with this REST API by providing easy-to-use helper functions and Python objects. `openml-python` closely follows the design of OpenML entities and represents each with separate classes in their own submodules which makes for a natural mapping.

A number of `list` functions allow light-weight access to data stored on OpenML. For example, it is possible to use the `list_datasets` function to query for datasets with specific characteristics, list tasks related to that dataset (using `list_tasks`), and query experimental results for that task (using `list_runs`). To upload new data to OpenML, entities can be created with `create` functions which create the Python objects which may be uploaded through a `publish` method.

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 `openml-python` and machine learning library code. An extension's responsibility is to convert between the libraries' models and OpenML flows, interact with its training interface and format predictions.

We created an extension for *scikit-learn* [184], 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 [40]. Concretely, the *scikit-learn* extension can convert an `OpenMLFlow` to an `scikit-learn 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).

While `openml-python` is best used with an internet connection, it does facilitate offline usage to a degree. All downloaded entities are cached locally, which makes it faster when e.g., fetching the same dataset across multiple sessions and allows subsequent loading of the same datasets without a connection to OpenML. For example, when running large scale experiments it is possible to first download all the required entities (e.g., datasets, tasks and flows), then conduct experiments offline on cached data, and later upload results when a connection is available again.

The package is developed publicly on Github, uses continuous integration, and features documentation with a mix of tutorials, examples and API documentation. It builds on standard open-source packages for scientific computing such as `numpy` [114], `scipy` [263], and `pandas` [235], which means it integrates well with ML in Python. The package is written in Python3 and open-sourced with a 3-Clause BSD License.⁴

4.2.2 Related Work

There are other Python packages for importing datasets, such as PMLB [178], and submodules of both `scikit-learn` [184] and `tensorflow` [1], but these offer no support for reproducible experiments such as provided train/test splits. On the other hand, `Kaggle.com`, an online platform mostly known for its company-sponsored competitions, provides a versatile platform to share and collaboratively work with datasets. `Kaggle.com` itself is closed source and cannot be extended and developed by the research community. Their Python API provides functionality to up- and download datasets and so-called kernels to their webserver. However, datasets are neither required to adhere to the same format and can therefore not be automatically ingested. Similarly, there is no central and consistently formatted storage of the experiments, which makes it hard to build on previous results and conduct large scale analyses.

4.2.3 Use Cases

Here follow a few of the use cases of OpenML and code examples of how to perform them with `openml-python`. Further information, including advanced examples on how OpenML-Python was used in previous publications, can be

⁴Source: <http://github.com/openml/openml-python>

found in the online documentation.⁵

Finding and downloading 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* [114], *scipy* [263] or *pandas* [235] 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.

Listing 4.1 shows how to query for datasets with specific characteristics and download one.⁶ Here we use `list_datasets` to query datasets that have between 100 and 200 instances (rows), more than 2 classes and no missing values, and request at most 5 results. If we are interesting in any one particular dataset, for example ‘iris’, we can download it with the `get_dataset` function and inspect the data.

Performing reproducible ML experiments OpenML tasks and flows together describe all aspects of an experimental setup. This can be used to conduct reproducible ML experiments, but it can be tedious (and error-prone) to initialize the models and setup the data splits through the basic building blocks. `openml-python` provides a simplified interface which automates much of this process. Listing 4.2 shows how to conduct a reproducible experiment which evaluates the predictive accuracy of a decision tree using 10-fold cross-validation on the ‘Iris’ dataset and upload the results to OpenML. To achieve this simple interface, `run_model_on_task` internally uses the `scikit-learn` extension to convert the `DecisionTreeClassifier` to a *flow*, reads the *task* to split the *dataset*, and use the `DecisionTreeClassifier`’s `fit` and `predict` functions.

Using published results Experiment data on OpenML is plentiful and allows interesting analysis of e.g., hyperparameter importance [255] or algorithm performance [233]. In Listing 4.3 we show how to produce a contour plot, as shown in Figure 4.2, which shows the effect of hyperparameters C and γ of an SVM on its accuracy for the letter dataset using experiment data already available on OpenML. In particular, lines 5-8 retrieve all experiment data for an SVM flow on a 10-fold cross-validation task on the ‘letter’ dataset with `openml-python`.

⁵Documentation, extensions and examples: <https://openml.github.io/openml-python>

⁶The example specifies the dataset by name for convenience. To guarantee the exact same version of the dataset is downloaded, a numeric identifier should be used.

Listing 4.1: Code for listing and retrieving datasets. In this listing only, the code (prefix: >>>) and output are interleaved. Output is abridged and formatted for display in this document.

```

1 >>> import openml
2 >>> openml.datasets.list_datasets(
3     output_format="dataframe",
4     number_instances="100..200",
5     number_missing_values="0",
6     size=5,
7 )
8
9 did name      NumberOfFeatures      NumberOfInstances  ...
10 10  lymph      19.0                148.0                ...
11 48  tae         6.0                151.0                ...
12 61  iris        5.0                150.0                ...
13 62  zoo        17.0                101.0                ...
14 164 mol        58.0                106.0                ...
15
16 >>> iris = openml.datasets.get_dataset("iris")
17 >>> iris.get_data()
18
19 SEPALLENGTH SEPALWIDTH PETALLENGTH PETALWIDTH CLASS
20 5.1          3.5         1.4          0.2      setosa
21 4.9          3.0         1.4          0.2      setosa
22 ...         ...         ...         ...      ...
23 6.2          3.4         5.4          2.3      virginica
24 5.9          3.0         5.1          1.8      virginica

```

Listing 4.2: Automatically performing 10-fold cross-validation with a decision tree on ‘iris’ (task 59) and uploading the results (requires an API key).

```

1 import sklearn.metrics, sklearn.tree, openml
2
3 iris_task = openml.tasks.get_task(59)
4 model = sklearn.tree.DecisionTreeClassifier()
5 run = openml.runs.run_model_on_task(model, iris_task)
6 run.publish()

```


The remainder, lines 9-17, only processes and visualizes the obtained data.

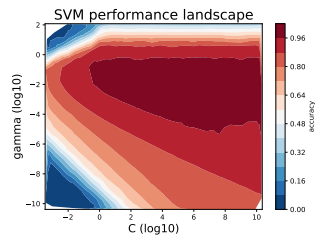


Figure 4.2: SVM hyperparameter contour plot generated by the code in Listing 4.3.

Listing 4.3: Code for retrieving the predictive accuracy of an SVM classifier on the ‘letter’ dataset and creating a contour plot with the results.

```
1 import openml
2 import numpy as np
3 import matplotlib.pyplot as plt
4
5 # Choose an SVM flow (e.g. 8353),
6 # and the dataset 'letter' (task 6).
7 df = openml.evaluations.list_evaluations_setups(
8     'predictive_accuracy', flows=[8353], tasks=[6],
9     output_format='dataframe',
10     parameters_in_separate_columns=True,
11 )
12 hp_names = [
13     'sklearn.svm.classes.SVC(16)_C',
14     'sklearn.svm.classes.SVC(16)_gamma'
15 ]
16 df[hp_names] = df[hp_names].astype(float).apply(np.log)
17 C, gamma, score = df[hp_names[0]], df[hp_names[1]], df['value']
18
19 cntr = plt.tricontourf(
20     C, gamma, score, levels=12, cmap='RdBu_r'
21 )
22 plt.colorbar(cntr, label='accuracy')
23 plt.xlim((min(C), max(C)))
24 plt.ylim((min(gamma), max(gamma)))
25 plt.xlabel('C (log10)', size=16)
26 plt.ylabel('gamma (log10)', size=16)
27 plt.title('SVM performance landscape', size=20)
```

4.3 Benchmarking Suites

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.

However, in practice machine learning researchers have benchmarked their algorithms on often ad-hoc subsets of dataset repositories such as UCI [66] or LivSVM [53]. 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 [2], are hard to reproduce [117, 183], and even lead to contradictory results [130]. 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 [219].

OpenML provides all the building blocks for creating curated benchmarks, such as meta-data rich datasets, tasks, flows and runs. However, OpenML did not yet facilitate the simple creation and sharing of well-designed benchmark suites and results of experiments ran on them. In this last part of the chapter 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 **Curated Classification benchmarking suite 2018** (OpenML-CC18)

4.3.1 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 al-

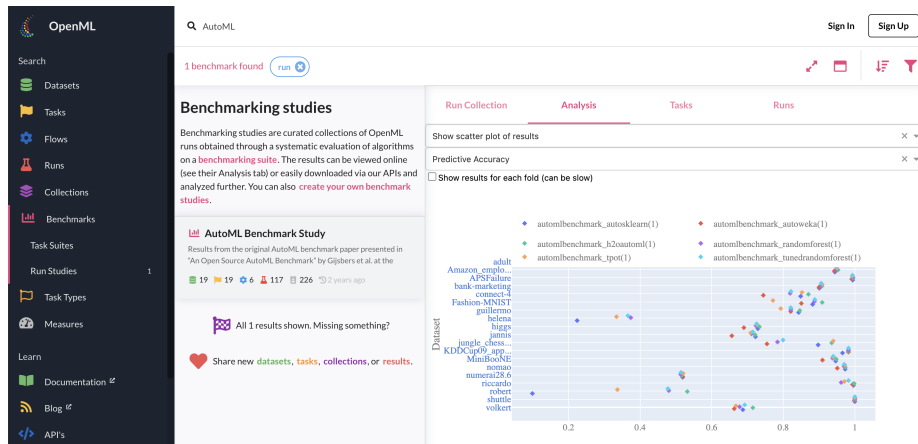


Figure 4.3: OpenML website showing a list of benchmark studies on the left, and interactive exploration of the results of the AutoML Benchmark (see Chapter 5) on the right. Can be viewed online at <https://www.openml.org/s/226>.

low 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 4.3.2): OpenML hosts thousands of datasets, and scientists can easily filter them down to those needed for their benchmarks.
- 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 4.3 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 4.4. 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.

4.3.2 How to Use OpenML Benchmarking Suites

To realize all these benefits, we have developed a series of extensions to the OpenML platform:⁷

⁷All code is open, BSD-3 licenced, and available on <https://github.com/openml>

- 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.
- 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 a code example on how to retrieve, iterate the contents of a benchmark suite and run machine learning algorithms on it in Listing 4.4.⁸

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.⁹ 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 A.2.

⁸<https://docs.openml.org/benchmark> has up-to-date instructions for Python, Java and R.

⁹Notebooks can be found at <https://github.com/openml/benchmark-suites>

Listing 4.4: Running a large scale benchmark study with openml-python

```
1  from sklearn import compose, impute, metrics, pipeline, tree
2  from sklearn.preprocessing import OneHotEncoder
3  from openml import config, study, tasks, runs, extensions
4  from openml.extensions.sklearn import cat, cont
5
6  var imputer = (impute.SimpleImputer(), cont)
7  var encoder = (OneHotEncoder(handle_unknown='ignore'), cat)
8  clf = pipeline.make_pipeline(
9      compose.make_column_transformer(imputer, encoder),
10     tree.DecisionTreeClassifier(max_depth=1)
11 )
12
13 benchmark_suite = study.get_suite('OpenML-CC18')
14 # config.apikey = 'OPENML_API_KEY' # For uploads
15
16 run_ids = []
17
18 for task_id in benchmark_suite.tasks:
19     task = tasks.get_task(task_id)
20     run = runs.run_model_on_task(clf, task)
21     score = run.get_metric_fn(metrics.accuracy_score)
22     print(f'{task.get_dataset().name}: {score.mean():.2} acc')
23     run.publish() # Requires API-Key
24     run_ids.append(run.id)
25
26 benchmark_study = study.create_study(
27     name="CC18-Example",
28     description="An example decision stump study.",
29     run_ids=run_ids,
30     benchmark_suite=benchmark_suite.id
31 )
32
33 benchmark_study.publish() # Requires API-key
34 print(f"Results stored at {benchmark_study.openml_url}")
```

Retrieving Existing Suites

Existing benchmark suites can be easily downloaded via any of the OpenML client libraries using its unique id or alias (see Listing 4.4). The tasks and datasets are all uniformly formatted, and come 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 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.

Running Benchmarks

After retrieving the tasks from a suite, new experiments can be conducted locally. As illustrated in Figure 4.4, this is easiest with the readily integrated machine learning libraries. We provide support for running experiments with `scikit-learn` [184] in Python, `mlr` [26] or its successor `mlr3` [145] in R, and `Weka` [111] in Java. Integrations for deep learning libraries are under development, and we welcome further open source integrations.¹⁰ 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 in Figure 4.4. 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 4.4 visualizes the results of 3.8 million runs collected on a single benchmarking suite, which we will discuss next.

4.3.3 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

¹⁰Development is carried out on GitHub, see: <https://docs.openml.org>.

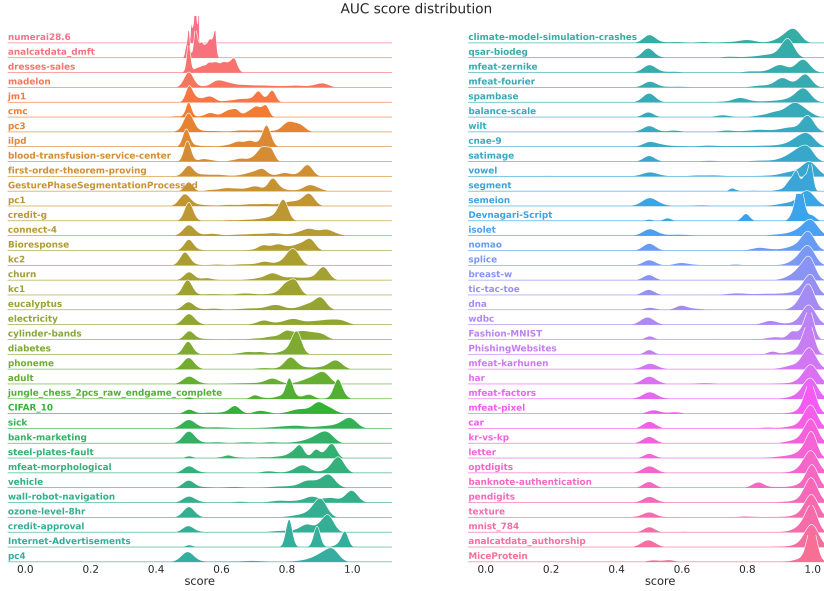


Figure 4.4: Distribution of the scores (average area under ROC curve, weighted by class support) of 3.8 million experiments with thousands of machine learning pipelines, shared on the CC18 benchmark tasks. Some tasks prove harder than others, some have wide score ranges, and for all there exist models that perform poorly (0.5 AUC). Code to reproduce this figure (for any metric) is available on GitHub.⁹

Table A.1 in the appendix. We first describe the design criteria of the OpenML-CC18 before discussing uses of the benchmark and success stories.

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. The selected datasets must be annotated with their source, contain data that is not artificially generated or derived from another dataset, and be small enough to allow for models to be trained on almost any computing hardware (i.e., 500 - 100k samples and less than 5000 features after one-hot encoding categorical variables). From the remaining datasets we select

only reasonably balanced classification tasks of which the observations may be assumed to be independently and identically distributed. Finally, to ensure that datasets are sufficiently challenging, we removed datasets which are easily solved by a decision tree.

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).

Usage of the OpenML-CC18

The OpenML-CC18 has been acknowledged and used in various studies. For instance, Van Wolputte and Blockeel [256] used it to study iterative imputation algorithms for imputing missing values, König, Hoos, and Rijn [136] used it to develop methods to improve upon uncertainty quantification of machine learning classifiers, and De Bie et al. [63] 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 [22, 23] and using the individual columns to test quantile sketch algorithms [166].

Cardoso et al. [47] 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. [79] 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 [255] and Probst, Boulesteix, and Bischl [196] used it to quantify the hyperparameter importance of machine learning algorithms, while Probst, Wright, and Boulesteix [197] used it to learn the best strategy for tuning random forest based on large-scale experiments (although Probst, Boulesteix, and Bischl [196] and Probst, Wright, and Boulesteix [197] 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.

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. A benchmark suite that contains trading prices and technical analysis features of various currency pairs for evaluating machine learning algorithms for Foreign Exchange was created by Schut, Rijn, and Hoos [216]. Strang et al. [233] investigate on which types of datasets linear classifiers can be competitive to non-linear classifiers. Since the hypothesis is that this happens on smaller datasets, they have replicated the OpenML-100 suite and relaxed the exclusion criteria to also including small datasets (starting with 10 data points). A large amount of datasets from PubChem have been annotated and made available as an OpenML benchmarking suite by Olier et al. [177]. Mantovani et al. [159] aim to predict when hyperparameter tuning improves SVM classifiers, and have made the datasets that they experiment on available as benchmark suite. Finally, there are the AutoML benchmarking suites which will be discussed in more detail in the next chapter.

4.4 Conclusion and Future Work

OpenML is a platform for collaborative machine learning which allows researchers to define reproducible experimental setups, share ML experiment results, and build on the results of others. In this chapter we introduced `openml-python`, which provides an easy-to-use interface to OpenML in Python, and OpenML benchmarking suites, which are a collection of curated tasks to evaluate algorithms under a precise set of conditions. 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, analyzed, and used by others in their research.

`openml-python` makes it easy for people to share and reuse datasets, meta-data, and empirical results of ML experiments. It has already been used to scale up studies with hundreds of consistently formatted datasets [85, 94], supply large amounts of meta-data for meta-learning [186], answer questions about algorithms such as hyperparameter importance [255] and facilitate large-scale comparisons of algorithms [233]. In the future we hope to improve support for deep learning experiments through e.g., extensions for frameworks such as tensorflow [1].

We introduced OpenML benchmarking suites, a new benchmarking layer on the OpenML platform that allows scientists to download, share, and compare results with just a few lines of code. We then introduced the OpenML-CC18, a benchmark suite created with these tools for general classification benchmarking. We reviewed how other scientists have adopted OpenML-CC18 and other 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.

Recently some conferences have recognized that creating a benchmarking suite requires a lot of work and introduced tracks such as NeurIPS' *datasets and benchmarks* and AutoMLConf's *Systems, Benchmarks and Challenges* which helps authors to expose their work and receive credit for it. However, the introduction of benchmarking suites is free-form which makes it harder for reviewers to evaluate their value, and for users to find a benchmarking suite relevant to their research. Similar to datasheets [97] which provide a rich context about a dataset, such as why it was created, how it was created, whether there are ethical concerns, and the intended use for the dataset, a standard sheet for benchmark-

ing suites could be constructed. This would help authors of benchmark sheets communicate with its users and further streamline the benchmark suite creation process, as the questionnaire helps the authors to reflect on each stage of the creation process. Such a ‘benchmarking sheet’ could be easily citable, integrated with the OpenML platform, further increase the quality of benchmarking suites and streamline their review process.

Big benchmarking suites help evaluate algorithms on a wide range of domains or dataset characteristics. However, for some purposes a large number of tasks might be unnecessary. Cardoso et al. [47] a post-hoc analysis is used to find a representative subset of tasks for the OpenML-CC18. The rich experimental data on OpenML could perhaps be used to shrink the benchmarking suite before publication by automatically analyzing results of previous experiments, or to propose tasks to add to a benchmarking suite to improve its expressiveness. Achieving a similar ability for a benchmarking suite to differentiate algorithms while using fewer tasks enables researchers with limited budgets and reduces the environmental impact benchmarking studies have.

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 [201]) and invite the community to extend existing suites with harder tasks, as done in NLP research [131].

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.

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.

Chapter 5

The AutoML Benchmark

With considerable effort being spent on developing and improving AutoML tools [285], as well as increased usage by practitioners [252], comes the need to compare the different tools and track progress in the field. However, comparing AutoML tools leaves much room for error. Issues may arise from not knowing how to correctly install, configure, or use ‘competitor’ frameworks, for instance by misunderstanding memory management and/or using insufficient compute resources [9], or failing to use comparable resource budgets [81]. Additionally, we observe that no common benchmarking suites are employed for evaluating AutoML frameworks and most published AutoML papers use a self-selected set of datasets on which to evaluate their methods. This inconsistency makes it hard to compare results across papers, and also allows for presenting cherry picked results.

In this chapter, we present an open source AutoML benchmark.¹ It consists of an easy to use benchmarking tool for reproducible research on a curated list of high quality datasets. The benchmarking tool can be used to perform fully automated AutoML evaluations, and integration of the AutoML frameworks is developed together with the original AutoML contributors to ensure correctness. We carry out a large scale evaluation of 9 AutoML frameworks across 71 classification and 33 regression tasks and report on the results from various

This chapter is based on a work-in-progress paper, scheduled to be submitted to JMLR. We presented a first look at this work at the ICML 2019 AutoML workshop:

Pieter Gijsbers et al. “An open source AutoML benchmark”. In: *arXiv preprint arXiv:1907.00909* (2019)

Since the workshop presentation, Stefan Coors and Marcos L. P. Bueno also joined the project.

¹<https://openml.github.io/automlbenchmark/>

perspectives. Finally, we provide an interactive visualization tool which may be used for further exploration of the results.

The rest of the chapter is structured as follows. We discuss related benchmarking literature in Section 5.1, followed by an overview of integrated AutoML frameworks in Section 5.2. In Section 5.3, we provide an overview of the benchmarking tool and how to use it. We then motivate our benchmark design choices and its limitations in Section 5.4 and report on the results obtained by running the benchmark in Section 5.5. In Section 5.6 conclude our paper and sketch directions for future work.

5.1 Related Work

Several benchmark suites have been developed in machine learning [28, 178, 254, 275]. These datasets often do not include problematic data characteristics found in real world tasks (e.g., missing values) because many ML algorithms are not able to handle them natively. By contrast, AutoML frameworks should be designed to handle these problematic data characteristics to be applicable to a wide range of data. This makes relaxing these practical restrictions on the selection of datasets not only possible, but indeed interesting as the way in which AutoML frameworks handle these issues provides new dimensions in which they can be compared. Moreover, runtime budgets are often not specified in traditional ML benchmarks as the algorithms can run to completion (one exception is performance studies, such as [138]), yet they are a requirement in an AutoML benchmark as most AutoML frameworks are designed to optimize until a given time budget is exhausted.

In the remainder of this section we will discuss some of the many experimental evaluations of AutoML frameworks. In the process we highlight some of the issues that are encountered. We stress that we do not mean to discredit their authors and similar issues can be found in other papers.

Balaji and Allen [9] conducted one of the first benchmark studies on AutoML tools. They evaluated four open-source frameworks on both classification and regression tasks sourced from OpenML, optimized for weighted F1 score and mean squared error, respectively. Unfortunately, they encountered technical issues with most AutoML tools which led to a questionable experimental evaluation. For example, `H2O AutoML` [148] was configured to optimize to a different metric (log loss as opposed to weighted F1 score) and ran with a different setup (unlike the others, `H2O AutoML` was not containerized), and `auto_ml` [182] had its hyperparameter optimization disabled.

A study on nearly 300 datasets across six different frameworks was conducted by Truong et al. [244]. Each experiment consisted of a single 80/20 hold out split on a 15-minute training time budget, which was chosen so that most tools returned a result on at least 70% of the datasets. We postulate it is reasonable to assume that the datasets for which no result is returned by a framework correlate strongly with datasets for which optimization is hard. For example, a big dataset might cause one framework to conduct only few evaluations while it completely halts another. Unfortunately this makes results uninterpretable when comparing aggregate performance (e.g., through the box-whisker plots used), because a tool could show better performance because it failed to return models on datasets for which optimization was hard. Truong et al. present their results across different subsets of the benchmark, e.g., few versus many categorical features, which helps highlighting differences between different frameworks. The authors also conduct small-scale experiments to analyze performance over time by running the tools on multiple time budgets on a subset of datasets, as well as the ‘robustness’ which denotes the variance in final performance given the same input data. Unfortunately, both experiments were conducted on only one dataset per sub-category, which does not lend to generalizing the results.

Zöller and Huber [285] present a survey and benchmark on AutoML and combined algorithm selection and hyperparameter optimization (CASH [238]) frameworks. Six CASH frameworks and five AutoML tools are compared across 137 classification tasks, the former have a limit of 325 iterations while the latter are constrained to a one hour time limit. The comparison of CASH frameworks gives insight into the effectiveness of different optimization strategies on the same search space (**hyperopt** [18] performed best though absolute differences were small between all optimizers). The AutoML tools are compared as they are, which means the comparison might reflect a real life use case more closely with the drawback that conclusions about the effectiveness of individual parts of the system are not possible. A number of errors are observed during the experiments, including memory constraint violations, segmentation faults and Java server crashes. When analyzing the generated pipelines from different tools, the authors find that current tools construct rather modest pipelines (few preprocessing operators) and suggest that perhaps search should be expanded to explore more complicated pipelines.

Kaggle.com, a platform for data science competitions, is sometimes used to compare AutoML tools to human data scientists [73, 285]. The comparison by Zöller and Huber [285] found that the best AutoML framework on the benchmark was different than the best in Kaggle competitions (**TPOT** and **H2O AutoML**, respectively). They find humans take approximately 8.5 hours to build a model

as good the best AutoML tool does in one hour, though the best AutoML model is still bad compared to the best human-made model. Erickson et al. [73] compare on a larger set of Kaggle tasks, including classification and regression, and find the tools are able to outperform anywhere from 20% (**AutoWEKA**) to 70% (**AutoGluon-Tabular**) of competitors on a 4 hour budget.

However, it is hard to interpret these results as it is typically unclear how to interpret scores on the Kaggle leaderboard. Submissions can range from serious attempts by ML experts to students or even people only testing the upload functionality. For example, in one report a framework outperformed “99.3% of participating data scientists” while 42.5% of all submissions did not outperform the baseline which always predicts the majority class. Similarly, when computing the time spent on a submission, it seems unreasonable to assume that all time between submissions is spent working on improving the model.

A benchmark on AutoML for multi-label classification, where a data instance can have multiple labels simultaneously, was presented by Wever, Mohr, and Hüllermeier [268]. The authors develop a framework with a configurable search space and optimizer which allows for new improvements to be proposed in isolation alongside an ablation study, as opposed to the common practice of changing multiple aspects of the AutoML pipeline at once (typically together as a new tool). The disadvantage is that existing tools can’t be directly evaluated, instead each of their components first need to be reimplemented or wrapped into the benchmark framework so that a tool can be reconstructed within the benchmark framework. Five different optimizers are compared across 24 datasets, finding that Hierarchical Task Network planning worked best, though the comparison restricts itself only to the CASH problem as opposed to finding complete machine learning pipelines that may include preprocessing steps.

In addition to AutoML benchmarks, a series of competitions for tabular AutoML was hosted [110]. The first two competitions focused on tabular AutoML where data is assumed to be independent and identically distributed. In the competitions participants had to submit code which automatically builds a model on given data and produce predictions for a test set. During the development phase, competitors could make use of a public leaderboard and several validation datasets. After the development phase, the latest submissions of each participant would be evaluated on a set of new datasets to determine the final ranking. Datasets consisted of a mix of both new data and data taken from public repositories, though they were reformatted to conceal their identity. In their analysis Guyon et al. reveal most methods fail to return results on at least some datasets due to practical issues (e.g., running out of memory).

5.2 AutoML Tools

Automated machine learning pipeline design was first explored by Statnikov, Aliferis, and Tsamardinos [230] to automate cancer diagnosis from gene expression data. Their method, later called **GEMS** (for Gene Expression Model Selector [231]), automatically performed pipeline design through grid search. Automated pipeline design was later independently explored in a domain-agnostic setting by Escalante, Montes, and Sucar [75] and the first prominent general-purpose AutoML framework was **Auto-WEKA** [238]. **Auto-WEKA** used Bayesian optimization to select and tune the algorithms in a machine learning pipeline based on **WEKA** [111]. Over time, countless new AutoML frameworks have been developed either by iteratively improving on old designs, or using novel approaches. In this section we will discuss the AutoML frameworks we evaluate in this chapter.

Unfortunately the cost of evaluating all frameworks is prohibitive, so we selected only 9 of them. Only open source tools were considered, and from those we made picks to cover a variety of different approaches. We considered both frameworks developed by industry and academia, and included packages whose authors proactively integrated their AutoML framework.

The most notable omission is **Auto-WEKA**, which we decided to exclude based on the performance in our 2019 evaluation and its lack of updates since then [100]. Other integrated tools which are not included in the evaluation are **autoxgboost** [237], because the author opted out due to the framework being built on deprecated software, and **ML-Plan** [169] and **mlr3automl**² because we experienced odd behavior when running the experiments.³ There are still many AutoML frameworks not yet integrated which we hope to include in the future, e.g., **Auto-Keras** [124], **AutoPyTorch** [284], and **BOHB** [80].

5.2.1 Integrated Frameworks

Table 5.1 offers an overview of the AutoML tools evaluated in this paper. These aspects are simplified, and we brief description with more detail of each framework below.

AutoGluon-Tabular **AutoGluon** automates machine learning across a variety of tasks including image, text and tabular data. The subsystem which

²<https://github.com/a-hanf/mlr3automl/>

³**ML-Plan** and **mlr3automl** are still planned to be included in the paper submission to JMLR.

framework	optimization	search space
autogluon	custom	predefined pipelines
autosklearn	Bayesian	scikit-learn pipelines
autosklearn 2	Bayesian	iterative algorithms
flaml	CFO	iterative algorithms
GAMA	Evolution	scikit-learn pipelines
H2OAutoML	Random Search	H2O pipelines
lightautoml	Bayesian	Linear model, GBM
mljarsupervised	custom	python modules
TPOT	Evolution	scikit-learn pipelines

Table 5.1: Used AutoML frameworks in the experiments.

automates machine learning on tabular data is called **AutoGluon-Tabular** [73], which for the remainder of this chapter we will simply refer to as **AutoGluon**. In contrast to other AutoML systems discussed here, it does not perform a pipeline search or hyperparameter tuning. Instead, it has a predetermined set of models which are combined through multi-layer stacking and ensembling.

AutoGluon's ensemble consists of three layers. The first layer are models from a range of model families trained directly on the data. In the second layer the same type of models are considered, but as a stacking learner trained with both the input data and the predictions of the first layer. In the final layer the predictions of the second-layer models are combined into an ensemble [49].

To adhere to time constraints **AutoGluon** may stop iterative algorithms prematurely or forgo training certain models altogether. Given more time **AutoGluon** will train additional models using the same algorithms and hyperparameter configurations on different data splits, which further improves the generalization of the stacking layer.

auto-sklearn Based on the design of **Auto-WEKA**, **auto-sklearn** [85] also uses Bayesian optimization but is instead implemented in Python and optimizes pipelines built with **scikit-learn** [184]. Additionally, it warm-starts optimization through meta-learning, starting pipeline search with the best pipelines for the most similar datasets [88]. After pipeline search has concluded, an ensemble is created from pipelines trained during search. **Auto-sklearn** has won two AutoML Challenges [110], though for both entries **auto-sklearn** was customized for the competition and not all changes are found in the public releases [83].

Based on experience from the challenges, ‘`auto-sklearn 2.0`’ was developed [82]. The most notable changes include reducing the search space to only iterative learning algorithms and excluding most preprocessing, use of successive halving [123], adaptive evaluation strategies, and replacing the data-specific warm-start strategy with a data-agnostic portfolio of pipelines. Because these changes make version 2.0 almost entirely different to 1.0, and 1.0 has been updated since our last evaluation, we evaluate both `auto-sklearn` versions in this paper. However, `autosklearn 2.0` does not yet support regression and its heavy use of meta-learning made it impossible for us to perform a ‘clean’ evaluation at this time (see Section 5.4.3).

FLAML FLAML [265], short for fast and lightweight AutoML library, which optimizes boosting frameworks (`xgboost`, `catboost`, and `lightgbm`) and a small selection of `scikit-learn` algorithms through a multi-fidelity randomized directed search [273]. This search is based on an expected cost for improvement, which tracks for each learner the expected computational cost of improving over the best found model so far. Only after choosing which learner to tune, hyperparameter optimization proceeds by a randomized directed search, sampling a new configuration from a unit sphere around the previous sample point. After evaluating its validation performance, the next sample point is moved to that direction (if better) or the opposite direction (if worse). FLAML positions itself as a fast AutoML framework that can find good models in minutes [265].

GAMA Described in detail in Chapter 3, GAMA is designed as a modular AutoML tool for researchers [103]. By default GAMA uses the asynchronous evolutionary optimization described in Section 3.2.1 to optimize `scikit-learn` pipelines, and ensembles them in a post-processing step as described in Section 3.2.2.

H2O AutoML Built on the scalable H2O machine learning platform, H2O AutoML [148] evaluates a portfolio of algorithm configurations and also performs a random search over the majority of the supervised learning algorithms offered in H2O. To maximize accuracy, H2O AutoML also trains two types of stacked ensemble models at various stages during the run: an ensemble using all available models at time t , and an ensemble with only the best models of each algorithm type at time t . H2O AutoML relies on high performance implementations of algorithms inside H2O, to cover a large search space quickly, and relies on stacking to boost model performance. H2O AutoML uses a predefined strategy for impu-

tation, normalization and categorical encoding for each algorithm and does not currently optimize over preprocessing pipelines. The **H2O AutoML** algorithm is designed to generate models that are very fast at inference time, rather than strictly focusing on maximizing model accuracy, with the goal of balancing these two competing factors to produce practical models that can be practically used in production environments.

