AutoMLBench: A Comprehensive Experimental Evaluation of Automated Machine Learning Frameworks

Hassan Eldeeb
Mohamed Maher
Radwa El Shawi
Sherif Sakr

Data Systems Group, Institute of Computer Science
University of Tartu, Tartu, 51009, Estonia

Abstract

With the booming demand for machine learning applications, it has been recognized that the number of knowledgeable data scientists cannot scale with the growing data volumes and application needs in our digital world. In response to this demand, several automated machine learning (AutoML) frameworks have been developed to fill the gap of human expertise by automating the process of building machine learning pipelines. Each framework comes with different heuristics-based design decisions. In this study, we present a comprehensive evaluation and comparison of the performance characteristics of six popular AutoML frameworks, namely, AutoWeka, AutoSKlearn, TPOT, Recipe, ATM and SmartML across 100 data sets from established AutoML benchmark suites. Our experimental evaluation considers different aspects for its comparison, including the performance impact of several design decisions, including time budget, size of search space, meta-learning, and ensemble construction. The results of our study reveal various interesting insights that can significantly guide and impact the design of AutoML frameworks.

1. Introduction

We are witnessing tremendous interest in artificial intelligence applications across governments, industries and research communities with a yearly cost of around 12.5 billion US dollars (International Data Corporation, 2017). The driver for this interest is the advent and increasing popularity of machine learning (ML) and deep learning (DL) techniques. The rise of generated data from different sources, processing capabilities, and ML algorithms opened the way for adopting ML in a wide range of real-world applications (Zomaya & Sakr, 2017). This situation is increasingly contributing towards a potential data science crisis, similar to the software crisis (Fitzgerald, 2012), due to the crucial need to have an increasing number of data scientists with solid knowledge and good experience so that they can keep up with harnessing the power of the massive amounts of data produced daily. Thus, we are witnessing a growing interest in automating the process of building ML pipelines where the presence of a human in the loop can be dramatically reduced. Research in the area of AutoML aims to alleviate both the computational cost and human expertise required for developing ML pipelines through automation with efficient algorithms. In particular, AutoML techniques enable the widespread use of ML techniques by domain experts and non-technical users.

Applying ML to real-world problems is a multi-stage process and highly iterative exploratory process. It aims to automatically produce the optimal ML pipeline that maximizes the predictive performance over the validation set of a dataset within a fixed computational budget(See Figure
The problem of AutoML can be formally stated as follows: For $i = 1, \ldots, n + m'$, let $x_i \in \mathbb{R}$ denote a feature vector and $y_i \in Y$ the corresponding target value. Given a training dataset $D_{\text{train}} = \{(x_1, y_1), \ldots, (x_{n'}, y_{n'})\}$ and the feature vectors $x_{n'+1}, \ldots, x_{n'+m'}$ of a test dataset $D_{\text{test}} = \{(x_{n'+1}, y_{n'+1}), \ldots, (x_{n'+m'}, y_{n'+m'})\}$ drawn from the same underlying data distribution, as well as a resource budget $b$ and a loss metric $L(., .)$, the AutoML problem is to automatically produce test set predictions $\hat{y}_{n'+1}, \ldots, \hat{y}_{n'+m'}$. The loss of a solution $\hat{y}_{n'+m'}$ to the AutoML problem is given by $\frac{1}{m'} \sum_{j=1}^{m'} L(\hat{y}_{n'+j}, y_{n'+j})$.

The budget $b$ would comprise computational resources (e.g., CPU and/or wallclock time, memory usage). In particular, solving the AutoML problem aims to select and tune an ML algorithm from a defined search space to achieve (near)-optimal performance in terms of the user-defined evaluation metric (e.g., accuracy, sensitivity, specificity, F1-score) within the user-defined budget for the search process, as shown in Figure 1. Additionally, different AutoML frameworks consider various design decisions. For example, SmartML (Maher & Sakr, 2019) adopts a meta-learning based mechanism to improve the performance of the automated search process by starting with the most promising classifiers that performed well with similar datasets in the past. Another example, AutoSKlearn (Feurer, Klein, Eggensperger, Springenberg, Blum, & Hutter, 2015) employs an option to take a weighted average of the predictions of an ensemble composed of the top trained models during the optimization process. Auto-Tuned Models (ATM) (Swearingen, Drevo, Cyphers, Cuesta-Infante, Ross, & Veeramachaneni, 2017) restricts the default search space into only three classifiers, namely, decision tree, K-nearest neighbors, and logistic regression. Nevertheless, there is no clear understanding of the impact of various design decisions of the different AutoML frameworks on the performance of the output pipeline. In this work, we aim to answer the following four questions:

(1) What is the impact of the time budget on the performance of different AutoML frameworks? Given more time budget, can AutoML frameworks guarantee consistent performance improvement?
(2) What is the impact of the search space size of the AutoML framework on the performance? How does limiting the search space to a predefined portfolio affect the predictive performance?

(3) Does meta-learning always yield a consistent performance improvement across different time budgets? Is there a relationship between the characteristics of the datasets and the improvement caused by employing the meta-learning version of the AutoML framework?

(4) Does ensemble construction yield better performance than single learners across different time budgets? Is there a relationship between the characteristics of the datasets and the improvement caused by employing the ensembling version of the AutoML framework?

This work is an extension of our initial work (Eldeeb, Matsuk, Maher, Eldallal, & Sakr, 2021) that mainly focused on studying the impact of different design decisions on the performance of AutoSKlearn. More specifically, in this work, we follow a holistic approach to design and conduct a comparative study of six AutoML frameworks, namely AutoWeka (Kotthoff, Thornton, Hoos, Hutter, & Leyton-Brown, 2017), AutoSKlearn, TPOT (Olson & Moore, 2016), Recipe (de Sá, Pinto, Oliveira, & Pappa, 2017), ATM and SmartML, focusing on comparing their general performance under various design decisions including time budget, size of search space, meta-learning and ensembling. For ensuring reproducibility as one of the main targets of this work, we provide access to the source codes and the detailed results for the experiments of our studies.¹

The remainder of this paper is organized as follows. The related work is reviewed in Section 2. Section 3 provides an overview of the evaluated frameworks included in our study. Section 4 describes our benchmark design. The evaluation of the general performance of the benchmark frameworks and the evaluation of the different design decisions on the performance of the benchmark frameworks are presented in Section 5. We discuss the results and future direction in Section 6 before we finally conclude the paper in Section 7.

2. Related Work

Recently, few research efforts have attempted to tackle the challenge of benchmarking different AutoML frameworks (Gijsbers, LeDell, Thomas, Poirier, Bischl, & Vanschoren, 2019; He, Zhao, & Chu, 2019; Shawi, Maher, & Sakr, 2019; Truong, Walters, Goodsitt, Hines, Bruss, & Farivar, 2019; Zöller & Huber, 2021). In general, most experimental evaluation and comparison studies show no framework always performed the best, as some trade-offs always need to be considered and optimized according to user-defined objectives. For example, Gijsbers et al. (Gijsbers, Bueno, Coors, LeDell, Poirier, Thomas, Bischl, & Vanschoren, 2022) conducted a study to compare the performance of 9 AutoML frameworks, namely, Autogluon-tabular (Erickson, Mueller, Shirkov, Zhang, Larroy, Li, & Smola, 2020), AutoSKlearn, AutoSKlearn 2 (Feurer, Eggensperger, Falkner, Lindauer, & Hutter, 2020), FLAML (Wang, Wu, Weimer, & Zhu, 2021), GANA (Gijsbers & Vanschoren, 2020), H2O AutoML (LeDell & Poirier, 2020), LightAutoMLs (Vakhreshev, Ryzhkov, Savchenko, Simakov, Damdinov, & Tuzhilin, 2021), MLjar (Płońska & Płoński, 2021), TPOT, across 71 classification and 33 regression tasks. The study includes techniques for comparing AutoML frameworks, including final model accuracy, inference time trade-offs, and failure analysis. Autogluon has a consistently higher average performance in this benchmark. Additionally, an interactive visualization tool is supported to explore further the results and reproducibility of the analyses performed. Gijsbers et al. (Gijsbers et al., 2019) have conducted an experimental study to compare the performance of 4 AutoML frameworks, namely, AutoWeka, AutoSKlearn, TPOT

¹.https://datasystemsgrouput.github.io/AutoMLBench/
and H2O on 39 datasets across two time budgets (60 minutes and 240 minutes). The results showed that no single AutoML framework outperformed others across all time budgets. Surprisingly, on some datasets, none of the frameworks outperformed the Random Forest model within 4 hours time budget. Truong et al. (Truong et al., 2019) compared the performance of 7 AutoML frameworks, namely, H2O, Auto-keras (Jin, Song, & Hu, 2019), AutoSKlearn, Ludwig, Darwin, TPOT and Auto-ml on 300 datasets across different time budgets. The results showed that no single framework outperformed all others on a plurality of tasks. Across the various evaluations and benchmarks, H2O, Auto-keras and AutoSKlearn performed better than the rest of the frameworks. In particular, H2O slightly outperformed other frameworks for binary classification and regression tasks while achieving poor performance on multi-class classification tasks. Auto-keras showed a stable performance across all tasks and slightly outperformed other frameworks on multi-class classification tasks while achieving poor performance on binary classification tasks.

Zöller and Huber (Zöller & Huber, 2021) compared the performance of different optimization techniques, namely, Grid Search, Random Search, ROBust Bayesian Optimization (ROBO) (Klein, Falkner, Mansur, & Hutter, 2017), Bayesian Tuning and Bandits (BTB) (Smith, Sala, Kanter, & Veeramachaneni, 2020), hyperopt (Bergstra, Yamins, & Cox, 2013b), SMAC (Hutter, Hoos, & Leyton-Brown, 2011), BOHB (Falkner, Klein, & Hutter, 2018) and Optunity (Smith et al., 2020). The results showed that all optimization techniques achieved comparable performance, and a simple search algorithm such as random search did not perform worse than other techniques. Thus, the study suggested that ranking optimization techniques on pure performance measures are not reasonable, and other aspects like scalability should also be considered. The study also compared the performance of 5 AutoML frameworks, namely, TPOT, hpsklearn (Komer, Bergstra, & Eliasmith, 2014), AutoSKlearn, ATM, and H2O on 73 real datasets. The study considered AutoSKLearn once with the default optimizer SMAC and once replacing SMAC with the random search while ensemble building and meta-learning options are disabled. The comparison results showed that, on average, all AutoML frameworks performed quite similar with a maximum performance difference of 2.2%.

To the best of our knowledge, our study is the first to investigate the impact of different AutoML design decisions on predictive performance. We benchmark six open-source, centralized, and distributed AutoML frameworks, namely, AutoWeka, AutoSKlearn, TPOT, Recipe, ATM and SmartML on 100 datasets from established AutoML benchmark suites. Differently from the previous benchmark studies focused only on comparing the performance of different AutoML frameworks, we take a holistic approach to studying the impact of various design decisions, including the size of the search space, time budget, meta-learning, and ensembling construction on the performance of the AutoML frameworks.

