DISSERTATIONES
INFORMATICAE
UNIVERSITATIS
TARTUENSIS
56

HASSAN ABDULGALEEL HASSAN
SALIM ELDEEB

Empowering Machine Learning Pipelines with Automated Feature Engineering

DISSERTATIONES INFORMATICAE UNIVERSITATIS TARTUENSIS
56

DISSERTATIONES INFORMATICAE UNIVERSITATIS TARTUENSIS
56

HASSAN ABDULGALEEL HASSAN SALIM ELDEEB

Empowering Machine Learning Pipelines with Automated Feature Engineering

UNIVERSITY OF TARTU
Press
1632

Institute of Computer Science, Faculty of Science and Technology, University of Tartu, Estonia.

Dissertation has been accepted for the commencement of the degree of Doctor of Philosophy (PhD) in Computer Science on September 3, 2024 by the Council of the Institute of Computer Science, University of Tartu.

Supervisor

Assoc. Prof. Radwa Mohamed El Emam El Shawi
Institute of Computer Science
Tartu University, Estonia

Opponents

Prof. Dr. Çağatay Çatal
Department of Computer Science and Engineering
Qatar University, Qatar

Prof. Dr. Ladjel Bellatreche
Laboratory of Computer Science and Automatic Control for Systems
National Engineering School for Mechanics and Aerotechnics, France

The public defense will take place on October 8, 2024 at 10:15 in Zoom.

The publication of this dissertation was financed by the Institute of Computer Science, University of Tartu.

ISSN 2613-5906 (print) ISSN 2806-2345 (pdf)
ISBN 978-9916-27-653-2 (print) ISBN 978-9916-27-654-9 (pdf)

University of Tartu Press
http://www.tyk.ee/

*To my beloved children, **Malek**, **Huzaifa**, and **Albara**, I pen these words with the deepest affection. Know that for your happiness and well-being, I am willing to devote every ounce of my being, sacrificing all, for you are the essence of my life's joy and fulfillment.*

ABSTRACT

With the increasing demand for machine learning applications, it has been recognized that the number of data scientists cannot scale with the growing data volumes and application needs in our digital world. To fill the gap of human expertise, several automated machine learning (AutoML) frameworks have been developed to automate the process of building machine learning pipelines.

However, building a well-performing machine learning pipeline is an iterative and complex process that requires a solid understanding of various techniques that can be used in each component of the machine learning pipeline. Additionally, feature engineering (FE) is one of the most time-consuming yet least automated steps in building machine learning pipelines. It requires a deep understanding of the domain and data exploration to discover relevant hand-crafted features from raw data.

In this work, 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.

Furthermore, we empirically evaluate the impact of integrating an automated feature extraction tool (AutoFeat) into two automated machine learning frameworks, namely, Auto-Sklearn and TPOT, on their predictive performance. Besides, we discuss the limitations of AutoFeat that need to be addressed to improve the predictive performance of the automated machine learning frameworks on real-world datasets.

Given the limitations of AutoFaat and other automated FE tools, including a lack of practical scalability and efficiency, we introduce BigFeat, a scalable and interpretable framework that streamlines critical phases of the machine learning pipeline: feature engineering, model selection, and hyperparameter tuning. BigFeat presents two execution options: as a standalone feature engineering framework, denoted as BigFeat-FE, and an AutoML framework, referred to as BigFeat-AutoML. BigFeat-FE optimizes input feature quality with the ultimate aim of maximizing predictive performance. It employs a dynamic feature generation and selection mechanism that systematically creates a set of expressive features. These features not only enhance prediction performance but also prioritize interpretability.

BigFeat-FE also employs a meta-learning technique to warm-start the optimization process, resulting in significant overall performance gains. BigFeat-AutoML, tailored for algorithm selection and hyperparameter tuning, harnesses a random search method over the space of interpretable models. We conducted extensive experiments, and the results demonstrate that BigFeat-FE consistently

outperforms state-of-the-art automated feature engineering frameworks, such as AutoFeat and SAFE, across a wide range of datasets, achieving an average performance improvement of 8.65% compared to AutoFeat and 4.71% compared to SAFE, respectively.

Additionally, BigFeat-AutoML demonstrates substantial performance improvement compared to TPOT and AutoSKlearn, with average improvements of 0.74% over TPOT and 2.25% over AutoSKlearn, respectively. Furthermore, BigFeat's scalability is affirmed through its linear time-and-space complexity and swift execution times, averaging 20 times faster than AutoFeat and 14 times faster than SAFE.

Our study provides crucial insights that can significantly guide and impact the design of AutoML frameworks and the integration of automated feature engineering tools to build efficient and accurate machine learning pipelines.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

3C	Three Classifier set
AB	AdaBoost
AF	AutoFeat
AFS	ANOVA Feature Selection
ANOVA	Analysis of Variance
Auto FE	Automated Feature Engineering
AutoML	Automated Machine Learning
BF	BigFeat
BF-FE	BigFeat-FE
BF-v	BigFeat-vanilla
BTB	Bayesian Tuning and Bandits
CASH	Combined Algorithm Selection and Hyperparameter Optimization
DL	Deep Learning
DT	Decision Tree
ET	Extremely Randomized Trees
FE	Feature Engineering
FG	Feature Generation
FS	Feature Selection
GBoost	Gradient Boosting
GUI	Graphical User Interface
HPO	Hyperparameter Optimization
IG	Information Gain
I^f	Feature Importance Scores
I^o	Operator Importance Scores
kNN	k-Nearest Neighbors
LR	Logistic Regression
ML	Machine Learning
MLP	Multi-Layer Perceptron
ORG	Original Feature
RF	Random Forest
RL	Reinforcement Learning
RFG	Random Feature Generation
ROBO	Robust Bayesian Optimization
SAFE	Scalable Automatic Feature Engineering
SD	Standard Deviation
SF	SAFE
SMAC	Sequential Model Algorithm Configuration
TPE	Tree-structured Parzen Estimator

1. LIST OF ORIGINAL PUBLICATIONS

I **Hassan Eldeeb**, Shota Amashukeli, and Radwa El Shawi. "An empirical analysis of integrating feature extraction to automated machine learning pipeline." In Pattern Recognition. ICPR International Workshops and Challenges: Virtual Event, January 10–15, 2021, Proceedings, Part IV, pp. 336-344. Springer International Publishing (2021) DOI: 10.1007/978-3-030-68799-1_24. *Lead Author: Conceptualization, Methodology, Software, Validation, Formal analysis, Investigation, Writing – original draft, Writing – review & editing, Visualization.*

II **Hassan Eldeeb**, Oleh Matsuk, Mohamed Maher, Abdelrhman Eldallal, and Sherif Sakr. "The impact of Auto-Sklearn's Learning Settings: Meta-learning, Ensembling, Time Budget, and Search Space Size." In EDBT/ICDT Workshops (2021). *Lead Author: Conceptualization, Methodology, Software, Validation, Formal analysis, Investigation, Writing – original draft, Writing – review & editing, Visualization.*

III **Hassan Eldeeb**, Shota Amashukeli, and Radwa ElShawi. "BigFeat: Scalable and Interpretable Automated Feature Engineering Framework." In 2022 IEEE International Conference on Big Data (Big Data), pp. 515-524. IEEE (2022). DOI: 10.1109/BigData55660.2022.10020768. *Lead Author: Conceptualization, Methodology, Software, Validation, Formal analysis, Investigation, Writing – original draft, Writing – review & editing, Visualization.*