Light AutoML **Light AutoML** is specifically designed with applications in the financial services industry in mind [249]. Pipelines are designed for quick inference and interpretability. Only linear models and GBMs are considered, and their hyperparameters are tuned in three steps. First, expert rules are used to evaluate likely good hyperparameter configurations. Second, Tree-structured Parzen Estimators [16] are used as the time-budget allows to optimize hyperparameters in a data-driven way. A final stage of tuning is performed with grid search. In the final model construction step, different models are combined in either a weighted voting ensemble (binary classification and regression) or with two levels of stacking (multi-class classification). In a special “compete” mode for larger time budgets, the AutoML pipeline is ran multiple times and their resulting models are ensembled with weighted voting.

MLJar Similar to H2O, search starts with a set of predetermined models and a limited random search. This is followed by a feature creation and selection step, after which a hill climbing algorithm is used to further tune the best pipelines. After search, the models can be stacked, used in a voting ensemble, or both. The search space contains many `scikit-learn` algorithms, but also the boosting frameworks `xgboost`, `catboost`, and `lightgbm` and the neural network frameworks `Keras` and `Tensorflow`.

TPOT Tree-based Pipeline Optimization Tool [179], or TPOT, optimizes pipelines using genetic programming. Using a grammar, machine learning pipelines can be expressed as trees where different branches represent distinct preprocessing pipelines. These pipelines are then optimized through evolutionary optimization. To reduce overfitting that may arise from the large search space, multi-objective optimization is used to minimize for pipeline complexity while optimizing for performance. It is also possible to reduce the search space by specifying a pipeline template [147], which dictates the high-level steps in the pipeline (e.g. “Selector-Transformer-Classifier”). Development has been focused around genomics studies, providing specific options for dealing with this type of

high dimensional data for which prior knowledge may be present [227]. While TPOT supports neural networks in its search [210], the default search space uses `scikit-learn` components and `XGBoost` [55] only.

5.2.2 Baselines

In addition to the integrated frameworks, the benchmark tool allows for running several baselines. The `constant predictor` always predicts the class prior or mean target value, regardless of the values of the independent variables. The `Random Forest` baseline builds a forest 10 trees at a time, until one of two criterion is met: we expect to exceed 90% of the memory limit or time limit by building 10 more trees, or 2000 trees have been built.

The `Tuned Random Forest` baseline improves on the `Random Forest` baseline by using an optimized `max features` value. The `max features` hyperparameter defines how many features are considered when determining the best split, and is found to be the most important hyperparameter [255]⁴. The value is optimized by evaluating up to 11 unique values for the hyperparameter with 5-fold cross-validation, before training a final model with the best found value. The `Tuned Random Forest` is our strongest baseline and could mimic the first effort of a data scientist.

Recently, Mohr and Wever [168] proposed to introduce a baseline which aims to emulate the optimization that a data scientist might perform. In several steps, including feature scaling, feature selection, hyperparameter tuning, and model selection. We omit it here because it does not support regression and it was published late into our preparation for the experiments.

5.3 Software

We developed an open source benchmark tool which may be used for reproducible AutoML benchmarking.⁵ It features robust automated experiment execution and has support for multiple AutoML frameworks, many of which are evaluated in this paper. The benchmark tool is implemented as a Python application consisting mainly of an *amlb* module and a *framework* folder hosting all the officially supported extensions, which have been developed together with

⁴`min. samples leaf` is more important, but not significantly. It is not obvious the absolute values used for `min. samples leaf` transfer as well to our datasets as the relative values used in `max features`.

⁵<https://github.com/openml/automlbenchmark>

AutoML framework developers. The main consideration for the design of the benchmark tool is to produce correct and reproducible evaluations. That is to say that the AutoML tools are used as intended by their authors with little to no room for user error, and the same evaluation conditions (e.g., framework version, dataset, resampling splits) and controlled computational environments can easily be recreated by anyone. The *amlb* module provides the following features:

- a data loader to retrieve and prepare data from OpenML or local datasets.
- various benchmark runner implementations:
 - a *local* runner: which runs the experiments directly on the machine. This is also the runner to which each runner below delegates the final execution.
 - *container* runners (*docker* and *singularity* are currently supported): this allows to preinstall the *amlb* application together with a full setup of one framework, and consistently run all benchmark tasks against the same setup. It also makes it possible to run multiple container instances in parallel.
 - an *aws* runner that allows the user to safely run the benchmark on several EC2 instances in parallel. Each EC2 instance can itself use a pre-built docker image, as used for this paper, or can configure the target framework on the fly, e.g., for experiments in a development environment.
- a job executor responsible to run and orchestrate all the tasks. When used with the *aws* runner, this allows to distribute the benchmark tasks across hundreds of EC2 instances in parallel, each one being monitored remotely by the host.
- a post-processor responsible for collecting and formatting the predictions returned by the frameworks, handling errors, and computing the scoring metrics before writing the information needed for post-analysis to a file.

5.3.1 Extensible Framework Structure

To make sure that the benchmark tool is easily extensible for new AutoML frameworks, we integrate each tool through a minimal interface. Each of the current tools require less than 200 lines of code across at most four files (most

of which is boilerplate). The integration code takes care of installation of the AutoML tool and its software stack, as well as providing it with data and recording predictions. The integration requirements are minimal, as both input data and predictions can be exchanged both in Python objects and common file formats, which makes integration across programming languages possible (currently integrated frameworks are written in C#, Java, Python and R). By keeping the integration requirements minimal, we hope that AutoML framework authors are encouraged to contribute integration scripts for their framework, and at the same time avoid influencing the methods or software used to design and develop new AutoML frameworks (as opposed to providing a generic starter kit which may bias the developed AutoML frameworks [110]). Frameworks may also be integrated completely locally, to allow for private benchmarking.⁶

5.3.2 Extensible Benchmarks

Benchmark suites define the datasets and one or more train/test splits which should be used to evaluate the AutoML frameworks. The benchmark tool can work directly with OpenML tasks and suites, allowing new evaluations without further changes to the tool or its configuration. This is the preferred way to use the benchmark tool for scientific experiments, because it guarantees that the exact evaluation procedure can be reproduced easily by others. However, it is also possible to use datasets stored in local files with manually defined splits, for example to benchmark private use cases.⁷

5.3.3 Running the tool

To benchmark an AutoML framework, the user first needs to identify and define:

- the *framework* against which the benchmark is executed,
- the *benchmark* suite listing the tasks to use in the evaluation, and
- the *constraint* that needs to be imposed on each task. This includes:
 - the maximum training time.

⁶<https://github.com/openml/automlbenchmark/blob/master/docs/HOWTO.md#add-an-automl-framework>

⁷<https://github.com/openml/automlbenchmark/blob/master/docs/HOWTO.md#add-a-benchmark>

- the amount of CPU cores that can be used by the framework: not all frameworks respect this constraint, but when run in *aws* mode, this constraint translates to specific EC2 instances, therefore limiting the total amount of CPUs available to the framework.
- the amount of memory that can be used by the framework: not all frameworks respect this constraint, but when run in *aws* mode, this constraint translates to specific EC2 instances, therefore limiting the total amount of memory available to the framework.
- the amount of disk volume that can be used by the framework (only respected in *aws* mode).

Those constraints must then be declared explicitly in a *constraints.yaml* file (also in the *resources* folder or as an external extension).

Commands

Once the previous parameters have been defined, the user can run a benchmark on the command line using the basic syntax:

```
$ python runbenchmark.py framework_id benchmark_id constraint_id
```

For example, to evaluate the tuned random forest baseline on the classification suite:

```
$ python runbenchmark.py tunedrandomforest openml/s/271 1h8c
```

Additional options may be used to specify e.g., the *mode* or the *parallelization*. For example, the following command may be used to evaluate the random forest baseline on the regression benchmark suite across 100 *aws* instances in parallel.

```
$ python runbenchmark.py randomforest openml/s/269 1h8c -m aws -p 100
```

5.4 Benchmark Design

In this section we discuss both the design of the *benchmark suite* (i.e., the chosen datasets and evaluation procedures [28]) and the experimental setup, as well as their limitations.

5.4.1 Benchmark Suites

To facilitate a reproducible experimental evaluation, we make use of OpenML Benchmark suites [28]. An OpenML benchmark suite is collection of OpenML tasks, which each reference a dataset, an evaluation procedure (e.g., k-fold CV) and its splits, the target feature, and the type of task (regression or classification). The benchmark suites are designed to reflect a wide range of realistic use-cases for which the AutoML tools are designed. Resource constraints are not part of the task definition. Instead, we define them separately in a local file so that each task can be evaluated with multiple resource constraints. Both the OpenML benchmark suite (and tasks) and the resource constraints are machine-readable to ensure automated and reproducible experiments.

Datasets

We created two benchmarking suites, one with 71 classification tasks, and one with 33 regression tasks. The datasets used in these tasks are selected from previous AutoML papers [238], competitions [109], and machine learning benchmarks [28], according to a predefined list of criteria as follows:

- **Difficulty** of the dataset has to be a sufficient. If a problem is easily solved by just about any algorithm, it will not be able to differentiate the various AutoML frameworks. This can mean that a simple models such as random forests, decision trees or logistic regression achieve a generalization error of zero, or that the performance of these models and all evaluated AutoML tools is identical.
- **Representative of real-world** data science problems to be solved with the tool. In particular we limit artificial problems. We included a small selection of such problems, either based on their widespread use (kr-vs-kp) or because they pose difficult problems. But we do not want them to be a large part of the benchmark. We also limit computer vision problems on raw pixel data because those problems are typically solved with dedicated deep learning solutions. However since they still make for real-world, interesting, and hard problems, we did not exclude them altogether.
- **No free form text features** that cannot reasonably be interpreted as a categorical feature. Most AutoML frameworks do not yet support feature engineering on text features and will process them as categorical features. For this reason we exclude text features even though we admit their prevalence in many interesting real-world problems. A first investigation and

benchmark of multimodal AutoML with text features has been carried out by Shi et al. [222].

- **Diversity** in the problem domains. We do not want the benchmark to skew towards any application domain in particular. There are various software quality problems in the OpenML-CC18 benchmark (jm1, kc1, kc2, pc1, pc3, pc4), but adopting them all would lead to a bias in the benchmark to this domain.
- **Independent and identically distributed** (i.i.d) data is required for each task. If the data is of temporal nature or repeated measurements have been conducted the task has been discarded. Both types of data are generally very interesting, but are currently not supported for most AutoML systems and we plan to extend the benchmark in the future in this direction.
- **Freely available** and hosted on OpenML. Datasets that can only be used on specific platforms like kaggle or not shared freely for any reasons are not included in the benchmark.
- **Miscellaneous** reasons to exclude a dataset included label-leakage, near-duplicates of other tasks in features (e.g., different only in categorical encoding or imputation) or target (e.g., binarization of a regression of multi-class task).

To study the differences between AutoML systems, the datasets vary in the number of samples and features by orders of magnitude, and vary in the occurrence of numeric features, categorical features and missing values. Figure 5.1 shows basic properties of the classification and regression tasks, including the distributions of the number of instances and features, the frequency of missing values and categorical features, and the number of target classes (for classification tasks). Other properties of the tasks are shown in Table A.2 and Table A.3 of Appendix A and can be explored interactively on OpenML.⁸ While the selection spans a wide range of data types and problem domains, we recognize that there is room for improvement. Restricting ourselves to open datasets without text features severely limits options, especially for big datasets.

All datasets are available in multiple formats for the AutoML frameworks, either as files (parquet, arff, or csv) or as Python object (pandas dataframe, numpy array). The used format depends on the framework, and in case a format

⁸Regression: www.openml.org/s/269, classification: www.openml.org/s/271

is used without column annotation (i.e., numpy arrays or csv) these annotations, i.e., type of column and levels, of may be provided to the framework separately.

Performance metrics

In our evaluation, we use area under the receiver operator curve (AUROC) for binary classification, log loss for multi-class classification and root mean-squared error (rmse) for regression⁹. We chose to use these metrics because they are generally reasonable, commonly used in practice and supported by most AutoML tools. The latter is especially important because it is imperative that AutoML systems optimize for the same metric they are evaluated on. However, our tool is not limited to these three metrics and a wide range of performance metrics can be specified by the user.

Missing Values

As will be discussed in more detail in Section 5.5.4, not all frameworks are equally well-behaved. There are times when search time budgets are exceeded or the AutoML frameworks crash outright, which results in missing performance estimates. There are multiple strategies to consider on how to deal with this missing data.

One naive approach may be to ignoring missing values, and aggregate over the obtained results. However, we see that failures do not occur at random. Failures are correlated to dataset properties, such as dataset size and class imbalance, which may be correlated with “problem difficulty” and thus performance. Ignoring missing values thus means that AutoML frameworks may fail on harder tasks or folds, and consequently obtain higher performance estimates. Imputing missing values with performance obtained by the same AutoML framework on other folds is subject to the same drawback. Moreover, both methods do not specify how to deal with missing values in case a framework fails to produce predictions on all folds of a task.

Instead, we propose to impute the missing values with an interpretable and reliable baseline. An argument may be made for using the random forest baseline, since this may be a strong fallback that AutoML frameworks could realistically implement. However, we observe that training a random forest (of the size used in the baseline) requires a non-significant amount of time on some datasets. Automatically providing this fallback by means of imputation would provide an unfair advantage to the AutoML frameworks which are well-behaved. Moreover,

⁹We use the implementations provided by `scikit-learn` 0.24.2

many failures would not be remedied by having a random forest to fall back on, since the AutoML frameworks crash irrecoverably due to e.g., segmentation faults.

Instead, we impute missing values with the constant predictor, or prior. This baseline returns the empirical class distribution for classification, and the empirical mean for regression. This is a very penalizing imputation strategy, as the constant predictor is often much worse than results obtained by the AutoML frameworks which produce predictions for the task or fold. However, we feel this penalization for ill-behaved systems is appropriate and fairer towards the well-behaved frameworks, and hope that it encourages a standard of robust, well-behaved AutoML frameworks.

5.4.2 Experimental Setup

Hardware

For comparable hardware and easy expandability we opt to conduct the benchmark on standard `m5.2xlarge`¹⁰ instances available on Amazon Web Services (AWS). These represent current commodity level hardware with 32 GB memory, 8 vCPUs (Intel Xeon Platinum 8000 series Skylake-SP processor with a sustained all core Turbo CPU clock speed of up to 3.1 GHz). 100 GB of gp3-SSD storage is available for storage, which can be necessary for storing a larger number of evaluated pipelines. The use of AWS also enables others to fully reproduce and extend our results, since the results do not depend on private computing infrastructure. As discussed in Section 5.3.1, the benchmark is not limited to AWS but can be run on any machine.

Framework Configuration

All AutoML frameworks are instantiated with their default configuration, with the following exceptions:

- **Runtime** for the search with one hour leeway for data loading, making predictions, and cleanup operations.
- **Resource constraints** which specify the number of CPU cores and amount of memory available.

¹⁰<https://aws.amazon.com/ec2/instance-types/m5/>

- **Target metric** to use for optimization. This is the same metric that is used for evaluation in the benchmark.
- **‘mode’** to declare the user intent. E.g., getting the best possible model versus finding an interpretable (less complex) model. The mode used to evaluate each AutoML framework is chosen by their developers.
- **‘output directory’** where any artifacts of the AutoML framework may be stored.

The benchmark intentionally does not allow further customization of other AutoML system configuration parameters to reflect how these systems are usually applied in practice as closely as possible.

5.4.3 Limitations

Both the design of the benchmark and the setup for the experiments described in this paper have some limitations when it comes to the interpretation of their results. Limitations in the design stem from the desire to keep the use of the tools as close as possible to the original vision and usage intended by developers, whereas the limitations in the experiments are caused by resource constraints and may be alleviated by running additional experiments with the benchmark software. In this section we highlight some important limitations, and stress that this paper and the results within do *not* state which AutoML tool ultimately is the best.

Limitations of the Design

Perhaps the biggest limitation of the design is the inability to attribute the performance of an AutoML tool to any one aspect of its build as it is often done with ablation studies. The evaluated AutoML tools differ among multiple design choices, such as underlying ML library, search space, preprocessing and search algorithm. Concretely, a performance difference between e.g., **auto-sklearn** and **TPOT** could be caused by **TPOT**’s built-in stacking, **auto-sklearn**’s ensembles, the difference in Bayesian Optimization versus genetic programming, the difference in how multiprocessing is employed, or a combination of these or any other difference between them. Software that would allow for such conclusions essentially requires each AutoML tool to be reimplemented on a shared set of algorithms for building models, search and evaluation. We acknowledge that this would be incredibly valuable for the research community, however it would

also no longer resemble the software as used in practice and thus be different work altogether. Note that it *is* possible to perform ablation studies with the benchmark tool for a specific AutoML framework, for example by comparing different framework configurations as done by Erickson et al. [73].

Another limitation stems from only recording results produced by the final model. Anytime performance, where information about the performance during optimization is captured as if they would be final models, can be very insightful. It allows for the distinction between a tool which converges quickly from one that does not, and may be especially important for users which are interested to use the systems with a human-in-the-loop, e.g., when designing a search space or data features. Unfortunately many tools do not support collection of anytime performance, and depending on how they are recorded might interfere with resources used during search. We hope to be able to record anytime performance in the future, but in this work we only approximate it by evaluating the tools under two different time constraints (1 and 4 hours).

Finally, the qualitative comparison of the tools is also limited. Certain quality of life features like analysis of the pipeline via interpretable machine learning (IML) methods, reports, usability or support are not evaluated, but are important to many users. For a qualitative analysis of those we refer the reader to one of the many existing overview papers on AutoML [244, 285].

Limitations of the Experiments

Most tools are highly configurable and allow the user to configure the search algorithm or its hyperparameters, among other aspects that affect the AutoML performance. Some tools even provide different configuration presets for different use cases, e.g., a performance-oriented competition mode, which we use, and a mode that produces fast or interpretable models at the cost of some performance. However, comparing the effect on model performance of tuning hyperparameters, or using different presets, quickly carries prohibitive costs. For this reason we have to limit our experiments to only use one mode specified by the frameworks’ developers. The mode selected for each tool was the most performance-oriented setting. It is likely that better results may be achieved by carefully meta-tuning the AutoML tool, or that the tool with the best performance in *competition* mode has relatively poor performance in *interpretable* mode. While it is cost prohibitive for us to evaluate many different scenarios, it is easy to run the benchmark with custom configurations for the various AutoML tools. This allows users to evaluate AutoML systems in a setting that reflects their interest.

Meta-learning

Many AutoML tools make use of meta-learning to better initialize and speed up the search. Since all data in the benchmark is publicly available and many of them are well known in the AutoML community, it is likely that there is a substantial overlap between data used by the developers for meta-learning and the data used in the benchmark. This is a very intricate problem as we consider AutoML tools as black boxes. Removing the effect of the dataset that is to be evaluated from the meta-learning procedure is not solvable in general.

In this paper specifically, both **auto-sklearn 1** and **auto-sklearn 2** use meta-learning. **Auto-sklearn 1**'s meta-learning uses 140 datasets from OpenML, each associated with well working ML pipelines. The search is initialized with a 25-nearest-dataset (KND) lookup using 38 meta-features [203]. **Auto-sklearn 1** can exclude datasets by name from the lookup¹¹ which we make use of in the benchmark. Even so, it cannot be guaranteed that identical datasets with a different name might be used for meta learning. A different approach would be to exclude data with a distance of zero (or extremely small value) in the KND initialization.

Auto-sklearn 2's meta-learning model is more complicated, it consists of: a) A static pipeline portfolio for warm-starting the search, which is computed across hundreds of datasets using a greedy forward selection. And b) A meta-model to predict the internal model selection strategy and budget allocation strategy. Single datasets cannot be excluded from these meta-learning procedures and it is not feasible retrain the meta-models and pipeline portfolio for each dataset in our benchmark. This ultimately means that the result of **auto-sklearn 1** and especially **auto-sklearn 2** needs to be considered very carefully. More research is required to address these issues and allow for the correct evaluation of AutoML systems that use meta-learning.

5.4.4 Overfitting the Benchmark

One last issue that plagues any widely adopted benchmark is the potential of algorithms overfitting on the datasets used in the benchmark. Since freely available, interesting and usable (c.f. Section 5.4.1 on our selection criteria) datasets are scarce, many AutoML developers use these datasets to benchmark and improve their systems iteratively. While this is not as direct of an issue as with meta-learning, these datasets can in general not be assumed to be truly unseen. The only practical way to avoid this is to collect a novel set of datasets

¹¹<https://automl.github.io/auto-sklearn/master/faq.html#meta-learning>

for the benchmark which would entail a prohibitive effort. Even more, after publishing this benchmark, the new datasets are published which again gives developers the possibility to use them to improve their systems. On the other hand, should you keep the benchmarking datasets private to avoid this issue, the benchmark is no longer entirely reproducible by independent researchers. We hope that the size of our benchmarking suites is large enough and their design general enough that overfitting is less of an issue, but this is hard to guarantee.

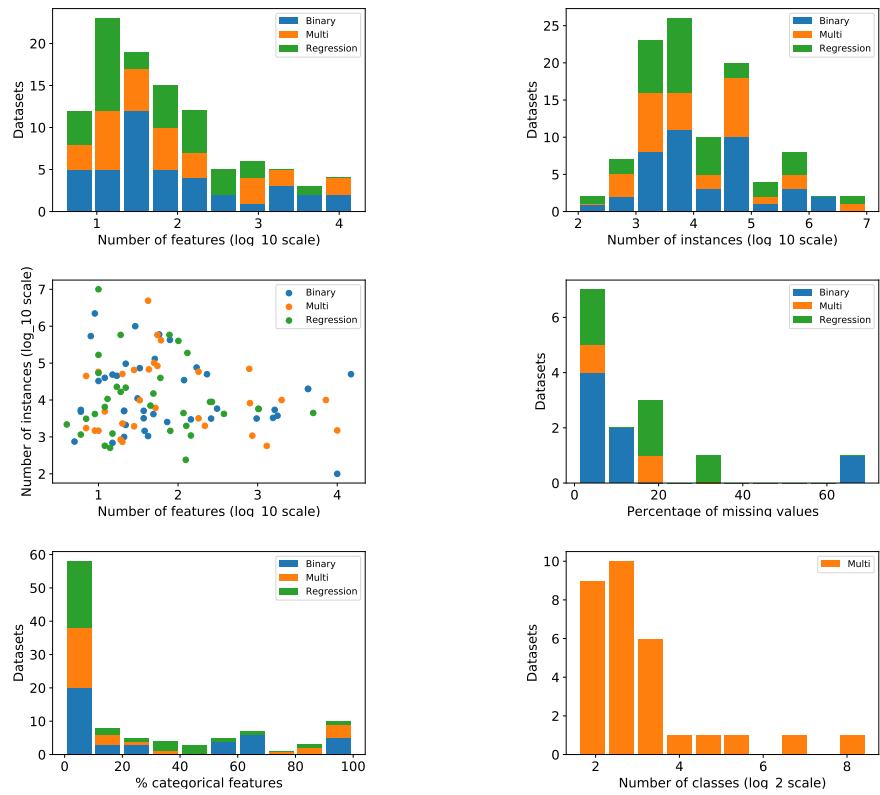


Figure 5.1: Properties of the tasks in both benchmarking suites.

5.5 Results

In this section, we provide an overview and analysis of the results obtained. This section is accompanied by an interactive visualization tool¹² and all data artifacts generated from these experiments¹³. We strongly encourage the reader to explore the data with the interactive visualization tool for a more comprehensive comparison than we can provide here.

5.5.1 Performance

To report on the results for many AutoML frameworks across whole benchmarking suites, we propose to use critical difference (CD) diagrams [65]. For each framework they show their average rank, as well as which ranks are statistically significantly different from each other. To calculate the average rank per task, we first impute any missing values with the constant predictor and then average the performance over all folds. We may then test for the presence of statistically significant differences in the average rank distributions using a non-parametric Friedman test at $p < 0.05$ (here, $p \approx 0$ for every diagram) and use a Nemenyi post-hoc test to find which pairs differ. For each benchmarking suite and time budget, the critical difference diagrams are shown in Figure 5.2. They display the rank of each framework (lower is better) averaged over all results from the given benchmarking suite and budget.

Overall, we observe that **AutoGluon** and **TPOT** respectively achieve the best and worst rank among AutoML frameworks in each setting with respect to model accuracy, though never by a statistically significant margin. In almost all cases, the baselines obtain lower ranks than any AutoML framework, though the tuned random forest is a strong baseline that is often not significantly worse than many of the AutoML frameworks. All AutoML frameworks except **AutoGluon** and **TPOT** are generally ranked close to each other, with small differences in order for the various suites and budgets.

To complement the CD diagrams, which obfuscate the relative performance differences, we show box plots of obtained results (after imputation) across all tasks in Figure 5.3. Because the performances are not commensurable across tasks, we first scale the all results per task between the random forest performance (-1) and the best observed performance (0) which means that higher scores are better. This also makes the scaled value interpretable, and scales

¹²<https://compstat-lmu.shinyapps.io/AutoML-Benchmark-Analysis/>

¹³<https://openml-test.win.tue.nl/amlb/>

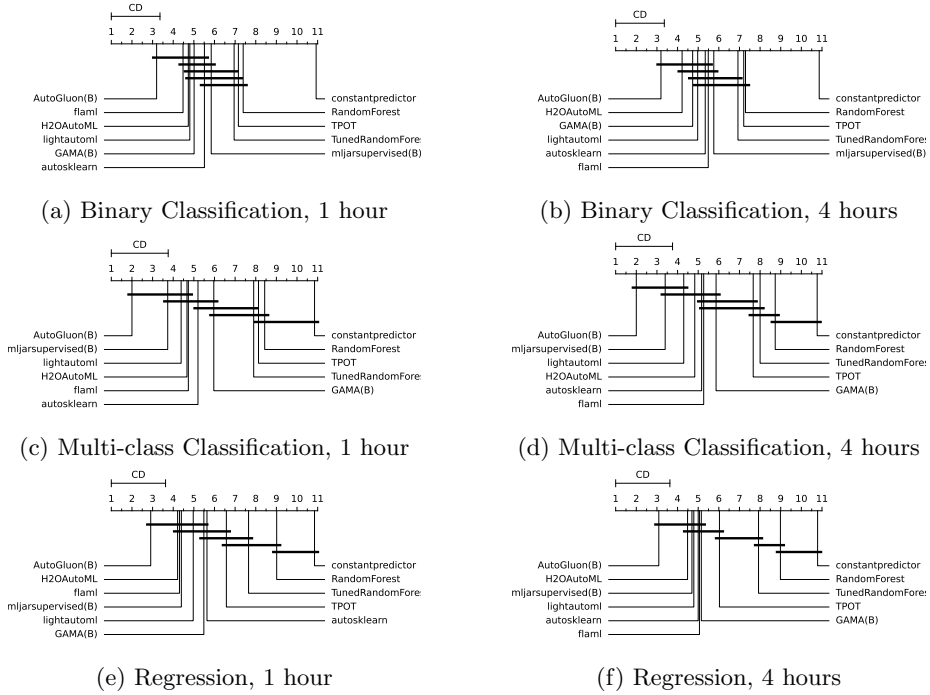


Figure 5.2: CD plots with Nemenyi post-hoc test after imputing missing values with the constant predictor baseline.

based on the improvement over the baseline that is observed to be achievable. While the boxplots are calculated over performance data on all tasks, the plots are cut off to allow a better visualization of the most relevant area. The number of outliers for each framework that are not shown in the plot are denoted on the x-axis.

Even if ranks are similar, the performance distribution might be noticeably different. For example, **GAMA**, **H2O AutoML**, and **Light AutoML** achieve very similar average ranks on the one hour binary classification tasks, but from the boxplots we observe that while **H2O AutoML** achieves lower median normalized performance in this segment, its worst observed performances are much better than that of **GAMA** and **Light AutoML**. Similarly, while **TPOT**'s average rank is generally close to that of the Tuned Random Forest baseline, **TPOT** exhibits much higher variance in its prediction quality.

5.5.2 BT-Trees

Bradley-Terry (BT) trees [234] can be used to statistically analyse benchmark experiments based on dataset characteristics [78]. These trees use dataset characteristics, such as the number of instances, the number of features, the ratio of missing values and others, to split paired performance comparisons of the framework to find statistically significant differences in performance. Bradley-Terry models are originated in psychology to analyze paired comparison experiments of subjects preferring one stimulus over another. For our benchmark, such a preference ranking can be easily derived by pairwise performance comparisons of all frameworks regarding the datasets and cross-validation folds.

The underlying algorithm of model-based recursive partitioning of Bradley-Terry models works as follows: in each split of the BT tree, a BT model is fitted for the paired comparisons based on the underlying dataset characteristics. Following Zeileis and Hornik [282] and Eugster, Leisch, and Strobl [78] the BT model performs a statistical test of parameter instability for the chosen data characteristics. If this test reveals a significant instability in the model parameters, the corresponding tree node splits the data according to the characteristic yielding the highest instability (lowest test p-value). The splitting cut-point is then determined such that it has the highest improvement of the model fit. This procedure is repeated until either no significant instability is left, a set tree depth is reached, or further splits would exceed a set minimum number of observations in the leaves.

Numeric values in the tree leafs are worth parameters which can be interpreted as preferences for the different frameworks [78]. Since they are in $[0, 1]$

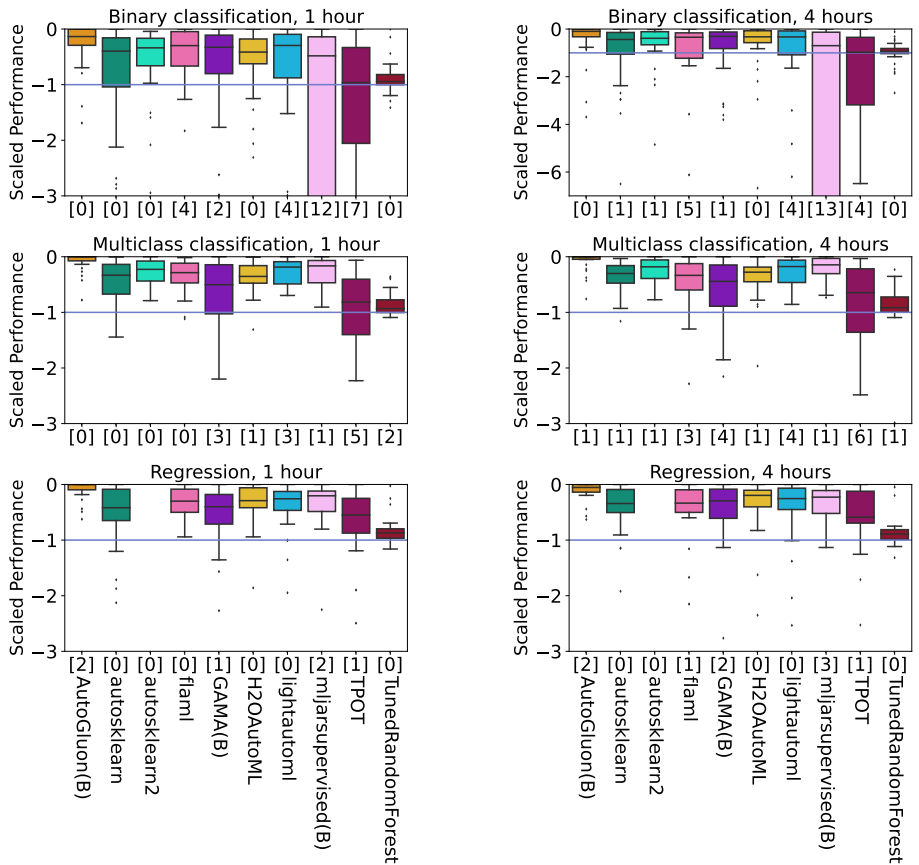


Figure 5.3: Boxplots of framework performance across tasks after scaling the performance values from random forest (-1) to best observed (0).

and sum up to 1 within a leaf, they can be understood as probability of a framework performing best given the data characteristics in the corresponding leaf.