3. AutoML Frameworks

This section provides an introduction to the evaluated AutoML frameworks used in this study in terms of popularity (measured in terms of the number of stars on GitHub), ML tool-box used, optimization technique, whether they use meta-learning to learn from previous experience, whether they perform post-processing (e.g., ensemble construction), whether they use Graphical User Inter-

---

2. https://github.com/uber/ludwig
3. https://www.sparkcognition.com/product/darwin/
4. https://github.com/ClimbsRocks/auto_ml
Table 1: Comparison table of the functionality of the AutoML frameworks considered in this study as of 2/3/2023

| Framework | Release Date | Popularity (## of stars on GitHub) | Optimization Technique | ML Tool Box | Meta-Learning | Post-processing | GUI | Data Pre-processing |
|-----------|--------------|-----------------------------------|------------------------|-------------|---------------|----------------|-----|---------------------|
| AutoWeka  | 2013         | 312                               | Bayesian optimization  | Weka        | ×             | ✓              | ✓   | ✓                   |
| AutoSKlearn | 2015       | 6.8k                              | Bayesian optimization  | Scikit-Learn| ✓             | Ensemble selection | ×   | ✓                   |
| TPOT      | 2016         | 9k                                | Evolutionary optimization | Scikit-Learn| ×             | ×              | ×   | ×                   |
| Recipe    | 2017         | 49                                | Grammar-based genetic algorithm | Scikit-Learn| ×             | ×              | ×   | ×                   |
| ATM       | 2017         | 522                               | Distributed Random search & Tree-Parzen estimators | Scikit-Learn| ×             | ×              | ×   | ✓                   |
| SmartML   | 2019         | 23                                | Bayesian optimization  | mlr, RWeka & other R packages | ✓             | Voting ensembles | ×   | ✓                   |

AutoWeka is implemented in Java on top of Weka, a popular ML library with a wide range of ML algorithms. AutoWeka employs Bayesian optimization using SMAC (Hutter et al., 2011) and TPE (Bergstra, Yamins, & Cox, 2013a) for algorithm selection and hyperparameter tuning. In particular, SMAC draws the relationship between algorithm performance and a given set of hyperparameters by estimating the predictive mean and variance of their performance along with the trees of a random forest model. TPE is a robust technique that separates low-performing parameter configurations from the best-performing ones.

AutoSKlearn is a tool for automating the process of building ML pipelines for classification and regression tasks. AutoSKlearn is implemented on top of Scikit-Learn (Buitinck, Louppe, Blondel, Pedregosa, Mueller, Grisel, Niculae, Prettenhofer, Gramfort, Grobler, Layton, VanderPlas, Joly, Holt, & Varoquaux, 2013), a popular Python ML package, and uses SMAC for algorithm selection and hyperparameter tuning. AutoSKlearn uses meta-learning to initialize the optimization procedure. Additionally, ensemble selection is implemented by combining the best pipelines to improve the performance of the output model. AutoSKlearn supports different execution options including the *vanilla* version (AutoSKlearn-v), the meta-learning version (AutoSKlearn-m), the ensembling selection version (AutoSKlearn-e), and the full version (AutoSKlearn), where all options are enabled.

TPOT is an AutoML framework for building classification and regression pipelines based on genetic algorithm. ML pipelines can be expressed as a computational graph, with different branches representing different preprocesing pipelines. These pipelines are then optimized using a multi-objective optimization technique to minimize pipeline complexity while optimizing for performance to reduce overfitting caused by the large search space (Olson, Bartley, Urbanowicz, & Moore, 2016).

Recipe is an AutoML framework for building machine learning pipelines for classification tasks. Recipe follows the same optimization procedure as TPOT, exploiting the advantages of a
global search. TPOT suffers from the unconstrained search problem in which resources can be spent on generating and evaluating invalid solutions. Recipe handles this problem by adding a grammar that reduces the generation of invalid pipelines and hence accelerates the optimization process.

ATM is a collaborative service for optimizing ML pipelines for classification tasks. In particular, ATM supports parallel execution through multiple nodes/cores with a shared model hub storing the results out of these executions and improving the selection of pipelines that may outperform the currently chosen ones. ATM is based on a hybrid Bayesian and multi-armed bandit optimization technique to traverse the search space and report the target pipeline.

SmartML is the first AutoML R package for classification tasks. In the algorithm selection phase, SmartML employs a meta-learning approach to identify the best-performing algorithms on similar datasets. The hyperparameter tuning of SmartML is based on SMAC. SmartML maintains the results of the new runs to continuously enrich its knowledge base to further improve the performance and robustness of future runs. SmartML supports two execution options which are the base version SmartML-m that employs meta-leaning for warm-starting, and the ensemble version SmartML-e that additionally employs a voting ensemble mechanism.

4. Benchmark Design

Each benchmark task consists of a dataset, a metric to optimize, and design decisions made by the user, including a specific time budget to use. We will briefly explain our choice for each.

Datasets We used 100 datasets collected from the popular OpenML repository (Vanschoren, van Rijn, Bischl, & Torgo, 2013), allowing users to query data for different use cases. Detailed descriptions of the datasets used in this study are given in Table 8 in Appendix A. To evaluate the AutoML frameworks on a variety of dataset characteristics, we selected multiple datasets according to different criteria, including the number of classes, number of features, number of instances, number of categorical features per sample, number of instances with missing values, and the class entropy. The datasets represent a mix of binary (50%) and multiclass (50%) classification tasks, where the size of the largest dataset is 643MB.

Performance metrics The benchmark can be run with a wide range of measures per user’s choice. The reported results in this paper are based on F1-score. AutoML frameworks are optimized for the same metric they are evaluated on. The measures are estimated with hold-out validation; each dataset is partitioned into two parts, 70% for training and 30% for testing. All AutoML frameworks are applied to the same training and testing splits on all datasets. To eliminate the effects of non-deterministic factors, the performance reported in each experiment is based on an average of 10 trials. We report a performance of 0 for any framework if the number of failed trials exceeds or is equal to 5.

Frameworks and design decisions The frameworks considered in this paper are selected based on ease of use, variety of underlying optimization techniques and ML toolboxes, popularity measured by the number of stars on GitHub, and citation count. All frameworks considered in this work are open source. A reference to the source code of each framework is given in Table 9 in Appendix B. We do plan to include more frameworks in future work. For AutoSKlearn, we consider four execution options; AutoSKlearn-v, AutoSKlearn-m, AutoSKlearn-e and AutoSKlearn. For SmartML, we consider two execution options including SmartML-m, and SmartML-e. We examined different design decisions, including the size of the search space, meta-learning, and ensemble construction as a post-processing step. We study the impact of these design
decisions for only AutoML frameworks that support configuring these decisions. It is important to highlight that the optimization technique is not consistent among all the frameworks, which prevents drawing a clear conclusion about this point in this benchmark. We consider the following versions of the frameworks: AutoSKLearn 0.11.0, AutoWeka 2.5, TPOT 0.11.6, Recipe 1.0, ATM 0.2.2, and SmartML 0.2.

**Baseline method** To assess the effectiveness of the different AutoML frameworks included in this work, we use a baseline method which is a simple pipeline consisting of an imputation of missing values and a random forest model (Pedregosa, Varoquaux, Gramfort, Michel, Thirion, Grisel, Blondel, Prettenhofer, Weiss, Dubourg, Vanderplas, Passos, Cournapeau, Brucher, Perrot, & Duchesnay, 2011).

**Time budget choice** All AutoML frameworks were used with four different time budgets. Each framework is limited by a soft time budget (10, 30, 60, and 240 minutes) and a hard one (10% more than soft time budget). If a framework exceeds the hard time budget, the run is terminated and considered failed. Setting a time budget for all experiments is not straightforward. While it is more favourable to un-set a time limit to guarantee the best performance for each framework, however, doing so for all the six evaluated frameworks, with different configurations, across the 100 datasets, with ten trails for each, is very time-consuming. Therefore we used four time-budgets which led to more than 40000 experiments to run for a total of more than 88366-hour EC2 run-time. To keep the experiment run-time and cost to practical limits, we tested the maximum cut-off timeouts of 4 and 8 hours on 14 randomly selected datasets. The results are reported in Table 10 in Appendix C. Additionally, the Wilcoxon signed-rank test was conducted to determine if a statistically significant difference in performance exists between the AutoML frameworks over the two-time budgets (See Table 10). The results confirm that the difference is not necessarily towards the 8-hour budget, and not statistically significant. Hence, the 8-hour budget is not further considered.

**Hardware choice and resource specifications** Our experiments were conducted on Google Cloud machines; each machine is configured with 2 vCPUs, 7.5 GB RAM and ubuntu-minimal-1804-bionic. Each machine uses Python 2.7.15, Python 3.6.8, scikit-learn 0.21.3, R 3.4.4, and Java 1.8. To avoid memory leakage, we have rebooted the machines after each run to ensure that each experiment has the same available memory size.

**5. Experimental Evaluation**

This section provides empirical evaluations of the different AutoML frameworks. We first compare the general performance of the different AutoML frameworks in Section 5.1. Next, we examine the impact of various design decisions on the performance of the different AutoML frameworks in Section 5.2.

**5.1 General Performance Evaluation**

In this section, we focus on evaluating and comparing the general performance of the benchmark frameworks. Our evaluation considers different aspects for its comparison, including (a) the number of successful runs, (b) the average performance of the final pipeline per AutoML framework across all datasets, (c) the significance of the performance difference between different frameworks across different time budgets, and (d) the robustness of the benchmark frameworks.

Figure 2(a) shows the number of datasets with successful runs of each framework on different time budgets. If an AutoML framework could not generate a model for a particular dataset
times or more, it is considered a failed experiment. Generally, the results show that increasing the time budget for the AutoML frameworks increases the number of successful runs. **AutoSKlearn** achieves the largest number of successful runs across all time budgets, as shown in Figure 2(a). Each of the different versions of **AutoSKlearn** successfully ran on 99 datasets across different time budgets. **SmartML-e** comes in second place in terms of the number of successful runs, followed by **AutoWeka** and **SmartML**. The genetic-based frameworks, **TPOT** and **Recipe**, come in the last place, as shown in Figure 2(a). For **Recipe** and **TPOT**, the number of successful runs achieved in the longest time budget, 240 minutes, is almost double that achieved for the smallest time budget of 10 minutes. Hence, larger budgets are preferable for **Recipe** and **TPOT**.

Figure 2: General performance trends of the benchmark AutoML frameworks.
Figure 3: Heatmaps show the number of datasets a given AutoML framework outperforms another in terms of predictive performance over different time budgets. Two frameworks are considered to have the same performance on a task if they achieve predictive performance with < 1% difference.