IV **Hassan Eldeeb**, Mohamed Maher, Radwa Elshawi, and Sherif Sakr. "AutoMLBench: A comprehensive experimental evaluation of automated machine learning frameworks." Expert Systems with Applications (2024). DOI: 10.1016/j.eswa.2023.122877. *Lead Author: Conceptualization, Methodology, Software, Validation, Formal analysis, Investigation, Writing – original draft, Writing – review & editing, Visualization.*

V **Hassan Eldeeb** and Radwa Elshawi. "Empowering Machine Learning with Scalable Feature Engineering and Interpretable AutoML." IEEE Transactions on Artificial Intelligence (2024). DOI 10.1109/TAI.2024.3400752. *Lead Author: Conceptualization, Methodology, Software, Validation, Formal analysis, Investigation, Writing – original draft, Writing – review & editing, Visualization.*

2. INTRODUCTION

The field of Automated Machine Learning (AutoML) has gained significant attention in recent years due to the increasing demand for automated solutions that can efficiently and effectively build machine learning models[110]. AutoML has shown great potential in reducing the time, effort, and expertise required to build accurate machine learning models. However, despite promising results, there are still many challenges that need to be systematically identified and addressed to make AutoML a widely adopted technology.

This thesis aims to systematically investigate these challenges and contribute to the advancement of the AutoML field. The research identifies and strengthens the weaknesses of AutoML frameworks, and further improves their performance by automating and enhancing the automated feature engineering stage. The proposed improvement addresses the need to enhance the predictive performance of the machine learning pipelines, reduce time and resource consumption, and improve the scalability and interpretability of the generated pipelines.

To illustrate the challenges in developing machine learning pipelines, consider a scenario in which a data scientist needs to build a predictive model for customer churn. The traditional process involves several steps: collecting and cleaning data, selecting and engineering features, choosing and tuning a machine learning model, and validating the model's performance. Each step is time-consuming and requires significant expertise. Automating these steps using an AutoML framework can drastically reduce the time and effort involved. However, current AutoML frameworks have limitations, particularly in the feature engineering stage, which is crucial for model performance. This example underscores the importance of improving AutoML frameworks to handle feature engineering more effectively.

This chapter introduces the study by first discussing the background and context of autoML, followed by the research problem, the research aims, objectives and questions, the significance, and finally, the limitations.

2.1. Automated Machine Learning (AutoML)

In the background section of AutoML, it's crucial to acknowledge the widespread interest in artificial intelligence, with an annual expenditure nearing 12.5 billion [47]. This interest is primarily fueled by the advancements in ML. The increase in data generation from various sources, enhanced processing capabilities, and advancements in ML algorithms have led to their extensive application in real-world scenarios [111]. This trend is leading towards a 'data science crisis', akin to the previously experienced software crisis [28], driven by the pressing need for a larger number of skilled data scientists to manage the ever-increasing data volumes.

Deep Learning, a subset of ML, has particularly gained prominence due to its ability to automatically learn hierarchical feature representations from raw data,

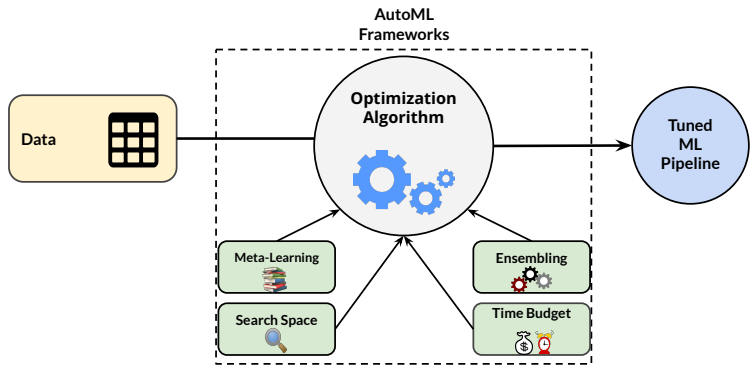


Figure 1: The general Workflow of autoML frameworks.

such as images, text, and speech [67, 35]. This capability has led to groundbreaking results in areas like computer vision, natural language processing, and speech recognition [61, 43, 3]. However, despite its success, DL often requires extensive computational resources and expertise in designing, training, and tuning complex neural network architectures [40, 97]. Integrating AutoML techniques with DL frameworks can provide several benefits. For instance, automated hyperparameter tuning and neural architecture search (NAS) can optimize DL models, making them more efficient and effective [112, 70]. Additionally, AutoML can assist in automating the preprocessing and feature engineering steps, which are often labor-intensive and require domain expertise [27, 57].

Consequently, there is a growing focus on automating the construction of ML pipelines to reduce the necessity of human involvement. Research in AutoML aims to lower both computational costs and the need for expert human intervention in developing ML pipelines, utilizing efficient automation algorithms. AutoML, notably, makes ML techniques more accessible to domain experts and those without technical backgrounds.

In applying ML to real-world issues, the process is multi-staged and highly iterative, aiming to automatically generate the optimal ML pipeline for maximizing predictive performance within a set computational budget (refer to Figure 1).

Different AutoML frameworks take various approaches. For instance, `Smart-ML` [72] employs meta-learning to initiate the search with promising classifiers, whereas `AutoSKlearn` [27] considers using an ensemble of top models from the optimization phase. `ATM` [96], in contrast, limits its search space to three classifiers. The impact of these varied design decisions on the performance of AutoML pipelines is yet to be fully understood.

2.2. Automated Feature Engineering (automated FE)

As Figure 2 indicates, the feature engineering (FE) phase involves transforming original features into engineered features through a systematic process. Initially,

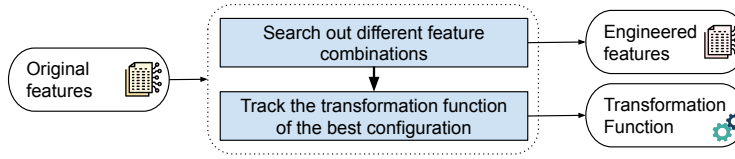


Figure 2: High-level illustration of the feature engineering phase in the context of AutoML.

a comprehensive search is conducted to explore various feature combinations, aiming to identify configurations that enhance model performance. Throughout this process, the transformation functions applied to the features are meticulously tracked to ensure reproducibility and interpretability. The outcome is a set of engineered features that are optimized for the machine learning model, along with the associated transformation functions that detail the steps taken to achieve these features.

FE is a critical component in the creation of machine learning pipelines for tabular data, bearing a significance akin to representation learning in varied data formats like images, text, or videos [4]. The process of FE involves complex tasks such as feature selection, combination, and arithmetic operations. Traditionally, this has been a labor-intensive, trial-and-error process for experts, beginning with the development of a FE strategy, followed by model training, evaluation, and iterative adjustments based on performance [62, 63].

Transitioning from this traditional approach, the field of Automated Feature Engineering (AutoFE) has emerged, characterized by its robust exploration and automation capabilities [91, 104, 105]. AutoFE tools [15, 45, 51, 107], which are increasingly being used in industrial applications, offer extensive and efficient exploration techniques. These tools not only streamline the FE process but also significantly reduce the associated labor costs, marking a significant shift from manual, expert-driven methods to automated, algorithm-driven approaches in FE.

2.3. Research Objectives and Scope

2.3.1. Research Gap

Currently, the performance of various Automated Machine Learning (AutoML) frameworks under differing conditions remains poorly understood. Frameworks like `SmartML` [72], `AutoSKlearn` [27], and `ATM` [96] differ in their methodologies, yet there is a lack of clarity on how time budgets, search space sizes, and specific methodologies like meta-learning and ensemble methods impact their performance. Moreover, in the field of automated Feature Engineering (FE), there is a need for more comprehensive tools that are user-friendly, interpretable, scalable, and model-agnostic.