Figure 5.4 shows a Bradley-Terry tree for classification tasks for a runtime of one hour. For simplicity reasons, in order to obtain an easy understanding tree, only the number of instances/features and the imbalance ratio was chosen as data characteristics. The first split distinguishes between datasets with more than 5832 instances and the ones equal or below that cut-point. Following the left child node, the imbalance ratio of 1.034 was chosen to define the two left tree leafs. The left one (Node 3) - small and very balanced classification datasets - indicate that in such situations GAMA is preferred over all other frameworks. Even though AutoGluon is less preferred for those kind of datasets than GAMA, it is still preferable to all other frameworks. On small, more im-

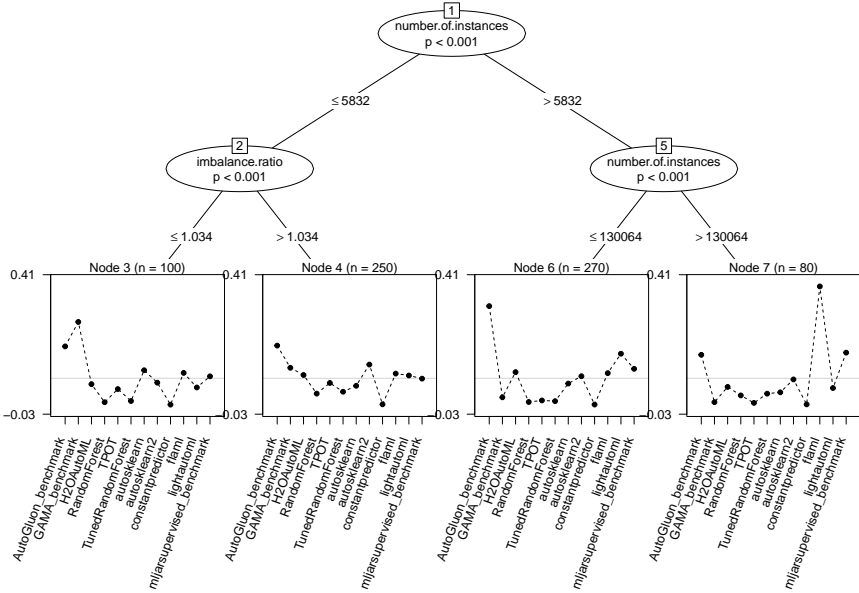


Figure 5.4: Bradley-Terry tree of depth three for classification tasks. Results from the one hour classification benchmark were used, and missing values were imputed by constant predictor performance. One observation within the BT tree equals the preference ranking of one fold on one dataset.

balanced datasets (Node 4), **Autogluon** is preferable to all other frameworks, followed by **autosklearn 2**.

The right half of the tree is again divided into medium and large datasets at a splitting value of 130064 observations. While on the left leaf (Node 6) **AutoGluon** is clearly the preferred framework, the same applies on large classification datasets to **FLAML** (Node 7), followed by **mljar supervised** and **AutoGluon**.

Figure B.1 and B.2 in the appendix show simpler Bradley-Terry trees with only the number of instances and features as dataset characteristics. The findings from the BT trees are essentially the same as those from Section 5.5.1, i.e., that overall **AutoGluon** is the preferred framework in most tree leafs. Moreover, the reader is strongly invited to explore the aforementioned interactive visualization tool with which deeper BT trees based on several more dataset characteristics can be constructed on various task types.

5.5.3 Model Accuracy vs. Inference Time Trade-offs

In terms of performance metrics, the development version of the AutoML Benchmark only measured model accuracy metrics (e.g., AUC, logloss). Some of the integrated frameworks offer a “compete” mode (e.g., **AutoGluon**, **MLJar**) which maximizes accuracy, typically at the cost of increased model complexity, similar to how you may compete in a Kaggle competition. This can lead to models being built that are highly accurate but are extremely slow at inference time and are therefore not practical in many real life use-cases. In order to evaluate the limitations of the models produced by each framework, we also measured “prediction duration,” or how long it took to produce predictions for the test set for each dataset in the benchmark. This metric provides important insight into the trade-offs that tool authors make in their algorithm designs.

Figure 5.5 shows aggregated inference times across all models, including total time to score the test set (predict duration) as well as the per-row prediction speed (predict duration divided by the number of rows in each test set). Outliers have been removed from both plots for visibility, as there are a handful of very extreme outliers. Both **AutoGluon** and **MLJar** stand out as orders of magnitude slower than the other AutoML tools, on average. **Light AutoML** is also slower than the remaining tools. **GAMA** is approximately as fast as **autosklearn** and **autosklearn 2**, which can be explained by the fact that they both build optimize **scikit-learn** pipelines and create an ensemble using the same algorithm [48, 49]. **H2O AutoML**, **FLAML** and **TPOT** stand out as having very fast inference times, however **TPOT** is much less accurate than the other tools, as we will see in more detail below.

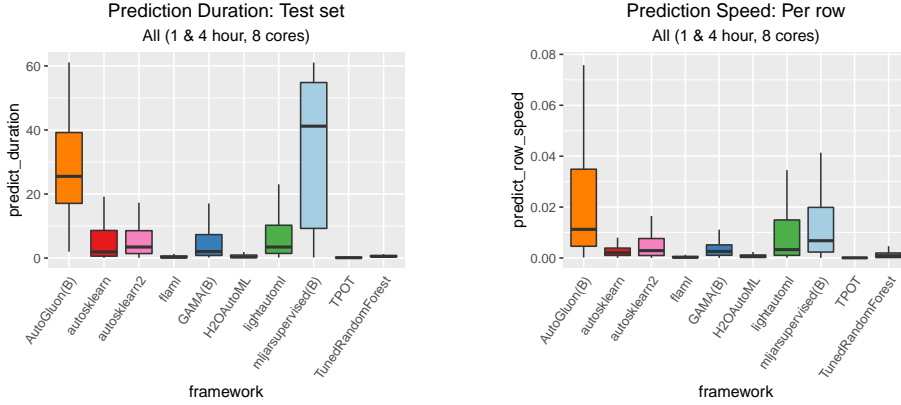


Figure 5.5: Prediction duration (on the test set) in seconds and prediction speed (per row) in seconds as divided by the total number of rows in the test set, aggregated across all runs (tasks, problem types and constraints).

In Figure 5.6 we show the Pareto Frontier for all six scenarios, demonstrating the average normalized model accuracy against corresponding average per-row prediction speeds. We can more clearly see that the frameworks that are getting the highest accuracy are doing so at the cost of inference time performance. This demonstrates that when contextualizing any type of model accuracy results, it is important to consider any trade-offs that may have been made to achieve the extra performance and how that will affect the framework’s usability in practice. Measuring accuracy in isolation does not give the complete picture of the overall utility of a particular framework.

5.5.4 Observed AutoML Failures

While most jobs completed successfully, we observed multiple framework errors during our experiments. In this section, we will discuss where AutoML frameworks fail, though we want to stress that development for these packages is ongoing. For that reason, it is likely that the same frameworks will not experience the same failures in the future (especially after gaining access to all experiment logs). We categorize the errors into the following categories:

Memory: The framework crashed due to exceeding available memory.

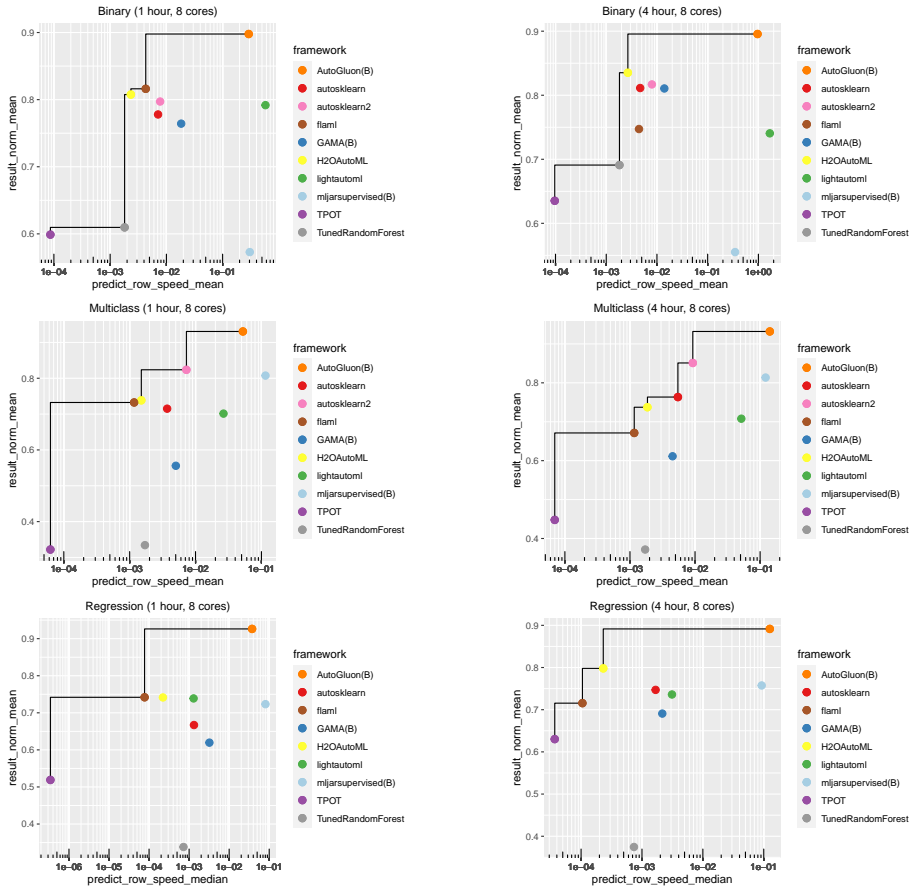


Figure 5.6: Pareto Frontiers of framework performance across tasks after scaling the performance values from the worst framework (0) to best observed (1).

Time: The framework exceeded the time limit past the leniency period.

Data: When errors are due to specific data characteristics (e.g., imbalanced data).

Implementation: Any errors caused by bugs in the AutoML framework code.

These categories are a bit crude and ultimately subjective, e.g., in the extreme case all errors are implementation errors. However, they serve for a quick overview and a more detailed overview can be found in Appendix B.2.

Figure 5.7 shows the errors by type on the left, and by task on the right. Overall, memory and time constraints are the main cause for errors with one major exception¹⁴. We observe that errors are far more common in the classification benchmark suite than the regression suite. This is largely accounted for by the difference in benchmarking suite size (33 and 71 tasks) and the fact that the largest datasets are mostly classification, both in number of instances and features. Unique to classification, we do observe several frameworks failing to produce models or predictions on highly imbalanced datasets (e.g., ‘yeast’). This is also the case for the failures on the two small classification datasets, where internal validation splits no longer contain all classes. Interestingly, AutoML frameworks fail more frequently on a larger time budget. Both memory and time constraint violations happen more frequently, which may potentially be explained by frameworks saving increasingly more models or building increasingly larger pipelines.

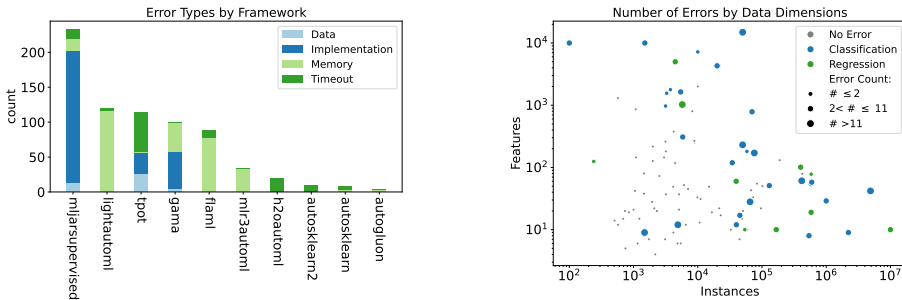


Figure 5.7: For each framework, errors by type are shown on the left, and errors by task are shown on the right.

¹⁴MLJarSupervised has 190 ‘implementation errors’ which are caused by only two distinct index errors.

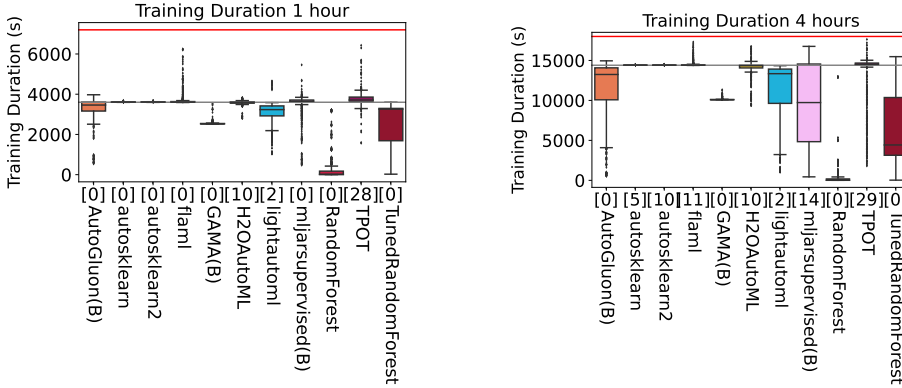


Figure 5.8: Time spent during search with a one hour budget (left) and four hour budget (right). The grey line indicates the specified time limit, and the red line denotes the end of the leniency period. The number of timeout errors for each framework are shown beside it.

Only when the framework exceeds the time budget by more than one hour do we record a time error. However, as we can see in Figure 5.8, not all AutoML frameworks adhere to the runtime constraints equally well, even if they finish within the leniency period. In the figure, the training duration for each job (task and fold combination) are aggregated and timeout errors are shown above each framework, missing values due to non-time errors are not included. These plots reveal design decisions around the specified runtime, with some frameworks never exceeding the limit by more than a few minutes, while others violate it by a larger margin with some regularity. Interestingly, we see that several frameworks consistently tend to stop far before the specified runtime limit.

5.6 Conclusion and Future Work

The benchmark tool makes producing rigorous reproducible research both easier and faster. We conducted a thorough comparison of 9 AutoML frameworks across 71 classification tasks and 33 regression tasks. A statistical analysis reveals that their *average* rank is generally not statistically significantly different. Overall, **AutoGluon** consistently has the highest average rank, in terms of model performance, in our benchmark. Also, in most scenarios, the AutoML frame-

works outperform even our strongest baseline.

Since inference time is an important factor in real-world applications, we also analyzed the trade-off between model accuracy and inference time. We found large difference in inference time, in some cases the difference spans orders of magnitude. Overall, better models also had slower inference time, but not all AutoML frameworks provide solutions that are Pareto optimal.

In the future we would like to extend the benchmark to support new problem types, such as multi-objective optimization, semi-supervised learning or non-i.i.d. settings (such as when temporal relationships are present in the data). We also want to continue to update the benchmark with current and real-world tasks so that it stays reflective of modern challenges, and in the process hopefully reduce the ability for AutoML frameworks to overfit to the benchmark. Even so, we would like to investigate whether AutoML frameworks start to overfit to the benchmark, as it may be used in framework development. This might be possible by, for example, benchmarking both the current and future version of the AutoML frameworks on new, unseen tasks, and comparing the relative improvements on both the benchmark tasks and the new tasks.

Contributing new datasets or integrating a new framework is possible through the open source and extensible design of the benchmark. We hope this motivates researchers to contribute their own dataset, framework integration or feedback to the open source AutoML benchmark so that it may be useful to the community for a long time to come.

Chapter 6

Meta-Learning for Symbolic Hyperparameter Defaults

As we have seen in Chapters 1 and 2, the performance of most machine learning algorithms is greatly influenced by their hyperparameter configuration [146] and various methods exist to automatically optimize them for a specific dataset [27]. This motivates that the optimal values of a hyperparameter are functionally dependent on properties of the data.

While various methods exist to automatically optimize hyperparameters, the additional complexity and effort cause many practitioners to forgo optimization. Hyperparameter defaults provide a fallback but are often static and do not take properties of the dataset into account. If we could learn the functional relationship between hyperparameter configurations and the data, we could express them as *symbolic default* configurations that work well across many datasets. These symbolic defaults would not only directly benefit users of the algorithms, but could also be used as a stronger baseline for further tuning in AutoML, or even to inform transformations of the search space.

Well-known examples for such symbolic defaults are already widely used: The random forest algorithm’s default $mtry = \sqrt{p}$ for the number of features

This chapter is derived from: Pieter Gijsbers et al. *Meta-Learning for Symbolic Hyperparameter Defaults*. 2021. arXiv: 2106.05767 [stat.ML]

and its short-form publication: Pieter Gijsbers et al. “Meta-learning for symbolic hyperparameter defaults”. In: *Proceedings of the Genetic and Evolutionary Computation Conference Companion* (July 2021). DOI: 10.1145/3449726.3459532. URL: <http://dx.doi.org/10.1145/3449726.3459532>

sampled in each split [38], the median distance between data points for the width¹ of the Gaussian kernel of an SVM [46], and many more. Unfortunately, it has not been studied how such formulas can be obtained in a principled, empirical manner, especially when multiple hyperparameters interact, and have to be considered simultaneously.

This chapter addresses a new meta-learning challenge: “Can we *learn* a vector of *symbolic* configurations for multiple hyperparameters of state-of-the-art machine learning algorithms?”. We propose an approach to learn such symbolic default configurations by optimizing over a grammar of potential expressions, in a manner similar to symbolic regression [140] using Evolutionary Algorithms. The proposed approach is general and can be used for any algorithm as long as their performance is empirically measurable on instances in a similar manner.

The rest of the chapter is structured as follows. We first give a motivating example in Section 6.1, after which we introduce relevant related work in Section 6.2 and define the resulting optimization problem in Section 6.3. In Section 6.4 we continue with describing the proposed method and we study the efficacy of our approach in a broad set of experiments across multiple machine learning algorithms in Sections 6.5 & 6.6.²

6.1 A Motivating Example

We motivate the intuitive idea that the optimal hyperparameter configurations depend on properties of the data with an example in Figure 6.1. The figure shows averaged response surfaces across 106 tasks for hyperparameters γ and *cost* (zoomed in to a relevant area of good performance). While the scale for the cost parameter is kept fixed in Figures 6.1(a) and 6.1(b), the x-axis displays the unchanged, direct scale for γ in (a), and multiples of $\frac{mkd}{xvar}$ in (b).³ This formula was found using the procedure that will be detailed in this chapter. The maximum performance across the grid in (a) is 0.859, while in (b) it is 0.904.

Empirically, we can observe several things. First, on average, a grid search over the *scaled* domain in (b) yields better models. Secondly, the average solu-

¹Or the inverse median for the inverse kernel width γ

²This work was carried out before/concurrent to the other work presented in this thesis. For this reason, and additional considerations described in the final section of this chapter, empirically evaluating the usefulness of symbolic hyperparameter defaults for AutoML remains future work.

³Values for $\frac{mkd}{xvar}$ range between $4.8 \cdot 10^{-5}$ and 0.55, this formula was found using the procedure that will be detailed in this chapter. Symbols are described in Table 6.2.

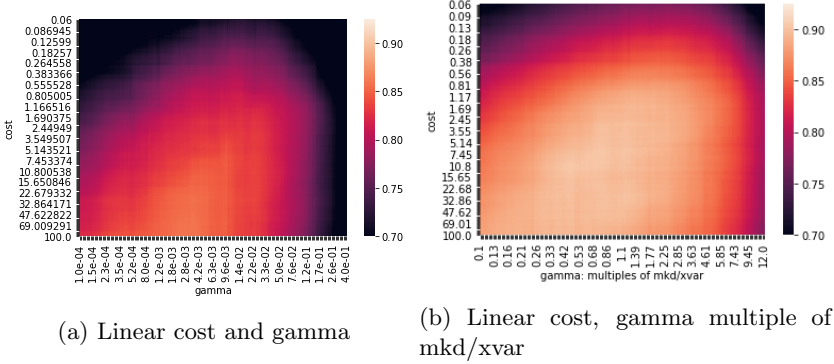


Figure 6.1: Performance of an RBF-SVM averaged across 106 datasets for different values of cost and gamma, unscaled (a) and with gamma as multiples of meta-features (b)

tion quality and the area where good solutions can be obtained is larger, and the response surface is therefore likely also more amenable towards other types of optimization. And thirdly, we can conjecture that introducing formulas e.g. $\gamma = \frac{mkd}{xvar}$ for each hyperparameter can lead to better defaults. Indeed, finding good defaults in our proposed methodology essentially corresponds to optimization on an algorithm’s response surface (averaged across several datasets). It should be noted that the manually defined heuristic used in `sklearn` [184], i.e. $\gamma = \frac{1}{p \cdot xvar}$, is strikingly similar.

6.2 Related Work

Symbolic defaults express a functional relationship between an algorithm hyperparameter value and dataset properties. Some example for such relationships are reported in literature, such as the previously mentioned formulas for the random forest [37] or the SVM [46]. Some of these are also implemented in ML workbenches such as `sklearn` [184], `WEKA` [111] or `mlr` [145]. It is often not clear and rarely reported how such relationships were discovered, nor does there seem to be a clear consensus between workbenches on which symbolic defaults to implement. Also, they are typically limited to a single hyperparameter, and do not take into account how multiple hyperparameters may interact.

Meta-learning approaches have been proposed to learn *static* (sets of) de-

faults for machine learning algorithms [160, 188, 196, 267, 269, 271] or neural network optimizers [162], to analyze which hyperparameters are most important to optimize [196, 255, 267], or to build meta-models to select the kernel or kernel width in SVMs [225, 233, 250].

An underlying assumption is that hyperparameter response surfaces across datasets behave similarly, and therefore settings that work well on some datasets also generalize to new datasets. Research conducted by warm-starting optimization procedures with runs from other datasets (cf. [154], [86]) suggest that this is the case for many datasets.

Previous work [209] on *symbolic* defaults proposed a simplistic approach towards obtaining those, concretely by doing an exhaustive search over a space of simple formulas composed of an *operator*, a *numeric value* and a single *meta-feature*. This significantly constricts the variety of formulas that can be obtained and might therefore not lead to widely applicable solutions.

6.3 Problem Definition

6.3.1 Supervised Learning and Risk of a Configuration

Consider a target variable y , a feature vector \mathbf{x} , and an unknown joint distribution \mathcal{P} on (\mathbf{x}, y) , from which we have sampled a dataset \mathcal{D} containing N observations. An ML model $\hat{f}(\mathbf{x})$ should approximate the functional relationship between \mathbf{x} and y . An ML algorithm $\mathcal{A}_{\lambda}(\mathcal{D})$ now turns the dataset (of size N) into a prediction model $\hat{f}(\mathbf{x})$. \mathcal{A}_{λ} is controlled by a multi-dimensional hyperparameter configuration $\lambda \in \Lambda$ of length M : $\lambda = \{\lambda_1, \dots, \lambda_M\}$, where $\Lambda = \Lambda_1 \times \dots \times \Lambda_M$ is a cross-product of (usually bounded) domains for all individual hyperparameters, so Λ_j is usually a bounded real or integer interval, or a finite set of categorical values. In order to measure prediction performance pointwise between a true label and a prediction, we define a loss function $L(y, \hat{y})$. We are interested in estimating the expected risk of the inducing algorithm w.r.t. λ on new data, also sampled from \mathcal{P} :

$$R_{\mathcal{P}}(\lambda) = E_{\mathcal{P}}(L(y, \mathcal{A}_{\lambda}(\mathcal{D})(\mathbf{x}))),$$

where the expectation above is taken over all datasets \mathcal{D} of size N from \mathcal{P} and the test observation (\mathbf{x}, y) . Thus, $R_{\mathcal{P}}(\lambda)$ quantifies the expected predictive performance associated with a hyperparameter configuration λ for a given data distribution, learning algorithm and performance measure.

6.3.2 Learning an Optimal Configuration

From a good *default* configuration λ we now expect that it performs well according to many of such risk mappings for many different data scenarios. Given K different datasets (or data distributions) $\mathcal{P}_1, \dots, \mathcal{P}_K$, we define K hyperparameter risk mappings:

$$R_k(\lambda) = E_{\mathcal{P}_k}(L(y, \mathcal{A}_\lambda(\mathcal{D})(x))), \quad k = 1, \dots, K.$$

We now define the average risk of λ over K data distributions:

$$R(\lambda) = \frac{1}{K} \sum_{k=1}^K R_k(\lambda).$$

Minimizing the above w.r.t λ over Λ defines an optimization problem for obtaining an optimal static configuration from K scenarios, where we assume that, given a large enough K , a configuration will also work well on new data situations \mathcal{P} .

6.3.3 Learning a Symbolic Configuration

We now allow our configurations to be symbolic, i.e., contain formulas instead of static values. Hence, we assume that $\lambda(\cdot)$ is no longer a static vector from Λ , but a function that maps a dataset, or its data characteristics, to an M -dimensional configuration.

$$\lambda = (\lambda_1, \dots, \lambda_M) : \mathcal{D} \rightarrow \Lambda$$

For this reason, we define a context-free grammar of transformations, which define the space of potential expressions for all component functions $\lambda_j(\cdot)$. This grammar consists of constant values, symbolic dataset meta-features and simple mathematical operators, detailed in Table 6.1.

Given a meta-training set of K data scenarios $\mathcal{D}_1, \dots, \mathcal{D}_K$, we include the computation of the configuration by $\lambda(\mathcal{D})$ as a first step into the algorithm $\mathcal{A}_\lambda(\mathcal{D})$ and change our risk definition to:

$$R_k(\lambda) = E_{\mathcal{P}_k}(L(y, \mathcal{A}_{\lambda(\mathcal{D})}(\mathcal{D})(x))), \quad k = 1, \dots, K.$$

and again average to obtain a global objective for $\lambda(\cdot)$:

$$R(\lambda) = \frac{1}{K} \sum_{k=1}^K R_k(\lambda),$$

where the optimization now runs over the space of all potential M-dimensional formulas induced by our grammar.

6.3.4 Metadata and Surrogates

In principle, it is possible to estimate $R_k(\boldsymbol{\lambda})$ empirically using cross-validation during the optimization. However, this is obviously costly, as we want to obtain results for a large number of configurations across many multiple datasets. Therefore, we propose to employ *surrogate models* that approximate $R_k(\boldsymbol{\lambda})$. We generate one surrogate for each dataset, ML algorithm and performance metric combination on a sufficiently large number of cross-validations experiments, with randomly planned design points for $\boldsymbol{\lambda}$. Such meta-data evaluations are often used in literature ([255, 270, 271]), and can for example be obtained from [143, 255] or [270]. This induces empirical surrogates, that map from static configurations to predicted performance values:

$$\hat{R}(\boldsymbol{\lambda}) : \Lambda \rightarrow \mathbb{R}$$

As our algorithm $\mathcal{A}_{\boldsymbol{\lambda}}(\mathcal{D})$ is now removed, we simply change our objective to a simplified version, too:

$$\hat{R}(\boldsymbol{\lambda}(.)) = \frac{1}{K} \sum_{k=1}^K R_k(\boldsymbol{\lambda}(\mathcal{D}_k))$$

This defines a global, average risk objective for arbitrary formulaic $\boldsymbol{\lambda}(\cdot)$ expressions that can be efficiently evaluated.

Considering the fact that performances on different datasets are usually not commensurable [65], an appropriate scaling is required before training surrogate models to enable a comparison between datasets. This is done in literature by resorting to ranking [12], or scaling [281] to standard deviations from the mean. We mitigate the problem of lacking commensurability between datasets by scaling performance results to $[0; 1]$ on a per-dataset basis as done in [162, 188]. After scaling, 1 corresponds to the best performance observed in the meta-data and 0 to the worst. A drawback to this is that some information regarding the absolute performance of the algorithm and the spread across different configurations is lost.

Dataset Characteristics In addition to the performance of random hyperparameter-configurations, OpenML [258] contains a range of dataset characteristics, i.e.,

meta-features. A full list of available characteristics is described by [207]. In order to obtain simple, concise and efficient formulas, we opted to include only simple dataset characteristics instead of working an extensive set as described by [207]. Table 6.2 contains an overview over used meta-features and their corresponding ranges over our meta training set described in Section 6.5. Meta-features are computed for each dataset after imputation, one-hot encoding of categoricals and scaling of numeric features. We include (among many others) the number of observations, the number of features and information regarding class balance. We denote the set of characteristics $\{c_1, c_2, \dots, c_L\}$ with C . For this, we can also reuse evaluations shared on OpenML.

Evaluation meta-data To learn symbolic defaults, we first gather meta-data that evaluates $R_k(\boldsymbol{\lambda})$ on all K datasets. For a given fixed algorithm with hyperparameter space Λ and performance measure, e.g., logistic loss, a large number of experiments of randomly sampled $\boldsymbol{\lambda}$ is run on datasets P_1, \dots, P_K , estimating the generalization error of $\boldsymbol{\lambda}$ via 10-fold Cross-Validation.

Symbol definition	Description
$\langle \text{configuration} \rangle ::= [\langle F \rangle \text{---} \langle I \rangle]^* N$	N: Number of hyperparameters Type depends on hyperparameter
$\langle I \rangle ::=$	
$\langle \text{unary} \rangle \langle F \rangle$	unary function
$\text{---} \langle \text{binary} \rangle 2^* \langle F \rangle$	binary function
$\text{---} \langle \text{quaternary} \rangle 4^* \langle F \rangle$	quaternary function
$\text{---} \langle i \rangle$	integer constant or symbol
$\langle F \rangle ::=$	
$\langle I \rangle$	
$\text{---} \langle f \rangle$	float constant or symbol
$\langle i \rangle ::=$	
$\langle \text{mfi} \rangle$	Integer meta-feature, see Table 6.2
$\text{---} c_i$	$[x]; x \sim \text{loguniform}(2^0, 2^{10})$
$\langle f \rangle ::=$	
$\langle \text{mff} \rangle$	Continuous meta-feature, see Table 6.2
$\text{---} c_f$	$x \sim \text{loguniform}(2^{-10}, 2^0)$
$\langle \text{unary} \rangle ::=$	
exp	$\exp(x)$
--- neg	$-x$
$\langle \text{binary} \rangle ::=$	
add	$x + y$
--- sub	$x - y$
--- mul	$x \cdot y$
--- truediv	x / y
--- pow	x^y
--- max	$\max(x, y)$
--- min	$\min(x, y)$
$\langle \text{quaternary} \rangle ::=$	
if_greater	if a > b: c else d

Table 6.1: BNF Grammar for symbolic defaults search. $\langle \text{configuration} \rangle$ is the start symbol.

symbol	explanation	min	median	max
$\langle \text{mfi} \rangle ::=$				
n	N. observations	100	4147	130064
po	N. features original	4	36	10000
p	N. features one-hot	4	54	71673
m	N. classes	2	2	100
$\langle \text{mff} \rangle ::=$				
rc	N. categorical / p	0.00	0.00	1.00
mcp	Majority Class %	0.01	0.52	1.00
mkd	Inv. Median Kernel Distance	0.00	0.01	0.55
xvar	Avg. feature variance	0.00	1.00	1.00

Table 6.2: Available meta-features with corresponding symbols

6.4 Finding Symbolic Defaults

The problem we aim to solve requires optimization over a space of mathematical expressions. Several options to achieve this exist, e.g., by optimizing over a pre-defined fixed-length set of functions [209]. One possible approach is to represent the space of functions as a grammar in Backus-Naur form and represent generated formulas as integer vectors where each entry represents which element of the right side of the grammar rule to follow [175]. We opt for a tree representation of individuals, where nodes correspond to operations and leaves to terminal symbols or numeric constants, and optimize this via genetic programming [140]. Our approach is inspired by symbolic regression [139], where the goal is to seek formulas that describe relationships in a dataset. In contrast, we aim to find a configuration (expressed via formulas), which minimizes $\hat{R}(\lambda(.))$.

We differentiate between real-valued ($\langle F \rangle$) and integer-valued ($\langle I \rangle$) terminal symbols to account for the difference in algorithm hyperparameters. This is helpful, as real-valued and integer hyperparameters typically vary over different orders of magnitude. Simultaneously, some meta-features might be optimal when set to a constant, which is enabled through ephemeral constants.

A relevant trade-off in this context is the bias induced via a limited set of operations, operation depth and available meta-features. Searching, e.g., only over expressions of the form $\langle binary \rangle (\langle mff \rangle, c_f)$ introduces significant bias towards the form and expressiveness of resulting formulas. Our approach using a grammar is agnostic towards the exact depth of resulting solutions, and bias is therefore only introduced via the choice of operators, meta-features and allowed depth.

Note that formulas can map outside of valid hyperparameter ranges. This complicates search on such spaces, as a large fraction of evaluated configurations might contain at least one infeasible setting. In order to reduce the likelihood of this happening, we define a set of mutation operators for the genetic algorithms that search locally around valid solutions. However if an infeasible setting is generated, the random forest surrogate effectively truncates it to the nearest observed value of that hyperparameter in the experiment meta-data, which is always valid.

Symbolic hyperparameters are interpretable and can lead to new knowledge i.e., about the interaction between dataset characteristics and performance.

6.4.1 Grammar

Table 6.1 shows the primitives and non-symbolic terminals of the grammar, and Table 6.2 shows the symbolic terminals whose values depend on the dataset. We define a set of *unary*, *binary* and *quaternary* operators which can be used to construct flexible functions for the different algorithm hyperparameters.

The definition start symbol $\langle \text{configuration} \rangle$ indicates the type and number of hyperparameters available in a configuration and depends on the algorithm for which we want to find a symbolic default configuration. In Table 6.3 we indicate for each considered hyperparameter of a learner whether it is real-valued or integer-valued (the latter denoted with an asterisk (*)). For example, when searching for a symbolic default configuration for the decision tree algorithm, $\langle \text{configuration} \rangle$ is defined as $\langle F \rangle \langle I \rangle \langle I \rangle \langle I \rangle$, because only the first hyperparameter is a float. Expressions for integer hyperparameters are also rounded after their expression has been evaluated. Starting from $\langle \text{configuration} \rangle$, placeholders $\langle I \rangle$ and $\langle F \rangle$ can now be iteratively replaced by either operators that have a return value of the same type or terminal symbols $\langle i \rangle$ and $\langle f \rangle$. Terminal symbols can either be meta-features (cf. Table 6.2) or ephemeral constants.