Figure 2(b) reports the performances of all AutoML frameworks averaged over all datasets over 240 minutes budget. It is apparent that all frameworks are able to outperform the random forest baseline on average. However, single results vary significantly. Figures 9 to 11 in Appendix D report the performance of all AutoML frameworks and the baseline across 10, 30, 60 minutes, respectively. We investigate pair-wise “outperformance” by calculating the number of datasets for which one framework outperforms another across different time budgets, shown in Figure 3. One framework outperforms another on a dataset if it has at least a 1% higher predictive performance.
Table 2: Wilcoxon pairwise test p-values for AutoML frameworks over different time budgets. Bold entries highlight significant differences \((p \leq 0.05)\). Highlighted entries in each row represent a given AutoML framework (row) outperforms another AutoML framework (column).

| Time Budget | AutoSKLearn-m | AutoSKLearn-v | AutoSKLearn-e | AutoWeka | Baseline | Recipe | SmartML-m | SmartML-e | TPOT |
|-------------|---------------|---------------|---------------|----------|----------|--------|-----------|-----------|------|
| 10 Minutes  | 0.0           | 0.0           | 0.0           | 0.0      | 0.0      | 0.0    | 0.0       | 0.0       | 0.0  |
|             | 0.0           | 0.0           | 0.0           | 0.0      | 0.0      | 0.0    | 0.0       | 0.0       | 0.0  |
|             | 0.0           | 0.0           | 0.0           | 0.0      | 0.0      | 0.0    | 0.0       | 0.0       | 0.0  |
|             | 0.0           | 0.0           | 0.0           | 0.0      | 0.0      | 0.0    | 0.0       | 0.0       | 0.0  |
|             | 0.0           | 0.0           | 0.0           | 0.0      | 0.0      | 0.0    | 0.0       | 0.0       | 0.0  |
|             | 0.0           | 0.0           | 0.0           | 0.0      | 0.0      | 0.0    | 0.0       | 0.0       | 0.0  |
|             | 0.0           | 0.0           | 0.0           | 0.0      | 0.0      | 0.0    | 0.0       | 0.0       | 0.0  |
|             | 0.0           | 0.0           | 0.0           | 0.0      | 0.0      | 0.0    | 0.0       | 0.0       | 0.0  |
|             | 0.0           | 0.0           | 0.0           | 0.0      | 0.0      | 0.0    | 0.0       | 0.0       | 0.0  |
|             | 0.0           | 0.0           | 0.0           | 0.0      | 0.0      | 0.0    | 0.0       | 0.0       | 0.0  |
|             | 0.0           | 0.0           | 0.0           | 0.0      | 0.0      | 0.0    | 0.0       | 0.0       | 0.0  |

For example, for the 10 minutes time budget, there are 2 datasets for which Recipe performs better than AutoSKlearn, despite being the overall worst- and best-ranked algorithms, respectively, as shown in Figure 3(a) On average, the results show that AutoSKlearn framework comes in the first place, outperforming other frameworks on the most significant number of datasets for different time budgets, followed by ATM framework, while Recipe comes in the last place, as shown in Figure 3 The Wilcoxon signed-rank test (Gehan, 1965) was conducted to determine if a statistically significant difference in performance exists between the AutoML frameworks including the baseline over different time budgets, the results of which are summarized in Table 2. The results representing a minimal threshold for performance improvement. In terms of “outperformance”, it is worth mentioning that no single AutoML framework performs best across all 100 datasets on all-time budgets.
show that all AutoML frameworks except Recipe statistically outperform the baseline across all time budgets with a significant difference. The results of the Wilcoxon test confirm the fact that there is no dominating winner, and the statistical significance in the performance difference among the AutoML frameworks can vary from one-time budget to another. The ensembling version and the full version of AutoSKLearn statistically outperform most of the other frameworks across all time budgets. The results show that SmartML-m, SmartML-e, and AutoWeka are statistically outperformed by the majority of the frameworks, as shown in Table 2. For longer time budgets of 60 and

Figure 4: Performance of the different AutoML frameworks based on the various characteristics of datasets and tasks over 240 minutes.
240 minutes, TPOT significantly outperforms AutoWeka, Recipe, SmartML-m, SmartML-e, AutoSKlearn-m, and AutoSKlearn-v.

We investigate the performance of the different AutoML frameworks based on the various characteristics of datasets and tasks. Figure 4 reports the mean performances of the AutoML frameworks on multi-class and binary-class classification tasks across 240 minutes budget. Notably, the improvement achieved by all AutoML on multi-class datasets is less significant than the average improvement on whole datasets. The second subgroup of datasets where autoML frameworks struggle to boost their performance contains datasets with a relatively large number of features and a small number of instances as shown in Figures 12 to 15 in Appendix D. These figures report the mean performance of the different AutoML frameworks on datasets with various characteristics, including a large number of instances and features, a small number of features and instances, a small number of features and a large number of instances, and a large number of features and a small number of instances. Additionally, we report the mean performance of all frameworks on binary classification tasks (See Figure 4(b) in Appendix D).

We test the robustness of the AutoML frameworks evaluated by the ability of the framework to achieve the same results across different runs on the same input dataset. For a randomly selected dataset, we run each AutoML framework for 10 different times on 10 minutes time budget. Figure 5 shows the robustness of the AutoML frameworks. The results show that the four versions of AutoSKlearn have the most stable runs, and Recipe and AutoWeka comes second. on contrast, the two versions of SmartML achieve the least stable runs.

5.2 Performance Evaluation of Different Design Decisions

In this section, we study the impact of different design decisions including time budget (Section 5.2.1), size of search space (Section 5.2.2), meta-learning (Section 5.2.3), and ensembling (Section 5.2.4) on the performance of the different AutoML frameworks across different time budgets. For each framework, the performance reported in each experiment is based on an average of 10 runs.
Table 3: Mean\textsubscript{succ}, Mean and standard deviation of the predictive performance of AutoML frameworks per time budget. Bold entries highlight highest Mean\textsubscript{succ}, mean and lowest standard deviation.

| Time Budget | Framework   | Mean\textsubscript{succ} | Mean | SD  | Time Budget | Framework   | Mean\textsubscript{succ} | Mean | SD  |
|-------------|-------------|---------------------------|------|-----|-------------|-------------|---------------------------|------|-----|
| 10 Min      | ATM         | 0.664                     | 0.886 | 0.120 | 60 Min      | ATM         | 0.700                     | 0.886 | 0.132 |
|             | AutoWeka    | 0.724                     | 0.842 | 0.165 |             | AutoWeka    | 0.743                     | 0.835 | 0.163 |
|             | Recipe      | 0.252                     | 0.764 | 0.221 |             | Recipe      | 0.568                     | 0.748 | 0.247 |
|             | AutoSKLearn-e | 0.859                   | 0.868 | 0.145 |             | AutoSKLearn-e | 0.870                   | 0.879 | 0.138 |
|             | AutoSKLearn-m | 0.855                 | 0.864 | 0.153 |             | AutoSKLearn-m | 0.861                   | 0.870 | 0.144 |
|             | AutoSKLearn-v | 0.853                 | 0.862 | 0.151 |             | AutoSKLearn-v | 0.861                   | 0.870 | 0.142 |
|             | AutoSKLearn  | 0.859                     | 0.868 | 0.152 |             | AutoSKLearn  | 0.868                     | 0.877 | 0.137 |
|             | SmartML-m    | 0.806                     | 0.799 | 0.212 |             | SmartML-m    | 0.790                     | 0.816 | 0.194 |
|             | SmartML-e    | 0.711                     | 0.831 | 0.176 |             | SmartML-e    | 0.726                     | 0.832 | 0.0172|
|             | TPOT         | 0.383                     | 0.890 | 0.121 |             | TPOT         | 0.620                     | 0.885 | 0.137 |
| 30 Min      | ATM         | 0.665                     | 0.899 | 0.121 | 240 Min     | ATM         | 0.768                     | 0.893 | 0.124 |
|             | AutoWeka    | 0.747                     | 0.839 | 0.166 |             | AutoWeka    | 0.771                     | 0.838 | 0.166 |
|             | Recipe      | 0.516                     | 0.748 | 0.254 |             | Recipe      | 0.645                     | 0.759 | 0.248 |
|             | AutoSKLearn-e | 0.866               | 0.875 | 0.141 |             | AutoSKLearn-e | 0.874                   | 0.883 | 0.132 |
|             | AutoSKLearn-m | 0.859               | 0.868 | 0.152 |             | AutoSKLearn-m | 0.864                   | 0.873 | 0.141 |
|             | AutoSKLearn-v | 0.858               | 0.867 | 0.149 |             | AutoSKLearn-v | 0.850                   | 0.867 | 0.156 |
|             | AutoSKLearn  | 0.862                     | 0.871 | 0.148 |             | AutoSKLearn  | 0.875                     | 0.884 | 0.132 |
|             | SmartML-m    | 0.804                     | 0.808 | 0.199 |             | SmartML-m    | 0.798                     | 0.826 | 0.169 |
|             | SmartML-e    | 0.727                     | 0.838 | 0.159 |             | SmartML-e    | 0.735                     | 0.840 | 0.165 |
|             | TPOT         | 0.518                     | 0.878 | 0.144 |             | TPOT         | 0.790                     | 0.888 | 0.131 |

5.2.1 IMPACT OF TIME BUDGET

Tuning the time budget is a crucial and challenging task in AutoML, as it requires a balance between the available computational resources and the desired level of performance. It is a task that involves careful consideration of trade-offs between generalization and over-fitting of the AutoML frameworks. We investigate the impact of time budget on the performance of various AutoML frameworks, examining the speed at which they can generate ML pipelines and their ability to consistently improve performance given more time. We assess each framework’s performance on successful runs under four different time budgets: 10, 30, 60, and 240 minutes. Table 3 presents the mean (Mean) and standard deviation (SD) of performance for all successful runs at each time budget. Furthermore, we report the mean predictive performance weighted by the percentage of successful runs (Mean\textsubscript{succ}).

\[
\text{Mean}_{\text{succ}} = \text{Mean} \times \frac{N}{T}
\]

where \( N \) is the number of successful runs and \( T \) is the total number of runs.

The results show that for the 10 and 240 minutes budgets, AutoSKlearn and AutoSKlearn-e have comparable Mean\textsubscript{succ}, while AutoSKlearn-e has the highest Mean\textsubscript{succ} over the rest of time budgets. In contrast, Recipe achieves the lowest mean performance and Mean\textsubscript{succ} over all time budgets, as shown in Table 3. Notably, the performance of genetic-based tools, i.e., Recipe and TPOT, improves over time as the Mean\textsubscript{succ} values show. Figures 18 to 27 in Appendix G show the impact of increasing the time budget for each AutoML framework on 100 datasets.

Extended time budgets do not necessarily lead to better performance, in contrast to the prior assumptions, as shown in Table 4. We report the gain (\( g \)) or loss (\( l \)) in the predictive performance of the frameworks when increasing the time budget. The gain is measured by the mean and maximum predictive performance improvement over all improved/declined datasets. When in-
Table 4: Summary of the impact of increasing the time budget. Bold entries highlight the highest mean gain, highest maximum gain, smallest mean loss, smallest maximum loss, and maximum and minimum number of datasets with gain > 1 and loss > 1, respectively.