2.3.2. Aims and Objectives

This research aims to address these gaps through two primary objectives: firstly, to conduct a detailed evaluation of various AutoML frameworks, assessing their performance under a range of conditions and configurations; and secondly, to develop BigFeat, an automated FE framework that meets critical criteria such as ease of use, interpretability, scalability, and model-agnostic functionality. The objectives include understanding the influence of operational parameters on AutoML frameworks and creating an automated FE tool capable of handling large datasets and improving performance across various ML algorithms.

2.3.3. Research Questions

1. What is the impact of the time budget on the performance of different AutoML frameworks? Can extended time budgets guarantee consistent performance improvement?
2. How does the size of the search space in the AutoML framework affect predictive performance?
3. Does employing meta-learning consistently improve performance across different time budgets and dataset characteristics?
4. Is there a correlation between the use of ensemble construction in AutoML frameworks and enhanced performance across various datasets?
5. How does the integration of automated Feature Engineering (FE) influence the overall performance of AutoML frameworks? Specifically, what is the impact of advanced feature generation techniques on the predictive accuracy and efficiency of these frameworks?
6. What are the effective strategies for utilizing various feature generation operators within the automated FE process to enhance the model's ability to capture complex relationships in the data?
7. How can the automated FE process be optimized to identify and prioritize important features, while eliminating those that may confuse the model or degrade its predictive performance?

2.3.4. Research Significance

This research is significant as it aims to bridge the gap between theoretical and practical aspects of AutoML. Enhancing understanding of AutoML framework performance and contributing to the development of advanced automated FE tools are expected to make AutoML more accessible and effective for a broader range of users, leading to improvements in various sectors reliant on machine learning.

2.3.5. Limitations

This study is primarily limited by its exclusive focus on tabular data, thereby not encompassing other data types such as images, text, or unstructured data. It is

also centered on classical, particularly supervised, machine learning techniques. These limitations define the scope of the study and highlight areas for future research, including adapting the developed principles and tools to other data types and machine learning methodologies.

2.4. Problem Formulation

Consider a dataset $\mathscr{D}$ comprising input-output pairs (X,Y), where X represents the input features and Y the target variable. Let each record in the input space be denoted as $X_i \in X$ with N features $\{X_i^1,...,X_i^N\}$, and the corresponding output label as $Y_i \in Y$. The training dataset, comprising M instances, is represented as $\mathscr{D}_{train} = \{X_{train}, Y_{train}\} = \{(X_1,Y_1),...,(X_M,Y_M)\}$, with analogous representations for the validation and testing datasets, $\mathscr{D}_{valid}$ and $\mathscr{D}_{test}$, respectively. In predictive modeling, a machine learning algorithm $\mathscr{A}$ inputs $\mathscr{D}_{train}$ and $\mathscr{D}_{valid}$ to produce a predictive function $\mathscr{F} : X \to Y$, predicting the output label. The loss of $\mathscr{F}$ on $\mathscr{D}_{valid}$ is denoted by $\mathscr{L}$.

The aim of automated FE is to devise a generation function $\Psi : X \to \mathbb{X}$ that transforms the original input features X into a new expressive feature representation $\mathbb{X}$, utilizing a set of operators $\mathscr{O} = \{o_1,...,o_{|\mathscr{O}|}\}$, thereby minimizing the loss $\mathscr{L}$. This can be formally expressed as:

$$\Psi^* = argmin_{\Psi} \mathscr{L}(\mathscr{F}(\Psi(X_{valid})), Y_{valid}) \tag{2.1}$$

$$F^* = argmin_{F} \mathscr{L}(\mathscr{F}(\Psi^*(X_{valid})), Y_{valid}) \tag{2.2}$$

where $\mathscr{F}$ is derived from $\mathscr{F} = \mathscr{A}(\mathscr{D}_{train\_new}, \mathscr{D}_{valid\_new})$, with $\mathscr{D}_{train\_new} = \Psi(X_{train}, Y_{train})$ representing the transformed training dataset. Similarly, $\mathscr{D}_{valid\_new}$ and $\mathscr{D}_{test\_new}$ can be generated.

Each generated feature $\mathbb{X}^i$ is obtained by applying an operation $o \in \mathscr{O}$ to one or more raw input features. These operations can be unary (e.g., $log(x)$, x^2), binary (e.g., arithmetic operations, logical conjunctions), ternary (e.g., conditional operators), or n-ary (e.g., MAX, MIN). Domain-specific operators, such as LAG in time series, are also considered, emphasizing the need for automated FE frameworks to be comprehensive and adaptable.

The Combined Algorithm Selection and Hyperparameter tuning (CASH) problem in the context of supervised tasks is formally stated as:

Let $A = \{A^{(1)},...,A^{(R)}\}$ be a set of ML algorithms, each with a hyperparameter domain $\Lambda^{(j)}$. With $\mathscr{D}_{train\_new} = \Psi(\mathscr{D}_{train})$ split into K cross-validation folds $\{\mathscr{D}_{valid}^{(1)},...,\mathscr{D}_{valid}^{(K)}\}$ and $\{\mathscr{D}_{train}^{(1)},...,\mathscr{D}_{train}^{(K)}\}$, the loss function $\mathscr{L}(A_{\lambda}^{(j)}, \mathscr{D}_{train}^{(i)}, \mathscr{D}_{valid}^{(i)})$ denotes the loss of algorithm $A^{(j)}$ on $\mathscr{D}_{valid}^{(i)}$ when trained on $\mathscr{D}_{train}^{(i)}$ with hyperparameters λ. The CASH problem is thus:

$$A^*, \lambda^* \in argmin_{A^{(j)} \in A, \lambda \in \Lambda^{(j)}} \frac{1}{K} \sum_{i=1}^{K} \mathscr{L}(A_{\lambda}^{(j)}, \mathscr{D}_{train}^{(i)}, \mathscr{D}_{valid}^{(i)}) \tag{2.3}$$

3. BACKGROUND AND RELATED WORK

3.1. Overview of Machine Learning

Machine Learning (ML) is an essential subset of artificial intelligence focused on enabling systems to learn from data, identify patterns, and make decisions with minimal human intervention. It plays a transformative role in various fields, including healthcare, finance, and technology, by providing innovative solutions to complex problems. The significance of ML lies in its ability to process and analyze large datasets, extract meaningful insights, and predict future trends or behaviors [35].

ML can be categorized into four main types:

- **Supervised Learning:** This involves training a model on a labeled dataset where the outcome variable is known. For instance, in spam detection, emails are labeled as "spam" or "not spam," and the model learns to predict these labels for new emails [89]. Another example is image recognition, where images are labeled with the objects they contain, allowing the model to identify these objects in new images.
- **Unsupervised Learning:** Deals with unlabeled data, focusing on discovering hidden patterns or structures within the input data. A common application is clustering, such as grouping customers based on purchasing behavior without prior labels [39]. Another example is dimensionality reduction, where high-dimensional data is reduced to lower dimensions while retaining essential patterns.
- **Semi-supervised Learning:** A hybrid approach that uses both labeled and unlabeled data for training. This is particularly useful in situations where acquiring a fully labeled dataset is challenging. For instance, in medical diagnosis, a small number of labeled medical images can be supplemented with a large number of unlabeled images to improve model performance [108].
- **Reinforcement Learning:** This method involves training models through a reward system. An example is training a model to play a game like Go, where the model receives rewards for winning and penalties for losing, learning to optimize its strategy over time [95].