6.4.2 Algorithm

We consider a genetic programming approach for optimizing the symbolic expressions. We use a plus-strategy algorithm to evolve candidate solutions, where we set population size to 20 and generate 100 offspring in each generation via crossover and mutation. Evolution is run for 1000 generations in our experiments. We perform multi-objective optimization, jointly optimizing for performance of solutions (normalized logloss) while preferring formulas with smaller structural depth. Concretely, we employ NSGA-II selection [64] with binary tournament selection for parents and select offspring in an elitist fashion by usual non-dominated sorting and crowding distance as described in [64]. For offspring created by crossover, the M vector components of λ_j are chosen at random from both parents (uniform crossover on components), though we enforce at least one component from each parent is chosen. This results in large, non-local changes to candidates. Mutations, on the other hand, are designed to cause more local perturbations. We limit their effect to one hyperparameter only, and include varying constant values, pruning or expanding the expression or replacing a node. Each offspring is created through either crossover or mutation, never a combination. Initial expressions are generated with a maximum depth of three. We do not limit the depth of expressions during evolution ex-

plicitly, though multi-objective optimization makes finding deep formulas less likely.

6.5 Experimental Setup

We aim to answer the following research questions:

RQ1: How good is the performance of symbolic defaults in a practical sense? In order to assess this, we compare symbolic defaults with the following baselines: *a)* existing defaults in current software packages *b)* static defaults found by the search procedure described in Section 6.4, disallowing meta-features as terminal symbols and *c)* a standard hyperparameter random search (on the surrogates) with different budgets. Note that existing defaults already include symbolic hyperparameters in several implementations. In contrast to defaults obtained from our method, existing implementation defaults are often not empirically evaluated, and it is unclear how they were obtained. This question is the core of our work, as discrepancy to evaluations on real data only arise from inaccurate surrogate models, which can be improved by tuning or obtaining more data.

RQ2: How good are the symbolic defaults we find, when evaluated on real data? We evaluate symbolic defaults found by our method with experiments on real data and compare them to existing implementation defaults and a simple meta-model baseline.

6.5.1 General setup

We investigate symbolic defaults for 6 ML algorithms using the possibly largest available set of meta-data, containing evaluations of between 106 and 119 datasets included either in the OpenML-CC18 [29] benchmark suite or the AutoML benchmark [100]. Datasets have between 100 and 130000 observations, between 3 and 10000 features and 2 – 100 classes. The number of datasets varies across algorithms as we restrict ourselves to datasets where the experimental data contains evaluations of at least 100 unique hyperparameter configurations available as well as surrogate models that achieve sufficient quality (Spearman’s $\rho > 0.8$). We investigate implementations of a diverse assortment of state-of-the-art ML algorithms, namely support vector machines [59], elastic net [287], approximate knn [158], decision trees [36], random forests [37] and extreme gradient boosting (xgboost, [54]). The full set of used meta-features can be obtained from Table 6.2. The choice of datasets and ML algorithms evaluated in our paper was based on the availability of high-quality metadata. A large number of random

algorithm	fixed	optimized
elastic net	-	α, λ
decision tree	-	$cp, \quad max.depth^*,$ $\quad \quad \quad minbucket^*,$ $\quad \quad \quad minsplit^*$
random forest	splitrule:gini, num.trees:500, replace:True	$mtry^*,$ $sample.fraction,$ $min.node.size^*$
svm	kernel:radial	C, γ
approx. knn	distance:l2	k^*, M^*, ef^*, efc^*
xgboost	booster:gbtrees	$\eta, \quad \lambda, \quad \gamma, \quad \alpha,$ $subsample,$ $max_depth^*,$ $min_child_weight,$ $colsample_bytree,$ $colsample_bylevel$

Table 6.3: Fixed and optimizable hyperparameters for different algorithms. Hyperparameters with an asterisk (*) are integers.

evaluations across the full configuration space of each algorithm for each dataset was obtained and used in order to fit a random forest surrogate model for each dataset / algorithm combination⁴. We optimize the average logistic loss across 10 cross-validation folds (normalized to $[0,1]$), as it is robust to class-imbalances, but our methodology trivially extends to other performance measures. The hyperparameters optimized for each algorithm are shown in Table 6.3. For the random forest, we set the number of trees to 500 as recommended in literature [195]. We perform 10 replications of each experiment for all stochastic algorithms and present aggregated results.

6.5.2 Experiments for RQ1 & RQ2

The evaluation strategy for both experiments is based on a leave-one-data-set-out strategy, where the (symbolic) defaults are learned on all but one dataset, and evaluated using the held-out dataset.

In *Experiment 1* for **RQ1**, we evaluate symbolic defaults found using our approach against baselines mentioned in **RQ1**. Our hold-out-evaluation is performed on a held-out surrogate.

In *Experiment 2* for **RQ2**, we now learn our symbolic defaults in the same manner as for *Experiment 2* on $K - 1$ surrogates, but now evaluate their performance via a true cross-validation on the held out dataset instead of simply querying the held out surrogate.

Our main experiment – Experiment 1 evaluates symbolic defaults on surrogates for a held-out task, which lets us measure whether our symbolic defaults can extrapolate to future datasets. If results from surrogate evaluation correspond to real data, we can also conclude that our surrogates approximate the relationship between hyperparameters and performance well enough to transfer to real-world evaluations. We conjecture, that, given the investigated search space, for some algorithms/hyperparameters symbolic defaults do not add additional benefits and constant defaults suffice. In those cases, we expect that our approach performs roughly as well as an approach that only takes into account constant values, because our approach can similarly yield purely constant solutions.

We employ a modified procedure, *optimistic random search* that simulates random search on each dataset which is described below. We consider random search with budgets of up to 32 iterations to be a strong baseline. Other base-

⁴<https://www.openml.org/d/4245>[4-9]

lines like Bayesian optimization are left out of scope, as we only evaluate a single symbolic default, which is not optimized for the particular dataset. In contrast to requiring complicated evaluation procedures such as nested cross-validation, our symbolic defaults can simply be implemented as software defaults. We further consider full AutoML systems such as auto-sklearn [85] to be out-of-scope for comparison, as those evaluate across pre-processing steps and various ML algorithms, while our work focuses on finding defaults for a single algorithm without any pre-processing.

Optimistic random search As we deal with random search in the order of tens of evaluations, obtaining reliable results would require multiple replications of random search across multiple datasets and algorithms. We therefore adapt a cheaper, optimistic random search procedure, which samples *budget* rows from the available metadata for a given dataset, computes the best performance obtained and returns it as the random search result. Note that this assumes that nested cross-validation performance generalizes perfectly to the outer cross-validation performance, which is why it is considered an optimistic procedure. It is therefore expected to obtain higher scores than a realistic random search would obtain. Nonetheless, as we will show, the single symbolic default will often outperform the optimistic random search procedure with 8-16 iterations. The optimistic random search procedure is repeated several times in order to obtain reliable estimates.

1-Nearest Neighbour In Experiment 2, where we conduct evaluations with 10-fold cross-validation on the data, we also compare to a simple meta-model for generating a candidate solution given the meta-dataset. We use the k-Nearest Neighbour approach used by auto-sklearn [85], which looks up the best known hyperparameter configuration for the *nearest* dataset. To find the nearest datasets each meta-feature is first normalized using min-max scaling, then distances to each dataset are computed using L_1 -norm. While auto-sklearn finds hyperparameter configuration candidates for each of the 25 nearest neighbours, we only use the best hyperparameter configuration from the first nearest neighbour.

The Python code for our experiments is available online ⁵ and makes use of the DEAP module [90] for genetic programming.

⁵<https://github.com/PGijsbers/symbolicdefaults>

6.6 Results

In this section, we will first analyse the quality of our surrogates models in order to ensure their reliability. Next, we evaluate the performance of the found symbolic defaults on both surrogates and real data.

6.6.1 Surrogates and Surrogate Quality

As described earlier, we use surrogate models to predict the performance of hyperparameter configurations to avoid expensive evaluations. It is important that the surrogate models perform well enough to substitute for real experiments. For this optimization task, the most important quality of the surrogate models is the preservation of relative order of hyperparameter configuration performance. Error on predicted performance is only relevant if it causes the optimization method to incorrectly treat a worse configuration as a better one (or vice versa). The performance difference itself is irrelevant.

For that reason, we evaluate our surrogate models first on rank correlation coefficients Spearman’s ρ and Kendall’s τ . For each task, we perform 10-fold cross validation on our meta data, and calculate the rank correlation between the predicted ranking and the true one. On the left in Figure 6.2 we show the distribution of Spearman’s ρ and Kendall’s τ across 106 tasks for the SVM surrogate. Due to the high number of observations all rank correlations have a p-value of near zero. We observe high rank correlation for both measurements on most tasks. That τ values are lower than ρ values indicates that the surrogate model is more prone to making small mistakes than big ones. This is a positive when searching for good performing configurations, but may prove detrimental when optimizing amongst good configurations.

While it does not directly impact search, we also look at the difference between the predicted and true performances. For each task 10 configurations were sampled as a test set, and a surrogate model was trained on the remainder of the meta data for that task. On the right in Figure 6.2 the predicted normalized performance is shown against the real normalized performance. Predictions closer to the diagonal line are more accurate, points under and over the diagonal indicate the surrogate model was optimistic and pessimistic respectively. Plots for the other models can be found in Appendix C.2.

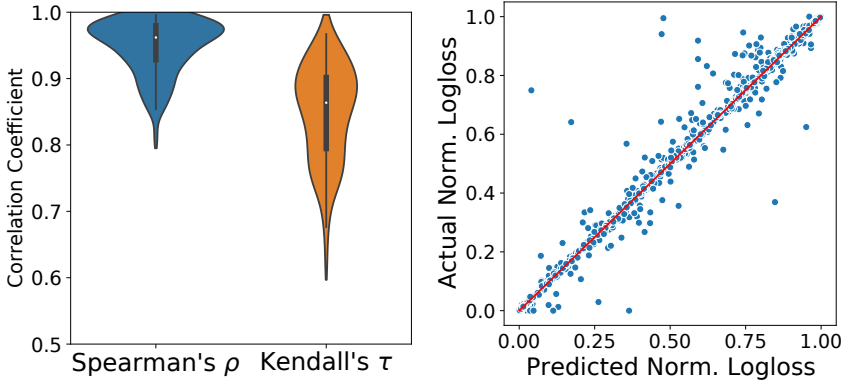


Figure 6.2: Comparison of performance predicted by the SVM surrogate against the real performance across tasks. On the left by their rank correlation coefficients, on the right in normalized performance.

6.6.2 Experiment 1 - Benchmark on surrogates

In order to answer **RQ1**, we compare the performance of symbolic defaults, constant defaults and existing implementation defaults on surrogates. Implementation defaults are default values currently used for the corresponding algorithm implementations and can be obtained from Table C.1 in the appendix. Note that random search in this context refers to per-task optimistic random-search as described in 6.6.1. In the following, we analyze results for the SVM and report normalized out-of-bag logistic loss on a surrogate if not stated otherwise. We conduct an analysis of the other algorithms mentioned in Table 6.3 in Appendix C.2.

A comparison to baselines *a – c*) for the SVM can be obtained from Figure 6.3. We compare symbolic defaults (blue), existing implementation defaults (green), constant defaults (purple) and several iterations of random search (orange). Symbolic defaults slightly outperform existing implementation defaults (mlr default and sklearn default) and compare favorably to random search with up to 8 evaluations.

For significance tests we use a non-parametric Friedman test for differences in samples at $\alpha = 0.05$ using and a post-hoc Nemenyi test. The corresponding critical differences diagram [65] is displayed in Figure 6.4. Methods are sorted on the x-axis by their average rank across tasks (lower is better). For methods

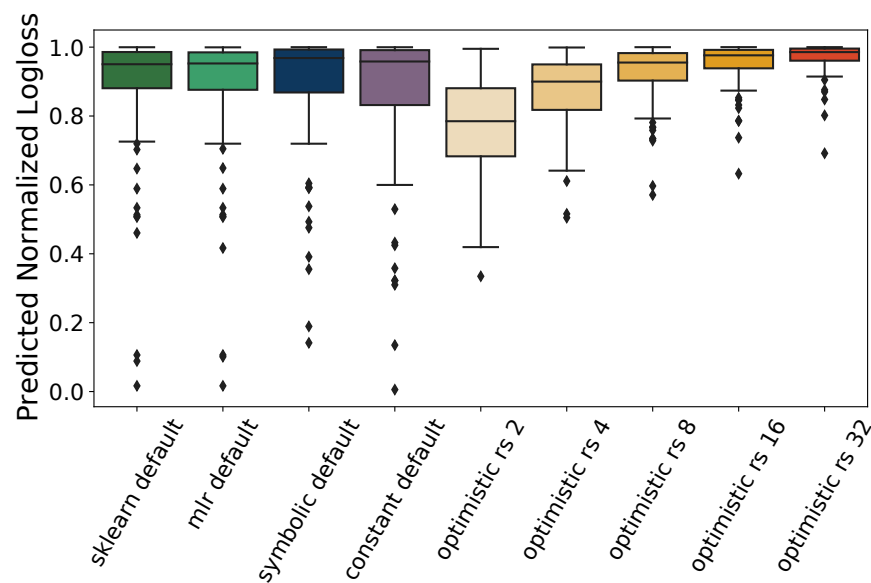


Figure 6.3: Symbolic, static and implementation defaults for SVM, comparing normalized logloss predicted by surrogates.

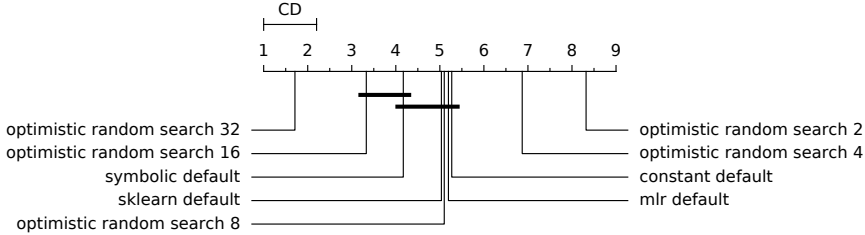


Figure 6.4: Critical Differences Diagram for symbolic, static and implementation defaults on surrogates

connected by a bold bar, significant differences could not be obtained. Symbolic defaults do not perform significantly worse than random search with a budget of 16 evaluations, however they also do not significantly outperform the hand-crafted implementation defaults or an optimized constant default.

Figure 6.5 shows comparisons to baselines, again using normalized logistic loss. The y-axis in both cases corresponds to symbolic defaults, while the x-axis corresponds to constant defaults (left) and random search with a budget of 8 (right). We conduct a similar analysis on all other algorithms in the appendix.

We summarize the results across all experiments in Table 6.4, which shows the mean normalized logistic loss and standard deviation across all tasks for each algorithm. The symbolic and constant column denote the performance of defaults found with our approach including and excluding symbolic terminals respectively. The package column shows the best result obtained from either the scikit-learn or mlr default, and the last column denotes the best found performance sampling 8 random real world scores on the task for the algorithm.

We find that the symbolic default mean rank is never significantly lower than that of other approaches, but in some cases it is significantly higher (in bold). While the mean performance for symbolic solutions is lower for glmnet, random forest and rpart, we observe that the average rank is only higher for glmnet (see Section C.2).

The only implementation default which does not score a significantly lower mean rank than the defaults found by search with symbolic terminals is the default for SVM, which has carefully hand-crafted defaults. This further motivates the use of experiment data for tuning default hyperparameter configurations. In three out of six cases the tuned defaults even outperform eight iterations of the

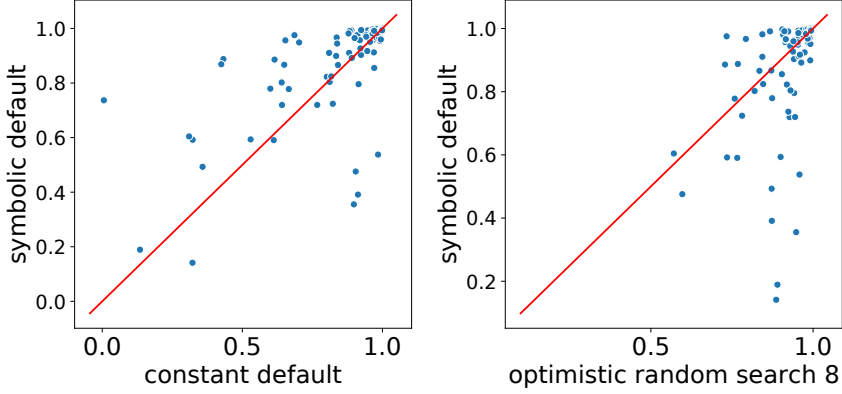


Figure 6.5: Performance comparison of symbolic defaults to constant defaults (left) and budget 8 random search (right). Points above the red line indicate symbolic defaults are better.

optimistic random search baseline, in the other cases they have a significantly higher mean rank than 4 iterations of random search.

6.6.3 Experiment 2 - Benchmark on real data

We run the defaults learned on $K - 1$ surrogates for each hold-out dataset with a true cross-validation and compare its performance to existing implementation defaults. We again analyze results for SVM and provide results on other algorithms in the appendix. Note, that instead of normalized log-loss (where 1 is the optimum), we report standard log-loss in this following section, which means lower is better. Figure 6.6 shows box plot and scatter plot comparisons between the better implementation default (sklearn) and symbolic defaults obtained from our method. The symbolic defaults found by our method performs slightly better to the two existing baselines in most cases, but outperforms the sklearn default on some datasets while never performing drastically worse. This small difference might not be all-too-surprising, as the existing sklearn defaults are already highly optimized symbolic defaults in their second iteration [217].

algorithm	symbolic	constant	package	opt. RS 8
glmnet	0.917(.168)	0.928(.158)	0.857(.154)	0.906(.080)
knn	0.954(.148)	0.947(.156)	0.879(.137)	0.995(.009)
rf	0.946(.087)	0.951(.074)	0.933(.085)	0.945(.078)
rpart	0.922(.112)	0.925(.093)	0.792(.141)	0.932(.082)
svm	0.889(.178)	0.860(.207)	0.882(.190)	0.925(.084)
xgboost	0.995(.011)	0.995(.011)	0.925(.125)	0.978(.043)

Table 6.4: Mean normalized log-loss (standard deviation) across all tasks with baselines. Boldface values indicate the average rank was not significantly worse than the best (underlined) of the four settings.

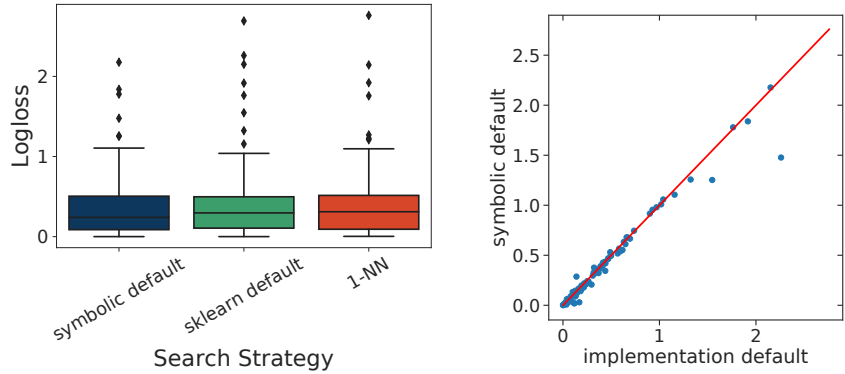


Figure 6.6: Comparison of symbolic and implementation default using log-loss across all datasets performed on real data. Box plots (right) and scatter plot (left)

6.7 Conclusion and Future Work

In this chapter, we consider the problem of finding data-dependent hyperparameter configurations, or *symbolic hyperparameter defaults*, that work well across datasets. We define a grammar that allows for complex expressions that can use data-dependent meta-features as well as constant values. Surrogate models are trained on a large meta dataset to efficiently optimize over symbolic expressions.

We find that the data-driven approach to finding default configurations leads to defaults as good as hand-crafted ones. The found defaults are generally better than the defaults set by algorithm implementations. Depending on the algorithm, the found defaults can be as good as performing 4 to 16 iterations of random search. In some cases, defaults benefit from being defined as a symbolic expression, i.e., in terms of data-dependent meta-features.

In future work, we aim to extend the search space in two ways: both in terms of the meta-features available and the grammar. Dataset characteristics have to reflect properties that are relevant to the algorithm hyperparameters, yet it is not immediately clear what those relevant properties are. It is straightforward to extend the number of meta-features, as many more have already been described in the literature (cf. [207]). This might not only serve to find even better symbolic defaults but it also reduces bias introduced by the small number of meta-features considered in our work. By extending the grammar described in Table 6.1 to include categorical terminals and operators more suitable for categorical hyperparameters (e.g., *if-else*), the described procedure can extend to categorical and hierarchical hyperparameters.

Another relevant aspect, which we do not study in this work is the *runtime* associated with a given default, as we typically want default values to be fast as well as good, and therefore this trade-off might be considered in optimizing symbolic defaults. In this work, we address this by restricting specific hyperparameters to specific values, in particular the xgboost `nrounds` parameter to 500. In future research, we aim to take this into consideration for all methods.

Finally, we want to evaluate ways in which these improved defaults may be used in AutoML design. Leveraging hyperparameter defaults to speed up AutoML has been exploited before, for example by sampling around the default values [278] or using the default values to shrink the search space [6]. It would be interesting to revisit these ideas with learned defaults that adapt to the task at hand, after first learning symbolic hyperparameters on a larger set of algorithms. While we learn the defaults over a large selection of datasets, it may be possible that these datasets are not representative of preprocessed data as found

in an ML pipeline, and thus it is possible that the currently learned symbolic hyperparameter defaults don't transfer as well to ML pipelines. Future work in this direction should evaluate the effect of preprocessing on the quality of the learned symbolic hyperparameter defaults and to see if adding additional experiments on preprocessed datasets to the meta-dataset is needed. Overall, having access to better, symbolic defaults, makes machine learning more accessible and robust to researchers from all domains.

Chapter 7

Conclusion and Future Work

Automated machine learning (AutoML) enables novice users by abstracting away the complexity of ML pipeline design and empowers the ML experts by saving time that would be spent tuning ML pipelines. When building an AutoML framework there are many design choices, such as which optimization algorithm to use or how to define the search space. Developing new methods and analyzing those design decisions is currently a very active area of research, with the first AutoML conference to take place in 2022.

Yet, in Chapter 1, we identified a few issues with current AutoML research. These include obfuscated comparisons across multiple design decisions, evaluations on inconsistent sets of datasets, and incorrect use of ‘competitor’ frameworks. In this chapter, we revisit the research questions we posed and summarize our contributions that address them in Section 7.1. In the two sections thereafter we discuss the limitations of our work and outline future research directions, respectively.

7.1 Conclusions

In this thesis, we presented work which we hope will improve the rate and quality of AutoML research. First, in Chapter 3, we introduced the modular AutoML tool **GAMA** to make implementing novel AutoML ideas easier (Q1). We observed that it was common for novel ideas to be evaluated against previous implementations that differed in more than one design decision, e.g., both the optimization algorithm and search space, which made it impossible to evaluate

the contribution of any individual component in AutoML design. We propose that this is because current AutoML tools were not designed on a level of abstraction that easily allowed researchers to investigate novel ideas, which made developing an entirely new AutoML tool an attractive idea. **GAMA** is designed to allow AutoML researchers to quickly develop and evaluate novel AutoML ideas in isolation and analyze their effectiveness.

By developing a modular AutoML tool that facilitates AutoML research, we not only significantly lower the barrier to developing and evaluating new AutoML ideas, but also ensure it is easy to evaluate each idea in isolation. Additionally, **GAMA** automatically tracks experiments and compiles data which researchers can use to better understand the workings of individual components, e.g., by visualizing their optimization trace. **GAMA** currently features three different optimization methods (random search, asynchronous successive halving, and an asynchronous evolutionary algorithm) and two different post-processing methods (fit the best pipeline, and ensemble construction through hill-climbing) which can be used in any combination.

The second issue we identified was the lack of standard benchmarking suites, i.e., the sets of datasets and evaluation procedures used to evaluate AutoML ideas. In Chapter 4, we presented an extension of the OpenML platform to allow for the creation and use of common benchmark suites (Q2). On the one hand, we provided easy programmatic access to the platform through `openml-python`, and on the other, we developed the concept of an OpenML benchmark suite. An OpenML benchmarking suite is a set of carefully selected OpenML tasks, which precisely define an evaluation procedure, including a reference to an exact dataset and evaluation splits.

We provided tools for researchers to construct their own benchmarking suites and used those tools to propose the OpenML Curated Classification suite (OpenML-CC18) for benchmarking classification algorithms on commodity hardware. For benchmarking AutoML systems we proposed two benchmarking suites (in Chapter 5), one with regression tasks and one with classification tasks. We saw that both the OpenML-CC18 and the AutoML¹ benchmarking suites have already been used in many other publications, a clear sign that they are useful but also that a continuous conversation with the research community is essential to evolve benchmarks and make them better and more useful over time.

To allow for the evaluation of AutoML tools in a correct and reproducible manner we developed the AutoML benchmark software tools and benchmarking suites (Q3). In Chapter 5 we give an overview of the developed software and

¹In particular an earlier version that was published at the ICML 2019 AutoML workshop.

report on the results of a large-scale evaluation of AutoML frameworks. We allow for correct and reproducible experiments by fully automating all aspects of the evaluation. We used `openml-python` to download and split the data in a reproducible manner and use integration scripts, which are developed together with the AutoML authors, to allow for the automated installation and usage of the AutoML frameworks, which avoids pitfalls such as a framework misconfiguration. The benchmarking framework can also build containers for even greater reproducibility, or to perform cross-platform benchmarking, and can distribute jobs to AWS which provides common hardware. Results are automatically aggregated and evaluated and can be analyzed with an interactive visualization tool.

We also proposed two OpenML benchmarking suites, one with 71 classification tasks, and one with 33 regression tasks. These benchmarking suites span a wide range of domains and dataset characteristics fit for tabular AutoML tools, unlike previously used sets of datasets which were typically small or not representative of the types of tasks the tools were designed to solve (e.g., image classification). We carried out a large-scale evaluation of 8 AutoML frameworks on these benchmarking suites and discuss the results, including the differences in performance and an analysis of the framework errors.

The fact that no single hyperparameter configuration is optimal across all tasks implies that there is a relationship between the dataset properties and the optimal hyperparameter configuration. In Chapter 6 we proposed a method based on symbolic regression to automatically find and leverage relationships between the dataset properties and good hyperparameter configurations, dubbed symbolic hyperparameter defaults, in a data-driven way through meta-learning over more than 100 tasks. To allow for the quick evaluation of symbolic hyperparameter defaults we trained surrogate models across tens of thousands of experiments across more than one hundred tasks for each ML algorithm. We showed that the proposed method is capable of finding symbolic hyperparameter defaults which are as good as hand-crafted ones, at least as good as constant hyperparameter defaults, and in almost all cases better than current implementation defaults. These defaults may be used to effectively warm-start search, but could also be used in other ways that may speed up AutoML, e.g., by using them in search space design (Q4).

7.2 Limitations

The methods proposed in this thesis come with limitations. Some of these limitations are inherent to the proposed methods or require further research, while others are merely a limitation that can be resolved through engineering effort. We will discuss the limitations in that order, with a focus on the former.

The core of this work focuses on enabling rigorous research on curated sets of tasks. While it has not yet been demonstrated in longitudinal studies, we assume that as more methods are being evaluated on benchmarking suites, overfitting on fixed suites is increasingly likely. To avoid a scenario where improved performance on the benchmarking suites no longer represents an improvement on other tasks, the benchmarking suites should be periodically updated.

Moreover, while benchmarking suites are an excellent tool to analyze quantitative differences between different methods, it provides no insight into the qualitative differences. In particular, for the AutoML benchmark, which evaluates tools designed to be used by end-users, an informed choice is often made on more than just performance reports. For example, the user may specifically be interested in the model’s interpretability, the level of support provided by the developers, or insight into why the final ML pipeline is designed the way it is.

The conducted set of experiments on AutoML frameworks in Chapter 5 are deliberately designed to reflect the out-of-the-box experience that regular users will encounter. This means that based on the experiments no conclusion can be drawn about the quality of any individual design decision. Another limitation of the experimental results is that only final performance is measured. While in some cases the final performance may not be statistically significantly different, it may be that one tool converges to a solution much faster than others.

In principle, the AutoML benchmark allows for performing ablation studies, though this also requires a high level of configurability of the AutoML frameworks. Modular AutoML tool **GAMA** features this configurability and may be used to evaluate along one design axis at a time. However, the measured performance is still affected by other choices in the design and experimental evaluation. If in an ablation study method A outperforms method B, this still comes with the caveat that the results only hold for e.g., the used search space or resource budget, and the extent to which they generalize across those decisions is unknown. It should be noted that this limitation is not unique to **GAMA**, but indeed any experiment with sufficiently many design decisions.

Finally, in our work on the automated discovery of symbolic hyperparame-

ter defaults, we used a limited set of meta-features and mathematical operators from which to compose the defaults. Given these design choices, we were unable to find suitable symbolic defaults for several algorithms and did not significantly outperform tuned constant defaults for them. Further research should include more dataset properties, though it is not immediately obvious which these should be. It also remains an open question if for all hyperparameters there even exists a symbolic default that uses only meta-features which can be computed efficiently.

There are also limitations imposed by the state of the software. The OpenML platform, and by extension `openml-python`, OpenML benchmarking suites, and the AutoML benchmark, offers only limited support for settings outside of i.i.d. classification and regression, such as clustering or time series prediction. **GAMA** allows for modular configuration and isolated development for search algorithms and post-processing, though it does not yet offer the same flexibility in other parts of the AutoML pipeline design. For example, the data sanitation step is fixed and the search space design assumes `scikit-learn` compatible workflows. None of these limitations are inherent to the respective designs and can be overcome with additional engineering effort.

7.3 Future Work

As outlined in the last section, we can overcome some limitations through additional engineering effort. In this section, we focus on interesting future work which can not be overcome by engineering alone.

7.3.1 Meta-learning for AutoML

In Chapter 6 we presented a method to use meta-learning to find symbolic hyperparameter defaults. How to incorporate these symbolic hyperparameter defaults in AutoML tools is an interesting open research question. Possible applications include transforming the search space, shrinking the search space to speed up the search, or using symbolic hyperparameter defaults to evaluate ML pipeline architecture design. We hope to find symbolic hyperparameter defaults for more algorithms and hyperparameters by extending the set of meta-features and the formulas which may be considered. Moreover, we aim to extend the notion of defaults into sets of defaults, which can serve as complementary starting points for hyperparameter optimization.

Many other approaches to include meta-learning in AutoML methods have already been proposed [82, 88, 144, 149, 279]. Unfortunately, it is unclear how to evaluate AutoML methods which use meta-learning on benchmarking suites in a practical manner. The task on which a method is evaluated should not be included in learning the meta-model used by the method. While some meta-learning methods, such as nearest-neighbor dataset lookups [88], allow for the easy exclusion of specific tasks from the meta-model, this is not the case for meta-models in general. A clean evaluation would then involve training as many meta-models as there are (chosen subsets of) tasks in the benchmarking suite, which may become prohibitively expensive for more complex meta-models. Additionally, it is not always easy to identify which task is being used in the evaluation. While the specific dataset may be easily identified, all variants derived from the dataset should also be accounted for and excluded from the meta-model. More research is required to address these issues and allow for the correct evaluation of AutoML systems that use meta-learning.

7.3.2 Benchmark Design

While the tools presented with the introduction of OpenML benchmarking suites allow for some automated curation of tasks, the proposed benchmarking suites are still mostly designed by humans. The design process may lead to unnecessarily large benchmarking suites, which is undesirable not only because it wastes resources, but also because the increased computational demand will prohibit some people from using the proposed suites. Some post-hoc analysis methods of benchmarking suites exist [47], but we hope additional techniques will be developed and in particular for them to already be applicable during the design process.

It remains an open question if and when methods may start to overfit to a static benchmarking suite. For this reason, and to keep the benchmarking suites reflective of modern challenges, we propose to periodically update the benchmarking suites (e.g., as done for computer vision research [201]) and invite the community to partake in this process. Developing new methods to analyze whether overfitting on benchmark suites occurs, and how many or which tasks would need to be replaced to alleviate the issue, is interesting future work.

7.3.3 Trust in AutoML

Interpretable [171] and explainable [240] ML has gained attention recently, in part because of new legislature that requires explainability [21], e.g., GDPR². As this pertains to the final model produced, AutoML can directly benefit from ideas and techniques for general interpretability and explainability in ML, such as training interpretable models to mimic complex ones found by AutoML [3, 133] or post-hoc model-agnostic explanation methods such as LIME [205]. However, AutoML may also be used to generate interpretable models, by using existing AutoML frameworks with an altered search space that produces interpretable models [91], or by using `autocompboost` [58], which is specifically developed to build interpretable models.

In AutoML not only the final model but also how it was found, is important for a user's trust [69, 220]. To this end, Moosbauer et al. [173] propose to use adapted partial dependence plots to visualize what the surrogate model learned about the search space. Providing the users with generated code that builds the final model also increases trust in the system, because it helps them understand the model that is used and to verify if specific changes affect the results as expected [266].