| Time Budget in minutes | Framework     | Gain ($g$) | #datasets with | Loss ($l$) |
|------------------------|---------------|------------|----------------|------------|
|                        |               | Mean       | $g > 1\%$ | $g \approx 0\%$ | $l > 1\%$ | Mean | Max |
| 10 → 30                | ATM           | 4.3        | 21.0         | 19         | 33         | 15 | -4.7 | -21.4 |
|                        | AutoWeka      | 5.8        | 20.1         | 16         | 62         | 7  | -3.8 | -11.9 |
|                        | Recipe        | 19.3       | 36.6         | 2          | 17         | 2  | -4.4 | -6.2  |
|                        | AutoSKLearn-e | 3.6        | 13.5         | 24         | 67         | 8  | -2.7 | -6.8  |
|                        | AutoSKLearn-m | 3.6        | 13.3         | 21         | 65         | 13 | -2.6 | -10.7 |
|                        | AutoSKLearn-v | 3.9        | 22.1         | 23         | 63         | 13 | -3.1 | -7.4  |
|                        | AutoSKLearn   | 3.5        | 15.2         | 17         | 72         | 10 | -2.6 | -4.8  |
|                        | SmartML-m     | 8.0        | 33.3         | 13         | 64         | 11 | -3.6 | -8.3  |
|                        | SmartML-e     | 7.9        | 85.2         | 18         | 67         | 11 | -6.3 | -16.9 |
|                        | TPOT          | 6.2        | 17.0         | 7          | 29         | 4  | -2.2 | -3.7  |
| 30 → 60                | ATM           | 5.0        | 16.5         | 15         | 34         | 20 | -6.7 | -28.6 |
|                        | AutoWeka      | 9.1        | 66.6         | 14         | 61         | 13 | -9.6 | -56.7 |
|                        | Recipe        | 4.7        | 17.2         | 6          | 58         | 3  | -19.2 | -29.1 |
|                        | AutoSKLearn-e | 4.9        | 19.6         | 16         | 66         | 17 | -2.2 | -3.7  |
|                        | AutoSKLearn-m | 4.9        | 23.4         | 16         | 67         | 16 | -4.1 | -13.3 |
|                        | AutoSKLearn-v | 4.1        | 14.2         | 23         | 62         | 14 | -4.6 | -13.9 |
|                        | AutoSKLearn   | 4.1        | 32.0         | 22         | 65         | 12 | -2.4 | -6.8  |
|                        | SmartML-m     | 12.2       | 40.0         | 10         | 73         | 6  | -6.2 | -18.3 |
|                        | SmartML-e     | 6.5        | 18.2         | 21         | 54         | 20 | -9.0 | -84.3 |
|                        | TPOT          | 3.7        | 8.7          | 6          | 42         | 8  | -2.8 | -7.7  |
| 60 → 240               | ATM           | 5.6        | 31.1         | 21         | 39         | 17 | -3.4 | -12.0 |
|                        | AutoWeka      | 4.1        | 8.7          | 17         | 61         | 8  | -3.8 | -11.5 |
|                        | Recipe        | 13.5       | 38.2         | 4          | 69         | 2  | -20.5 | -40.0 |
|                        | AutoSKLearn-e | 4.3        | 39.0         | 21         | 62         | 16 | -3.8 | -12.7 |
|                        | AutoSKLearn-m | 4.0        | 13.3         | 20         | 59         | 20 | -2.7 | -6.0  |
|                        | AutoSKLearn-v | 3.6        | 12.5         | 22         | 63         | 13 | -8.7 | -25.3 |
|                        | AutoSKLearn   | 4.8        | 36.5         | 22         | 63         | 14 | -3.2 | -9.9  |
|                        | SmartML-m     | 10.6       | 59.6         | 19         | 59         | 10 | -6.6 | -19.4 |
|                        | SmartML-e     | 9.1        | 22.3         | 23         | 53         | 19 | -6.9 | -18.8 |
|                        | TPOT          | 2.6        | 5.6          | 18         | 47         | 5  | -4.1 | -7.7  |

Increasing the time budget from 10 to 30 minutes, Recipe achieves the highest mean gain of 19.3 on 2 datasets, followed by SmartML-m, while AutoSKlearn comes in the last place achieving a mean gain of 3.5 on 17 datasets. It is noticeable that Recipe has the smallest number of datasets that witnessed performance improvement and performance degradation when increasing the time budget. AutoSKlearn-v have the largest number of datasets that witnessed performance improvement when increasing the time budget from 30 to 60 minutes and from 60 to 240 minutes, while AutoSKlearn-e witnessed performance improvement across the largest number of datasets when increasing the time budget from 10 to 30 minutes. In contrast, ATM has the most significant number of datasets with performance degradation when increasing the time budget from 10 to 30 minutes and from 30 to 60 minutes.
### Table 5: Wilcoxon test p-values for all the AutoML frameworks over different time budgets. Bold entries highlight significant difference.

| Framework | Time Budget 1 | Time Budget 2 | Avg. Acc. Diff | P-value | Framework | Time Budget 1 | Time Budget 2 | Avg. Acc. Diff | P-value |
|-----------|--------------|--------------|----------------|---------|-----------|--------------|--------------|----------------|---------|
| AutoWeka  | 30           | 10           | 0.008          | 0.016   | AutoSKLearn | 30           | 10           | 0.003          | 0.338   |
|           | 60           | 10           | 0.003          | 0.834   |           | 60           | 10           | 0.010          | 0.001   |
|           | 60           | 30           | 0.000          | 0.885   |           | 60           | 30           | 0.007          | 0.034   |
|           | 240          | 10           | 0.009          | 0.000   |           | 240          | 10           | 0.016          | 0.004   |
|           | 240          | 30           | 0.005          | 0.039   |           | 240          | 30           | 0.011          | 0.018   |
|           | 240          | 60           | 0.005          | 0.042   |           | 240          | 60           | 0.008          | 0.125   |
| TPOT      | 30           | 10           | 0.009          | 0.717   |           | 30           | 10           | 0.005          | 0.175   |
|           | 60           | 10           | 0.008          | 0.588   |           | 60           | 10           | 0.008          | 0.001   |
|           | 60           | 30           | 0.001          | 0.428   |           | 60           | 30           | 0.003          | 0.088   |
|           | 240          | 30           | 0.013          | 0.016   |           | 240          | 30           | 0.005          | 0.008   |
|           | 240          | 60           | 0.006          | 0.035   |           | 240          | 30           | 0.000          | 0.040   |
| Recipe    | 30           | 10           | 0.014          | 0.866   |           | 30           | 10           | 0.008          | 0.000   |
|           | 60           | 10           | -0.002         | 0.955   |           | 60           | 10           | 0.012          | 0.000   |
|           | 60           | 30           | -0.004         | 0.535   |           | 60           | 30           | 0.004          | 0.904   |
|           | 240          | 10           | 0.023          | 0.093   |           | 240          | 10           | 0.015          | 0.000   |
|           | 240          | 30           | 0.003          | 0.067   |           | 240          | 30           | 0.007          | 0.038   |
|           | 240          | 60           | 0.002          | 0.345   |           | 240          | 60           | 0.003          | 0.291   |
| ATM       | 30           | 10           | 0.001          | 0.583   |           | 30           | 10           | 0.004          | 0.156   |
|           | 60           | 10           | -0.007         | 0.254   |           | 60           | 10           | 0.006          | 0.105   |
|           | 60           | 30           | -0.008         | 0.497   |           | 60           | 30           | 0.002          | 0.873   |
|           | 240          | 10           | 0.003          | 0.585   |           | 240          | 10           | 0.009          | 0.210   |
|           | 240          | 30           | -0.001         | 0.799   |           | 240          | 30           | 0.004          | 0.920   |
|           | 240          | 60           | 0.008          | 0.394   |           | 240          | 60           | 0.003          | 0.660   |
| SmartML-m | 30           | 10           | 0.007          | 0.636   |           | 30           | 10           | 0.008          | 0.521   |
|           | 60           | 10           | 0.009          | 0.832   |           | 60           | 10           | 0.003          | 0.584   |
|           | 60           | 30           | 0.009          | 0.597   |           | 60           | 30           | 0.004          | 0.674   |
|           | 240          | 10           | 0.026          | 0.121   |           | 240          | 10           | 0.011          | 0.092   |
|           | 240          | 30           | 0.025          | 0.050   |           | 240          | 30           | 0.004          | 0.182   |
|           | 240          | 60           | 0.015          | 0.071   |           | 240          | 60           | 0.008          | 0.305   |

The Wilcoxon signed-rank test is conducted to determine if the average performance difference when extending the time budget is statistically significant, as shown in table 5. The impact of increasing the time budget varies from one framework to another. For example, AutoSKLearn-m, Recipe, ATM, SmartML-m and SmartML-e does not witness significant performance difference. While in most of the cases of AutoWeka, TPOT and all versions of AutoSKLearn except AutoSKLearn-m, the differences are statistically significant. These results show that end-users should always carefully consider the trade-off between time budget and performance for the benchmark frameworks based on their specific goals.

#### 5.2.2 Impact of the Size of Search Space

Search space defines the structural paradigm that the different optimization methods can explore; thus, designing a good search space is a vital but challenging problem. Figure 6 provides an overview of the most frequent ML models commonly used by the different AutoML frameworks. By analyzing the returned best-performing models, it is notable that there is no single ML algorithm that dominates all AutoML frameworks; however, it is apparent the tree-based models are the most frequent across all frameworks for all time budgets. For example, the returned pipelines by AutoWeka, AutoSKLearn-v, and SmartML-m show that random forest is the most frequently used classifier, as shown in Figures 6(a) 6(c) and 6(e). The most frequent classifier for AutoSKLearn-m, TPOT, and Recipe is gradient boosting, as shown in Figures 6(d) 6(f) and 6(g) respectively. To efficiently utilize the time budget, ATM limits its default search space to only
Figure 6: The frequency of using different machine learning models by the different AutoML frameworks.
Figure 7: The impact of using a static portfolio on each AutoML framework. Green markers represent better performance with FC search space, blue markers represent comparable performance with a difference less than 1%, red markers represent better performance with 3C search space, yellow markers on the left represent failed runs with FC but successful with 3C, yellow markers on the right represent failed runs with 3C but successful with FC, and yellow markers in the middle represent failed runs with both FC and 3C.

Finding an optimal solution to the time-bounded optimization problem of AutoML requires defining the underlying search space and searching for well-performing ML pipelines as efficiently as possible. Often these search spaces are chosen arbitrarily without any validation, sometimes leading to bloated spaces and the inability to find optimal results (Zöller & Huber, 2021). In the following, we examine the impact of a budget allocation strategy as a complementary design decision for AutoML frameworks. The strategy is based on using a static portfolio (Kotthoff, 2016) – a set of configurations that covers as many diverse datasets as possible and minimizes the risk of failure when facing a new task. So, we construct a portfolio consisting of the top three performing classifiers over the 100 datasets and supported by all AutoML frameworks. These classifiers are support vector machine, random forest, and decision tree. Then for a dataset at hand, all algorithms in this portfolio based on different hyperparameters are evaluated. For the AutoML frameworks included in this work that allows configuring the search space, ATM, AutoSKlearn, and TPOT, we compare the performance of using the full search space including all available classifiers (FC) to the performance when using the static portfolio (3C) on 30 minutes time budget, the results of which are summarized in Figure 7.
For AutoSKlearn, the results show that the performance of the FC outperforms 3C on 28 datasets with an average predictive performance gain of 3.3%. However, the performance achieved using the 3C outperforms that achieved using the FC on 21 datasets by 5.9%, as shown in Figure 7(a). This performance discrepancy is attributed to the AutoML framework’s focus on tuning classifiers that have yielded good performance, thereby evaluating more hyperparameters of these classifiers. Hence, AutoML frameworks concentrate on promising regions in the search space while disregarding unimportant ones. The performance of both FC and 3C is comparable on 50 datasets, with predictive performance differences of less than 1%. For TPOT, 23 datasets failed to run using the 3C, while 20 failed using the FC. Both search spaces failed to produce results for 12 datasets, as shown in Figure 7(b). For successful runs, the FC outperformed the 3C on 21 datasets, with an average predictive performance improvement of 9.6%. In contrast, the performance of both search spaces was comparable on 18 datasets. Notably, the 3C search space achieved better performance than the FC search space on six datasets, with an average predictive performance difference of 8.8%. For ATM, the 3C outperformed the FC search space on 17 datasets, with an average predictive performance improvement of 4%. In contrast, the FC search space outperformed the 3C search space on 15 datasets, with an average performance improvement of 9.3%. Both search spaces achieved comparable performance on 22 datasets, as depicted in Figure 7(c). Notably, the FC failed to produce results for 19 datasets in which the 3C search space succeeded. In contrast, the 3C failed for ten datasets that the FC was successful. The Wilcoxon signed-rank test was conducted to determine if a statistically significant difference in performance exists between the FC and 3C across all datasets. For TPOT, the test results show that the difference in performance between the two search spaces is statistically significant with more than 95% level of confidence (p-value=0.003). However, no statistically significant difference in performance exists between the two search spaces for AutoSKlearn and ATM.