Classical ML involves manual feature selection and employs algorithms like decision trees, linear regression, and support vector machines [10]. In contrast, Deep Learning, a branch of ML, operates on neural networks with multiple layers to learn from data, especially effective for large amounts of unstructured data, as seen in computer vision and natural language processing [67].

	Release Date	Popularity (#of stars on GitHub)	Optimization Technique	ML Tool Box	Meta-Learning	Post-processing	GUI	Data Pre-processing
AutoWeka	2013	314	Bayesian optimization	Weka	×	×	✓	✓
AutoSKlearn	2015	7k	Bayesian optimization	Scikit-Learn	✓	Ensemble selection	×	✓
TPOT	2016	9.1k	Evolutionary optimization	Scikit-Learn	×	×	×	×
Recipe	2017	50	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	×	✓

Table 1: Comparison table of the functionality of the AutoML frameworks considered in this study as of Jul 9, 2023.

3.2. Feature Engineering in Machine Learning

Feature Engineering (FE) is a crucial step in the machine learning pipeline, where raw data is transformed and enriched to improve the performance of learning algorithms. Effective FE can significantly enhance the accuracy of models and is often considered more important than the choice of the model itself [54].

Traditionally, FE has been a manual and domain-specific process, requiring substantial expert knowledge and intuition to identify relevant features. This process involves selecting the most relevant features, creating new features, and transforming existing features to better capture the underlying patterns in the data [36].

Traditional Feature Engineering: Manual feature engineering has been a labor-intensive process requiring domain expertise. Traditional methods involve the manual creation of new features through transformation, extraction, and selection processes. For instance, in financial data analysis, features like moving averages or financial ratios are manually created to improve model performance.

Automated Feature Engineering: Tools such as AutoFeat automate this process, generating and selecting features algorithmically to reduce human effort and improve model performance. AutoFeat, for example, can automatically create polynomial features and interactions between existing features, significantly enhancing model accuracy in various

3.3. Automated Machine Learning Frameworks

This section provides an introduction to the evaluated AutoML frameworks used in this study. The evaluation focuses on aspects such as popularity (measured in terms of the number of stars on GitHub), the machine learning toolbox used, optimization techniques employed, utilization of meta-learning to learn from previous experience, performance of post-processing tasks such as ensemble construction, availability of a Graphical User Interface (GUI), and capability for data pre-processing. A brief comparative overview across these frameworks is presented in Table 1, with more detailed comparisons to follow.

AutoWeka is built on top of Weka, a comprehensive ML library in Java, and uses Bayesian optimization techniques like SMAC and TPE for algorithm selection and hyperparameter tuning. SMAC is known for its efficiency in drawing relationships between algorithm performance and hyperparameters, while TPE effectively distinguishes between low and high-performing parameter configurations [98].

AutoSKlearn, implemented in Python on top of Scikit-Learn, focuses on automating the creation of ML pipelines for classification and regression tasks. It uses SMAC for optimization and features meta-learning to enhance the initialization process. AutoSKlearn also includes an ensemble selection mechanism, improving the performance of the final model. The framework offers several execution options, each tailored to specific needs. These include the vanilla version (`AutoSKlearn-v`) without meta-learning or ensemble features, the meta-learning version (`AutoSKlearn-m`), the ensembling version (`AutoSKlearn-e`), and the full version (`AutoSKlearn`) that incorporates both meta-learning and ensembling features [27].

TPOT is an evolutionary AutoML framework that represents ML pipelines as computational graphs. These graphs are optimized using a multi-objective approach, which simultaneously minimizes pipeline complexity and maximizes performance. This approach helps mitigate overfitting, a common issue in large search spaces [81].

Recipe operates similarly to TPOT but with a focus on classification tasks. It employs a grammar-based genetic algorithm, which enhances the search process by reducing the generation of invalid pipelines, thus expediting the optimization procedure [90].

ATM is designed for optimizing ML pipelines in classification tasks. It supports distributed execution across multiple nodes and cores, featuring a shared model hub that stores execution results. This hub plays a crucial role in refining the selection of potentially superior pipelines. ATM leverages a hybrid approach combining Bayesian optimization and multi-armed bandit techniques [96].

SmartML, the first R package for AutoML in classification tasks, uses meta-learning in the algorithm selection phase, where it compares the features of the input dataset with those in its knowledge base. Hyperparameter tuning in SmartML is based on SMAC, and it continuously updates its knowledge base with new runs to improve future performance. SmartML offers a base meta-learning version (`SmartML-m`) and an ensemble version featuring a voting mechanism (`SmartML-e`) [72].

3.4. Related Work in Efforts in Benchmarking

Recently, few research efforts have attempted to tackle the challenge of benchmarking different AutoML frameworks [34, 41, 92, 99, 109]. In general, most experimental evaluation and comparison studies show no framework always per-

formed the best, as some trade-offs always need to be considered and optimized according to user-defined objectives. For example, Gijsbers et al. [33] conducted a study to compare the performance of 9 AutoML frameworks, namely, Autogluon-tabular [21], `AutoSKlearn`, `AutoSKlearn 2` [26], `FLAML` [103], `GAMA` [32], `H2O AutoML` [68], `LightAutoMLs` [100], `MLjar` [88], and `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. [34] have conducted an experimental study to compare the performance of 4 AutoML frameworks, namely, `AutoWeka`, `AutoSKlearn`, `TPOT` 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. [99] compared the performance of 7 AutoML frameworks, namely, `H2O`, `Auto-keras` [48], `AutoSKlearn`, `Ludwig`[1], `Darwin`[2], `TPOT` and `Auto-ml`[3] 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 [109] compared the performance of different optimization techniques, namely, *Grid Search*, *Random Search*, *RObust Bayesian Optimization* (ROBO) [58], *Bayesian Tuning and Bandits* (BTB) [94], *hyperopt* [7], *SMAC* [46], *BOHB* [22] and *Optunity* [94]. 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` [59], `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,

[1] `https://github.com/uber/ludwig`

[2] `https://www.sparkcognition.com/product/darwin/`

[3] `https://github.com/ClimbsRocks/auto_ml`

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.

3.5. Automated Feature Engineering

3.5.1. Introduction to Automated Feature Engineering

Automated Feature Engineering (Automated FE) encompasses tasks such as feature selection, combination, and arithmetic operations, which are crucial in machine learning. Unlike traditional approaches that often involve a manual and iterative process, Automated FE introduces a level of automation that significantly enhances the efficiency and effectiveness of feature engineering. Experts traditionally engage in a trial-and-error method, starting with a feature engineering plan followed by model training and evaluations [62, 63]. In contrast, Automated FE is known for its robust exploration capabilities and potential to reduce labor costs [91, 104, 105].

Frameworks developed for Automated FE, such as those by Chen et al. [15], Horn et al. [45], Kanter et al. [51], and Zhu et al. [107], have been particularly well-received in industry due to their exhaustive exploration schemes and comprehensive feature generation capabilities. In the following, we introduce two of the most used automated FE frameworks.

`AutoFeat` is a Python package designed to enhance linear regression and classification models, similar to scikit-learn, by automating the generation and selection of features [44]. This library is versatile for single-table datasets. The developers of `AutoFeat` suggest that neural networks and deep learning are essentially linear classifiers. They believe that these models achieve their performance through complex, though non-transparent, features. Accordingly, they argue that similar results could be achieved with linear models by creating informative and interpretable features. The process to develop these new features employs a 'generate-and-select' method. In the first phase, the tool generates all feasible combinations from a set of predefined nonlinear functions (like $log(x)$, $\sqrt{x}$, $1/x$, x^2, x^3, $sin(x)$, e^x) applied to the input features. These generated features are then merged using the primary arithmetic operations ($+, -, *$, and $/$). In

the next step, these features undergo normalization, followed by training a linear model on them to identify and select those with significant coefficients.