In certain settings it is important that the model follows some notion of fairness [13], e.g., when the model affects humans, it shouldn't discriminate. This can be expressed through metrics that make a distinction between a protected and unprotected group. Examples include demographic parity [42], which stipulates the average predictions for the two groups should be equal, and equalized odds [113], which dictate the false negative and positive rates should be equal between the groups. However, it should be noted that while different notions of fairness exist, they may not all be satisfied simultaneously [57, 134]. While this is also true for performance metrics, the choice of fairness metric has a significant effect on how the model treats the protected group.

To allow AutoML to find fairer models, it has been treated as a multi-objective optimization problem, optimizing a fairness metric and a performance metric together (e.g., [60, 214, 215]). However, this approach ignores the development of fairness specific preprocessing, in-processing, and post-processing algorithms (e.g., [44], [19], and [113], respectively), which seems like it would lead to sub-optimal pipelines³.

Still, only changing the optimization objective, or even the search space,

²<https://gdpr-info.eu/>

³To the best of my knowledge, there is no AutoML system which includes these algorithms in its search space, so there is no evidence that excluding them leads to worse solutions.

largely ignores the problems present in other parts of the model creation. Blind optimization without regard for other aspects, such as the data and its collection process or the users ultimately using the model, may only lead to perceived progress [11]. On the one hand, AutoML may exacerbate that problem. If, at times, even ML experts fail to identify biases in their models [96], how will the novice AutoML user pick up on these errors? On the other hand, AutoML may alleviate some of these issues by allowing the domain experts themselves to build models. With a much better understanding of the data, the relevant performance metrics, and the ability to assess model predictions, domain experts using AutoML may be able to deploy better models than an ML expert could. These two scenarios are not mutually exclusive, and both user groups would benefit from support for fair learning and interpretability in both the AutoML procedure and the model it produces.

Bibliography

- [1] 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.
- [2] D. W. Aha. “Generalizing from case studies: A case study”. In: *Proceedings of the International Conference on Machine Learning (ICML)* (1992), pp. 1–10.
- [3] Ahmed Alaa and Mihaela van der Schaar. “AutoPrognosis: Automated Clinical Prognostic Modeling via Bayesian Optimization with Structured Kernel Learning”. In: *Proceedings of the 35th International Conference on Machine Learning*. Ed. by Jennifer Dy and Andreas Krause. Vol. 80. Proceedings of Machine Learning Research. PMLR, July 2018, pp. 139–148. URL: <https://proceedings.mlr.press/v80/aaa18b.html>.
- [4] 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.” In: *Journal of Multiple-Valued Logic and Soft Computing* 17.2-3 (2010), pp. 255–287.
- [5] Edesio Alcobaça, Felipe Siqueira, Adriano Rivolli, Luís Paulo F Garcia, Jefferson Tales Oliva, André CPLF de Carvalho, et al. “MFE: Towards reproducible meta-feature extraction.” In: *J. Mach. Learn. Res.* 21 (2020), pp. 111–1.
- [6] Marie Anastacio, Chuan Luo, and Holger Hoos. “Exploitation of default parameter values in automated algorithm configuration”. In: *Workshop Data Science meets Optimisation (DSO), IJCAI*. 2019.

- [7] Noor Awad, Neeratyoy Mallik, and Frank Hutter. “DEHB: Evolutionary Hyberband for Scalable, Robust and Efficient Hyperparameter Optimization”. In: *arXiv preprint arXiv:2105.09821* (2021).
- [8] Claudine Badue, Rânik Guidolini, Raphael Vivacqua Carneiro, Pedro Azevedo, Vinicius B Cardoso, Avelino Forechi, Luan Jesus, Rodrigo Berriel, Thiago M Paixao, Filipe Mutz, et al. “Self-driving cars: A survey”. In: *Expert Systems with Applications* 165 (2021), p. 113816.
- [9] Adithya Balaji and Alexander Allen. “Benchmarking Automatic Machine Learning Frameworks”. In: (Aug. 2018). arXiv: 1808.06492 [cs.LG].
- [10] Wolfgang Banzhaf, Peter Nordin, Robert E Keller, and Frank D Francone. *Genetic programming: an introduction: on the automatic evolution of computer programs and its applications*. Morgan Kaufmann Publishers Inc., 1998.
- [11] Michelle Bao, Angela Zhou, Samantha Zottola, Brian Brubach, Sarah Desmarais, Aaron Horowitz, Kristian Lum, and Suresh Venkatasubramanian. “It’s COMPASlicated: The Messy Relationship between RAI Datasets and Algorithmic Fairness Benchmarks”. In: *CoRR* abs/2106.05498 (2021). arXiv: 2106.05498. URL: <https://arxiv.org/abs/2106.05498>.
- [12] Rémi Bardenet, Mátyás Brendel, Balázs Kégl, and Michèle Sebag. “Collaborative Hyperparameter Tuning”. In: *Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28*. ICML’13. Atlanta, GA, USA: JMLR.org, 2013, pp. II-199–II-207. URL: <http://dl.acm.org/citation.cfm?id=3042817.3042916>.
- [13] Solon Barocas, Moritz Hardt, and Arvind Narayanan. “Fairness in machine learning”. In: *Nips tutorial 1* (2017), p. 2017.
- [14] Hilan Bensusan and Alexandros Kalousis. “Estimating the predictive accuracy of a classifier”. In: *European Conference on Machine Learning*. Springer. 2001, pp. 25–36.
- [15] J. Bergstra, N. Pinto, and D.D. Cox. “SkData: data sets and algorithm evaluation protocols in Python”. In: *Computational Science & Discovery* 8.1 (2015).
- [16] James Bergstra, Rémi Bardenet, B Kégl, and Y Bengio. “Implementations of algorithms for hyper-parameter optimization”. In: *NIPS Workshop on Bayesian optimization*. 2011, p. 29.
- [17] James Bergstra and Yoshua Bengio. “Random search for hyper-parameter optimization.” In: *Journal of machine learning research* 13.2 (2012).

- [18] James Bergstra, Daniel Yamins, and David Cox. “Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures”. In: *International conference on machine learning*. PMLR. 2013, pp. 115–123.
- [19] Richard Berk, Hoda Heidari, Shahin Jabbari, Matthew Joseph, Michael Kearns, Jamie Morgenstern, Seth Neel, and Aaron Roth. “A convex framework for fair regression”. In: *arXiv preprint arXiv:1706.02409* (2017).
- [20] Hans-Georg Beyer and Hans-Paul Schwefel. “Evolution strategies—a comprehensive introduction”. In: *Natural computing* 1.1 (2002), pp. 3–52.
- [21] Adrien Bibal, Michael Lognoul, Alexandre De Streel, and Benoît Frénay. “Legal requirements on explainability in machine learning”. In: *Artificial Intelligence and Law* 29.2 (2021), pp. 149–169.
- [22] Aurélien Bibaut, Antoine Chambaz, Maria Dimakopoulou, Nathan Kallus, and Mark van der Laan. “Post-Contextual-Bandit Inference”. In: *arXiv:2106.00418 [stat.ML]* (2021).
- [23] Aurélien Bibaut, Antoine Chambaz, Maria Dimakopoulou, Nathan Kallus, and Mark van der Laan. “Risk Minimization from Adaptively Collected Data: Guarantees for Supervised and Policy Learning”. In: *arXiv:2106.01723 [stat.ML]* (2021).
- [24] Albert Bifet and Ricard Gavaldà. “Adaptive learning from evolving data streams”. In: *International Symposium on Intelligent Data Analysis*. Springer. 2009, pp. 249–260.
- [25] Mauro Birattari, Thomas Stützle, Luis Paquete, Klaus Varrentrapp, et al. “A Racing Algorithm for Configuring Metaheuristics.” In: *Gecco*. Vol. 2. 2002. 2002.
- [26] B. Bischl, M. Lang, L. Kotthoff, J. Schiffner, J. Richter, E. Studerus, G. Casalicchio, and Z. M. Jones. “mlr: Machine learning in R”. In: *Journal of Machine Learning Research* 17.170 (2016).
- [27] Bernd Bischl, Martin Binder, Michel Lang, Tobias Pielok, Jakob Richter, Stefan Coors, Janek Thomas, Theresa Ullmann, Marc Becker, Anne-Laure Boulesteix, Difan Deng, and Marius Lindauer. “Hyperparameter Optimization: Foundations, Algorithms, Best Practices and Open Challenges”. In: (July 2021). arXiv: 2107.05847 [stat.ML].

- [28] Bernd Bischl, Giuseppe Casalicchio, Matthias Feurer, Pieter Gijsbers, Frank Hutter, Michel Lang, Rafael Gomes Mantovani, Jan N van Rijn, and Joaquin Vanschoren. “OpenML Benchmarking Suites”. In: *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*. 2021.
- [29] Bernd Bischl, Giuseppe Casalicchio, Matthias Feurer, Frank Hutter, Michel Lang, Rafael G. Mantovani, Jan N. van Rijn, and Joaquin Vanschoren. “OpenML Benchmarking Suites”. In: (Aug. 2017). arXiv: 1708.03731 [stat.ML].
- [30] Bernd Bischl, Pascal Kerschke, Lars Kotthoff, Marius Lindauer, Yuri Malitsky, Alexandre Fréchette, Holger Hoos, Frank Hutter, Kevin Leyton-Brown, Kevin Tierney, and Joaquin Vanschoren. “ASlib: A benchmark library for algorithm selection”. en. In: *Artificial Intelligence* 237 (Aug. 2016), pp. 41–58. ISSN: 0004-3702. DOI: 10.1016/j.artint.2016.04.003. URL: <https://www.sciencedirect.com/science/article/pii/S0004370216300388> (visited on 10/21/2021).
- [31] Jesús Bobadilla, Fernando Ortega, Antonio Hernando, and Abraham Gutiérrez. “Recommender systems survey”. In: *Knowledge-based systems* 46 (2013), pp. 109–132.
- [32] Bernhard E Boser, Isabelle M Guyon, and Vladimir N Vapnik. “A training algorithm for optimal margin classifiers”. In: *Proceedings of the fifth annual workshop on Computational learning theory*. 1992, pp. 144–152.
- [33] Pavel Brazdil, João Gama, and Bob Henery. “Characterizing the applicability of classification algorithms using meta-level learning”. In: *European conference on machine learning*. Springer. 1994, pp. 83–102.
- [34] Pavel B Brazdil and Carlos Soares. “A comparison of ranking methods for classification algorithm selection”. In: *European conference on machine learning*. Springer. 2000, pp. 63–75.
- [35] Pavel B Brazdil, Carlos Soares, and Joaquim Pinto Da Costa. “Ranking learning algorithms: Using IBL and meta-learning on accuracy and time results”. In: *Machine Learning* 50.3 (2003), pp. 251–277.
- [36] L Breiman, JH Friedman, R Olshen, and CJ Stone. “Classification and Regression Trees”. In: (1984).
- [37] Leo Breiman. “Random Forests”. In: *Mach. Learn.* 45.1 (Oct. 2001), pp. 5–32. ISSN: 0885-6125. DOI: 10.1023/A:1010933404324. URL: <https://doi.org/10.1023/A:1010933404324>.

- [38] Leo Breiman and Adele Cutler. *Random Forests Manual*. 2020. URL: https://www.stat.berkeley.edu/~breiman/RandomForests/cc_home.htm (visited on 05/01/2020).
- [39] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. “OpenAI Gym”. In: *arXiv:1606.01540 [cs.LG]* (2016).
- [40] L. Buitinck, 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. “API design for machine learning software: experiences from the scikit-learn project”. In: *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*. 2013, pp. 108–122.
- [41] Richard H Byrd, Peihuang Lu, Jorge Nocedal, and Ciyu Zhu. “A limited memory algorithm for bound constrained optimization”. In: *SIAM Journal on scientific computing* 16.5 (1995), pp. 1190–1208.
- [42] Toon Calders and Sicco Verwer. “Three naive Bayes approaches for discrimination-free classification”. In: *Data mining and knowledge discovery* 21.2 (2010), pp. 277–292.
- [43] Tadeusz Caliński and Jerzy Harabasz. “A dendrite method for cluster analysis”. In: *Communications in Statistics-theory and Methods* 3.1 (1974), pp. 1–27.
- [44] Flavio P Calmon, Dennis Wei, Bhanukiran Vinzamuri, Karthikeyan Nateisan Ramamurthy, and Kush R Varshney. “Optimized pre-processing for discrimination prevention”. In: *Proceedings of the 31st International Conference on Neural Information Processing Systems*. 2017, pp. 3995–4004.
- [45] Israel Campero Jurado and Joaquin Vanschoren. “Multi-fidelity optimization method with Asynchronous Generalized Island Model for AutoML”. In: *(to appear) Proceedings of the Genetic and Evolutionary Computation Conference Companion* (July 2022).
- [46] B. Caputo, K. Sim, F. Furesjo, and A. Smola. “Appearance-based Object Recognition using SVMs: Which Kernel Should I Use?” In: *Proceedings of NIPS workshop on Statistical methods for computational experiments in visual processing and computer vision, Whistler*. 2002, pp. 1–10.
- [47] Lucas FF Cardoso, Vitor CA Santos, Regiane S Kawasaki Francês, Ricardo BC Prudêncio, and Ronnie CO Alves. “Data vs classifiers, who wins?” In: *arXiv:2107.07451 [cs.LG]* (2021).

- [48] Rich Caruana, Art Munson, and Alexandru Niculescu-Mizil. “Getting the most out of ensemble selection”. In: *Sixth International Conference on Data Mining (ICDM’06)*. IEEE. 2006, pp. 828–833.
- [49] Rich Caruana, Alexandru Niculescu-Mizil, Geoff Crew, and Alex Ksikes. “Ensemble selection from libraries of models”. In: *Proceedings of the twenty-first international conference on Machine learning*. 2004, p. 18.
- [50] Giuseppe Casalicchio, Jakob Bossek, Michel Lang, Dominik Kirchhoff, Pascal Kerschke, Benjamin Hofner, Heidi Seibold, Joaquin Vanschoren, and Bernd Bischl. “OpenML: An R package to connect to the machine learning platform OpenML”. In: 34 (2019), pp. 977–991. issn: 0943-4062. doi: 10.1007/s00180-017-0742-2.
- [51] Bilge Celik, Prabhant Singh, and Joaquin Vanschoren. *Online AutoML: An adaptive AutoML framework for online learning*. 2022. arXiv: 2201.09750 [cs.LG].
- [52] Bilge Celik and Joaquin Vanschoren. “Adaptation strategies for automated machine learning on evolving data”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2021).
- [53] C. C. Chang and C. J. Lin. “LIBSVM: A library for support vector machines”. In: *ACM Transactions on Intelligent Systems and Technology (TIST)* 2.3 (2011), p. 27.
- [54] Tianqi Chen and Carlos Guestrin. “XGBoost: A Scalable Tree Boosting System”. In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD ’16. San Francisco, California, USA: ACM, 2016, pp. 785–794. isbn: 978-1-4503-4232-2. doi: 10.1145/2939672.2939785. url: <http://doi.acm.org/10.1145/2939672.2939785>.
- [55] Tianqi Chen, Tong He, Michael Benesty, Vadim Khotilovich, Yuan Tang, Hyunsu Cho, et al. “Xgboost: extreme gradient boosting”. In: *R package version 0.4-2* 1.4 (2015), pp. 1–4.
- [56] Y. Chen, E. Keogh, B. Hu, N. Begum, A. Bagnall, A. Mueen, and G. Batista. *The UCR Time Series Classification Archive*. www.cs.ucr.edu/~eamonn/time_series_data/. July 2015.
- [57] Alexandra Chouldechova. “Fair prediction with disparate impact: A study of bias in recidivism prediction instruments”. In: *Big data* 5.2 (2017), pp. 153–163.

- [58] Stefan Coors, Daniel Schalk, Bernd Bischl, and David Rügamer. “Automatic Componentwise Boosting: An Interpretable AutoML System”. In: *arXiv preprint arXiv:2109.05583* (2021).
- [59] Corinna Cortes and Vladimir Vapnik. “Support-vector networks”. In: *Machine learning* 20.3 (1995), pp. 273–297.
- [60] André F Cruz, Pedro Saleiro, Catarina Belém, Carlos Soares, and Pedro Bizarro. “A Bandit-Based Algorithm for Fairness-Aware Hyperparameter Optimization”. In: *arXiv preprint arXiv:2010.03665* (2020).
- [61] Casey Davis and Christophe Giraud-Carrier. “Annotative experts for hyperparameter selection”. In: *AutoML Workshop at ICML*. 2018.
- [62] George De Ath, Richard M Everson, Alma AM Rahat, and Jonathan E Fieldsend. “Greed is good: Exploration and exploitation trade-offs in Bayesian optimisation”. In: *ACM Transactions on Evolutionary Learning and Optimization* 1.1 (2021), pp. 1–22.
- [63] Gwendoline De Bie, Herilalaina Rakotoarison, Gabriel Peyré, and Michèle Sebag. “Distribution-Based Invariant Deep Networks for Learning Meta-Features”. In: *arXiv:2006.13708 [stat.ML]* (2020).
- [64] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and TAMT Meyarivan. “A fast and elitist multiobjective genetic algorithm: NSGA-II”. In: *IEEE transactions on evolutionary computation* 6.2 (2002), pp. 182–197.
- [65] J. Demšar. “Statistical Comparisons of Classifiers over Multiple Data Sets”. In: *The Journal of Machine Learning Research* 7 (2006), pp. 1–30.
- [66] D. Dheeru and E. Karra Taniskidou. *UCI Machine Learning Repository*. 2017. URL: <http://archive.ics.uci.edu/ml>.
- [67] Elizabeth Ditton, Anne Swinbourne, Trina Myers, and Mitchell Scovell. “Applying Semi-Automated Hyperparameter Tuning for Clustering Algorithms”. In: *arXiv preprint arXiv:2108.11053* (2021).
- [68] Iddo Drori, Yamuna Krishnamurthy, Remi Rampin, Raoni Lourenço, Jorge One, Kyunghyun Cho, Claudio Silva, and Juliana Freire. “AlphaD3M: Machine learning pipeline synthesis”. In: *5th ICML Workshop on Automated Machine Learning (AutoML)*. 2018.

- [69] Jaimie Drozdal, Justin Weisz, Dakuo Wang, Gaurav Dass, Bingsheng Yao, Changruo Zhao, Michael Muller, Lin Ju, and Hui Su. “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. Cagliari, Italy: Association for Computing Machinery, 2020, pp. 297–307. ISBN: 9781450371186. DOI: 10.1145/3377325.3377501. URL: <https://doi.org/10.1145/3377325.3377501>.
- [70] Russell Eberhart and James Kennedy. “Particle swarm optimization”. In: *Proceedings of the IEEE international conference on neural networks*. Vol. 4. Citeseer. 1995, pp. 1942–1948.
- [71] Katharina Eggensperger, Matthias Feurer, Frank Hutter, James Bergstra, Jasper Snoek, Holger Hoos, Kevin Leyton-Brown, et al. “Towards an empirical foundation for assessing bayesian optimization of hyperparameters”. In: *NIPS workshop on Bayesian Optimization in Theory and Practice*. Vol. 10. 3. 2013.
- [72] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. “Neural architecture search: A survey”. In: *The Journal of Machine Learning Research* 20.1 (2019), pp. 1997–2017.
- [73] Nick Erickson, Jonas Mueller, Alexander Shirkov, Hang Zhang, Pedro Larroy, Mu Li, and Alexander Smola. “AutoGluon-Tabular: Robust and Accurate AutoML for Structured Data”. In: (Mar. 2020). arXiv: 2003.06505 [stat.ML].
- [74] Kutluhan Erol, James A Hendler, and Dana S Nau. “UMCP: A Sound and Complete Procedure for Hierarchical Task-network Planning.” In: *Aips*. Vol. 94. 1994, pp. 249–254.
- [75] Hugo Jair Escalante, Manuel Montes, and Luis Enrique Sucar. “Particle swarm model selection.” In: *Journal of Machine Learning Research* 10.2 (2009).
- [76] Hugo Jair Escalante, Wei-Wei Tu, Isabelle Guyon, Daniel L. Silver, Evelyn Viegas, Yuqiang Chen, Wenyuan Dai, and Qiang Yang. “AutoML @ NeurIPS 2018 Challenge: Design and Results”. In: *The NeurIPS '18 Competition*. Ed. by Sergio Escalera and Ralf Herbrich. Cham: Springer International Publishing, 2020, pp. 209–229. ISBN: 978-3-030-29135-8.

- [77] Suilan Estévez-Velarde, Yoan Gutiérrez, Yudivián Almeida-Cruz, and Andrés Montoyo. “General-purpose hierarchical optimisation of machine learning pipelines with grammatical evolution”. In: *Information Sciences* 543 (2021), pp. 58–71.
- [78] Manuel J.A. Eugster, Friedrich Leisch, and Carolin Strobl. “(Psycho-)Analysis of Benchmark Experiments”. In: *Comput. Stat. Data Anal.* 71.C (Mar. 2014), pp. 986–1000. ISSN: 0167-9473.
- [79] Raül Fabra-Boluda, Cesar Ferri, Fernando Martínez-Plumed, José Hernández-Orallo, and M José Ramírez-Quintana. “Family and prejudice: A behavioural taxonomy of machine learning techniques”. In: *ECAI 2020 - 24th European Conference on Artificial Intelligence*. IOS Press, 2020, pp. 1135–1142.
- [80] Stefan Falkner, Aaron Klein, and Frank Hutter. “BOHB: Robust and efficient hyperparameter optimization at scale”. In: *International Conference on Machine Learning*. PMLR. 2018, pp. 1437–1446.
- [81] Mauricio Ferreira, Rafaela Ventorim, Eduardo Almeida, Sabrina Silveira, and Wendel Silveira. “Protein Abundance Prediction Through Machine Learning Methods”. In: *Journal of molecular biology* 433.22 (2021), p. 167267.
- [82] Matthias Feurer, Katharina Eggensperger, Stefan Falkner, Marius Lindauer, and Frank Hutter. “Auto-Sklearn 2.0: Hands-free AutoML via Meta-Learning”. In: *arXiv:2007.04074 [cs.LG]* (2020).
- [83] Matthias Feurer, Katharina Eggensperger, Stefan Falkner, Marius Lindauer, and Frank Hutter. “Practical automated machine learning for the automl challenge 2018”. In: *International Workshop on Automatic Machine Learning at ICML*. 2018, pp. 1189–1232.
- [84] Matthias Feurer and Frank Hutter. “Hyperparameter Optimization”. In: *Automated Machine Learning*. Springer, Cham, 2019, pp. 3–33.
- [85] Matthias Feurer, Aaron Klein, Katharina Eggensperger, Jost Tobias Springenberg, Manuel Blum, and Frank Hutter. “Efficient and Robust Automated Machine Learning”. In: *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*. Ed. by Corinna Cortes, Neil D. Lawrence, Daniel D. Lee, Masashi Sugiyama, and Roman Garnett. 2015, pp. 2962–2970. URL: <https://proceedings.neurips.cc/paper/2015/hash/11d0e6287202fced83f79975ec59a3a6-Abstract.html>.

- [86] Matthias Feurer, Benjamin Letham, and Eytan Bakshy. “Scalable meta-learning for Bayesian optimization”. In: *stat* 1050 (2018), p. 6.
- [87] Matthias Feurer, Jan N van Rijn, Arlind Kadra, Pieter Gijsbers, Neeratyoy Mallik, Sahithya Ravi, Andreas Mueller, Joaquin Vanschoren, and Frank Hutter. “Openml-python: an extensible python api for openml”. In: *Journal of Machine Learning Research* 22:100 (2021), pp. 1–5.
- [88] Matthias Feurer, Jost Springenberg, and Frank Hutter. “Initializing bayesian hyperparameter optimization via meta-learning”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 29. 1. 2015.
- [89] Chelsea Finn, Pieter Abbeel, and Sergey Levine. “Model-agnostic meta-learning for fast adaptation of deep networks”. In: *International conference on machine learning*. PMLR. 2017, pp. 1126–1135.
- [90] Félix-Antoine Fortin, François-Michel De Rainville, Marc-André Gardner, Marc Parizeau, and Christian Gagné. “DEAP: Evolutionary Algorithms Made Easy”. In: *Journal of Machine Learning Research* 13 (July 2012), pp. 2171–2175.
- [91] Alex A Freitas. “Automated machine learning for studying the trade-off between predictive accuracy and interpretability”. In: *International Cross-Domain Conference for Machine Learning and Knowledge Extraction*. Springer. 2019, pp. 48–66.
- [92] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. “Regularization Paths for Generalized Linear Models via Coordinate Descent”. In: *Journal of Statistical Software* 33.1 (2010), pp. 1–22. URL: <http://www.jstatsoft.org/v33/i01/>.
- [93] Jerome H Friedman. “Greedy function approximation: a gradient boosting machine”. In: *Annals of statistics* (2001), pp. 1189–1232.
- [94] N. Fusi, R. Sheth, and M. Elibol. “Probabilistic Matrix Factorization for Automated Machine Learning”. In: *Proc. of NeurIPS’18*. 2018.
- [95] João Gama, Indrė Žliobaitė, Albert Bifet, Mykola Pechenizkiy, and Abdelhamid Bouchachia. “A survey on concept drift adaptation”. In: *ACM computing surveys (CSUR)* 46.4 (2014), pp. 1–37.
- [96] Megan Garcia. “Racist in the Machine”. In: *World Policy Journal* 33.4 (2016), pp. 111–117.
- [97] Timnit Gebru, Jamie Morgenstern, Briana Vecchione, Jennifer Wortman Vaughan, Hanna Wallach, Hal Daumé III, and Kate Crawford. “Datasheets for datasets”. In: *arXiv:1803.09010 [cs.DB]* (2018).

- [98] Malik Ghallab, Dana Nau, and Paolo Traverso. *Automated Planning: theory and practice*. Elsevier, 2004.
- [99] P Gijsbers, J Vanschoren, and R Olson. “Layered TPOT: speeding up tree-based pipeline optimization”. In: *European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases, September 18–22, 2017, Skopje, Macedonia*. CEUR-WS. org. 2017, pp. 49–68.
- [100] Pieter Gijsbers, Erin LeDell, Janek Thomas, Sébastien Poirier, Bernd Bischl, and Joaquin Vanschoren. “An open source AutoML benchmark”. In: *arXiv preprint arXiv:1907.00909* (2019).
- [101] Pieter Gijsbers, Florian Pfisterer, Jan N. van Rijn, Bernd Bischl, and Joaquin Vanschoren. *Meta-Learning for Symbolic Hyperparameter Defaults*. 2021. arXiv: 2106.05767 [stat.ML].
- [102] Pieter Gijsbers, Florian Pfisterer, Jan N. van Rijn, Bernd Bischl, and Joaquin Vanschoren. “Meta-learning for symbolic hyperparameter defaults”. In: *Proceedings of the Genetic and Evolutionary Computation Conference Companion* (July 2021). DOI: 10.1145/3449726.3459532. URL: <http://dx.doi.org/10.1145/3449726.3459532>.
- [103] Pieter Gijsbers and Joaquin Vanschoren. “GAMA: A General Automated Machine Learning Assistant”. In: *Machine Learning and Knowledge Discovery in Databases. Applied Data Science and Demo Track*. Ed. by Yuxiao Dong, Georgiana Ifrim, Dunja Mladenić, Craig Saunders, and Sofie Van Hoecke. Cham: Springer International Publishing, 2021, pp. 560–564. ISBN: 978-3-030-67670-4.
- [104] Pieter Gijsbers and Joaquin Vanschoren. “GAMA: genetic automated machine learning assistant”. In: *Journal of Open Source Software* 4.33 (2019), p. 1132.
- [105] Yolanda Gil, Ke-Thia Yao, Varun Ratnakar, Daniel Garijo, Greg Ver Steeg, Pedro Szekely, Rob Brekelmans, Mayank Kejriwal, Fanghao Luo, and I-Hui Huang. “P4ML: A phased performance-based pipeline planner for automated machine learning”. In: *5th ICML Workshop on Automated Machine Learning (AutoML)*. 2018.
- [106] Alex Goldstein, Adam Kapelner, Justin Bleich, and Emil Pitkin. “Peeking inside the black box: Visualizing statistical learning with plots of individual conditional expectation”. In: *journal of Computational and Graphical Statistics* 24.1 (2015), pp. 44–65.

- [107] Heitor M Gomes, Albert Bifet, Jesse Read, Jean Paul Barddal, Fabrício Enembreck, Bernhard Pfahringer, Geoff Holmes, and Talel Abdesslem. “Adaptive random forests for evolving data stream classification”. In: *Machine Learning* 106.9 (2017), pp. 1469–1495.
- [108] Silvio B Guerra, Ricardo BC Prudêncio, and Teresa B Ludermir. “Predicting the performance of learning algorithms using support vector machines as meta-regressors”. In: *International Conference on Artificial Neural Networks*. Springer. 2008, pp. 523–532.
- [109] Isabelle Guyon, Imad Chaabane, Hugo Jair Escalante, Sergio Escalera, Damir Jajetic, James Robert Lloyd, Núria Macià, Bisakha Ray, Lukasz Romaszko, Michèle Sebag, Alexander R. Statnikov, Sébastien Treguer, and Evelyne Viegas. “A brief Review of the ChaLearn AutoML Challenge: Any-time Any-dataset Learning without Human Intervention”. In: *Proceedings of the 2016 Workshop on Automatic Machine Learning, AutoML 2016, co-located with 33rd International Conference on Machine Learning (ICML 2016), New York City, NY, USA, June 24, 2016*. Ed. by Frank Hutter, Lars Kotthoff, and Joaquin Vanschoren. Vol. 64. JMLR Workshop and Conference Proceedings. JMLR.org, 2016, pp. 21–30. URL: http://proceedings.mlr.press/v64/guyon_review_2016.html.
- [110] Isabelle Guyon, Lisheng Sun-Hosoya, Marc Boullé, Hugo Jair Escalante, Sergio Escalera, Zhengying Liu, Damir Jajetic, Bisakha Ray, Mehreen Saeed, Michèle Sebag, Alexander Statnikov, Wei-Wei Tu, and Evelyne Viegas. “Analysis of the AutoML Challenge Series 2015–2018”. In: *Automated Machine Learning: Methods, Systems, Challenges*. Ed. by Frank Hutter, Lars Kotthoff, and Joaquin Vanschoren. Springer International Publishing, 2019, pp. 177–219.
- [111] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. “The WEKA data mining software”. In: *ACM SIGKDD Explorations Newsletter* 11.1 (Nov. 2009), pp. 10–18. DOI: 10.1145/1656274.1656278.
- [112] Nikolaus Hansen, Anne Auger, Raymond Ros, Olaf Mersman, Tea Tušar, and Dimo Brockhoff. “COCO: A Platform for Comparing Continuous Optimizers in a Black-Box Setting”. In: *Optimization Methods and Software* (2020).
- [113] Moritz Hardt, Eric Price, and Nati Srebro. “Equality of opportunity in supervised learning”. In: *Advances in neural information processing systems* 29 (2016), pp. 3315–3323.

- [114] Charles R. Harris, K. Jarrod Millman, Stéfan J van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. “Array programming with NumPy”. In: *Nature* 585 (2020), pp. 357–362. DOI: 10.1038/s41586-020-2649-2.
- [115] Sepp Hochreiter, A. Steven Younger, and Peter R. Conwell. “Learning to Learn Using Gradient Descent”. In: *Artificial Neural Networks — ICANN 2001*. Ed. by Georg Dorffner, Horst Bischof, and Kurt Hornik. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 87–94. ISBN: 978-3-540-44668-2.
- [116] Jeroen van Hoof and Joaquin Vanschoren. “Hyperboost: Hyperparameter Optimization by Gradient Boosting surrogate models”. In: *arXiv preprint arXiv:2101.02289* (2021).
- [117] M. Hutson. “Missing data hinder replication of artificial intelligence studies”. In: *Science News* (2018). URL: <https://www.science.org/content/article/missing-data-hinder-replication-artificial-intelligence-studies>.
- [118] Frank Hutter, Holger Hoos, and Kevin Leyton-Brown. “An Efficient Approach for Assessing Hyperparameter Importance”. In: *Proceedings of the 31st International Conference on Machine Learning*. Ed. by Eric P. Xing and Tony Jebara. Vol. 32. Proceedings of Machine Learning Research 1. Beijing, China: PMLR, June 2014, pp. 754–762. URL: <https://proceedings.mlr.press/v32/hutter14.html>.
- [119] Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. “Sequential Model-Based Optimization for General Algorithm Configuration”. In: *Learning and Intelligent Optimization*. Ed. by Carlos A. Coello Coello. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 507–523. ISBN: 978-3-642-25566-3.
- [120] Frank Hutter, Lars Kotthoff, and Joaquin Vanschoren. “Automated machine learning: methods, systems, challenges”. In: Springer Nature, 2019. Chap. Preface.
- [121] Frank Hutter, Lin Xu, Holger H Hoos, and Kevin Leyton-Brown. “Algorithm runtime prediction: Methods & evaluation”. In: *Artificial Intelligence* 206 (2014), pp. 79–111.