5.2.3 IMPACT OF META-LEARNING

One way to define meta-learning is the process of learning from previous experience gained during applying various learning algorithms on different ML tasks, reducing the time needed to learn new tasks (Vanschoren, 2018). In the following, we study the impact of meta-learning on the performance of AutoML frameworks. The only framework that supports configuring meta-learning is AutoSKlearn. Furthermore, we investigate the relationship between the characteristics of the different datasets and the improvement caused by employing the vanilla version or the meta-learning version of the AutoSKlearn.

AutoSKlearn applies a meta-learning mechanism based on a knowledge base storing the meta-features of datasets as well as the best-performing pipelines on these datasets. AutoSKlearn uses 38 meta-features, including statistical, information-theoretic and simple meta-features. In an offline phase, the meta-features and the empirically best-performing pipelines are stored for each dataset in their repository (140 datasets from the OpenML repository). In an online phase, for any new dataset, the framework extracts the meta-features of the new dataset and searches for the most similar datasets in the knowledge base. It returns the top $k$ best-performing pipelines on these similar datasets. These $k$ pipelines are used as a warm start for the Bayesian optimization algorithm used in the optimization process. To assess the impact of the meta-learning mechanism, we compare the performance of AutoSKlearn-v and AutoSKlearn-m on 100 datasets across different time budgets, as shown in Figure 8. The results show that using meta-learning is not
necessarily associated with performance improvement. On average, the performance of the vanilla and the meta-learning versions is very comparable on the 4 time budgets. In particular, both versions perform similarly on 64, 55, 65, and 69 datasets for 10 minutes, 30 minutes, 60 minutes and 240 minutes, respectively. Table 6 summarizes the performance of both of AutoSKlearn-m and AutoSKlearn-v, in addition to the number of datasets achieved improvement in performance by employing AutoSKlearn-m over AutoSKlearn-v on different time budgets. The improvement achieved by employing the meta-learning version decreases for extended time budgets. For example, the number of datasets that achieved performance improvement by using meta-learning dropped from 28 for the 30 minutes budget to 14 for the 240 minutes budget, as shown in Table 6. We use Wilcoxon statistical test to assess the significance of the performance difference between the vanilla version and the meta-learning version. The results show that the impact of the meta-learning is statistically significant only for the tiniest time budget of 10 minutes with more than 95% level of confidence (pvalue=0.004).

In the following, we explore the relationship between the characteristics of datasets and the improvement achieved by utilizing the meta-learning version of AutoSKlearn over different time budgets. We train a model that takes as input the meta-features of datasets, i.e., their characteristics, and predicts whether meta-learning can improve the performance. To develop this model, we label each dataset as Class 1 if utilizing meta-learning improves performance over the vanilla version and Class 0 otherwise. We implement a total of 42 meta-features from the literature, including simple, information-theoretic, and statistical meta-features (Kalousis, 2002; Mitchell, Buchanan,
Table 6: The performance of AutoSklearn-v and AutoSklearn-m and the gain in performance achieved by employing the meta-learning on 100 datasets over different time budgets.

| Time Budget | Framework   | Predictive Performance | Performance Gain | #datasets with gain > 1% |
|-------------|-------------|------------------------|------------------|--------------------------|
|             |             | Mean | SD   | Mean | Max |                  |                  |
| 10          | AutoSKlearn-m | 0.864 | 0.153 | 1.1% | 2.9% | 7.1% | 28               |
| 10          | AutoSKlearn-v | 0.862 | 0.151 | 1.1% | 5.4% | 15.5% | 12               |
| 30          | AutoSKlearn-m | 0.868 | 0.152 | 1.1% | 3.1% | 20.6% | 28               |
| 30          | AutoSKlearn-v | 0.8867 | 0.149 | 1.1% | 5.1% | 16.7% | 15               |
| 60          | AutoSKlearn-m | 0.870 | 0.144 | 1.1% | 3.3% | 18.8% | 20               |
| 60          | AutoSKlearn-v | 0.870 | 0.142 | 1.1% | 4.3% | 14.0% | 17               |
| 240         | AutoSKlearn-m | 0.873 | 0.141 | 1.1% | 7.7% | 31.6% | 14               |
| 240         | AutoSKlearn-v | 0.867 | 0.156 | 1.1% | 2.7% | 8.4% | 20               |

DeJong, Dietterich, Rosenbloom, & Waibel, 1990), such as statistics about the number of data points, features, and classes, as well as data skewness and entropy of the targets. All meta-features are listed in Appendix E, Table 11. Using the extracted information from the knowledge base, for our 100 datasets, we fit a shallow decision tree of depth 4 using the meta-feature variables as predictors. We considered decision tree classifier due to its interpretable nature, that allows rules to be derived from a root-leaf path in the tree. Given a new dataset, we compute its meta-features and use the decision tree model to recommend whether meta-learning is likely to improve performance or not. Our model achieves the following performance metrics: Recall = 0.85 and F1 Score = 0.85.

Rules for Class 1 and Class 0 can be represented as follows, where ∧ is the logical AND:

- **R1:** $\min(\frac{\xi}{n}) > 0.5 \implies$ Class 1
- **R2:** $\min(\frac{\xi}{n}) < 0.27 \land noise\text{-}signal\text{ ratio} > 8.57 \land p < 845 \implies$ Class 1
- **R3:** $0.1 < \min(\frac{\xi}{n}) < 0.27 \land noise\text{-}signal\text{ ratio} < 8.57 \implies$ Class 1
- **R4:** $0.27 < \min(\frac{\xi}{n}) < 0.5 \implies$ Class 0
- **R5:** $\min(\frac{\xi}{n}) < 0.27 \land noise\text{-}signal\text{ ratio} > 8.57 \land p > 845 \implies$ Class 0
- **R6:** $\min(\frac{\xi}{n}) < 0.1 \land noise\text{-}signal\text{ ratio} < 8.57 \implies$ Class 0

It is clear from the extracted rules that the number of features $p$, the percentage of the minority class to the number of instances ($\min(\frac{\xi}{n})$), and noisiness of data ($noise\text{-}signal\text{ ratio}$) are quite important features for the prediction.

### 5.2.4 Impact of Ensembling

Ensembling (Dietterich, 2000) is the process of combining multiple ML base models for the same task to produce a better predictive model. These base models can be combined in different techniques, including simple voting (averaging), weighted voting, bagging, and boosting (Dietterich,
Table 7: Performance comparison between vanilla/base version vs ensembling version of AutoSKlearn and SmartML different time budgets.

| Time Budget | Framework | Predictive Performance Mean | SD | Performance Gain Min | Mean | Max | #datasets with gain > 1% |
|-------------|-----------|----------------------------|----|-----------------------|------|-----|-------------------------|
| 10          | AutoSKlearn-e | 0.868 | 0.145 | 1.1% | 3.9% | 16.7% | 24 |
|             | AutoSKlearn-v | 0.868 | 0.151 | 1.1% | 3.2% | 8.9%  | 14 |
|             | SmartML-e    | 0.831 | 0.176 | 1.1% | 12.6% | 64.2% | 30 |
|             | SmartML      | 0.176 | 0.176 | 1.1% | 10.2% | 36.1% | 14 |
| 30          | AutoSKlearn-e | 0.875 | 0.141 | 1.1% | 3.2% | 13.9% | 32 |
|             | AutoSKlearn-v | 0.867 | 0.149 | 1.1% | 2.9% | 11.1% | 11 |
|             | SmartML-e    | 0.838 | 0.159 | 1.1% | 12.5% | 73.2% | 33 |
|             | SmartML      | 0.838 | 0.199 | 1.1% | 10.6% | 39.4% | 16 |
| 60          | AutoSKlearn-e | 0.879 | 0.138 | 1.1% | 4.7% | 12.7% | 25 |
|             | AutoSKlearn-v | 0.870 | 0.142 | 1.1% | 2.6% | 6.1%  | 13 |
|             | SmartML-e    | 0.832 | 0.172 | 1.1% | 11.2% | 55.8% | 28 |
|             | SmartML      | 0.816 | 0.194 | 1.1% | 10.5% | 31.1% | 18 |
| 240         | AutoSKlearn-e | 0.883 | 0.132 | 1.1% | 8.0% | 69.7% | 24 |
|             | AutoSKlearn-v | 0.867 | 0.156 | 1.1% | 3.1% | 8.4%  | 14 |
|             | SmartML-e    | 0.842 | 0.165 | 1.1% | 10.2% | 34.5% | 28 |
|             | SmartML      | 0.826 | 0.169 | 1.1% | 11.9% | 37.2% | 16 |

2000). In the following, we explore the impact of ensembling on the performance of the AutoML frameworks allowing enabling and disabling post-processing ensemble. Such frameworks include AutoSKlearn and SmartML-m. Furthermore, we investigate whether there is a relationship between the characteristics of the different datasets and the improvement caused by employing the vanilla version or the ensembling version of the AutoML framework. During the optimization process of AutoSKlearn and SmartML, the frameworks store the generated models instead of just keeping the best-performing one. These models are used in a post-processing phase to construct an ensemble model. This automatic ensemble construction avoids relying on a single hyperparameter setting which makes the generated model more robust to overfitting. AutoSKlearn uses the ensemble selection methodology introduced by Caruana et al. (Caruana, Niculescu-Mizil, Crew, & Ksikes, 2004), while SmartML uses majority voting (Lam & Suen, 1997). Ensemble selection is a greedy technique that starts with an empty ensemble and iteratively adds base models to the ensemble in a way that maximizes the validation performance. The technique uses uniform weights; however, it allows repetitions. Majority voting is considered the simplest scheme. It adheres to democratic principles, i.e., the class with the most votes wins. We kept the default setting of AutoSKlearn and SmartML using 50 and 5 base models in the ensemble, respectively.

To assess the impact of the ensembling, we compare the mean performance of vanilla/base version of each of AutoSKlearn and SmartML to their ensembling versions across different time budgets, as shown in Table 7. More detailed performance comparisons over all datasets across all time budgets are given in Figures 16 and 17 in Appendix F.
**AutoSKlearn**: The results show that ensembling does not always contribute to better performance than the vanilla version. However, it achieved mean improvement of 3.9%, 3.2%, 4.7%, and 8.0% on 24, 32, 25 and 24 datasets over 10, 30, 60, and 240 minutes budgets, respectively, as shown in Table 7. We use Wilcoxon statistical test to assess the significance of the performance difference between AutoSKlearn-e and AutoSKlearn-v. The results show that ensembling enhances the performance with a statistically significant gain of more than 95% level of confidence ($p$ value < 0.05) on the 4 time budgets. The level of confidence is almost 99% over all the time budgets combined.

**SmartML**: SmartML-e slightly improved the performance over the SmartML-m by average performance of 12.6%, 12.5%, 11.2%, and 10.2% on 30, 33, 28, and 28 datasets for 10, 30, 60, and 240 minutes time budgets, respectively, as shown in Table 7. We also use Wilcoxon statistical test to assess the significance of the performance difference between the base (meta-learning) and the ensembling versions of SmartML. The results show that the ensembling version enhance the performance with a statistically significant gain of more than 95% level of confidence ($p$ value < 0.05) on the 4 time budgets.