SAFE (Scalable Automatic Feature Engineering) offers an innovative approach for automatic feature engineering, particularly tailored for industrial classification tasks. It introduces a staged method that enhances both efficiency and scalability. This is achieved by focusing on the generation of original feature pairs with a high likelihood of producing effective new features, coupled with a comprehensive feature selection pipeline. This pipeline evaluates individual features, redundancy in feature pairs, and feature importance assessed through typical tree models, making it adaptable to various datasets and machine learning algorithms [93].

3.5.2. Desired Properties of Automated FE Frameworks

An effective Automated FE framework should satisfy key criteria to address the complexities of modern machine learning tasks:

- **Ease of Use:** Ensures the framework is accessible to non-experts. Minimal requirements include an intuitive user interface and pre-configured settings for common tasks [42]. For example, AutoSklearn's user-friendly interface allows users to quickly set up and run experiments without deep ML knowledge.
- **Interpretability:** Crucial for applications requiring transparency, such as healthcare [17]. For instance, providing clear explanations that link the generated feature to the original feature set can help healthcare professionals trust and adopt the models.
- **Scalability:** Necessary for handling large datasets, as data volumes continue to grow [49]. A scalable framework like H2O AutoML can efficiently process massive datasets in distributed computing environments.
- **Model-Agnostic Functionality:** Allows the framework to be used with various ML models [101]. For example, a model-agnostic framework can switch between different types of algorithms (e.g., decision trees, Support vector machine) without the need to change the engineered feature set.
- **Effectiveness:** Demonstrated improved predictive performance across diverse datasets. For example, benchmarks showing that a framework consistently outperforms others in various tasks validate its effectiveness.

These properties are foundational for the development of efficient and comprehensive Automated FE frameworks, guiding the evolution of this field.

3.5.3. FE for different characteristics of Big Data

FE is influenced heavily by the input data's characteristics, often described by the five V's of Big Data: Volume, Velocity, Variety, Veracity, and Value. The **Volume** of the data suggests a need for scalable FE techniques due to increased computational complexity. Approaches like distributed computing with Apache

Spark, sampling for smaller datasets, and dimensionality reduction through PCA and SVD are essential [111, 74, 38]. **Velocity** requires FE techniques that adapt and transform data swiftly, with online learning algorithms and stream processing frameworks like Apache Kafka and Apache Flink playing crucial roles [14, 30]. **Variety** in data types necessitates unified data processing pipelines and automated data wrangling tools to handle data seamlessly [38, 30]. Addressing **Veracity** involves implementing robust preprocessing, anomaly detection, and imputation methods to ensure data quality [30, 111, 38]. Lastly, **Value** focuses on extracting meaningful features through domain knowledge, feature selection techniques, and automated FE tools, enhancing model performance and facilitating insightful decision-making [111, 74, 30]. Addressing these five V's is crucial for developing effective FE strategies that cater to the complexities of big data, ensuring the generation of reliable and valuable features for machine learning models.

3.5.4. Advancements and Studies in Automated FE

Recent advancements in Automated FE have focused on enhancing the feature generation and selection processes to improve machine learning models' performance. Several studies have explored various frameworks and methodologies, analyzing their effectiveness in different machine learning scenarios. For example, research efforts have evaluated Automated FE tools based on criteria like scalability, interpretability, and the ability to integrate with existing machine learning pipelines [15, 45, 51, 107]. These studies contribute to a growing body of knowledge that underscores the importance of efficient feature engineering in achieving optimal model performance and offers insights into the future direction of Automated FE development.

3.6. Related Work in Automated Feature Engineering

Deep Learning (DL) techniques, while successful in handling data such as text, tabular data, video, and speech, have certain limitations in the context of feature engineering [4, 67]. DL models are adept at learning expressive representations from input features but face challenges, including the need for large datasets, limited success outside specific data types, substantial computational resource requirements, and lack of interpretability in the extracted features. Efforts to improve the interpretability of neural networks are ongoing [18, 78], but practical applications, especially in critical domains, often demand more interpretable and efficient solutions [19]. These challenges underline the importance of relying on interpretable automated feature engineering techniques.

3.6.1. Heuristic Search-Based Approaches in Automated FE

The traditional manual approach to feature engineering has been supplemented by heuristic search-based methods. Exhaustive search-oriented algorithms, for exam-

ple, systematically explore transformation paths, but they suffer from significant computational complexity [51, 44, 87, 23, 53]. Cognito improves upon traditional methods by incorporating well-known algorithms like BFS or DFS to reduce computational demands [57]. SAFE introduces a staged approach for industrial classification tasks, focusing on effective feature exploration and improved feature selection mechanisms [93]. AutoFeat, meanwhile, employs local region search through iterative down-sampling [44].

3.6.2. Learning-Based Approaches in Automated FE

Learning-based approaches in Automated FE have gained increasing popularity in recent years. The LFE algorithm parametrizes feature transformations, simplifying the action space but neglecting high-order feature generation [80]. An algorithm representing the feature engineering process as a graph addresses the challenge of feature explosion [56]. The NFS framework uses reinforcement learning (RL) with an RNN-based controller for feature generation [15]. The Catch framework employs hierarchical-policy reinforcement learning for collaborative feature engineering [71], while DIFER explores features within a limited action space [107].

3.6.3. Meta-Learning Approaches in Automated FE

Meta-learning techniques have also been employed in Automated FE frameworks. ExploreKit uses meta-learning to evaluate and select new features, though it faces scalability challenges [52]. The LFE framework adopts a meta-learning approach using meta-features from prior tasks to recommend useful transformations [80]. Despite advancements, the efficiency and scalability of these frameworks are still not sufficient for widespread real-world application. The demand for high-performing, efficient, scalable, and interpretable Automated FE frameworks remains a significant area for future development in machine learning.

4. EVALUATION AND COMPARISON OF AUTOML FRAMEWORKS

4.1. Introduction

In this chapter, we present a comprehensive evaluation and comparison of various Automated Machine Learning (AutoML) frameworks. Our analysis focuses on six prominent frameworks, namely `AutoSKlearn`, `ATM`, `TPOT`, `SmartML`, `Recipe`, and `AutoWeka`, assessed over 100 datasets from OpenML. This evaluation aims to understand each framework's performance characteristics across different time budgets, highlighting their strengths and limitations in various scenarios of their design decisions.

4.2. Benchmark Design

We have selected specific datasets, optimization metrics, and design decisions for each benchmark task. Additionally, we have allocated a specific time budget for each task. Below, we will provide a brief explanation for our choices.

Datasets In this study, a total of 100 datasets sourced from the well-known OpenML repository [102] were utilized to evaluate the performance of AutoML frameworks. The datasets were chosen based on various criteria such as the number of classes, features, instances, categorical features, instances with missing values, and class entropy. Detailed descriptions of the datasets used can be found in Table 16 in Appendix A.1. The datasets were a combination of binary and multiclass classification tasks, and the largest dataset size was found to be 643MB.