- [122] Carl Hvarfner, Danny Stoll, Artur Souza, Luigi Nardi, Marius Lindauer, and Frank Hutter. “ π BO: Augmenting Acquisition Functions with User Beliefs for Bayesian Optimization”. In: *International Conference on Learning Representations*. 2021.
- [123] Kevin Jamieson and Ameet Talwalkar. “Non-stochastic best arm identification and hyperparameter optimization”. In: *Artificial Intelligence and Statistics*. PMLR. 2016, pp. 240–248.
- [124] Haifeng Jin, Qingquan Song, and Xia Hu. “Auto-keras: An efficient neural architecture search system”. In: *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 2019, pp. 1946–1956.
- [125] Hadi S Jomaa, Lars Schmidt-Thieme, and Josif Grabocka. “Dataset2vec: Learning dataset meta-features”. In: *Data Mining and Knowledge Discovery* 35.3 (2021), pp. 964–985.
- [126] Donald R Jones. “A taxonomy of global optimization methods based on response surfaces”. In: *Journal of global optimization* 21.4 (2001), pp. 345–383.
- [127] Donald R Jones, Matthias Schonlau, and William J Welch. “Efficient global optimization of expensive black-box functions”. In: *Journal of Global optimization* 13.4 (1998), pp. 455–492.
- [128] Zohar Karnin, Tomer Koren, and Oren Somekh. “Almost optimal exploration in multi-armed bandits”. In: *International Conference on Machine Learning*. PMLR. 2013, pp. 1238–1246.
- [129] Marios Kefalas, Mitra Baratchi, Asteris Apostolidis, Dirk van den Herik, and Thomas Bäck. “Automated Machine Learning for Remaining Useful Life Estimation of Aircraft Engines”. In: *2021 IEEE International Conference on Prognostics and Health Management (ICPHM)*. 2021, pp. 1–9. DOI: 10.1109/ICPHM51084.2021.9486549.
- [130] E. Keogh and S. Kasetty. “On the need for time series data mining benchmarks: A survey and empirical demonstration”. In: *Data Mining and Knowledge Discovery* 7.4 (2003), pp. 349–371.
- [131] Douwe Kiela, Max Bartolo, Yixin Nie, Divyansh Kaushik, Atticus Geiger, Zhengxuan Wu, Bertie Vidgen, Grusha Prasad, Amanpreet Singh, Pratik Ringshia, Zhiyi Ma, Tristan Thrush, Sebastian Riedel, Zeerak Waseem, Pontus Stenetorp, Robin Jia, Mohit Bansal, Christopher Potts, and Adina Williams. “Dynabench: Rethinking Benchmarking in NLP”. In: *Pro-*

- ceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Association for Computational Linguistics, 2021, pp. 4110–4124.
- [132] Jungtaek Kim, Saehoon Kim, and Seungjin Choi. “Learning to warm-start Bayesian hyperparameter optimization”. In: *arXiv preprint arXiv:1710.06219* (2017).
- [133] SW Kim, Mira Jeong, and Byoung Chul Ko. “Is the surrogate model interpretable?”. In: *Proc. Adv. Neural Inf. Process. Syst. Workshops (NIPSW)*. 2020, pp. 1–5.
- [134] Jon Kleinberg, Sendhil Mullainathan, and Manish Raghavan. “Inherent trade-offs in the fair determination of risk scores”. In: *arXiv preprint arXiv:1609.05807* (2016).
- [135] Levente Kocsis and Csaba Szepesvári. “Bandit based monte-carlo planning”. In: *European conference on machine learning*. Springer. 2006, pp. 282–293.
- [136] Matthias König, Holger H Hoos, and Jan 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.
- [137] Igor Kononenko. “Machine learning for medical diagnosis: history, state of the art and perspective”. In: *Artificial Intelligence in Medicine* 23.1 (2001), pp. 89–109. ISSN: 0933-3657. DOI: [https://doi.org/10.1016/S0933-3657\(01\)00077-X](https://doi.org/10.1016/S0933-3657(01)00077-X). URL: <https://www.sciencedirect.com/science/article/pii/S093336570100077X>.
- [138] Helena Kotthaus, Ingo Korb, Michel Lang, Bernd Bischl, Jörg Rahnenführer, and Peter Marwedel. “Runtime and memory consumption analyses for machine learning R programs”. In: *Journal of Statistical Computation and Simulation* 85.1 (2015), pp. 14–29. DOI: 10.1080/00949655.2014.925192.
- [139] J. Koza, M. A. Keane, and J. P. Rice. “Performance improvement of machine learning via automatic discovery of facilitating functions as applied to a problem of symbolic system identification”. In: *IEEE International Conference on Neural Networks*. 1993, 191–198 vol.1.
- [140] John R Koza and John R Koza. *Genetic programming: on the programming of computers by means of natural selection*. Vol. 1. MIT press, 1992.

- [141] Tomas Kren, Martin Pilat, and Roman Neruda. “Evolving Workflow Graphs Using Typed Genetic Programming”. In: *2015 IEEE Symposium Series on Computational Intelligence*. 2015, pp. 1407–1414. DOI: 10.1109/SSCI.2015.200.
- [142] Tomáš Křen, Martin Pilát, and Roman Neruda. “Multi-objective evolution of machine learning workflows”. In: *2017 IEEE Symposium Series on Computational Intelligence (SSCI)*. IEEE. 2017, pp. 1–8.
- [143] D. Kühn, P. Probst, J. Thomas, and B. Bischl. “Automatic Exploration of Machine Learning Experiments on OpenML”. In: *arXiv preprint arXiv:1806.10961* (2018).
- [144] Doron Laadan, Roman Vainshtein, Yarden Curiel, Gilad Katz, and Lior Rokach. “MetaTPOT: enhancing a tree-based pipeline optimization tool using meta-learning”. In: *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*. 2020, pp. 2097–2100.
- [145] Michel Lang, Martin Binder, Jakob Richter, Patrick Schratz, Florian Pfisterer, Stefan Coors, Quay Au, Giuseppe Casalicchio, Lars Kotthoff, and Bernd Bischl. “mlr3: A modern object-oriented machine learning framework in R”. In: *Journal of Open Source Software* 4.44 (2019), p. 1903.
- [146] N. Lavesson and P. Davidsson. “Quantifying the impact of learning algorithm parameter tuning”. In: *AAAI*. Vol. 6. 2006, pp. 395–400.
- [147] Trang T Le, Weixuan Fu, and Jason H Moore. “Scaling tree-based automated machine learning to biomedical big data with a feature set selector”. In: *Bioinformatics* 36.1 (2020), pp. 250–256.
- [148] Erin LeDell and Sebastien Poirier. “H2o automl: Scalable automatic machine learning”. In: *Proceedings of the AutoML Workshop at ICML*. Vol. 2020. 2020.
- [149] Rui Leite, Pavel Brazdil, and Joaquin Vanschoren. “Selecting classification algorithms with active testing”. In: *International workshop on machine learning and data mining in pattern recognition*. Springer. 2012, pp. 117–131.
- [150] Yu-Feng Li, Hai Wang, Tong Wei, and Wei-Wei Tu. “Towards automated semi-supervised learning”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 33. 01. 2019, pp. 4237–4244.

- [151] Liam Li, Kevin Jamieson, Afshin Rostamizadeh, Ekaterina Gonina, Moritz Hardt, Benjamin Recht, and Ameet Talwalkar. *A System for Massively Parallel Hyperparameter Tuning*. 2020. arXiv: 1810.05934 [cs.LG].
- [152] Liam Li and Ameet Talwalkar. “Random Search and Reproducibility for Neural Architecture Search”. In: *CoRR* abs/1902.07638 (2019). arXiv: 1902.07638. URL: <http://arxiv.org/abs/1902.07638>.
- [153] Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. “Hyperband: A Novel Bandit-Based Approach to Hyperparameter Optimization”. In: *Journal of Machine Learning Research* 18 (2018) 1-52 (Mar. 2016). arXiv: 1603.06560 [cs.LG].
- [154] M. Lindauer and F. Hutter. “Warmstarting of Model-based Algorithm Configuration”. In: *Proceedings of the AAAI conference*. Feb. 2018, pp. 1355–1362.
- [155] Yue Liu, Shuang Li, and Wenjie Tian. “AutoCluster: Meta-learning Based Ensemble Method for Automated Unsupervised Clustering.” In: *PAKDD* (3). Springer. 2021, pp. 246–258.
- [156] Manuel López-Ibáñez, Jérémie Dubois-Lacoste, Leslie Pérez Cáceres, Mauro Birattari, and Thomas Stützle. “The irace package: Iterated racing for automatic algorithm configuration”. In: *Operations Research Perspectives* 3 (2016), pp. 43–58.
- [157] Y. A. Malkov and D. A. Yashunin. “Efficient and Robust Approximate Nearest Neighbor Search Using Hierarchical Navigable Small World Graphs”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 42.4 (2020), pp. 824–836.
- [158] Yury A. Malkov and D. A. Yashunin. “Efficient and robust approximate nearest neighbor search using Hierarchical Navigable Small World graphs”. In: *CoRR* abs/1603.09320 (2016). arXiv: 1603.09320. URL: <http://arxiv.org/abs/1603.09320>.
- [159] Rafael G Mantovani, Andre LD Rossi, Edesio Alcobaca, Joaquin Vanschoren, and Andre CPLF de Carvalho. “A meta-learning recommender system for hyperparameter tuning: Predicting when tuning improves SVM classifiers”. In: *Information Sciences* 501 (2019), pp. 193–221.

- [160] Rafael Gomes Mantovani, André Luis Debiasio Rossi, Edesio Alcobaça, Jadson Castro Gertrudes, Sylvio Barbon Junior, and André Carlos Ponce de Leon Ferreira de Carvalho. *Rethinking Default Values: a Low Cost and Efficient Strategy to Define Hyperparameters*. 2020. arXiv: 2008.00025 [cs.LG].
- [161] O. Maron and A.W. Moore. “The Racing Algorithm: Model Selection for Lazy Learners”. In: *Artificial Intelligence Review* 11.1-5 (1997). Cited by: 142, pp. 193–225. DOI: 10.1007/978-94-017-2053-3_8. URL: https://www.scopus.com/inward/record.uri?eid=2-s2.0-0031069121&doi=10.1007%2f978-94-017-2053-3_8&partnerID=40&md5=8906d29c0cc61ac2a77deb467618f72d.
- [162] Luke Metz, Niru Maheswaranathan, Ruoxi Sun, C. Freeman, Ben Poole, and Jascha Sohl-Dickstein. “Using a thousand optimization tasks to learn hyperparameter search strategies”. In: (Feb. 2020).
- [163] David Meyer, Evgenia Dimitriadou, Kurt Hornik, Andreas Weingessel, and Friedrich Leisch. *e1071: Misc Functions of the Department of Statistics, Probability Theory Group (Formerly: E1071), TU Wien*. R package version 1.7-3. 2019. URL: <https://CRAN.R-project.org/package=e1071>.
- [164] Daniele Micci-Barreca. “A Preprocessing Scheme for High-Cardinality Categorical Attributes in Classification and Prediction Problems”. In: *SIGKDD Explor. Newsl.* 3.1 (July 2001), pp. 27–32. ISSN: 1931-0145. DOI: 10.1145/507533.507538. URL: <https://doi.org/10.1145/507533.507538>.
- [165] Donald Michie, David J Spiegelhalter, and Charles C Taylor. “Machine learning, neural and statistical classification”. In: (1994).
- [166] Rory Mitchell, Eibe Frank, and Geoffrey Holmes. “An Empirical Study of Moment Estimators for Quantile Approximation”. In: *ACM Transactions on Database Systems* 46.1 (2021).
- [167] Tom M. Mitchell. *Machine Learning*. New York: McGraw-Hill, 1997. ISBN: 978-0-07-042807-2.
- [168] Felix Mohr and Marcel Wever. “Replacing the Ex-Def Baseline in AutoML by Naive AutoML”. In: *8th ICML Workshop on Automated Machine Learning (AutoML)*. 2021.
- [169] Felix Mohr, Marcel Wever, and Eyke Hüllermeier. “ML-Plan: Automated machine learning via hierarchical planning”. In: *Machine Learning* 107.8-10 (July 2018), pp. 1495–1515. DOI: 10.1007/s10994-018-5735-z.

- [170] Felix Mohr, Marcel Wever, Alexander Tornede, and Eyke Hullermeier. “Predicting Machine Learning Pipeline Runtimes in the Context of Automated Machine Learning”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2021).
- [171] Christoph Molnar, Giuseppe Casalicchio, and Bernd Bischl. “Interpretable machine learning—a brief history, state-of-the-art and challenges”. In: *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer. 2020, pp. 417–431.
- [172] Jacob Montiel, Max Halford, Saulo Martiello Mastelini, Geoffrey Bolmier, Raphael Sourty, Robin Vaysse, Adil Zouitine, Heitor Murilo Gomes, Jesse Read, Talel Abdessalem, and Albert Bifet. *River: machine learning for streaming data in Python*. 2020. arXiv: 2012.04740 [cs.LG].
- [173] Julia Moosbauer, Julia Herbinger, Giuseppe Casalicchio, Marius Lindauer, and Bernd Bischl. “Explaining Hyperparameter Optimization via Partial Dependence Plots”. In: *Advances in Neural Information Processing Systems* 34 (2021).
- [174] Avaniika Narayan, Piero Molino, Karan Goel, Willie Neiswanger, and Christopher Re. “Personalized Benchmarking with the Ludwig Benchmarking Toolkit”. In: *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*. 2021.
- [175] M. O’Neill and C. Ryan. “Grammatical evolution”. In: *IEEE Transactions on Evolutionary Computation* 5.4 (Aug. 2001), pp. 349–358. ISSN: 1089-778X.
- [176] ChangYong Oh, Efstratios Gavves, and Max Welling. “Bock: Bayesian optimization with cylindrical kernels”. In: *International Conference on Machine Learning*. PMLR. 2018, pp. 3868–3877.
- [177] Ivan Olier, Nouredin Sadawi, G Richard Bickerton, Joaquin Vanschoren, Crina Grosan, Larisa Soldatova, and Ross D King. “Meta-QSAR: a large-scale application of meta-learning to drug design and discovery”. In: *Machine Learning* 107.1 (2018), pp. 285–311.
- [178] 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”. In: *BioData Mining* 10.36 (2017).
- [179] Randal S. Olson, Nathan Bartley, Ryan J. Urbanowicz, and Jason H. Moore. *Evaluation of a Tree-based Pipeline Optimization Tool for Automating Data Science*. 2016. DOI: 10.1145/2908812.2908918.

- [180] Florian Pargent, Florian Pfisterer, Janek Thomas, and Bernd Bischl. *Regularized target encoding outperforms traditional methods in supervised machine learning with high cardinality features*. 2021. arXiv: 2104.00629 [stat.ML].
- [181] Laurent Parmentier, Olivier Nicol, Laetitia Jourdan, and Marie-Eleonore 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)*. IEEE. 2019, pp. 471–478.
- [182] Preston Parry. *auto_ml*. https://github.com/ClimbsRocks/auto_ml. 2018.
- [183] T. Pedersen. “Empiricism is not a matter of faith”. In: *Computational Linguistics* 34 (2008), pp. 465–470.
- [184] Fabian Pedregosa. “Scikit-learn: Machine learning in Python”. In: *the Journal of machine Learning research* 12 (2011), pp. 2825–2830.
- [185] Yonghong Peng, Peter A Flach, Carlos Soares, and Pavel Brazdil. “Improved dataset characterisation for meta-learning”. In: *International Conference on Discovery Science*. Springer. 2002, pp. 141–152.
- [186] V. Perrone, R. Jenatton, M. Seeger, and C. Archambeau. “Scalable Hyperparameter Transfer Learning”. In: *Proc. of NeurIPS’18*. 2018.
- [187] Bernhard Pfahringer, Hilan Bensusan, and Christophe G Giraud-Carrier. “Meta-Learning by Landmarking Various Learning Algorithms.” In: *ICML*. 2000, pp. 743–750.
- [188] Florian Pfisterer, Jan N van Rijn, Philipp Probst, Andreas C Mueller, and Bernd Bischl. “Learning multiple defaults for machine learning algorithms”. In: *Proceedings of the Genetic and Evolutionary Computation Conference Companion*. 2021, pp. 241–242.
- [189] Hieu Pham, Melody Guan, Barret Zoph, Quoc Le, and Jeff Dean. “Efficient neural architecture search via parameters sharing”. In: *International Conference on Machine Learning*. PMLR. 2018, pp. 4095–4104.
- [190] Martin Pilát, Tomáš Křen, and Roman Neruda. “Asynchronous Evolution of Data Mining Workflow Schemes by Strongly Typed Genetic Programming”. In: *2016 IEEE 28th International Conference on Tools with Artificial Intelligence (ICTAI)*. 2016, pp. 577–584. DOI: 10.1109/ICTAI.2016.0094.

- [191] Bruno Almeida Pimentel and Andre CPLF De Carvalho. “A new data characterization for selecting clustering algorithms using meta-learning”. In: *Information Sciences* 477 (2019), pp. 203–219.
- [192] Aleksandra Płońska and Piotr Płoński. *MLJAR: State-of-the-art Automated Machine Learning Framework for Tabular Data. Version 0.10.3*. Lapy, Poland, 2021. URL: <https://github.com/mljar/mljar-supervised>.
- [193] Yannis Poulakis, Christos Doulkeridis, and Dimosthenis Kyriazis. “AutoClust: A Framework for Automated Clustering based on Cluster Validity Indices”. In: *2020 IEEE International Conference on Data Mining (ICDM)*. IEEE, 2020, pp. 1220–1225.
- [194] Kenneth Price, Rainer M Storn, and Jouni A Lampinen. *Differential evolution: a practical approach to global optimization*. Springer Science & Business Media, 2006.
- [195] Philipp Probst and Anne-Laure Boulesteix. “To Tune or Not to Tune the Number of Trees in Random Forest”. In: *J. Mach. Learn. Res.* 18.1 (Jan. 2017), pp. 6673–6690. ISSN: 1532-4435.
- [196] Philipp Probst, Anne-Laure Boulesteix, and Bernd Bischl. “Tunability: Importance of Hyperparameters of Machine Learning Algorithms”. In: *Journal of Machine Learning Research* 20.53 (2019), pp. 1–32. ISSN: 1533-7928. URL: <http://jmlr.org/papers/v20/18-444.html> (visited on 10/21/2021).
- [197] Philipp Probst, Marvin N Wright, and Anne-Laure Boulesteix. “Hyperparameters and tuning strategies for random forest”. In: *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 9.3 (2019), e1301.
- [198] Herilalaina Rakotoarison, Louisot Milijaona, Andry Rasoanaivo, Michèle Sebag, and Marc Schoenauer. “Learning Meta-features for AutoML”. In: *International Conference on Learning Representations*. 2021.
- [199] Herilalaina Rakotoarison, Marc Schoenauer, and Michèle Sebag. “Automated Machine Learning with Monte-Carlo Tree Search”. In: *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*. International Joint Conferences on Artificial Intelligence Organization, July 2019, pp. 3296–3303. DOI: 10.24963/ijcai.2019/457. URL: <https://doi.org/10.24963/ijcai.2019/457>.
- [200] Carl Edward Rasmussen. “Gaussian processes in machine learning”. In: *Summer school on machine learning*. Springer, 2003, pp. 63–71.

- [201] Benjamin Recht, Rebecca Roelofs, Ludwig Schmidt, and Vaishaal Shankar. “Do ImageNet Classifiers Generalize to ImageNet?” In: (Feb. 2019). arXiv: 1902.10811 [cs.CV].
- [202] Payam Refaeilzadeh, Lei Tang, and Huan Liu. “Cross-validation.” In: *Encyclopedia of database systems* 5 (2009), pp. 532–538.
- [203] Matthias Reif, Faisal Shafait, and Andreas Dengel. “Meta-learning for evolutionary parameter optimization of classifiers”. In: *Machine learning* 87.3 (2012), pp. 357–380.
- [204] Matthias Reif, Faisal Shafait, Markus Goldstein, Thomas Breuel, and Andreas Dengel. “Automatic classifier selection for non-experts”. In: *Pattern Analysis and Applications* 17.1 (2014), pp. 83–96.
- [205] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. “” Why should i trust you?” Explaining the predictions of any classifier”. In: *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*. 2016, pp. 1135–1144.
- [206] Rich Caruana. *Research Opportunities in AutoML*. URL: <https://indico.ijclab.in2p3.fr/event/2914/contributions/6481/attachments/6048/7173/CaruanaAutoMLWorkshopICML2015rev4.pdf>. July 2015.
- [207] J. N. van Rijn. “Massively collaborative machine learning”. PhD thesis. Leiden University, 2016.
- [208] Jan N van Rijn, Geoffrey Holmes, Bernhard Pfahringer, and Joaquin Vanschoren. “Having a blast: Meta-learning and heterogeneous ensembles for data streams”. In: *2015 IEEE International Conference on Data Mining*. IEEE. 2015, pp. 1003–1008.
- [209] Jan N. van Rijn, Florian Pfisterer, Janek Thomas, Andreas Mueller, Bernd Bischl, and Joaquin Vanschoren. “Meta learning for defaults: symbolic defaults”. In: *Workshop on Meta-Learning @ NeurIPS2018*. 2018.
- [210] Joseph D Romano, Trang T Le, Weixuan Fu, and Jason H Moore. “TPOT-NN: augmenting tree-based automated machine learning with neural network estimators”. In: *Genetic Programming and Evolvable Artificial Machines* (2021), pp. 1–21.
- [211] Alex GC de Sá, Alex A Freitas, and Gisele L Pappa. “Automated selection and configuration of multi-label classification algorithms with grammar-based genetic programming”. In: *International Conference on Parallel Problem Solving from Nature*. Springer. 2018, pp. 308–320.

- [212] Alex GC de Sá, Walter José GS Pinto, Luiz Otavio VB Oliveira, and Gisele L Pappa. “RECIPE: a grammar-based framework for automatically evolving classification pipelines”. In: *European Conference on Genetic Programming*. Springer. 2017, pp. 246–261.
- [213] Shubhra Kanti Karmaker Santu, Md. Mahadi Hassan, Micah J. Smith, Lei Xu, Chengxiang Zhai, and Kalyan Veeramachaneni. “AutoML to Date and Beyond: Challenges and Opportunities”. In: *ACM Comput. Surv.* 54.8 (Oct. 2021). ISSN: 0360-0300. DOI: 10.1145/3470918. URL: <https://doi.org/10.1145/3470918>.
- [214] Robin Schmucker, Michele Donini, Valerio Perrone, Muhammad Bilal Zafar, and Cédric Archambeau. “Multi-Objective Multi-Fidelity Hyperparameter Optimization with Application to Fairness”. In: *NeurIPS Workshop on Meta-Learning*. Vol. 2. 2020.
- [215] Robin Schmucker, Michele Donini, Muhammad Bilal Zafar, David Salinas, and Cédric Archambeau. *Multi-objective Asynchronous Successive Halving*. 2021. arXiv: 2106.12639 [stat.ML].
- [216] 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.
- [217] scikit-learn developers scikit-learn. *sklearn.svm.SVC*. URL: <https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html>.
- [218] Eric O. Scott and Kenneth A. De Jong. “Understanding Simple Asynchronous Evolutionary Algorithms”. In: *Proceedings of the 2015 ACM Conference on Foundations of Genetic Algorithms XIII*. FOGA ’15. Aberystwyth, United Kingdom: Association for Computing Machinery, 2015, pp. 85–98. ISBN: 9781450334341. DOI: 10.1145/2725494.2725509. URL: <https://doi.org/10.1145/2725494.2725509>.
- [219] D Sculley, J Snoek, A Wiltschko, and A Rahimi. *Winner’s curse? On pace, progress, and empirical rigor*. en. Feb. 2018. URL: <https://openreview.net/forum?id=rJWF0Fywfwf> (visited on 11/30/2021).
- [220] Alex Serban, Koen van der Blom, Holger Hoos, and Joost Visser. “Adoption and effects of software engineering best practices in machine learning”. In: *Proceedings of the 14th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*. 2020, pp. 1–12.

- [221] Ali Sharif Razavian, Hossein Azizpour, Josephine Sullivan, and Stefan Carlsson. “CNN features off-the-shelf: an astounding baseline for recognition”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*. 2014, pp. 806–813.
- [222] Xingjian Shi, Jonas Mueller, Nick Erickson, Mu Li, and Alexander J Smola. “Benchmarking Multimodal AutoML for Tabular Data with Text Fields”. In: *arXiv preprint arXiv:2111.02705* (2021).
- [223] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. “Practical bayesian optimization of machine learning algorithms”. In: *Advances in neural information processing systems* 25 (2012).
- [224] Jasper Snoek, Oren Rippel, Kevin Swersky, Ryan Kiros, Nadathur Satish, Narayanan Sundaram, Mostofa Patwary, Mr Prabhat, and Ryan Adams. “Scalable bayesian optimization using deep neural networks”. In: *International conference on machine learning*. PMLR. 2015, pp. 2171–2180.
- [225] C. Soares, P. Brazdil, and P. Kuba. “A Meta-Learning Method to Select the Kernel Width in Support Vector Regression”. In: *Mach. Learn.* 54 (2004), pp. 195–209.
- [226] Carlos Soares and Pavel B Brazdil. “Zoomed ranking: Selection of classification algorithms based on relevant performance information”. In: *European conference on principles of data mining and knowledge discovery*. Springer. 2000, pp. 126–135.
- [227] Andrew Sohn, Randal S Olson, and Jason H Moore. “Toward the automated analysis of complex diseases in genome-wide association studies using genetic programming”. In: *Proceedings of the genetic and evolutionary computation conference*. 2017, pp. 489–496.
- [228] Marcilio C. P. de Souto, Ricardo B. C. Prudencio, Rodrigo G. F. Soares, Daniel S. A. de Araujo, Ivan G. Costa, Teresa B. Ludermir, and Alexander Schliep. “Ranking and selecting clustering algorithms using a meta-learning approach”. In: *2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence)*. 2008, pp. 3729–3735. doi: 10.1109/IJCNN.2008.4634333.
- [229] Artur Souza, Luigi Nardi, Leonardo B Oliveira, Kunle Olukotun, Marius Lindauer, and Frank Hutter. “Bayesian Optimization with a Prior for the Optimum”. In: *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer. 2021, pp. 265–296.

- [230] Alexander Statnikov, Constantin F Aliferis, and Ioannis Tsamardinos. “Methods for multi-category cancer diagnosis from gene expression data: a comprehensive evaluation to inform decision support system development”. In: *MEDINFO 2004*. IOS Press. 2004, pp. 813–817.
- [231] Alexander Statnikov, Ioannis Tsamardinos, Yerbolat Dosbayev, and Constantin F Aliferis. “GEMS: a system for automated cancer diagnosis and biomarker discovery from microarray gene expression data”. In: *International journal of medical informatics* 74.7-8 (2005), pp. 491–503.
- [232] Douglas Steinley. “Properties of the Hubert-Arable Adjusted Rand Index.” In: *Psychological methods* 9.3 (2004), p. 386.
- [233] 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.
- [234] Carolin Strobl, Florian Wickelmaier, and Achim Zeileis. “Accounting for Individual Differences in Bradley-Terry Models by Means of Recursive Partitioning”. In: *Journal of Educational and Behavioral Statistics* 36.2 (2011), pp. 135–153. DOI: 10.3102/1076998609359791. eprint: <https://doi.org/10.3102/1076998609359791>. URL: <https://doi.org/10.3102/1076998609359791>.
- [235] The pandas development team. *pandas-dev/pandas: Pandas*. Version latest. Feb. 2020. DOI: 10.5281/zenodo.3509134. URL: <https://doi.org/10.5281/zenodo.3509134>.
- [236] Terry Therneau and Beth Atkinson. *rpart: Recursive Partitioning and Regression Trees*. R package version 4.1-13. 2018. URL: <https://CRAN.R-project.org/package=rpart>.
- [237] Janek Thomas, Stefan Coors, and Bernd Bischl. “Automatic Gradient Boosting”. In: (July 2018). arXiv: 1807.03873 [stat.ML].
- [238] Chris Thornton, Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. “Auto-WEKA: combined selection and hyperparameter optimization of classification algorithms”. In: *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*. KDD ’13. Chicago, Illinois, USA: Association for Computing Machinery, Aug. 2013, pp. 847–855. ISBN: 9781450321747. DOI: 10.1145/2487575.2487629. URL: <https://doi.org/10.1145/2487575.2487629>.

- [239] Ryan J Tibshirani and Robert Tibshirani. “A bias correction for the minimum error rate in cross-validation”. In: *The Annals of Applied Statistics* 3.2 (2009), pp. 822–829.
- [240] Erico Tjoa and Cuntai Guan. “A Survey on Explainable Artificial Intelligence (XAI): Toward Medical XAI”. In: *IEEE Transactions on Neural Networks and Learning Systems* 32.11 (2021), pp. 4793–4813. DOI: 10.1109/TNNLS.2020.3027314.
- [241] Eric J Topol. “High-performance medicine: the convergence of human and artificial intelligence”. In: *Nature medicine* 25.1 (2019), pp. 44–56.
- [242] Tanja Tornede, Alexander Tornede, Marcel Wever, and Eyke Hüllermeier. “Coevolution of Remaining Useful Lifetime Estimation Pipelines for Automated Predictive Maintenance”. In: *Proceedings of the Genetic and Evolutionary Computation Conference*. New York, NY, USA: Association for Computing Machinery, 2021, pp. 368–376. ISBN: 9781450383509. URL: <https://doi.org/10.1145/3449639.3459395>.
- [243] Tanja Tornede, Alexander Tornede, Marcel Wever, Felix Mohr, and Eyke Hüllermeier. “Automl for predictive maintenance: One tool to rul them all”. In: *IoT Streams for Data-Driven Predictive Maintenance and IoT, Edge, and Mobile for Embedded Machine Learning*. Springer, 2020, pp. 106–118.
- [244] Anh Truong, Austin Walters, Jeremy Goodsitt, Keegan E. Hines, C. Bayan Bruss, and Reza Farivar. “Towards Automated Machine Learning: Evaluation and Comparison of AutoML Approaches and Tools”. In: *31st IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2019, Portland, OR, USA, November 4-6, 2019*. IEEE, 2019, pp. 1471–1479. DOI: 10.1109/ICTAI.2019.00209.
- [245] Ioannis Tsamardinos, Paulos Charonyktakis, Georgios Papoutsoglou, Giorgos Borboudakis, Kleanthi Lakiotaki, Jean Claude Zenklusen, Hartmut Juhl, Ekaterini Chatzaki, and Vincenzo Lagani. “Just Add Data: Automated Predictive Modeling for Knowledge Discovery and Feature Selection”. In: *(to appear) npj Precision Oncology* (2022).
- [246] Ioannis Tsamardinos, Elissavet Greasidou, and Giorgos Borboudakis. “Bootstrapping the out-of-sample predictions for efficient and accurate cross-validation”. In: *Machine Learning* 107.12 (2018), pp. 1895–1922.

- [247] Ioannis Tsamardinos, Amin Rakhshani, and Vincenzo Lagani. “Performance-estimation properties of cross-validation-based protocols with simultaneous hyper-parameter optimization”. In: *International Journal on Artificial Intelligence Tools* 24.05 (2015), p. 1540023.
- [248] G. Tsoumakas, E. Spyromitros-Xioufis, J. Vilcek, and I. Vlahavas. “MULAN: A Java Library for Multi-Label Learning”. In: *Journal of Machine Learning Research* (July 2011), pp. 2411–2414.
- [249] Anton Vakhrushev, Alexander Ryzhkov, Maxim Savchenko, Dmitry Simakov, Rinchin Damdinov, and Alexander Tuzhilin. “LightAutoML: AutoML Solution for a Large Financial Services Ecosystem”. In: *arXiv:2109.01528 [cs, stat]* (Sept. 2021). arXiv: 2109.01528. URL: <http://arxiv.org/abs/2109.01528> (visited on 10/18/2021).
- [250] Roberto Valerio and Ricardo Vilalta. “Kernel selection in support vector machines using gram-matrix properties”. In: *NIPS Workshop on Modern Nonparametrics: Automating the Learning Pipeline*. Vol. 14. 2014.
- [251] John Joseph Valletta, Colin Torney, Michael Kings, Alex Thornton, and Joah Madden. “Applications of machine learning in animal behaviour studies”. In: *Animal Behaviour* 124 (2017), pp. 203–220.
- [252] Koen Van der Blom, Alex Serban, Holger Hoos, and Joost Visser. “AutoML Adoption in ML Software”. In: *8th ICML Workshop on Automated Machine Learning (AutoML)*. 2021.
- [253] Mark J Van der Laan, Eric C Polley, and Alan E Hubbard. “Super learner”. In: *Statistical applications in genetics and molecular biology* 6.1 (2007).
- [254] Tony Van Gestel, Johan AK Suykens, Bart Baesens, Stijn Viaene, Jan Vanthienen, Guido Dedene, Bart De Moor, and Joos Vandewalle. “Benchmarking least squares support vector machine classifiers”. In: *Machine learning* 54.1 (2004), pp. 5–32.
- [255] Jan N Van Rijn and Frank Hutter. “Hyperparameter importance across datasets”. In: *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 2018, pp. 2367–2376.
- [256] Elia Van Wolputte and Hendrik Blockeel. “Missing Value Imputation with MERCS: A Faster Alternative to MissForest”. In: *Discovery Science - 23rd International Conference*. Vol. 12323. Lecture Notes in Computer Science. Springer. 2020, pp. 502–516.