In the following, we explore the relationship between the characteristics of the datasets and the improvement achieved by utilizing the ensembling version of AutoSKlearn over different time budgets. To this end, we followed the same approach in Section 5.2.3 and trained a decision tree of depth 3 that takes the meta-features of 100 datasets as input and provides prediction to whether using ensembling can improve the performance (Class 1) or not (Class 0). So, given a new dataset, we compute its meta-features and use the decision tree model to recommend whether ensembling will likely improve performance. Our model for AutoSKlearn has achieved the following performance: Recall = 0.70 and F1 Score = 0.70. AutoSKlearn rules for Class 1 and Class 0 can be represented as follows:

- **R1**: $\mu(\rho) > 0.13 \land \sigma(\rho) > 0.27 \land \max(\rho) > 0.98 \implies \text{Class 1}$
- **R2**: $\mu(\rho) > 0.13 \land \sigma(\rho) \leq 0.27 \land \min(\rho) > 0.44 \implies \text{Class 1}$
- **R3**: $\mu(\rho) \leq 0.13 \land \mu(\pi_i) > 3.11 \land \sigma(\frac{\pi_i}{\rho}) > 0.03 \implies \text{Class 1}$
- **R4**: $\mu(\rho) \leq 0.13 \land \mu(\pi_i) = 3.11 \land \min(Mutual\ inform.) > 0.03 \implies \text{Class 1}$
- **R5**: $\mu(\rho) > 0.13 \land \sigma(\rho) > 0.27 \land \max(\rho) \leq 0.98 \implies \text{Class 0}$
- **R6**: $\mu(\rho) > 0.13 \land \sigma(\rho) \leq 0.27 \land \min(\rho) \leq 0.44 \implies \text{Class 0}$
- **R7**: $\mu(\rho) \leq 0.13 \land \mu(\pi_i) > 3.11 \land \sigma(\frac{\pi_i}{\rho}) \leq 0.03 \implies \text{Class 0}$
- **R8**: $\mu(\rho) \leq 0.13 \land \mu(\pi_i) \leq 3.11 \land \min(Mutual\ inform.) \leq 0.03 \implies \text{Class 0}$

Clearly, the following features are important to the prediction of the model; the mean of the pairwise correlation between features ($\mu(\rho)$), standard deviation of the pairwise correlation between features ($\sigma(\rho)$), maximum of the pairwise correlation between features ($\max(\rho)$), minimum of the pairwise correlation between features ($\min(\rho)$), the standard deviation of the ratio between number of instances and the number of features ($\sigma(\frac{\pi_i}{\rho})$), the minimum of the mutual information between features and class ($Mutual\ inform.$), and the mean of the unique categorical values of features ($\mu(\pi_i)$).
For SmartML, we trained multiple models; however, none of the models could capture the relation of the meta-features and the performance improvement caused by employing ensembling.

6. Discussion and Future Direction

The global average performance, weighted by the percentage of successful runs, shows that Auto-SKlearn-e and AutoSKlearn achieve the highest performance, while Recipe comes in the last place. Overall, AutoSklearn achieves the highest number of successful runs across different time budgets and witnessed performance improvement over the most significant number of datasets when increasing the time budget. Our analysis reveals that the impact of meta-learning declines over longer time budgets (i.e., 60 mins, 240 mins). In contrast, ensembling achieves consistent performance improvement across all time budgets. For AutoSKlearn, the analysis reveals a relationship between the characteristics of the datasets (e.g., number of features, noisiness of data, mutual information between features and class) and the improvement achieved by utilizing meta-learning or ensembling. Generally, AutoML frameworks considered in this work build pipelines with an average length of 2. TPOT yields the shortest pipelines with an average length of 1.5. A possible explanation could be that TPOT generates pipelines that optimize both the pipelines’ performance and complexity. Additionally, AutoSKlearn, ATM, and TPOT achieve the highest performance on multi-class classification tasks. For datasets with a large number of instances and a small number of features, ATM is a clear winner.

For some datasets, the performance of the different versions of AutoSKlearn varies significantly across different iterations. These datasets are characterized by having far fewer instances than features. Analyzing the pipelines of the different versions of AutoSKlearn on these datasets across multiple iterations shows that data preprocessing component is responsible for the significant performance variance between the different pipelines. For example, the performance difference between AutoSKlearn-v and AutoSKlearn-m on phpdo58hj varies significantly between 6% to 13% across different iterations. The two generated pipelines for AutoSKlearn-v and AutoSKlearn-m used the same model (LDA) with the same set of hyperparameters but different preprocessors. For large datasets, meta-learning shows significant performance improvement. For example, AutoSKlearn-m achieves significantly better performance than AutoSKlearn-v on CovPokElec. A possible explanation is that meta-learning warm-starts the optimization process and increases the chances of finding a well-performing configuration in the limited attempts during the defined time budget.

Specifying the time budget needs to be considered carefully as significantly increasing the time budget for the search process (e.g., from 60 minutes to 240 minutes) may not significantly improve the predictive performance. This decision varies from one scenario/application to another. For some applications, spending a long budget to achieve an additional predictive performance of 1% could be crucial while less important for other applications. However, more extended time budgets may lead to over-fitting. Carefully selecting a small search space with few top-performing classifiers can lead to a comparable performance with a search space that includes many classifiers, which is the case for AutoSKlearn and ATM frameworks.

Intuitively, an extensive systematic search for a well-performing machine learning pipeline should bear a high risk of over-fitting, and previous AutoML frameworks have confirmed this intuition (Thornton, Hutter, Hoos, & Leyton-Brown, 2013). AutoML tools are on the right extreme of the bias-variance spectrum as they choose among all learners and even construct new and arbitrary
large ones using ensemble methods (Mohr, Wever, & Hüllermeier, 2018). Notably, SmartML and AutoWeka witnessed performance degradation when increasing the time budget from 30 to 60 minutes. One possible explanation is that the data available for the search process is not sufficiently substantial and representative of “real” data. Hence, the danger of over-fitting is higher than for basic learning algorithms. This insight calls for developing novel and more efficient mechanisms to prevent over-fitting.

While AutoML frameworks optimize predictive performance, many exceed the specified time budget by more than 10%. This violation of the time constraints caused many runs to be terminated and considered as failed. This problem is observed in all frameworks except for AutoSKLearn, which calls for a robust implementation and careful consideration of the time constraint.

Most of the current work on AutoML considered automating the preprocessing, algorithm selection and hyperparameter tuning while ignoring the feature engineering part. In practice, the feature engineering part consumes most of the Engineer’s time to build ML pipelines and significantly affects the performance. The proper feature engineering phase could turn the feature space into a linearly separable space, so even naive classifiers could achieve relatively high predictive performance. On the other hand, skipping this phase or using the wrong feature engineering preprocessors makes it harder to achieve relatively high predictive performance, even for the most efficient classifiers. Hence, further research in this area can improve the overall performance of the resulting AutoML pipelines.

7. Conclusion

In this paper, we present a comprehensive evaluation and comparison of the performance characteristics of six AutoML frameworks on 100 datasets from OpenML. Our analysis reveals that no single winning framework outperforms others over all time budgets. Across various evaluations, AutoSklearn, ATM, and TPOT are the top-performing frameworks. The results also show that genetic-based frameworks (TPOT and Recipe) have high frequent failure rates for short time budgets while their success rates are steadily increasing as the time budget increases. We also find that meta-learning has a significant impact on small-time budgets, and such impact declines as the time budget increases. In contrast, ensembling consistently improves performance significantly across all time budgets. Furthermore, carefully selecting a small search space with few top-performing classifiers can lead to a comparable performance with a search space that includes many classifiers. Furthermore, increasing the time budget does not necessarily improve predictive performance. We believe that the results of our analysis are beneficial for guiding and improving the design process of future AutoML techniques.

Data Availability

The datasets generated during and/or analysed during the current study are available in the AutoML-Bench repository, https://datasystemsgrouput.github.io/AutoMLBench/datasets

Acknowledgments

We would like to acknowledge support for this project. This work was supported by European Social Fund via “ICT programme measure”. The authors would like to thank the students Oleh Matsuk, Abdelrahman Aldallal for their involvement in some of the experiments of this work.
Appendix A. Evaluated Datasets

Table [8] shows the datasets used in evaluating all the AutoML frameworks included in this work.