Performance metrics The benchmark can be executed utilizing various measures as per the user's preference. For this study, the results are reported based on F1-score. AutoML frameworks are optimized for the same metric on which they are evaluated. The measures are estimated with hold-out validation, where each dataset is divided into two parts - 70% for training and 30% for testing. All AutoML frameworks are applied to the same training and testing splits across all datasets. To eliminate the effects of non-deterministic factors, the performance reported in each experiment is based on an average of 10 trials. If the number of failed trials exceeds or is equal to 5, we report a performance of 0 for any framework.

Frameworks and design decisions The selection of frameworks for this study was based on several criteria, including ease of use, variety of underlying optimization techniques and ML toolboxes, popularity assessed by the number of stars on GitHub, and citation count. All the frameworks considered in this study are open source, and the source code reference for each of them is available in Table 17 in Appendix A.2. We intend to include more frameworks in future work. For AutoSKlearn, we explore four execution options, namely, AutoSKlearn-v, AutoSKlearn-m, AutoSKlearn-e, and AutoSKlearn. Similarly, for SmartML, we

investigate two execution options, SmartML-m and SmartML-e. We analyze several design decisions, such as the size of the search space, meta-learning, and ensemble construction as a post-processing step. However, we only investigate these design decisions for AutoML frameworks that support configuring these decisions. It is crucial to note that the optimization technique is not uniform across all the frameworks, which makes it challenging to draw definitive conclusions about this aspect in this benchmark. We evaluate 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 In order to evaluate the efficacy of various AutoML frameworks that are incorporated in this study, a baseline approach is employed. This approach comprises of a basic pipeline that involves imputation of absent values and a random forest model, as mentioned in [84].

Time budget choice Four different time budgets were employed to evaluate the performance of various AutoML frameworks. Each framework was subject to a soft time limit (10, 30, 60, and 240 minutes) and a corresponding hard limit (10% above the soft limit). If the hard time limit is exceeded, the run is terminated and considered failed. While it would be ideal to remove time limits for each framework to ensure optimal performance, doing so for all six frameworks with different configurations across 100 datasets with ten trials each would be highly time-consuming. Therefore, four time budgets were used, resulting in over 40000 experiments that ran for a total of more than 88366-hour EC2 run-time. To maintain practical constraints on experiment run-time and cost, the maximum cut-off timeouts of four and eight hours were tested on 14 randomly selected datasets. The results, presented in Table 18 in Appendix A.3, show no statistically significant difference in performance between the AutoML frameworks under the two-time budgets. Thus, the eight-hour budget is not further considered. 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 18).

Statistical Test The Wilcoxon test serves as a powerful non-parametric statistical tool, primarily used when comparing two related samples or repeated measurements on a single sample to assess whether their population mean ranks differ. Essentially, its main goal is to evaluate if the observed differences between two sets of data can be attributed to something other than mere chance or randomness. This characteristic makes it particularly valuable in our benchmarking study, where ensuring that the differences in performance metrics aren't just due to random variation is crucial. Since it does not rely on the assumption of normally distributed data, it offers a versatile approach for analysis under various data conditions. This ensures that our conclusions are grounded in statistically significant differences, boosting the confidence in the decisions and insights drawn from our benchmarking efforts.

Hardware choice and resource specifications Our research experiments were

carried out utilizing Google Cloud machines, each of which was configured with 2 vCPUs, 7.5 GB RAM, and the ubuntu-minimal-1804-bionic operating system. The machines employed Python 2.7.15, Python 3.6.8, scikit-learn 0.21.3, R 3.4.4, and Java 1.8. To circumvent memory leakage issues, we performed a reboot of the machines after each run, ensuring that every experiment was executed with the same available memory size.

4.3. Experimental Evaluation

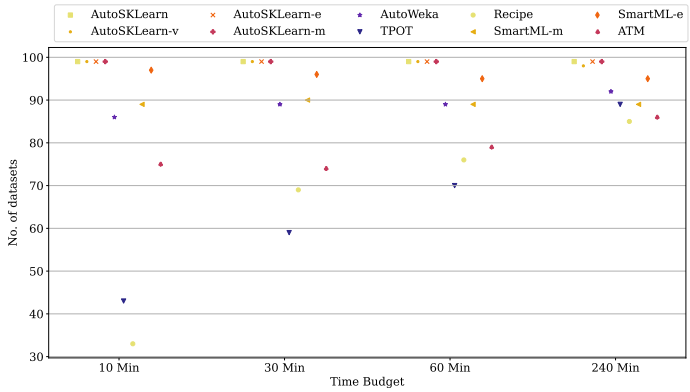
In this section, we present a set of empirical assessments for various AutoML frameworks. Initially, we conduct a comparison of the overall performance of the different AutoML frameworks in Section 4.4. Then, in Section 4.5, we investigate the influence of different design choices on the performance of the distinct AutoML frameworks.

4.4. General Performance Evaluation

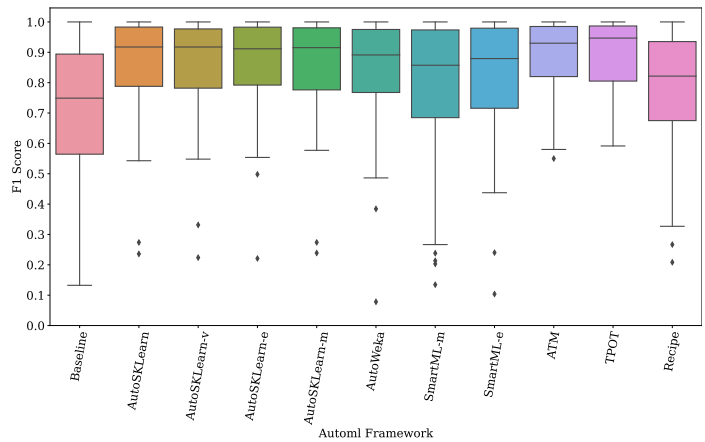
The present section aims to evaluate and compare the overall performance of benchmark frameworks, focusing on several aspects, including (a) the number of successful runs, (b) the average performance of the final pipeline per AutoML framework across all datasets, (c) the significance of the performance difference between different frameworks across different time budgets, (d) the robustness of the benchmark frameworks, and (e) the quality of the machine learning pipelines generated by the benchmark frameworks.

In Figure 3(a), it is illustrated the number of datasets with successful runs of each framework on various time budgets. A failed experiment is considered when an AutoML framework could not generate a model for a specific dataset five times or more. The analysis shows that the increase in the time budget for the AutoML frameworks leads to an increase in the number of successful runs. Specifically, `AutoSKlearn` exhibits the largest number of successful runs across all time budgets, successfully running on 99 datasets for different time budgets. In the second place, we have `SmartML-e`, followed by `AutoWeka` and `SmartML`. Genetic-based frameworks, `TPOT` and `Recipe`, show the lowest number of successful runs, as indicated in Figure 3(a). For `Recipe` and `TPOT`, the number of successful runs achieved in the longest time budget of 240 minutes is almost double that achieved with the smallest time budget of 10 minutes. Therefore, larger budgets are preferable for `Recipe` and `TPOT`.

The average performances of all AutoML frameworks across various datasets over 240 minutes budget are reported in Figure 3(b). It is observed that all frameworks outperform the random forest baseline, but individual results vary significantly. The performance of all AutoML frameworks and the baseline across 10, 30, and 60 minutes is reported in Appendix A.4, with Figures 22, 23, and 24 respectively.



(a) Number of successful runs.



(b) Performance of the final pipeline per AutoML framework for 240 minutes.

Figure 3: General performance trends of the benchmark AutoML frameworks.