- [257] Joaquin Vanschoren. “Meta-learning”. In: *Automated Machine Learning*. Springer, Cham, 2019, pp. 35–61.
- [258] Joaquin Vanschoren, Jan N Van Rijn, Bernd Bischl, and Luis Torgo. “OpenML: networked science in machine learning”. In: *ACM SIGKDD Explorations Newsletter* 15.2 (2014), pp. 49–60.
- [259] Sudhir Varma and Richard Simon. “Bias in error estimation when using cross-validation for model selection”. In: *BMC bioinformatics* 7.1 (2006), pp. 1–8.
- [260] Alfredo Vellido. “The importance of interpretability and visualization in machine learning for applications in medicine and health care”. In: *Neural computing and applications* 32.24 (2020), pp. 18069–18083.
- [261] Carlos Vieira, Adelson de Araújo, José E Andrade, and Leonardo CT Bezerra. “iSklearn: Automated Machine Learning with irace”. In: *2021 IEEE Congress on Evolutionary Computation (CEC)*. IEEE. 2021, pp. 2354–2361.
- [262] Nguyen Xuan Vinh, Julien Epps, and James Bailey. “Information Theoretic Measures for Clusterings Comparison: Variants, Properties, Normalization and Correction for Chance”. In: *J. Mach. Learn. Res.* 11 (Dec. 2010), pp. 2837–2854. ISSN: 1532-4435.
- [263] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. “SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python”. In: *Nature Methods* 17 (2020), pp. 261–272. DOI: 10.1038/s41592-019-0686-2.
- [264] Kiri Wagstaff. “Machine learning that matters”. In: *arXiv preprint arXiv:1206.4656* (2012).
- [265] Chi Wang, Qingyun Wu, Markus Weimer, and Erkang Zhu. “FLAML: a fast and lightweight AutoML Library”. In: *Proceedings of Machine Learning and Systems* 3 (2021), pp. 434–447.

- [266] Dakuo Wang, Josh Andres, Justin D. Weisz, Erick Oduor, and Casey Dugan. “AutoDS: Towards Human-Centered Automation of Data Science”. In: *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*. CHI ’21. Yokohama, Japan: Association for Computing Machinery, 2021. ISBN: 9781450380966. DOI: 10.1145/3411764.3445526. URL: <https://doi.org/10.1145/3411764.3445526>.
- [267] Hilde JP Weerts, Andreas C Mueller, and Joaquin Vanschoren. “Importance of tuning hyperparameters of machine learning algorithms”. In: *arXiv preprint arXiv:2007.07588* (2020).
- [268] Marcel Wever, Felix Mohr, and Eyke Hüllermeier. *Automated Multi-Label Classification based on ML-Plan*. 2018. arXiv: 1811.04060 [cs.LG].
- [269] Fela Winkelmolen, Nikita Ivkin, H. Furkan Bozkurt, and Zohar Karnin. *Practical and sample efficient zero-shot HPO*. 2020. arXiv: 2007.13382 [stat.ML].
- [270] Martin Wistuba, Nicolas Schilling, and Lars Schmidt-Thieme. “Hyperparameter search space pruning—a new component for sequential model-based hyperparameter optimization”. In: *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer. 2015, pp. 104–119.
- [271] Martin Wistuba, Nicolas Schilling, and Lars Schmidt-Thieme. “Learning hyperparameter optimization initializations”. In: *Data Science and Advanced Analytics (DSAA), 2015. 36678 2015. IEEE International Conference on*. IEEE. 2015, pp. 1–10.
- [272] Marvin N. Wright and Andreas Ziegler. “ranger: A Fast Implementation of Random Forests for High Dimensional Data in C++ and R”. In: *Journal of Statistical Software* 77.1 (2017), pp. 1–17. DOI: 10.18637/jss.v077.i01.
- [273] Qingyun Wu, Chi Wang, and Silu Huang. “Frugal optimization for cost-related hyperparameters”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 35. 12. 2021, pp. 10347–10354.
- [274] Qingyun Wu, Chi Wang, John Langford, Paul Mineiro, and Marco Rossi. “ChaCha for Online AutoML”. In: *2021 International Conference on Machine Learning (ICML 2021)*. July 2021. URL: <https://www.microsoft.com/en-us/research/publication/chacha-for-online-automl/>.

- [275] Zhenqin Wu, Bharath Ramsundar, Evan N. Feinberg, Joseph Gomes, Caleb Geniesse, Aneesh S. Pappu, Karl Leswing, and Vijay Pande. “MoleculeNet: a benchmark for molecular machine learning”. en. In: *Chemical Science* 9.2 (2018), pp. 513–530. DOI: 10.1039/C7SC02664A. URL: <https://pubs.rsc.org/en/content/articlelanding/2018/sc/c7sc02664a> (visited on 10/21/2021).
- [276] Iordanis Xanthopoulos, Ioannis Tsamardinos, Vassilis Christophides, Eric Simon, and Alejandro Salinger. “Putting the Human Back in the AutoML Loop.” In: *EDBT/ICDT Workshops*. 2020.
- [277] Zhen Xu, Wei-Wei Tu, and Isabelle Guyon. “AutoML Meets Time Series Regression Design and Analysis of the AutoSeries Challenge”. In: *Machine Learning and Knowledge Discovery in Databases. Applied Data Science Track*. Ed. by Yuxiao Dong, Nicolas Kourtellis, Barbara Hammer, and Jose A. Lozano. Cham: Springer International Publishing, 2021, pp. 36–51. ISBN: 978-3-030-86517-7.
- [278] Anatoly Yakovlev, Hesam Fathi Moghadam, Ali Moharrer, Jingxiao Cai, Nikan Chavoshi, Venkatanathan Varadarajan, Sandeep R. Agrawal, Sam Idicula, Tomas Karnagel, Sanjay Jinturkar, and Nipun Agarwal. “Oracle AutoML: A Fast and Predictive AutoML Pipeline”. In: *Proc. VLDB Endow.* 13.12 (Aug. 2020), pp. 3166–3180. ISSN: 2150-8097. DOI: 10.14778/3415478.3415542. URL: <https://doi.org/10.14778/3415478.3415542>.
- [279] Chengrun Yang, Yuji Akimoto, Dae Won Kim, and Madeleine Udell. “OBOE: Collaborative Filtering for AutoML Initialization”. In: *CoRR* abs/1808.03233 (2018). arXiv: 1808.03233. URL: <http://arxiv.org/abs/1808.03233>.
- [280] Murat Onur Yildirim, Elif Ceren Gök, Pieter Gijsbers, and Joaquin Vanschoren. *GAMACluster: Robust Automated Clustering*. (to appear) arXiv. 2022.
- [281] Dani Yogatama and Gideon Mann. “Efficient Transfer Learning Method for Automatic Hyperparameter Tuning”. In: *Proceedings of the Seventeenth International Conference on Artificial Intelligence and Statistics*. Ed. by Samuel Kaski and Jukka Corander. Vol. 33. Proceedings of Machine Learning Research. Reykjavik, Iceland: PMLR, Apr. 2014, pp. 1077–1085.

- [282] Achim Zeileis and Kurt Hornik. “Generalized M-fluctuation tests for parameter instability”. In: *Statistica Neerlandica* 61.4 (2007), pp. 488–508. DOI: <https://doi.org/10.1111/j.1467-9574.2007.00371.x>. eprint: <https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1467-9574.2007.00371.x>. URL: <https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1467-9574.2007.00371.x>.
- [283] Zhao Zhong, Junjie Yan, Wei Wu, Jing Shao, and Cheng-Lin Liu. “Practical block-wise neural network architecture generation”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 2423–2432.
- [284] Lucas Zimmer, Marius Lindauer, and Frank Hutter. “Auto-Pytorch: Multi-Fidelity MetaLearning for Efficient and Robust AutoDL”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 43.9 (2021), pp. 3079–3090.
- [285] Marc-André Zöller and Marco F Huber. “Benchmark and survey of automated machine learning frameworks”. In: *Journal of Artificial Intelligence Research* 70 (2021), pp. 409–472.
- [286] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. “Learning transferable architectures for scalable image recognition”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 8697–8710.
- [287] Hui Zou and Trevor Hastie. “Regularization and variable selection via the elastic net”. In: *Journal of the royal statistical society: series B (statistical methodology)* 67.2 (2005), pp. 301–320.

Appendix A

OpenML Benchmarking Suites

This appendix contains useful links for creating OpenML benchmarking suites, a suggested curation protocol (both from Bischl et al. [28]), and an overview of the different benchmarking suites presented in this thesis, including the full list of tasks and corresponding meta-data.

A.1 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)

A.2 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.

A.3 OpenML-CC18

A classification benchmarking suite for benchmarking ML algorithms.

Table A.1: Tasks OpenML-CC18.

Task ID	name	instances	features	classes	class ratio
7592	adult	48842	15	2	0.31
3549	analcatauthorship	841	71	4	0.17
3560	analcata_dmt	797	5	6	0.79
11	balance-scale	625	5	3	0.17
14965	bank-marketing	45211	17	2	0.13
10093	banknote-authentication	1372	5	2	0.80
9910	Bioresponse	3751	1777	2	0.84
10101	blood-transfusion-serv...	748	5	2	0.31
15	breast-w	699	10	2	0.53
146821	car	1728	7	4	0.05
167141	churn	5000	21	2	0.16
167124	CIFAR_10	60000	3073	10	1.00
146819	climate-model-simulati...	540	21	2	0.09
23	cmc	1473	10	3	0.53
9981	cnae-9	1080	857	9	1.00
146195	connect-4	67557	43	3	0.15
29	credit-approval	690	16	2	0.80
31	credit-g	1000	21	2	0.43
14954	cylinder-bands	540	40	2	0.73
167121	Devnagari-Script	92000	1025	46	1.00
37	diabetes	768	9	2	0.54
167140	dna	3186	181	3	0.46
125920	dressess-sales	500	13	2	0.72
219	electricity	45312	9	2	0.74
2079	eucalyptus	736	20	5	0.49
146825	Fashion-MNIST	70000	785	10	1.00
9985	first-order-theorem-pr...	6118	52	6	0.19

Tasks OpenML-CC18 (continued).

Task ID	name	instances	features	classes	class ratio
14969	GesturePhaseSegmentati...	9873	33	5	0.34
14970	har	10299	562	6	0.72
9971	ilpd	583	11	2	0.40
167125	Internet-Advertisements	3279	1559	2	0.16
3481	isolet	7797	618	26	0.99
3904	jm1	10885	22	2	0.24
167119	jungle_chess_2pcs_raw....	44819	7	3	0.19
3917	kc1	2109	22	2	0.18
3913	kc2	522	22	2	0.26
3	kr-vs-kp	3196	37	2	0.91
6	letter	20000	17	26	0.90
9976	madelon	2600	501	2	1.00
12	mfeat-factors	2000	217	10	1.00
14	mfeat-fourier	2000	77	10	1.00
16	mfeat-karhunen	2000	65	10	1.00
18	mfeat-morphological	2000	7	10	1.00
146824	mfeat-pixel	2000	241	10	1.00
22	mfeat-zernike	2000	48	10	1.00
146800	MiceProtein	1080	82	8	0.70
3573	mnist_784	70000	785	10	0.80
9977	nomao	34465	119	2	0.40
167120	numerai28.6	96320	22	2	0.98
28	optdigits	5620	65	10	0.97
9978	ozone-level-8hr	2534	73	2	0.07
3918	pc1	1109	22	2	0.07
3903	pc3	1563	38	2	0.11
3902	pc4	1458	38	2	0.14
32	pendigits	10992	17	10	0.92
14952	PhishingWebsites	11055	31	2	0.80
9952	phoneme	5404	6	2	0.42
9957	qsar-biodeg	1055	42	2	0.51

Tasks OpenML-CC18 (continued).

Task ID	name	instances	features	classes	class ratio
2074	satimage	6430	37	6	0.41
146822	segment	2310	20	7	1.00
9964	semeion	1593	257	10	0.96
3021	sick	3772	30	2	0.07
43	spambase	4601	58	2	0.65
45	splice	3190	61	3	0.46
146817	steel-plates-fault	1941	28	7	0.08
125922	texture	5500	41	11	1.00
49	tic-tac-toe	958	10	2	0.53
53	vehicle	846	19	4	0.91
3022	vowel	990	13	11	1.00
9960	wall-robot-navigation	5456	25	4	0.15
9946	wdbc	569	31	2	0.59
146820	wilt	4839	6	2	0.06

A.4 AutoML Benchmark Regression

A regression benchmarking suite for benchmarking AutoML frameworks.

Table A.2: Tasks in the AutoML regression suite.

Task ID	name	instances	features
359944	abalone	4177	9
359929	Airlines_DepDelay_10M	10000000	10
233212	Allstate_Claims_Severity	188318	131
359937	black_friday	166821	10
359950	boston	506	14
359938	Brazilian_houses	10692	13
233213	Buzzinsocialmedia_Twit...	583250	78
359942	colleges	7063	45

Tasks in the AutoML regression suite (continued).

Task ID	name	instances	features
233211	diamonds	53940	10
359936	elevators	16599	19
359952	house_16H	22784	17
359951	house_prices_nominal	1460	80
359949	house_sales	21613	22
233215	Mercedes-Benz_Greener_...	4209	377
360945	MIP-2016-regression	1090	145
167210	Moneyball	1232	15
359943	nyc-taxi-green-dec-2016	581835	19
359941	OnlineNewsPopularity	39644	60
359946	pol	15000	49
360933	QSAR-TID-10980	5766	1026
360932	QSAR-TID-11	5742	1026
359930	quake	2178	4
233214	Santander_transaction_...	4459	4992
359948	SAT11-HAND-runtime-reg...	4440	117
359931	sensory	576	12
359932	socmob	1156	6
359933	space_ga	3107	7
359934	tecator	240	125
359939	topo_2_1	8885	267
359945	us_crime	1994	127
359935	wine_quality	6497	12
317614	Yolanda	400000	101
359940	yprop_4_1	8885	252

A.5 AutoML Benchmark Classification

A classification benchmarking suite for benchmarking AutoML frameworks. Compared to OpenML-CC18, it features bigger datasets which harder data characteristics (e.g., greater class imbalances).

Table A.3: Tasks in the AutoML classification suite.

Task ID	name	instances	features	classes	class ratio
190411	ada	4147	49	2	0.33
359983	adult	48842	15	2	0.31
189354	airlines	539383	8	2	0.80
189356	albert	425240	79	2	1.00
10090	amazon-commerce-reviews	1500	10001	50	1.00
359979	Amazon_employee_access	32769	10	2	0.06
168868	APSFailure	76000	171	2	0.02
190412	arcene	100	10001	2	0.79
146818	Australian	690	15	2	0.80
359982	bank-marketing	45211	17	2	0.13
359967	Bioresponse	3751	1777	2	0.84
359955	blood-transfusion-serv...	748	5	2	0.31
359960	car	1728	7	4	0.05
359973	christine	5418	1637	2	1.00
359968	churn	5000	21	2	0.16
359992	Click_prediction_small	39948	12	2	0.20
359959	cmc	1473	10	3	0.53
359957	cnae-9	1080	857	9	1.00
359977	connect-4	67557	43	3	0.15
7593	covertype	581012	55	7	0.01
168757	credit-g	1000	21	2	0.43
211986	Diabetes130US	101766	50	3	0.21
168909	dilbert	10000	2001	5	0.93
189355	dionis	416188	61	355	0.36
359964	dna	3186	181	3	0.46
359954	eucalyptus	736	20	5	0.49
168910	fabert	8237	801	7	0.26
359976	Fashion-MNIST	70000	785	10	1.00
359969	first-order-theorem-pr...	6118	52	6	0.19
359970	GesturePhaseSegmentati...	9873	33	5	0.34
189922	gina	3153	971	2	0.97

Tasks in the AutoML classification suite (continued).

Task ID	name	instances	features	classes	class ratio
359988	guillermo	20000	4297	2	0.67
359984	helen	65196	28	100	0.03
360114	Higgs	1000000	29	2	0.89
359966	Internet-Advertisements	3279	1559	2	0.16
211979	jannis	83733	55	4	0.04
168911	jasmine	2984	145	2	1.00
359981	jungle_chess_2pcs_raw....	44819	7	3	0.19
359962	kc1	2109	22	2	0.18
360975	KDDCup09-Upselling	50000	14892	2	0.08
3945	KDDCup09_appetency	50000	231	2	0.02
360112	KDDCup99	4898431	42	23	0.00
359991	kick	72983	33	2	0.14
359965	kr-vs-kp	3196	37	2	0.91
190392	madeline	3140	260	2	0.99
359961	mfeat-factors	2000	217	10	1.00
359953	micro-mass	571	1301	20	0.18
359990	MiniBooNE	130064	51	2	0.39
359980	nomao	34465	119	2	0.40
167120	numera128.6	96320	22	2	0.98
359993	okcupid-stem	50789	20	3	0.13
190137	ozone-level-8hr	2534	73	2	0.07
359958	pc4	1458	38	2	0.14
190410	philippine	5832	309	2	1.00
359971	PhishingWebsites	11055	31	2	0.80
168350	phoneme	5404	6	2	0.42
360113	porto-seguro	595212	58	2	0.04
359956	qsar-biodeg	1055	42	2	0.51
359989	riccardo	20000	4297	2	0.33
359986	robert	10000	7201	10	0.92
359975	Satellite	5100	37	2	0.01
359963	segment	2310	20	7	1.00

Tasks in the AutoML classification suite (continued).

Task ID	name	instances	features	classes	class ratio
359994	sf-police-incidents	2215023	9	2	0.14
359987	shuttle	58000	10	7	0.00
168784	steel-plates-fault	1941	28	7	0.08
359972	sylvine	5124	21	2	1.00
190146	vehicle	846	19	4	0.91
359985	volkert	58310	181	10	0.11
146820	wilt	4839	6	2	0.06
359974	wine-quality-white	4898	12	7	0.00
2073	yeast	1484	9	10	0.01

Appendix B

AutoML Benchmark Results

B.1 Simple Bradley-Terry Trees

The Bradley-Terry trees in Figure B.1 and B.2 show trees generated on all results from 1 and 4 hour experiments, respectively. Only ‘instances’ and ‘features’ were considered as split criteria in making these trees and the maximum tree depth was kept at 3 to keep the trees interpretable.

In congruence with general results, **AutoGluon** is typically the preferred framework. The one exception is **FLAML** which is identified as more useful for large datasets (more than $\approx 180k$ instances) with a one hour constraint. This is in line with **FLAML**’s design goal to work especially well in time constrained settings (as the datasets are relatively large for the time budget).

As we saw in the Section 5.5, different preferences may be obtained for other subsets of the data (e.g., binary classification) and considering other meta-features (e.g., balance ratio). When using the BT trees to interpret the results, one should pay attention to the number of tasks in each leaf. Leafs based on particularly small sets of tasks may not generalize that well.

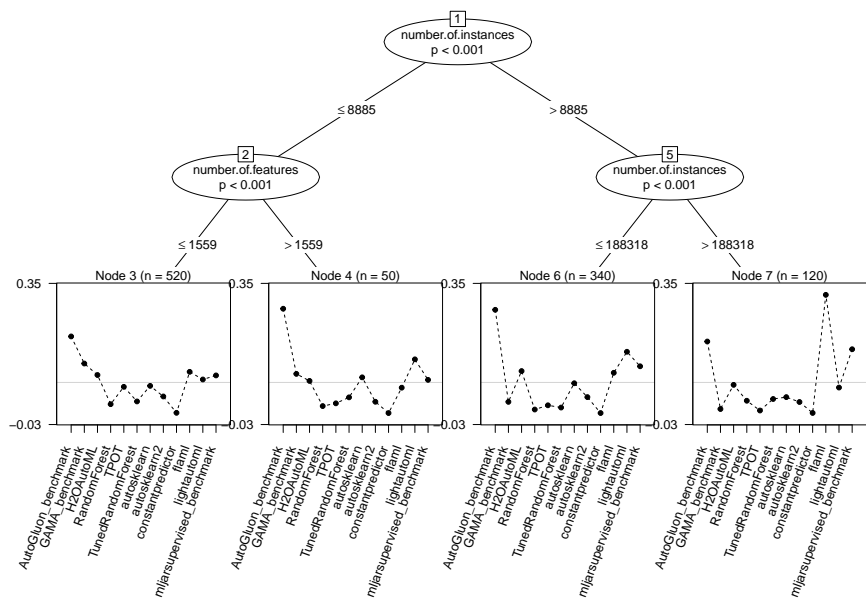


Figure B.1: A BT tree generated with only ‘features’ and ‘instances’ for split criteria, based on all results for one hour experiments.

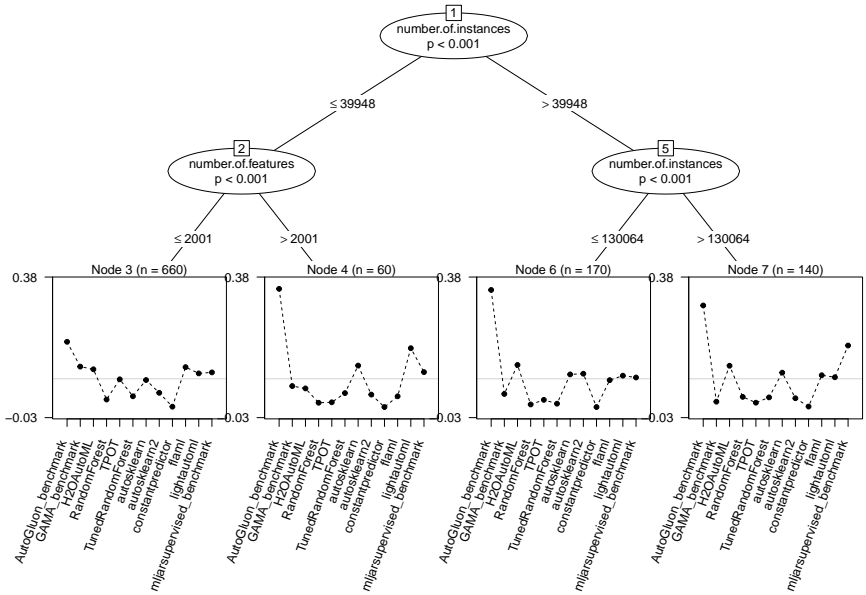


Figure B.2: A BT tree generated with only ‘features’ and ‘instances’ for split criteria, based on all results for four hour experiments.

B.2 AutoML Framework Errors

B.2.1 Class Imbalance

The two classification tasks with a large amount of failures despite being small are ‘yeast’ and ‘wine-quality-white’, which feature a minority class with only 5 instances. This means that within the 10-fold cross-validation we perform in our experiments, either 4 or 5 of those instances are available in the training splits. We see that only in the case where one of those samples is in the test set failures occur. The exact error message differs per framework, though they indicate that evaluating pipelines fails. This is likely due to e.g., using 5-fold cross-validation out of the box. Failure on these specific datasets (and folds) is only observed for GAMA, LightAutoML, and TPOT.

B.2.2 MLJarSupervised

Two thirds of all ‘implementation errors’ observed are failures of **MLJarSupervised**. All 190 failures are caused by variations of the following two unique errors:

```
25 times ['Ensemble_prediction_0_for_neg_1_for_pos', ...,
         '2.DecisionTree_prediction_0_for_neg_1_for_pos'] not in index"
```

```
165 times catboost/libs/data/model_dataset_compatibility.cpp:81:
         At position 6 should be feature with name 60_NeuralNetwork_prediction_0_for_
         (found 60_NeuralNetwork_prediction).
```

While we can only guess, we assume it is related to the extensive AutoML pipeline **MLJarSupervised** has. It includes 10 different steps, including three steps for feature generation and selection and three steps for ensembling and stacking. These steps are not turned on by default¹, feature engineering is only turned on for ‘performance’ and ‘compete’, and ensembling and stacking is only used in ‘compete’ mode, which we used.

B.2.3 Errors by Framework and Benchmark

Tables B.1 and B.2 display for each framework the number of errors per benchmark and time constraint. The ‘task’ column denotes in how many unique tasks an error was encountered and the ‘total’ column denotes the total number

¹<https://supervised.mljar.com/features/modes/>

Table B.1: An overview of errors for framework (A-H). ‘tasks’ denote how many unique tasks are affected, and ‘total’ how frequent the error occurred in total.

framework	task type	constraint	error	Tasks	Total
autogluon	Classification	4h	Memory	2	3
autosklearn	Classification	1h	Memory	1	1
		4h	Timeout	1	5
	Regression	1h	Data	1	1
		4h	Data	1	1
autosklearn2	Classification	4h	Timeout	1	10
flaml	Classification	1h	Memory	4	18
		4h	Memory	10	51
			Timeout	4	8
	Regression	4h	Memory	6	7
			Timeout	1	3
gama	Classification	1h	Data	1	2
			Implementation	4	26
			Memory	2	9
		4h	Data	1	2
			Implementation	4	28
	Regression	4h	Memory	5	25
h2oautoml	Classification	4h	Memory	2	7
		1h	Timeout	1	10
		4h	Timeout	1	10

of failures. Because the experiments are 10-fold cross-validation, at most 10 failures per task (per framework per constraint) may occur.

Table B.2: An overview of errors for framework (I-Z). ‘tasks’ denote how many unique tasks are affected, and ‘total’ how frequent the error occurred in total.

framework	task type	constraint	error	Tasks	Total
lightautoml	Classification	1h	Memory	8	41
			Timeout	2	2
		4h	Memory	10	69
			Timeout	2	2
	Regression	1h	Memory	1	1
		4h	Memory	1	5
mljarsupervised	Classification	1h	Implementation	1	1
			Memory	14	72
		4h	Implementation	1	7
			Memory	15	118
	Regression	1h	Timeout	2	9
			Data	3	13
		4h	Data	1	8
			Data	1	5
tpot	Classification	1h	Timeout	1	1
			Data	3	12
			Implementation	4	12
		4h	Timeout	6	27
			Data	3	14
			Implementation	5	10
	Regression	1h	Timeout	9	29
			Implementation	3	4
		4h	Timeout	1	1
			Implementation	1	4

Appendix C

Symbolic Hyperparameter Defaults

C.1 Implementation defaults

Table C.1 contains existing implementation defaults used in our experiments. They have been obtained from the current versions of the implementations. We analyze algorithms from the following algorithm implementations: Elastic Net: `glmnet` [92], Decision Trees: `rpart` [236], Random Forest: `ranger` [272], SVM: `LibSVM` via `e1071` ([53], [163]) and `xgboost` [55]. We investigate `HNSW` [157] as an approximate k-Nearest-Neighbours algorithm. Additional details on the exact meaning of the different hyperparameters can be obtained from the respective software’s documentation. We assume that small differences due to implementation details e.g. between the `LibSVM` and `sklearn` implementations exist, but try to compare to existing default settings nonetheless, as they might serve as relevant baselines.

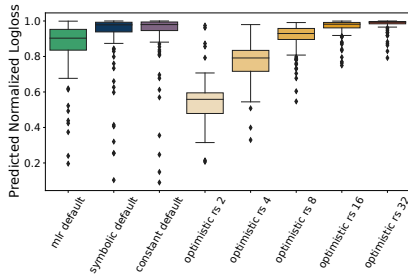
Algorithm	Package	Default
Elastic Net	glmnet	$\alpha : 1, \lambda : 0.01$
Decision Tree	rpart	$cp : 0.01, max.depth : 30,$ $minbucket : 1, minsplit : 20$
Random Forest	ranger	$mtry : \sqrt{po}, sample.fraction : 1,$ $min.node.size : 1$
SVM	e1071	$C : 1, \gamma : \frac{1}{po}$
	sklearn	$C : 1, \gamma : \frac{1}{p*xxvar}$
Approx. kNN	mlr	$k : 10, M : 16, ef : 10, efc : 200$
Gradient Boosting	xgboost	$\eta : 0.1, \lambda : 1, \gamma : 0, \alpha : 0, subsample : 1,$ $max_depth : 3, min_child_weight : 1,$ $colsample_bytree : 1, colsample_bylevel : 1$

Table C.1: Baseline *b*): Existing defaults for algorithm implementations. Fixed parameters described in Table 6.3 apply.

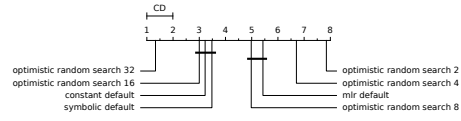
C.2 Experimental Results

The following section describes the results of the Experiments conducted to answer **RQ1** and **RQ2** across all other algorithms analyzed in this paper. Results and a more detailed analysis for the SVM can be obtained from Section 6.6.2.

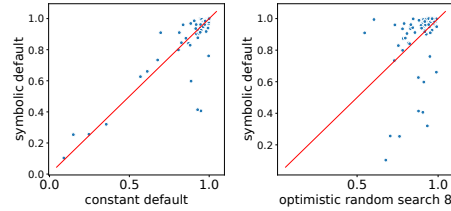
C.2.1 Elastic Net



(a) Symbolic, static and implementation defaults, comparing scaled logloss predicted by surrogates.



(b) Critical Differences Diagram of symbolic, static and implementation defaults on surrogates



(c) Performance comparison of symbolic defaults to constant defaults (left) and budget 8 optimistic random search (right).

Figure C.1: Results for the elastic net algorithm on surrogate data.

C.2.2 Decision Trees

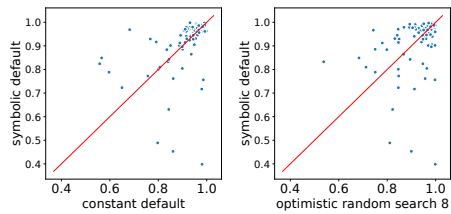
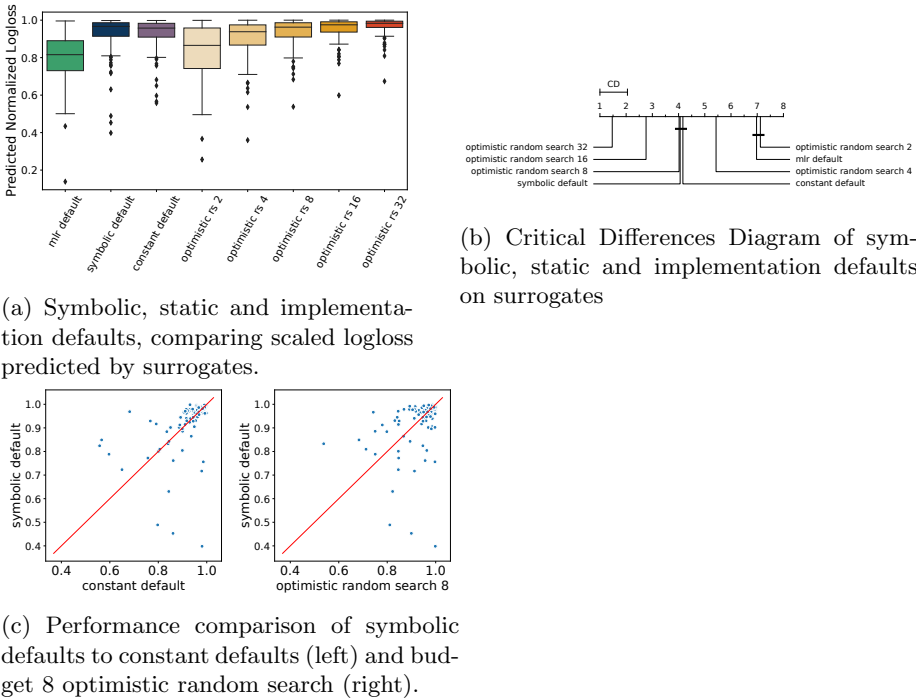
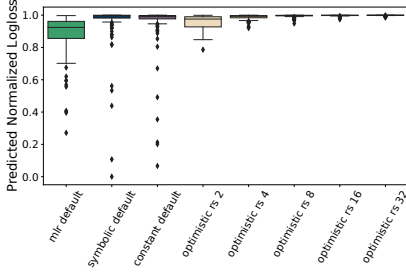
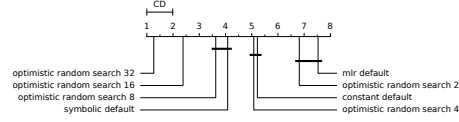


Figure C.2: Results for the decision tree algorithm on surrogate data.

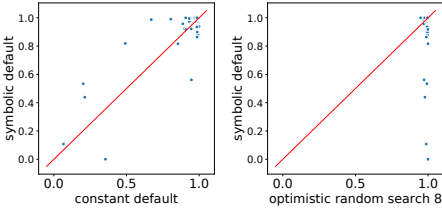
C.2.3 Approximate k-Nearest Neighbours



(a) Symbolic, static and implementation defaults, comparing scaled logloss predicted by surrogates.