| Dataset Name (openml id) | Nr features | Nr instances | Nr classes | Nr missing values | Nr categorical features | Class entropy |
|--------------------------|-------------|--------------|------------|-------------------|-------------------------|--------------|
| AirlinesCodrnaAdult (1240) | 30          | 1076790      | 2          | 11896             | 1                       | 1.00         |
| Amazon (1457)             | 10001       | 1500         | 50         | 0                 | 0                       | 0.93         |
| analcatdata_authorship (458) | 71          | 841          | 4          | 0                 | 0                       | 0.99         |
| AP_Breast_Lung (1150)     | 10937       | 470          | 2          | 0                 | 0                       | 0.99         |
| AP_Omentum_Ovary (1156)   | 10937       | 275          | 2          | 0                 | 1                       | 0.93         |
| AP_Prostate_Ovary (1152)  | 10937       | 267          | 2          | 0                 | 0                       | 2.18         |
| arrhythmia (1017)         | 263         | 452          | 2          | 0                 | 1                       | 1.58         |
| audiology (999)           | 70          | 226          | 2          | 0                 | 1                       | 3.70         |
| avila-tr (42932)          | 11          | 20867        | 12         | 114               | 10                      | 2.27         |
| churn (40701)             | 21          | 5000         | 2          | 0                 | 1                       | 1.00         |
| cifar-10 (40927)          | 3073        | 60000        | 10         | 0                 | 70                      | 0.81         |
| connect-4 (1591)          | 43          | 67557        | 3          | 0                 | 1                       | 0.82         |
| CovPokElec (149)          | 65          | 1455525      | 10         | 0                 | 1                       | 0.86         |
| dataset_183_adult (179)   | 15          | 48842        | 2          | 0                 | 0                       | 2.21         |
| dataset_185_yeast (181)   | 9           | 1484         | 10         | 0                 | 1                       | 2.19         |
| dataset_186_satimage (182) | 37         | 6430         | 6          | 1668              | 3                       | 1.00         |
| dataset_187_abalone (183) | 9           | 4177         | 28         | 0                 | 1                       | 0.94         |
| dataset_189_baseball (185) | 18         | 1340         | 3          | 0                 | 0                       | 3.32         |
| dataset_194_eucalyptus (188) | 20       | 736          | 5          | 816               | 1                       | 0.99         |
| dataset_24_mushroom (24)  | 22          | 8124         | 2          | 0                 | 1                       | 0.84         |
| dataset_26_nursery (26)   | 9           | 12960        | 5          | 0                 | 0                       | 4.20         |
| dataset_28_optdigits (28) | 63          | 5620         | 10         | 0                 | 0                       | 4.28         |
| dataset_31_credit-g (31)  | 21          | 1000         | 2          | 0                 | 1                       | 2.58         |
| dataset_36_segment (36)   | 19          | 2310         | 7          | 0                 | 36                      | 3.84         |
| dataset_39_ecoli (39)     | 8           | 336          | 8          | 32                | 1                       | 0.93         |
| dataset_40_sonar (40)     | 61          | 208          | 2          | 896               | 6                       | 2.26         |
| dataset_42_soybean (42)   | 36          | 683          | 19         | 0                 | 1                       | 1.79         |
| dataset_44_spambase (44)  | 58          | 4601         | 2          | 0                 | 1                       | 2.00         |
| dataset_54_vehicle (54)   | 19          | 846          | 4          | 0                 | 4                       | 0.44         |
| dataset_59 Ionosphere (59) | 34          | 351          | 2          | 0                 | 14                      | 0.88         |
| dataset_6_letter (6)      | 17          | 20000        | 26         | 0                 | 0                       | 4.65         |
| dataset_60_waveform-5000 (60) | 41      | 5000         | 3          | 0                 | 1                       | 0.83         |
| dataset_61_iris (61)      | 5           | 150          | 3          | 0                 | 0                       | 0.92         |
| dataset_69_autos (9)      | 26          | 205          | 6          | 2792              | 4                       | 2.99         |
| devnagari (40923)         | 785         | 92000        | 46         | 0                 | 0                       | 3.17         |
| electricity-normalized (151) | 9        | 45312        | 2          | 0                 | 0                       | 1.00         |
| eye_movements (1044)      | 28          | 10936        | 3          | 40                | 2                       | 0.54         |
| GCM (1106)                | 16064       | 190          | 14         | 0                 | 1                       | 2.49         |
| gina_agnostic (1038)      | 971         | 3468         | 2          | 0                 | 1                       | 5.64         |
| Dataset              | Number of Examples | Number of Features | Dimensionality | Number of Classes | Number of Attributes | Sparsity |
|---------------------|--------------------|--------------------|----------------|------------------|----------------------|----------|
| hiva_agnostic       | 1039               | 2                  | 0              | 0                | 3,32                 |
| ipums_la_99-small   | 378                | 9                  | 0              | 0                | 6,64                 |
| jm1                 | 1053               | 2                  | 0              | 0                | 1,71                 |
| jungle_chess_2pcs   | 40997              | 3                  | 0              | 0                | 6,64                 |
| KDDCup99            | 1113               | 23                 | 0              | 0                | 6,64                 |
| kin8nm              | 189                | 2                  | 0              | 1                | 2,41                 |
| leukemia            | 1104               | 2                  | 0              | 1                | 0,47                 |
| lymphoma_2classes   | 1101               | 2                  | 0              | 1                | 2,81                 |
| MagicTelescope      | 1120               | 2                  | 0              | 0                | 0,34                 |
| mfeat-pixel         | 20                 | 2                  | 0              | 0                | 1,00                 |
| mnist               | 784                | 100                | 2              | 0                | 1,53                 |
| page-blocks         | 30                 | 2                  | 0              | 1                | 3,60                 |
| php0FyS2T           | 1492               | 100                | 0              | 0                | 0,22                 |
| php3CTpq            | 1509               | 22                 | 0              | 0                | 0,97                 |
| php5OMDBD           | 40971              | 30                 | 0              | 6                | 1,16                 |
| php5s7Ep8           | 40982              | 7                  | 0              | 1                | 1,86                 |
| php7KLval           | 1547               | 2                  | 0              | 0                | 0,59                 |
| phpB0xrNj           | 300                | 26                 | 0              | 0                | 1,58                 |
| phpBL64U            | 1476               | 6                  | 0              | 1                | 0,11                 |
| phpchCuL5           | 40966              | 8                  | 0              | 0                | 1,71                 |
| phpCXS3fx           | 1491               | 100                | 0              | 1                | 0,48                 |
| phpdo58hj           | 1562               | 64                 | 0              | 0                | 3,32                 |
| phpDEp6S            | 1487               | 30                 | 0              | 6                | 2,30                 |
| phpEZ030X           | 1561               | 2                  | 0              | 0                | 2,48                 |
| phpFLuQE4           | 1485               | 2                  | 0              | 0                | 1,00                 |
| phpfrPbS            | 1568               | 4                  | 0              | 0                | 1,00                 |
| phpGReJU            | 40985              | 20                 | 0              | 1                | 4,70                 |
| phpGUriE90          | 1494               | 2                  | 0              | 22               | 1,00                 |
| phpHEck0            | 1502               | 2                  | 0              | 1                | 1,00                 |
| phpHyLSNF           | 1515               | 20                 | 0              | 26               | 0,48                 |
| phpkixsf            | 1461               | 2                  | 0              | 1                | 2,19                 |
| phpmcGu2X           | 1468               | 9                  | 0              | 1                | 0,94                 |
| phpmmPOD5A          | 4135               | 2                  | 0              | 0                | 0,71                 |
| phpnljVve           | 310                | 2                  | 0              | 0                | 1,57                 |
| phpN4gaxw           | 1477               | 6                  | 0              | 0                | 0,16                 |
| phpNewWwl           | 40477              | 5                  | 0              | 0                | 1,71                 |
| phpoOxxNn           | 1493               | 100                | 0              | 9                | 1,72                 |
| phpwo7Dbi           | 1566               | 2                  | 0              | 0                | 2,55                 |
| phpPBCMyg           | 1475               | 6                  | 0              | 0                | 2,55                 |
| phprAExmK           | 4535               | 2                  | 0              | 1                | 0,94                 |
| phpSZIq5T           | 1514               | 10                 | 0              | 1                | 4,70                 |
| phpTd5jYj           | 1501               | 2                  | 0              | 1                | 2,64                 |
| phpTJRsq            | 40498              | 10                 | 0              | 0                | 0,32                 |
| phpvcoG8S           | 1169               | 7                  | 0              | 9                | 0,52                 |
| phpVeNa5j           | 1497               | 2                  | 0              | 1                | 0,98                 |
| phpvtdNpu           | 1079               | 4                  | 0              | 0                | 4,25                 |
| phpWfyMlu           | 1496               | 2                  | 0              | 9                | 0,79                 |
| phpxjbp            | 1507               | 5                  | 0              | 0                | 0,96                 |
| phpYLeYdd           | 4538               | 2                  | 0              | 0                | 3,32                 |
| phpZrCzJR           | 40900              | 5                  | 0              | 0                | 1,22                 |
| Dataset                               | Classes | Features | Instances | Missing Values | Categorical Features | Entropy |
|---------------------------------------|---------|----------|-----------|----------------|----------------------|---------|
| pokerhand-normalized (155)            | 11      | 829201   | 10        | 0              | 0                    | 3.32    |
| schizo (466)                          | 14      | 340      | 2         | 0              | 1                    | 5.52    |
| shuttle (40685)                       | 10      | 58000    | 7         | 0              | 0                    | 3.99    |
| solar-flare_1 (40686)                 | 13      | 315      | 5         | 0              | 0                    | 0.74    |
| synthetic_control (377)               | 61      | 600      | 6         | 0              | 29                   | 0.34    |
| tumors_C (1107)                       | 7130    | 60       | 2         | 0              | 4                    | 1.56    |
| umistfacescropped (41084)             | 10305   | 575      | 20        | 0              | 3                    | 0.99    |
| vowel (307)                           | 14      | 990      | 2         | 0              | 0                    | 1.42    |
| wine-quality-red (40691)              | 12      | 1599     | 6         | 0              | 11                   | 0.99    |
| aaaData_for_UCI_named (43007)         | 14      | 10000    | 2         | 0              | 0                    | 1.59    |

Table 8: List of all tested datasets including information about (abbreviated) name and OpenML id for each data set together with the number of classes, the number of features, the number of instances, how many values are missing in total (Missing values), number of categorical features per sample, and the class entropy.

Appendix B. Framework and Source Code

Table 9 lists the Github repositories of all the open-source AutoML frameworks considered in this work. Some frameworks are still under active development and may differ from the evaluated versions.

| AutoML Framework   | Source Code                                      |
|--------------------|--------------------------------------------------|
| AutoSKlearn        | https://automl.github.io/auto-sklearn/           |
| TPOT               | https://github.com/EpistasisLab/tpot             |
| ATM                | https://github.com/HDI-Project/ATM               |
| Recipe             | https://github.com/laic-ufmg/Recipe              |
| AutoWeka           | https://github.com/automl/AutoWeka               |
| SmartML            | https://github.com/DataSystemsGroupUT/SmartML    |

Appendix C. Cut-off time Budget

We tested the cut-off timeouts of 4 and 8 hours on 14 randomly selected datasets. Table 10 shows the mean performance difference between the 8 and 4 hours (Avg. diff) over the 14 datasets. Additionally, we report the results of the Wilcoxon signed-rank test to determine if a statistically significant difference in performance exists between the AutoML frameworks over the two-time budgets.

Appendix D. General Performance Evaluation

Figures 9 to 11 shows the average performance of all frameworks for time budgets 10, 30, and 60 minutes compared the average performance of the baseline. Figures 4(b) and 13 to 15 show the AutoML frameworks’ average performance on subsets of the datasets with special characteristics, namely binary-class, large number of features and instances, small number of features and instances, and small number of features and large number of instances.
| Framework | P value | Avg. diff |
|-----------|---------|-----------|
| AutoSKlearn | 0.084 | -0.026 |
| AutoSKlearn-e | **0.039** | -0.025 |
| AutoSKlearn-m | 0.382 | -0.031 |
| AutoSKlearn-v | 0.272 | -0.008 |
| AutoWeka | 0.133 | -0.005 |
| Recipe | 0.480 | 0.007 |
| SmartML | 0.594 | -0.003 |
| SmartML-e | 0.753 | -0.009 |
| TPOT | 0.092 | -0.050 |

Table 10: Performance comparison between the 8 and 4 hours budgets on 14 randomly selected datasets.

Figure 9: Average performance of all frameworks (10 Min) compared to the baseline.

Appendix E. Impact of Meta Learning

Table 11 lists a total of 42 meta-features including simple, information-theoretic and statistical meta-features.

Appendix F. Impact of Ensembling

Figures 16 and 17 shows the performance difference between the ensembling version and the vanilla/base version of AutoSKlearn and SmartML, respectively over 10, 30, 60 and 240 minutes time budgets.

Appendix G. Impact of time budget

Figures 18 to 27 show the impact of increasing the time budget on the performance of the all the AutoML frameworks considered in this work.
Figure 10: Average performance of all frameworks (30 Min) compared to the baseline.

Figure 11: Average performance of all frameworks (60 Min) compared to the baseline.
Figure 12: Performance of the final pipeline for datasets with large number of features and small number of instances.

Figure 13: Performance of the final pipeline for datasets with large number of features and large number of instances.
Figure 14: Performance of the final pipeline for datasets with small number of features and small number of instances.