The pairwise "outperformance" analysis is conducted by calculating the number of datasets for which one framework outperforms another in different time budgets, as shown in Figure 4. The results reveal that no single AutoML framework performs best across all 100 datasets on all-time budgets. On average, the AutoSKlearn framework performs the best, followed by the ATM framework, while the Recipe framework comes last.

The Wilcoxon signed-rank test is carried out to determine if a statistically significant difference in performance exists between the AutoML frameworks and the baseline over different time budgets, with results summarized in Table 2. The results show that all AutoML frameworks except Recipe statistically outperform the baseline across all time budgets with a significant difference.

The ensembling version and the full version of AutoSKlearn statistically outperform most of the other frameworks across all time budgets. However, SmartML-m, SmartML-e, and AutoWeka are statistically outperformed by the majority

(a) 10 minutes time budget

(b) 30 minutes time budget

(c) 60 minutes time budget

(d) 240 minutes time budget

Figure 4: 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.

of the frameworks. Moreover, for longer time budgets of 60 and 240 minutes, TPOT significantly outperforms AutoWeka, Recipe, SmartML-m, SmartML-e, AutoSKlearn-m, and AutoSKlearn-v.

We analyze the performance of different AutoML frameworks based on various characteristics of datasets and tasks. In Figure 5, we present the mean performances of AutoML frameworks on both multi-class and binary-class classification tasks with a 240-minute budget.

It is important to note that all AutoML frameworks show less significant improvements on multi-class datasets than on whole datasets. Although `ATM` achieves the highest mean performance, `Recipe` also shows a comparable performance to the baseline.

We divide the datasets into different subgroups based on their characteristics. The second subgroup of datasets comprises those with a relatively large number of

10 Minutes

	Baseline	ATM	AutoWeka	Recipe	AutoSKLearn-e	AutoSKLearn-m	AutoSKLearn-v	AutoSKLearn	SmartML-m	SmartML-e	TPOT
Baseline		0.0	0.0	0.15	0.0	0.0	0.0	0.0	0.0	0.0	0.0
ATM	**0.0**		**0.008**	0.088	0.768	0.516	0.299	0.587	0.064	0.062	0.879
AutoWeka	**0.0**	0.008		0.156	0.0	0.0	0.004	0.0	0.748	0.096	0.06
Recipe	0.15	0.088	0.156		0.002	0.002	0.004	0.0	0.417	0.248	0.013
AutoSKLearn-e	**0.0**	0.768	**0.0**	**0.002**		0.569	**0.014**	0.001	**0.023**	0.203	0.492
AutoSKLearn-m	**0.0**	0.516	**0.0**	**0.002**	0.569		**0.009**	0.009	**0.004**	0.1	0.33
AutoSKLearn-v	**0.0**	0.299	**0.004**	**0.004**	0.014	0.009		0.0	**0.042**	0.663	**0.026**
AutoSKLearn	**0.0**	0.587	**0.0**	**0.0**	**0.001**	**0.009**	**0.0**		**0.001**	**0.035**	0.258
SmartML-m	**0.0**	0.064	0.748	0.417	0.023	0.004	0.042	0.001		0.014	0.022
SmartML-e	**0.0**	0.062	0.096	0.248	0.203	0.1	0.663	0.035	**0.014**		0.452
TPOT	**0.0**	0.879	0.06	**0.013**	0.492	0.33	0.026	0.258	**0.022**	0.452	

30 Minutes

	Baseline	ATM	AutoWeka	Recipe	AutoSKLearn-e	AutoSKLearn-m	AutoSKLearn-v	AutoSKLearn	SmartML-m	SmartML-e	TPOT
Baseline		0.0	0.0	0.346	0.0	0.0	0.0	0.0	0.0	0.0	0.0
ATM	**0.0**		**0.034**	**0.0**	0.898	0.408	0.159	0.85	**0.009**	**0.015**	0.902
AutoWeka	**0.0**	0.034		**0.004**	0.0	0.0	0.003	0.0	0.92	0.195	0.003
Recipe	0.346	0.0	0.004		0.0	0.0	0.0	0.0	0.134	0.007	0.0
AutoSKLearn-e	**0.0**	0.898	**0.0**	**0.0**		**0.015**	**0.0**	0.694	**0.001**	**0.005**	0.94
AutoSKLearn-m	**0.0**	0.408	**0.0**	**0.0**	0.015		**0.03**	0.152	**0.006**	0.075	0.316
AutoSKLearn-v	**0.0**	0.159	**0.003**	**0.0**	0.0	0.03		0.0	0.064	0.112	0.005
AutoSKLearn	**0.0**	0.85	**0.0**	**0.0**	0.694	0.152	**0.0**		**0.002**	**0.014**	0.337
SmartML-m	**0.0**	0.009	0.92	0.134	0.001	0.006	0.064	0.002		0.015	0.002
SmartML-e	**0.0**	0.015	0.195	**0.007**	0.005	0.075	0.112	0.014	**0.015**		0.065
TPOT	**0.0**	0.902	**0.003**	**0.0**	0.94	0.316	**0.005**	0.337	**0.002**	0.065	

60 Minutes

	Baseline	ATM	AutoWeka	Recipe	AutoSKLearn-e	AutoSKLearn-m	AutoSKLearn-v	AutoSKLearn	SmartML-m	SmartML-e	TPOT
Baseline		0.0	0.0	0.201	0.0	0.0	0.0	0.0	0.0	0.0	0.0
ATM	**0.0**		**0.017**	**0.0**	0.075	0.358	0.424	0.149	**0.005**	**0.004**	0.064
AutoWeka	**0.0**	0.017		**0.015**	0.0	0.0	0.0	0.0	0.59	0.083	0.0
Recipe	0.201	0.0	0.015		0.0	0.0	0.0	0.0	0.054	0.003	0.0
AutoSKLearn-e	**0.0**	0.075	**0.0**	**0.0**		**0.046**	**0.003**	0.319	**0.003**	**0.011**	0.198
AutoSKLearn-m	**0.0**	0.358	**0.0**	**0.0**	0.046		0.474	0.0	**0.012**	0.052	0.067
AutoSKLearn-v	**0.0**	0.424	**0.0**	**0.0**	0.003	0.474		0.0	**0.039**	0.201	0.01
AutoSKLearn	**0.0**	0.149	**0.0**	**0.0**	0.319	**0.0**	**0.0**		**0.001**	**0.015**	0.86
SmartML-m	**0.0**	0.005	0.59	0.054	0.003	0.012	0.039	0.001		0.047	0.0
SmartML-e	**0.0**	0.004	0.083	**0.003**	0.011	0.052	0.201	0.015	**0.047**		0.007
TPOT	**0.0**	0.064	**0.0**	**0.0**	0.198	0.067	**0.01**	0.86	**0.0**	**0.007**	