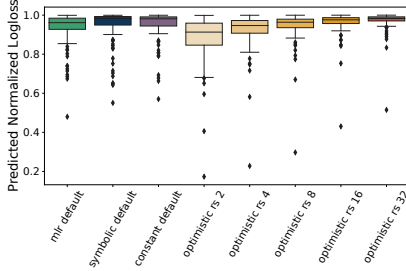
(b) Critical Differences Diagram of symbolic, static and implementation defaults on surrogates



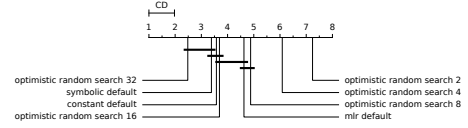
(c) Performance comparison of symbolic defaults to constant defaults (left) and budget 8 optimistic random search (right).

Figure C.3: Results for the approximate k-nearest neighbours algorithm on surrogate data.

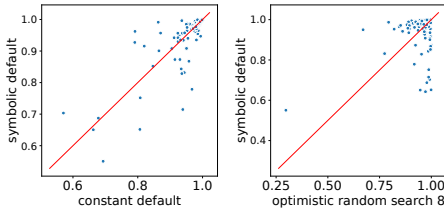
C.2.4 Random Forest



(a) Symbolic, static and implementation defaults, comparing scaled logloss predicted by surrogates.



(b) Critical Differences Diagram of symbolic, static and implementation defaults on surrogates



(c) Performance comparison of symbolic defaults to constant defaults (left) and budget 8 optimistic random search (right).

Figure C.4: Results for the random forest algorithm on surrogate data.

C.2.5 eXtreme Gradient Boosting (XGBoost)

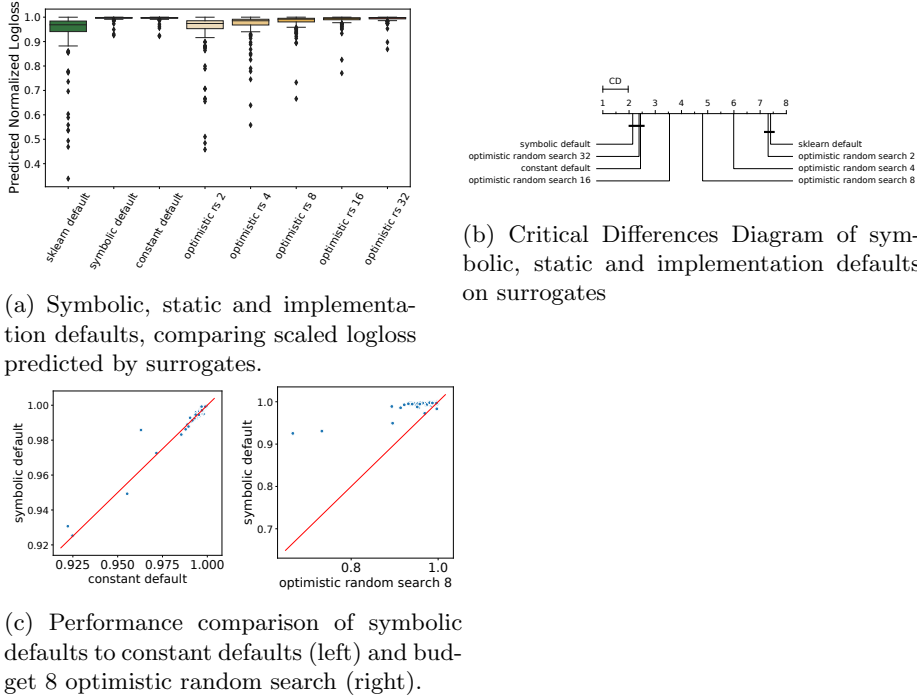


Figure C.5: Results for the XGBoost algorithm on surrogate data.

C.3 Real Data Experiments

In analogy to the presentation of the results for the SVM of the main text, we present results for Decision Tree and Elastic Net here.

C.3.1 Decision Tree

C.3.2 Elastic Net

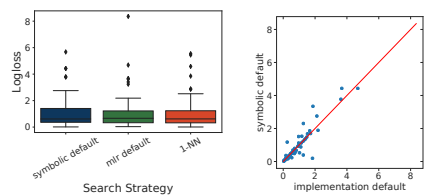


Figure C.6: Results for the decision tree algorithm. Comparison of symbolic and implementation default using log-loss across all datasets performed on real data. Box plots (right) and scatter plot (left)

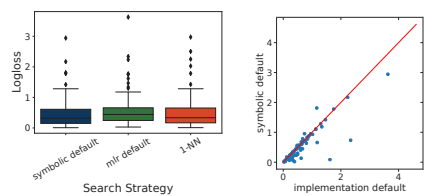


Figure C.7: Results for the Elastic Net algorithm. Comparison of symbolic and implementation default using log-loss across all datasets performed on real data. Box plots (right) and scatter plot (left)

Curriculum Vitae

Pieter Gijsbers was born in Eindhoven, the Netherlands on the 26th of June 1992. He obtained his bachelor's degree in 2015 at Fontys University of Applied Sciences with a focus on software engineering for embedded systems. For this bachelor's thesis, he interned at Van Dores MES, Veghel, where he stayed as a part-time manufacturing execution system developer until 2017.

In 2017 Pieter obtained his master's degree at Eindhoven University of Technology in Eindhoven, the Netherlands within the data mining group on automated machine learning. In 2017 he started a PhD project at Eindhoven University of Technology of which the results are presented in this dissertation. Since 2021 Pieter is employed at Eindhoven University of Technology as AI engineer.

Pieter joined the OpenML project as a contributor in 2016 and is now one of two `openml-python` core contributors, co-organized OpenML hackathons, and joined the OpenML steering committee in 2021. He obtained 8th place at the ChaLearn Lifelong AutoML Challenge in 2018. During his PhD trajectory he started two new open source projects, the modular AutoML tool **GAMA** and the open source AutoML benchmark. Together with Joaquin Vanschoren he has given invited talks at multiple editions of the Open Data Science Conference, where he demonstrated how to use OpenML and AutoML in Python.



List of Publications

A chronologically ordered list of publications by Pieter Gijsbers, papers used in the making of this thesis indicated with ‘▷’:

▷ B. Bischl, G. Casalicchio, M. Feurer, **P. Gijsbers**, F. Hutter, M. Lang, R. G. Mantovani, J. N. van Rijn, and J. Vanschoren, “Openml benchmarking suites,” in *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*, 2021

▷ **P. Gijsbers**, F. Pfisterer, J. N. van Rijn, B. Bischl, and J. Vanschoren, “Meta-learning for symbolic hyperparameter defaults,” *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, Jul. 2021.

▷ M. Feurer, J. N. van Rijn, A. Kadra, **P. Gijsbers**, N. Mallik, S. Ravi, A. Müller, and J. Vanschoren, “Openml-python: An extensible python api for openml,” *Journal of Machine Learning Research*, vol. 22, no. 100, pp. 1–5, 2021

▷ **P. Gijsbers** and J. Vanschoren, “Gama: A general automated machine learning assistant,” *Lecture Notes in Computer Science*, pp. 560–564, 2021, **issn**: 1611-3349

▷ **P. Gijsbers**, E. LeDell, J. Thomas, S. Poirier, B. Bischl, and J. Vanschoren, “An open source automl benchmark,” *arXiv preprint arXiv:1907.00909*, 2019, accepted at ICML 2019 AutoML workshop.

▷ **P. Gijsbers** and J. Vanschoren, “Gama: Genetic automated machine learning assistant,” *Journal of Open Source Software*, vol. 4, no. 33, p. 1132, 2019

M. Konzack, **P. Gijsbers**, F. Timmers, E. van Loon, M. A. Westenberg, and K. Buchin, “Visual exploration of migration patterns in gull data,” *Information Visualization*, vol. 18, no. 1, pp. 138–152, 2019.

P. Gijsbers, J. Vanschoren, and R. Olson, “Layered tpot: Speeding up tree-based pipeline optimization,” in *European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases, September 18–22, 2017, Skopje, Macedonia*, CEUR-WS. org, 2017, pp. 49–68

SIKS Dissertations

2016

- 01 Syed Saiden Abbas (RUN), Recognition of Shapes by Humans and Machines
- 02 Michiel Christiaan Meulendijk (UU), Optimizing medication reviews through decision support: prescribing a better pill to swallow
- 03 Maya Sappelli (RUN), Knowledge Work in Context: User Centered Knowledge Worker Support
- 04 Laurens Rietveld (VU), Publishing and Consuming Linked Data
- 05 Evgeny Sherkhonov (UVA), Expanded Acyclic Queries: Containment and an Application in Explaining Missing Answers
- 06 Michel Wilson (TUD), Robust scheduling in an uncertain environment
- 07 Jeroen de Man (VU), Measuring and modeling negative emotions for virtual training
- 08 Matje van de Camp (TiU), A Link to the Past: Constructing Historical Social Networks from Unstructured Data
- 09 Archana Nottamkandath (VU), Trusting Crowdsourced Information on Cultural Artefacts
- 10 George Karafotias (VUA), Parameter Control for Evolutionary Algorithms
- 11 Anne Schuth (UVA), Search Engines that Learn from Their Users
- 12 Max Knobbout (UU), Logics for Modelling and Verifying Normative Multi-Agent Systems
- 13 Nana Baah Gyan (VU), The Web, Speech Technologies and Rural Development in West Africa - An ICT4D Approach

- 14 Ravi Khadka (UU), Revisiting Legacy Software System Modernization
- 15 Steffen Michels (RUN), Hybrid Probabilistic Logics - Theoretical Aspects, Algorithms and Experiments
- 16 Guangliang Li (UVA), Socially Intelligent Autonomous Agents that Learn from Human Reward
- 17 Berend Weel (VU), Towards Embodied Evolution of Robot Organisms
- 18 Albert Meroño Peñuela (VU), Refining Statistical Data on the Web
- 19 Julia Efremova (Tu/e), Mining Social Structures from Genealogical Data
- 20 Daan Odijk (UVA), Context & Semantics in News & Web Search
- 21 Alejandro Moreno Céleri (UT), From Traditional to Interactive Playspaces: Automatic Analysis of Player Behavior in the Interactive Tag Playground
- 22 Grace Lewis (VU), Software Architecture Strategies for Cyber-Foraging Systems
- 23 Fei Cai (UVA), Query Auto Completion in Information Retrieval
- 24 Brend Wanders (UT), Repurposing and Probabilistic Integration of Data; An Iterative and data model independent approach
- 25 Julia Kiseleva (TU/e), Using Contextual Information to Understand Searching and Browsing Behavior
- 26 Dilhan Thilakarathne (VU), In or Out of Control: Exploring Computational Models to Study the Role of Human Awareness and Control in Behavioural Choices, with Applications in Aviation and Energy Management Domains
- 27 Wen Li (TUD), Understanding Geo-spatial Information on Social Media
- 28 Mingxin Zhang (TUD), Large-scale Agent-based Social Simulation - A study on epidemic prediction and control
- 29 Nicolas Höning (TUD), Peak reduction in decentralised electricity systems - Markets and prices for flexible planning
- 30 Ruud Mattheij (UvT), The Eyes Have It
- 31 Mohammad Khelghati (UT), Deep web content monitoring
- 32 Eelco Vriezekolk (UT), Assessing Telecommunication Service Availability Risks for Crisis Organisations
- 33 Peter Bloem (UVA), Single Sample Statistics, exercises in learning from just one example

- 34 Dennis Schunselaar (TUE), Configurable Process Trees: Elicitation, Analysis, and Enactment
- 35 Zhaochun Ren (UVA), Monitoring Social Media: Summarization, Classification and Recommendation
- 36 Daphne Karreman (UT), Beyond R2D2: The design of nonverbal interaction behavior optimized for robot-specific morphologies
- 37 Giovanni Sileno (UvA), Aligning Law and Action - a conceptual and computational inquiry
- 38 Andrea Minuto (UT), Materials that Matter - Smart Materials meet Art & Interaction Design
- 39 Merijn Bruijnes (UT), Believable Suspect Agents; Response and Interpersonal Style Selection for an Artificial Suspect
- 40 Christian Detweiler (TUD), Accounting for Values in Design
- 41 Thomas King (TUD), Governing Governance: A Formal Framework for Analysing Institutional Design and Enactment Governance
- 42 Spyros Martzoukos (UVA), Combinatorial and Compositional Aspects of Bilingual Aligned Corpora
- 43 Saskia Koldijk (RUN), Context-Aware Support for Stress Self-Management: From Theory to Practice
- 44 Thibault Sellam (UVA), Automatic Assistants for Database Exploration
- 45 Bram van de Laar (UT), Experiencing Brain-Computer Interface Control
- 46 Jorge Gallego Perez (UT), Robots to Make you Happy
- 47 Christina Weber (UL), Real-time foresight - Preparedness for dynamic innovation networks
- 48 Tanja Buttler (TUD), Collecting Lessons Learned
- 49 Gleb Polevoy (TUD), Participation and Interaction in Projects. A Game-Theoretic Analysis
- 50 Yan Wang (UVT), The Bridge of Dreams: Towards a Method for Operational Performance Alignment in IT-enabled Service Supply Chains

2017

-
- 01 Jan-Jaap Oerlemans (UL), Investigating Cybercrime
 - 02 Sjoerd Timmer (UU), Designing and Understanding Forensic Bayesian Networks using Argumentation

- 03 Daniël Harold Telgen (UU), Grid Manufacturing; A Cyber-Physical Approach with Autonomous Products and Reconfigurable Manufacturing Machines
- 04 Mrunal Gawade (CWI), Multi-core Parallelism in a Column-store
- 05 Mahdieh Shadi (UVA), Collaboration Behavior
- 06 Damir Vandić (EUR), Intelligent Information Systems for Web Product Search
- 07 Roel Bertens (UU), Insight in Information: from Abstract to Anomaly
- 08 Rob Konijn (VU) , Detecting Interesting Differences: Data Mining in Health Insurance Data using Outlier Detection and Subgroup Discovery
- 09 Dong Nguyen (UT), Text as Social and Cultural Data: A Computational Perspective on Variation in Text
- 10 Robby van Delden (UT), (Steering) Interactive Play Behavior
- 11 Florian Kunneman (RUN), Modelling patterns of time and emotion in Twitter #anticipointment
- 12 Sander Leemans (TUE), Robust Process Mining with Guarantees
- 13 Gijs Huisman (UT), Social Touch Technology - Extending the reach of social touch through haptic technology
- 14 Shoshannah Tekofsky (UvT), You Are Who You Play You Are: Modelling Player Traits from Video Game Behavior
- 15 Peter Berck (RUN), Memory-Based Text Correction
- 16 Aleksandr Chuklin (UVA), Understanding and Modeling Users of Modern Search Engines
- 17 Daniel Dimov (UL), Crowdsourced Online Dispute Resolution
- 18 Ridho Reinanda (UVA), Entity Associations for Search
- 19 Jeroen Vuurens (UT), Proximity of Terms, Texts and Semantic Vectors in Information Retrieval
- 20 Mohammadbashir Sedighi (TUD), Fostering Engagement in Knowledge Sharing: The Role of Perceived Benefits, Costs and Visibility
- 21 Jeroen Linssen (UT), Meta Matters in Interactive Storytelling and Serious Gaming (A Play on Worlds)
- 22 Sara Magliacane (VU), Logics for causal inference under uncertainty
- 23 David Graus (UVA), Entities of Interest — Discovery in Digital Traces
- 24 Chang Wang (TUD), Use of Affordances for Efficient Robot Learning

- 25 Veruska Zamborlini (VU), Knowledge Representation for Clinical Guidelines, with applications to Multimorbidity Analysis and Literature Search
- 26 Merel Jung (UT), Socially intelligent robots that understand and respond to human touch
- 27 Michiel Joosse (UT), Investigating Positioning and Gaze Behaviors of Social Robots: People's Preferences, Perceptions and Behaviors
- 28 John Klein (VU), Architecture Practices for Complex Contexts
- 29 Adel Alhuraibi (UvT), From IT-BusinessStrategic Alignment to Performance: A Moderated Mediation Model of Social Innovation, and Enterprise Governance of IT"
- 30 Wilma Latuny (UvT), The Power of Facial Expressions
- 31 Ben Ruijl (UL), Advances in computational methods for QFT calculations
- 32 Thaer Samar (RUN), Access to and Retrievability of Content in Web Archives
- 33 Brigit van Loggem (OU), Towards a Design Rationale for Software Documentation: A Model of Computer-Mediated Activity
- 34 Maren Scheffel (OU), The Evaluation Framework for Learning Analytics
- 35 Martine de Vos (VU), Interpreting natural science spreadsheets
- 36 Yuanhao Guo (UL), Shape Analysis for Phenotype Characterisation from High-throughput Imaging
- 37 Alejandro Montes Garcia (TUE), WiBAF: A Within Browser Adaptation Framework that Enables Control over Privacy
- 38 Alex Kayal (TUD), Normative Social Applications
- 39 Sara Ahmadi (RUN), Exploiting properties of the human auditory system and compressive sensing methods to increase noise robustness in ASR
- 40 Altaf Hussain Abro (VUA), Steer your Mind: Computational Exploration of Human Control in Relation to Emotions, Desires and Social Support For applications in human-aware support systems
- 41 Adnan Manzoor (VUA), Minding a Healthy Lifestyle: An Exploration of Mental Processes and a Smart Environment to Provide Support for a Healthy Lifestyle
- 42 Elena Sokolova (RUN), Causal discovery from mixed and missing data with applications on ADHD datasets
- 43 Maaïke de Boer (RUN), Semantic Mapping in Video Retrieval

- 44 Garm Lucassen (UU), Understanding User Stories - Computational
Linguistics in Agile Requirements Engineering
- 45 Bas Testerink (UU), Decentralized Runtime Norm Enforcement
- 46 Jan Schneider (OU), Sensor-based Learning Support
- 47 Jie Yang (TUD), Crowd Knowledge Creation Acceleration
- 48 Angel Suarez (OU), Collaborative inquiry-based learning

2018

-
- 01 Han van der Aa (VUA), Comparing and Aligning Process Repre-
sentations
 - 02 Felix Mannhardt (TUE), Multi-perspective Process Mining
 - 03 Steven Bosems (UT), Causal Models For Well-Being: Knowledge
Modeling, Model-Driven Development of Context-Aware Applica-
tions, and Behavior Prediction
 - 04 Jordan Janeiro (TUD), Flexible Coordination Support for Diagnosis
Teams in Data-Centric Engineering Tasks
 - 05 Hugo Huurdeman (UVA), Supporting the Complex Dynamics of the
Information Seeking Process
 - 06 Dan Ionita (UT), Model-Driven Information Security Risk Assess-
ment of Socio-Technical Systems
 - 07 Jieting Luo (UU), A formal account of opportunism in multi-agent
systems
 - 08 Rick Smetsers (RUN), Advances in Model Learning for Software
Systems
 - 09 Xu Xie (TUD), Data Assimilation in Discrete Event Simulations
 - 10 Julienka Mollee (VUA), Moving forward: supporting physical ac-
tivity behavior change through intelligent technology
 - 11 Mahdi Sargolzaei (UVA), Enabling Framework for Service-oriented
Collaborative Networks
 - 12 Xixi Lu (TUE), Using behavioral context in process mining
 - 13 Seyed Amin Tabatabaei (VUA), Computing a Sustainable Future
 - 14 Bart Joosten (UVT), Detecting Social Signals with Spatiotemporal
Gabor Filters
 - 15 Naser Davarzani (UM), Biomarker discovery in heart failure
 - 16 Jaebok Kim (UT), Automatic recognition of engagement and emo-
tion in a group of children
 - 17 Jianpeng Zhang (TUE), On Graph Sample Clustering
 - 18 Henriette Nakad (UL), De Notaris en Private Rechtspraak

- 19 Minh Duc Pham (VUA), Emergent relational schemas for RDF
- 20 Manxia Liu (RUN), Time and Bayesian Networks
- 21 Aad Slootmaker (OUN), EMERGO: a generic platform for author-
ing and playing scenario-based serious games
- 22 Eric Fernandes de Mello Araujo (VUA), Contagious: Modeling the
Spread of Behaviours, Perceptions and Emotions in Social Networks
- 23 Kim Schouten (EUR), Semantics-driven Aspect-Based Sentiment
Analysis
- 24 Jered Vroon (UT), Responsive Social Positioning Behaviour for
Semi-Autonomous Telepresence Robots
- 25 Riste Gligorov (VUA), Serious Games in Audio-Visual Collections
- 26 Roelof Anne Jelle de Vries (UT), Theory-Based and Tailor-Made:
Motivational Messages for Behavior Change Technology
- 27 Maikel Leemans (TUE), Hierarchical Process Mining for Scalable
Software Analysis
- 28 Christian Willemse (UT), Social Touch Technologies: How they feel
and how they make you feel
- 29 Yu Gu (UVT), Emotion Recognition from Mandarin Speech
- 30 Wouter Beek, The "K" in "semantic web" stands for "knowledge":
scaling semantics to the web

2019

-
- 01 Rob van Eijk (UL), Web privacy measurement in real-time bidding
systems. A graph-based approach to RTB system classification
 - 02 Emmanuelle Beauxis Aussalet (CWI, UU), Statistics and Visualiza-
tions for Assessing Class Size Uncertainty
 - 03 Eduardo Gonzalez Lopez de Murillas (TUE), Process Mining on
Databases: Extracting Event Data from Real Life Data Sources
 - 04 Ridho Rahmadi (RUN), Finding stable causal structures from clin-
ical data
 - 05 Sebastiaan van Zelst (TUE), Process Mining with Streaming Data
 - 06 Chris Dijkshoorn (VU), Nichesourcing for Improving Access to
Linked Cultural Heritage Datasets
 - 07 Soude Fazeli (TUD), Recommender Systems in Social Learning
Platforms
 - 08 Frits de Nijs (TUD), Resource-constrained Multi-agent Markov De-
cision Processes

- 09 Fahimeh Alizadeh Moghaddam (UVA), Self-adaptation for energy efficiency in software systems
- 10 Qing Chuan Ye (EUR), Multi-objective Optimization Methods for Allocation and Prediction
- 11 Yue Zhao (TUD), Learning Analytics Technology to Understand Learner Behavioral Engagement in MOOCs
- 12 Jacqueline Heinerman (VU), Better Together
- 13 Guanliang Chen (TUD), MOOC Analytics: Learner Modeling and Content Generation
- 14 Daniel Davis (TUD), Large-Scale Learning Analytics: Modeling Learner Behavior & Improving Learning Outcomes in Massive Open Online Courses
- 15 Erwin Walraven (TUD), Planning under Uncertainty in Constrained and Partially Observable Environments
- 16 Guangming Li (TUE), Process Mining based on Object-Centric Behavioral Constraint (OCBC) Models
- 17 Ali Hurriyetoglu (RUN), Extracting actionable information from microtexts
- 18 Gerard Wagenaar (UU), Artefacts in Agile Team Communication
- 19 Vincent Koeman (TUD), Tools for Developing Cognitive Agents
- 20 Chide Groenouwe (UU), Fostering technically augmented human collective intelligence
- 21 Cong Liu (TUE), Software Data Analytics: Architectural Model Discovery and Design Pattern Detection
- 22 Martin van den Berg (VU), Improving IT Decisions with Enterprise Architecture
- 23 Qin Liu (TUD), Intelligent Control Systems: Learning, Interpreting, Verification
- 24 Anca Dumitrache (VU), Truth in Disagreement - Crowdsourcing Labeled Data for Natural Language Processing
- 25 Emiel van Miltenburg (VU), Pragmatic factors in (automatic) image description
- 26 Prince Singh (UT), An Integration Platform for Sychromodal Transport
- 27 Alessandra Antonaci (OUN), The Gamification Design Process applied to (Massive) Open Online Courses
- 28 Esther Kuindersma (UL), Cleared for take-off: Game-based learning to prepare airline pilots for critical situations

- 29 Daniel Formolo (VU), Using virtual agents for simulation and training of social skills in safety-critical circumstances
- 30 Vahid Yazdanpanah (UT), Multiagent Industrial Symbiosis Systems
- 31 Milan Jelisavcic (VU), Alive and Kicking: Baby Steps in Robotics
- 32 Chiara Sironi (UM), Monte-Carlo Tree Search for Artificial General Intelligence in Games
- 33 Anil Yaman (TUE), Evolution of Biologically Inspired Learning in Artificial Neural Networks
- 34 Negar Ahmadi (TUE), EEG Microstate and Functional Brain Network Features for Classification of Epilepsy and PNES
- 35 Lisa Facey-Shaw (OUN), Gamification with digital badges in learning programming
- 36 Kevin Ackermans (OUN), Designing Video-Enhanced Rubrics to Master Complex Skills
- 37 Jian Fang (TUD), Database Acceleration on FPGAs
- 38 Akos Kadar (OUN), Learning visually grounded and multilingual representations

2020

-
- 01 Armon Toubman (UL), Calculated Moves: Generating Air Combat Behaviour
 - 02 Marcos de Paula Bueno (UL), Unraveling Temporal Processes using Probabilistic Graphical Models
 - 03 Mostafa Deghani (UvA), Learning with Imperfect Supervision for Language Understanding
 - 04 Maarten van Gompel (RUN), Context as Linguistic Bridges
 - 05 Yulong Pei (TUE), On local and global structure mining
 - 06 Preethu Rose Anish (UT), Stimulation Architectural Thinking during Requirements Elicitation - An Approach and Tool Support
 - 07 Wim van der Vegt (OUN), Towards a software architecture for reusable game components
 - 08 Ali Mirsoleimani (UL), Structured Parallel Programming for Monte Carlo Tree Search
 - 09 Myriam Traub (UU), Measuring Tool Bias and Improving Data Quality for Digital Humanities Research
 - 10 Alifah Syamsiyah (TUE), In-database Preprocessing for Process Mining

- 11 Sepideh Mesbah (TUD), Semantic-Enhanced Training Data AugmentationMethods for Long-Tail Entity Recognition Models
- 12 Ward van Breda (VU), Predictive Modeling in E-Mental Health: Exploring Applicability in Personalised Depression Treatment
- 13 Marco Virgolin (CWI), Design and Application of Gene-pool Optimal Mixing Evolutionary Algorithms for Genetic Programming
- 14 Mark Raasveldt (CWI/UL), Integrating Analytics with Relational Databases
- 15 Konstantinos Georgiadis (OUN), Smart CAT: Machine Learning for Configurable Assessments in Serious Games
- 16 Ilona Wilmont (RUN), Cognitive Aspects of Conceptual Modelling
- 17 Daniele Di Mitri (OUN), The Multimodal Tutor: Adaptive Feedback from Multimodal Experiences
- 18 Georgios Methenitis (TUD), Agent Interactions & Mechanisms in Markets with Uncertainties: Electricity Markets in Renewable Energy Systems
- 19 Guido van Capelleveen (UT), Industrial Symbiosis Recommender Systems
- 20 Albert Hankel (VU), Embedding Green ICT Maturity in Organisations
- 21 Karine da Silva Miras de Araujo (VU), Where is the robot?: Life as it could be
- 22 Maryam Masoud Khamis (RUN), Understanding complex systems implementation through a modeling approach: the case of e-government in Zanzibar
- 23 Rianne Conijn (UT), The Keys to Writing: A writing analytics approach to studying writing processes using keystroke logging
- 24 Lenin da Nobrega Medeiros (VUA/RUN), How are you feeling, human? Towards emotionally supportive chatbots
- 25 Xin Du (TUE), The Uncertainty in Exceptional Model Mining
- 26 Krzysztof Leszek Sadowski (UU), GAMBIT: Genetic Algorithm for Model-Based mixed-Integer opTimization
- 27 Ekaterina Muravyeva (TUD), Personal data and informed consent in an educational context
- 28 Bibeg Limbu (TUD), Multimodal interaction for deliberate practice: Training complex skills with augmented reality
- 29 Ioan Gabriel Bucur (RUN), Being Bayesian about Causal Inference
- 30 Bob Zadok Blok (UL), Creatief, Creatieve, Creatiefst
- 31 Gongjin Lan (VU), Learning better – From Baby to Better

- 32 Jason Rhuggenaath (TUE), Revenue management in online mar-
kets: pricing and online advertising
- 33 Rick Gilsing (TUE), Supporting service-dominant business model
evaluation in the context of business model innovation
- 34 Anna Bon (MU), Intervention or Collaboration? Redesigning Infor-
mation and Communication Technologies for Development
- 35 Siamak Farshidi (UU), Multi-Criteria Decision-Making in Software
Production

2021

-
- 01 Francisco Xavier Dos Santos Fonseca (TUD), Location-based Games
for Social Interaction in Public Space
 - 02 Rijk Mercur (TUD), Simulating Human Routines: Integrating So-
cial Practice Theory in Agent-Based Models
 - 03 Seyyed Hadi Hashemi (UVA), Modeling Users Interacting with
Smart Devices
 - 04 Ioana Jivet (OU), The Dashboard That Loved Me: Designing adap-
tive learning analytics for self-regulated learning
 - 05 Davide Dell'Anna (UU), Data-Driven Supervision of Autonomous
Systems
 - 06 Daniel Davison (UT), "Hey robot, what do you think?" How chil-
dren learn with a social robot
 - 07 Armel Lefebvre (UU), Research data management for open science
 - 08 Nardie Fanchamps (OU), The Influence of Sense-Reason-Act Pro-
gramming on Computational Thinking
 - 09 Cristina Zaga (UT), The Design of Robothings. Non-
Anthropomorphic and Non-Verbal Robots to Promote Children's
Collaboration Through Play
 - 10 Quinten Meertens (UvA), Misclassification Bias in Statistical Learn-
ing
 - 11 Anne van Rossum (UL), Nonparametric Bayesian Methods in
Robotic Vision
 - 12 Lei Pi (UL), External Knowledge Absorption in Chinese SMEs
 - 13 Bob R. Schadenberg (UT), Robots for Autistic Children: Under-
standing and Facilitating Predictability for Engagement in Learning
 - 14 Negin Samaeemofrad (UL), Business Incubators: The Impact of
Their Support

- 15 Onat Ege Adali (TU/e), Transformation of Value Propositions into Resource Re-Configurations through the Business Services Paradigm
- 16 Esam A. H. Ghaleb (UM), BIMODAL EMOTION RECOGNITION FROM AUDIO-VISUAL CUES
- 17 Dario Dotti (UM), Human Behavior Understanding from motion and bodily cues using deep neural networks
- 18 Remi Wieten (UU), Bridging the Gap Between Informal Sense-Making Tools and Formal Systems - Facilitating the Construction of Bayesian Networks and Argumentation Frameworks
- 19 Roberto Verdecchia (VU), Architectural Technical Debt: Identification and Management
- 20 Masoud Mansoury (TU/e), Understanding and Mitigating Multi-Sided Exposure Bias in Recommender Systems
- 21 Pedro Thiago Timbó Holanda (CWI), Progressive Indexes
- 22 Sihang Qiu (TUD), Conversational Crowdsourcing
- 23 Hugo Manuel Proença (LIACS), Robust rules for prediction and description
- 24 Kaijie Zhu (TUE), On Efficient Temporal Subgraph Query Processing
- 25 Eoin Martino Grua (VUA), The Future of E-Health is Mobile: Combining AI and Self-Adaptation to Create Adaptive E-Health Mobile Applications
- 26 Benno Kruit (CWI & VUA), Reading the Grid: Extending Knowledge Bases from Human-readable Tables
- 27 Jelte van Waterschoot (UT), Personalized and Personal Conversations: Designing Agents Who Want to Connect With You
- 28 Christoph Selig (UL), Understanding the Heterogeneity of Corporate Entrepreneurship Programs

2022

-
- 1 Judith van Stegeren (UT), Flavor text generation for role-playing video games
 - 2 Paulo da Costa (TU/e), Data-driven Prognostics and Logistics Optimisation: A Deep Learning Journey
 - 3 Ali el Hassouni (VUA), A Model A Day Keeps The Doctor Away: Reinforcement Learning For Personalized Healthcare

- 4 Ünal Aksu (UU), A Cross-Organizational Process Mining Framework
 - 5 Shiwei Liu (TU/e), Sparse Neural Network Training with In-Time Over-Parameterization
 - 6 Reza Refaei Afshar (TU/e), Machine Learning for Ad Publishers in Real Time Bidding
 - 7 Sambit Praharaj (OU), Measuring the Unmeasurable? Towards Automatic Co-located Collaboration Analytics
 - 8 Maikel L. van Eck (TU/e), Process Mining for Smart Product Design
 - 9 Oana Andreea Inel (VUA), Understanding Events: A Diversity-driven Human-Machine Approach
 - 10 Felipe Moraes Gomes (TUD), Examining the Effectiveness of Collaborative Search Engines
 - 11 Mirjam de Haas (UT), Staying engaged in child-robot interaction, a quantitative approach to studying preschoolers' engagement with robots and tasks during second-language tutoring
 - 12 Guanyi Chen (UU), Computational Generation of Chinese Noun Phrases
 - 13 Xander Wilcke (VUA), Machine Learning on Multimodal Knowledge Graphs: Opportunities, Challenges, and Methods for Learning on Real-World Heterogeneous and Spatially-Oriented Knowledge
 - 14 Michiel Overeem (UU), Evolution of Low-Code Platforms
 - 15 Jelmer Jan Koorn (UU), Work in Process: Unearthing Meaning using Process Mining
 - 16 Pieter Gijsbers (TU/e), Systems for AutoML Research**
-