Figure 15: Performance of the final pipeline for datasets with small number of features and large number of instances.
Table 11: Overview of the used meta-features. Groups from top to bottom: simple, statistical, and information-theoretic. Continuous features $X$ and target $Y$ have mean $\mu_X$, stdev $\sigma_X$, variance $\sigma_X^2$. Categorical features $X$ and class $C$ have categorical values $\pi_i$, conditional probabilities $\pi_{i|j}$, joint probabilities $\pi_{i,j}$, marginal probabilities $\pi_{i+} = \sum_j \pi_{i,j}$, entropy $H(X) = -\sum_i \pi_{i+} \log_2(\pi_{i+})$. (Vanschoren, 2018)

![Figure 16](image-url):

The performance difference between the AutoSKlern-e and AutoSKlern-v over different time budgets. Green markers represent better performance with AutoSKlern-e, blue markers represent comparable performance with a difference less than 1%, red markers represent better performance with AutoSKlern-v, and yellow markers represent failed runs on both versions.
Figure 17: The performance difference between the SmartML-m and SmartML-e. Green markers represent better performance with SmartML-e, blue markers represent comparable performance with a difference less than 1%, red markers represent better performance with SmartML, yellow markers on the right represent failed runs with SmartML-m but successful with SmartML-e, yellow markers on the left represent failed runs with SmartML-e but successful with SmartML-m and yellow markers in the middle represent failed runs with both SmartML-m and SmartML-e.
Figure 18: The impact of increasing the time budget on AutoSKLearn-v performance from $x$ to $y$ minutes ($x$-$y$). Green markers represent better performance with $y$ time budget, blue markers mean that the difference between $x$ and $y$ is $< 1$. Red markers represent better performance on $x$ time budget.
Figure 19: The impact of increasing the time budget on AutoSKlearn-m performance from $x$ to $y$ minutes ($x$-$y$). Green markers represent better performance with $y$ time budget, blue markers means that the difference between $x$ and $y$ is $< 1$. Red markers represent better performance on $x$ time budget.
Figure 20: The impact of increasing the time budget on AutoSKlearn-e performance from $x$ to $y$ minutes ($x-y$). Green markers represent better performance with $y$ time budget, blue markers mean that the difference between $x$ and $y$ is $< 1$. Red markers represent better performance on $x$ time budget.
Figure 21: The impact of increasing the time budget on AutoSKlearn performance from $x$ to $y$ minutes ($x$-$y$). Green markers represent better performance with $y$ time budget, blue markers mean that the difference between $x$ and $y$ is $< 1$. Red markers represent better performance on $x$ time budget.
Figure 22: The impact of increasing the time budget on TPOT performance from $x$ to $y$ minutes ($x$-$y$). Green markers represent better performance with $y$ time budget, blue markers mean that the difference between $x$ and $y$ is < 1. Red markers represent better performance on $x$ time budget.
Figure 23: The impact of increasing the time budget on ATM performance from \( x \) to \( y \) minutes (\( x-y \)). Green markers represent better performance with \( y \) time budget, blue markers mean that the difference between \( x \) and \( y \) is \( < 1 \). Red markers represent better performance on \( x \) time budget.
Figure 24: The impact of increasing the time budget on SmartML-m performance from \( x \) to \( y \) minutes (\( x-y \)). Green markers represent better performance with \( y \) time budget, blue markers means that the difference between \( x \) and \( y \) is \(< 1\). Red markers represent better performance on \( x \) time budget.
Figure 25: The impact of increasing the time budget on SmartML-e performance from $x$ to $y$ minutes ($x$-$y$). Green markers represent better performance with $y$ time budget, blue markers means that the difference between $x$ and $y$ is $< 1$. Red markers represent better performance on $x$ time budget.
Figure 26: The impact of increasing the time budget on AutoWeka performance from \( x \) to \( y \) minutes (\( x-y \)). Green markers represent better performance with \( y \) time budget, blue markers means that the difference between \( x \) and \( y \) is \(< 1\). Red markers represent better performance on \( x \) time budget.
Figure 27: The impact of increasing the time budget on recipe performance from $x$ to $y$ minutes ($x$-$y$). Green markers represent better performance with $y$ time budget, blue markers means that the difference between $x$ and $y$ is $< 1$. Red markers represent better performance on $x$ time budget.
References

Bergstra, J., Yamins, D., & Cox, D. D. (2013a). Hyperopt: A python library for optimizing the hyperparameters of machine learning algorithms. In Proceedings of the 12th Python in science conference, pp. 13–20. Citeseer.

Bergstra, J., Yamins, D., & Cox, D. D. (2013b). Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures.

Buitinck, L., Louppe, G., Blondel, M., Pedregosa, F., Mueller, A., Grisel, O., Niculae, V., Prettenhofer, P., Gramfort, A., Grobler, J., Layton, R., VanderPlas, J., Joly, A., Holt, B., & Varoquaux, G. (2013). API design for machine learning software: experiences from the scikit-learn project. In ECML PKDD Workshop: Languages for Data Mining and Machine Learning, pp. 108–122.

Caruana, R., Niculescu-Mizil, A., Crew, G., & Ksikes, A. (2004). Ensemble selection from libraries of models. In ICML.

Castiello, C., Castellano, G., & Fanelli, A. M. (2005). Meta-data: Characterization of input features for meta-learning. In International Conference on Modeling Decisions for Artificial Intelligence, pp. 457–468. Springer.

de Sá, A. G. C., Pinto, W. J. G. S., Oliveira, L. O. V. B., & Pappa, G. L. (2017). RECIPE: A grammar-based framework for automatically evolving classification pipelines. In EuroGP. Vol. 10196 of Lecture Notes in Computer Science, pp. 246–261.

Dietterich, T. G. (2000). Ensemble methods in machine learning. In International workshop on multiple classifier systems.

Eldeeb, H., Matsuk, O., Maher, M., Eldallal, A., & Sakr, S. (2021). The impact of auto-sklearns learning settings: Meta-learning, ensembling, time budget, and search space size. In EDBT/ICDT Workshops.

Erickson, N., Mueller, J., Shirkov, A., Zhang, H., Larroy, P., Li, M., & Smola, A. (2020). Autogluon-tabular: Robust and accurate autotml for structured data.

Falkner, S., Klein, A., & Hutter, F. (2018). Bohb: Robust and efficient hyperparameter optimization at scale.

Feurer, M., Eggensperger, K., Falkner, S., Lindauer, M., & Hutter, F. (2020). Auto-sklearrn 2.0: The next generation. arXiv preprint arXiv:2007.04074, 24.

Feurer, M., Klein, A., Eggensperger, K., Springenberg, J., Blum, M., & Hutter, F. (2015). Efficient and robust automated machine learning. In Advances in neural information processing systems, pp. 2962–2970.

Fitzgerald, B. (2012). Software crisis 2.0. Computer, 45(4).

Gehan, E. A. (1965). A generalized wilcoxon test for comparing arbitrarily singly-censored samples. Biometrika, 52(1-2), 203–224.

Gijsbers, P., Bueno, M. L., Coors, S., LeDell, E., Poirier, S., Thomas, J., Bischl, B., & Vanschoren, J. (2022). Amlb: an autotml benchmark.

Gijsbers, P., LeDell, E., Thomas, J., Poirier, S., Bischl, B., & Vanschoren, J. (2019). An open source autotml benchmark.

Gijsbers, P., & Vanschoren, J. (2020). Gama: A general automated machine learning assistant. In Joint European Conference on Machine Learning and Knowledge Discovery in Databases, pp. 560–564. Springer.

He, X., Zhao, K., & Chu, X. (2019). Autotml: A survey of the state-of-the-art.

Hutter, F., Hoos, H. H., & Leyton-Brown, K. (2011). Sequential model-based optimization for general algorithm configuration. In International conference on learning and intelligent optimization.

International Data Corporation (2017). Worldwide semiannual cognitive/artificial intelligence systems spending guide. Tech. rep.

Jin, H., Song, Q., & Hu, X. (2019). Auto-keras: An efficient neural architecture search system. In ACM KDD.

68
Kalousis, A. (2002). *Algorithm selection via meta-learning*. Ph.D. thesis, University of Geneva.

Klein, A., Falkner, S., Mansur, N., & Hutter, F. (2017). Robo: A flexible and robust bayesian optimization framework in python. In *NIPS 2017 Bayesian Optimization Workshop*.

Komer, B., Bergstra, J., & Eliasmith, C. (2014). Hyperopt-sklearn: automatic hyperparameter configuration for scikit-learn. In *ICML workshop on AutoML*, Vol. 9. Citeseer.

Kotthoff, L. (2016). Algorithm selection for combinatorial search problems: A survey. In *Data mining and constraint programming*, pp. 149–190. Springer.

Kotthoff, L., Thornton, C., Hoos, H. H., Hutter, F., & Leyton-Brown, K. (2017). Auto-weka 2.0: Automatic model selection and hyperparameter optimization in weka. *JMLR, 18*(1).

Lam, L., & Suen, S. (1997). Application of majority voting to pattern recognition: an analysis of its behavior and performance. *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans, 27*(5), 553–568.

LeDell, E., & Poirier, S. (2020). H2O AutoML: Scalable automatic machine learning..

Maher, M., & Sakr, S. (2019). Smartml: A meta learning-based framework for automated selection and hyperparameter tuning for machine learning algorithms. In *EDBT: 22nd International Conference on Extending Database Technology*.

Michie, D., Spiegelhalter, D. J., & Taylor, C. C. (1994). *Machine learning, neural and statistical classification*..

Mitchell, T., Buchanan, B., DeJong, G., Dietterich, T., Rosenbloom, P., & Waibel, A. (1990). Machine learning. *Annual review of computer science, 4*(1), 417–433.

Mohr, F., Wever, M., & Hüllermeier, E. (2018). Mi-plan: Automated machine learning via hierarchical planning. *Machine Learning, 107*(8-10), 1495–1515.

Olson, R. S., Bartley, N., Urbanowicz, R. J., & Moore, J. H. (2016). Evaluation of a tree-based pipeline optimization tool for automating data science. In *Proceedings of the genetic and evolutionary computation conference 2016*, pp. 485–492.

Olson, R. S., & Moore, J. H. (2016). Tpot: A tree-based pipeline optimization tool for automating machine learning. In *Workshop on automatic machine learning*, pp. 66–74. PMLR.

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

Płońska, A., & Płoński, P. (2021). Mljar: State-of-the-art automated machine learning framework for tabular data. version 0.10.3..

Shawi, R. E., Maher, M., & Sakr, S. (2019). Automated machine learning: State-of-the-art and open challenges. *CoRR, abs/1906.02287*.

Smith, M. J., Sala, C., Kanter, J. M., & Veeramachaneni, K. (2020). The machine learning bazaar: Harnessing the ml ecosystem for effective system development. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, pp. 785–800.

Swearingen, T., Drevo, W., Cyphers, B., Cuesta-Infante, A., Ross, A., & Veeramachaneni, K. (2017). ATM: A distributed, collaborative, scalable system for automated machine learning. In *2017 IEEE International Conference on Big Data (Big Data)*.

Thornton, C., Hutter, F., Hoos, H. H., & Leyton-Brown, K. (2013). Auto-weka: Combined selection and hyperparameter optimization of classification algorithms. In *ACM KDD*, pp. 847–855. ACM.

Truong, A., Walters, A., Goodsit, J., Hines, K., Bruss, C. B., & Farivar, R. (2019). Towards automated machine learning: Evaluation and comparison of automl approaches and tools. In *2019 IEEE 31st international conference on tools with artificial intelligence (ICTAI)*, pp. 1471–1479. IEEE.

Vakhrushev, A., Ryzhkov, A., Savchenko, M., Simakov, D., Damdinov, R., & Tuzhilin, A. (2021). Lightautoml: Automl solution for a large financial services ecosystem.
Vanschoren, J. (2018). Meta-learning: A survey.
Vanschoren, J., van Rijn, J. N., Bischl, B., & Torgo, L. (2013). Openml: Networked science in machine learning. *SIGKDD Explorations*, 15(2), 49–60.
Wang, C., Wu, Q., Weimer, M., & Zhu, E. (2021). Flaml: A fast and lightweight automl library. *Proceedings of Machine Learning and Systems*, 3, 434–447.
Zöller, M.-A., & Huber, M. F. (2021). Benchmark and survey of automated machine learning frameworks. *Journal of artificial intelligence research*, 70, 409–472.
Zomaya, A. Y., & Sakr, S. (2017). Handbook of big data technologies.