4 Hours

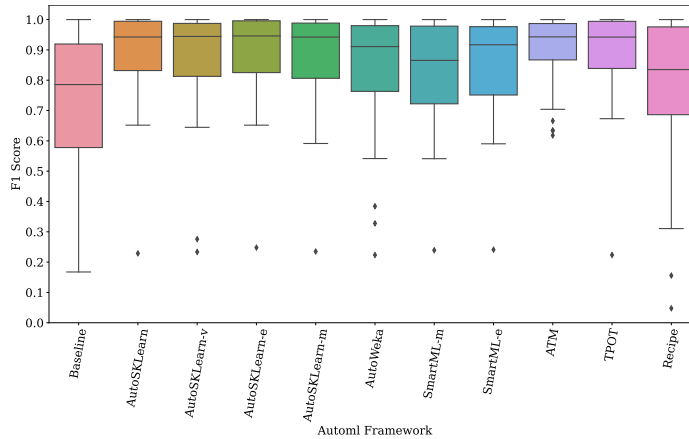
	Baseline	ATM	AutoWeka	Recipe	AutoSKLearn-e	AutoSKLearn-m	AutoSKLearn-v	AutoSKLearn	SmartML-m	SmartML-e	TPOT
Baseline		0.0	0.0	0.039	0.0	0.0	0.0	0.0	0.0	0.0	0.0
ATM	**0.0**		**0.046**	**0.0**	0.637	0.943	0.969	0.754	**0.002**	0.061	0.153
AutoWeka	**0.0**	0.046		**0.027**	0.0	0.001	0.002	0.0	0.773	0.389	0.0
Recipe	**0.039**	0.0	0.027		0.0	0.0	0.0	0.0	0.024	0.004	0.0
AutoSKLearn-e	**0.0**	0.637	**0.0**	**0.0**		**0.015**	**0.021**	0.447	**0.001**	**0.007**	0.152
AutoSKLearn-m	**0.0**	0.943	**0.001**	**0.0**	0.015		0.852	0.0	**0.006**	**0.043**	0.001
AutoSKLearn-v	**0.0**	0.969	**0.002**	**0.0**	0.021	0.852		0.001	**0.004**	0.06	0.0
AutoSKLearn	**0.0**	0.754	**0.0**	**0.0**	0.447	**0.0**	**0.001**		**0.0**	**0.002**	0.119
SmartML-m	**0.0**	0.002	0.773	**0.024**	0.001	0.006	0.004	0.0		0.031	0.0
SmartML-e	**0.0**	0.061	0.389	**0.004**	0.007	0.043	0.06	0.002	**0.031**		0.001
TPOT	**0.0**	0.153	**0.0**	**0.0**	0.152	**0.001**	**0.0**	0.119	**0.0**	**0.001**	

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).

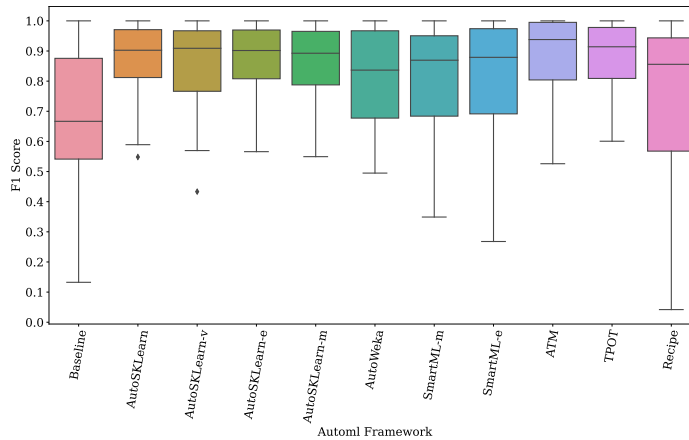
features and a small number of instances. Figures 26, 27, 28, and 25 in Appendix A.4 display the mean performance of different AutoML frameworks on datasets with various characteristics. Additionally, we present the mean performance of all frameworks on binary classification tasks in Figure 5(b) in Appendix A.4.

To test the robustness of the evaluated AutoML frameworks, we assess their ability to achieve the same results across different runs on the same input dataset. For a randomly selected dataset, we run each AutoML framework ten times with a ten-minute time budget. Figure 6 illustrates the robustness of the AutoML frameworks. The results indicate that the four versions of `AutoSKlearn` have the most stable runs, followed by `Recipe` and `AutoWeka`. On the other hand, the two versions of `SmartML` produce the least stable runs.

In this study, we evaluate the performance of machine learning pipelines created by AutoML frameworks, utilizing a set of precise evaluation metrics. This approach, inspired by the methodology in [2], aims to provide a careful assessment of AutoML frameworks, focusing on their efficiency, inference capabilities,



(a) Performance of the final pipeline on multi-class classification tasks.



(b) Performance of the final pipeline on binary classification tasks.

Figure 5: Performance of the different AutoML frameworks based on the various characteristics of datasets and tasks over 240 minutes.

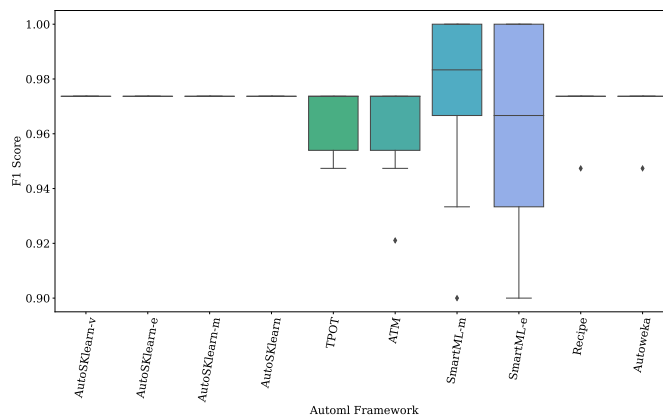


Figure 6: Evaluation of AutoML frameworks for robustness on (dataset_61_iris).

Framework	Training Efficiency (s)	Inference Performance (s)	Robustness to Noise	Model Stability
ATM	43.25 ± 1.17	0.100 ± 0.007	0.07	0.0014
AutoWeka	46.85 ± 1.28	0.103 ± 0.008	0.05	0.0016
Recipe	43.10 ± 0.95	0.098 ± 0.007	0.04	0.0013
AutoSKLearn-e	66.50 ± 1.11	0.106 ± 0.007	0.10	0.0017
AutoSKLearn-m	45.20 ± 0.98	0.099 ± 0.007	0.10	0.0015
AutoSKLearn-v	44.80 ± 1.15	0.101 ± 0.008	0.06	0.0018
AutoSKLearn	67.20 ± 1.02	0.105 ± 0.007	0.07	0.0016
SmartML-m	53.00 ± 0.93	0.098 ± 0.007	0.04	0.0014
SmartML-e	73.50 ± 1.10	0.107 ± 0.008	0.07	0.0017
TPOT	48.80 ± 1.19	0.104 ± 0.008	0.09	0.0015

Table 3: Performance evaluation of benchmark AutoML frameworks using key metrics: training efficiency, inference performance, robustness to noise and model stability.

stability, and noise resilience. The average performances of the frameworks across various datasets, based on these criteria, are tabulated in Table3. Below are the definitions for each of these metrics:

- **Training Efficiency:** This metric evaluates the time efficiency of model training, specifically the CPU wall-clock time taken. It reflects the computational resources expended during training. Table 3 indicates that ATM and Recipe are more efficient, making them suitable for environments with limited computational capacity or where swift model creation is crucial. Conversely, AutoSKLearn and SmartML-e exhibit longer training durations.
- **Inference Performance:** Here, we measure the model's inference speed by the CPU wall-clock time it takes to make a single prediction. Table 3 shows SmartML-m and Recipe leading in inference speed, an important aspect for applications requiring quick responses, like real-time recommendations or autonomous driving systems. In contrast, SmartML-e records the longest times for inference.
- **Model Stability:** This metric gauges a model's consistency by calculating the standard deviation of performance when retrained on identical datasets. Recipe, SmartML-m, and ATM show superior stability, with lower standard deviations, as per Table 3.
- **Robustness to Noise:** Assesses a model's tolerance to data noise. This is measured by observing changes in the model's output when noise is added to the test set, using the Cohen's Kappa metric [65]. The noise is introduced differently for continuous (Gaussian noise) and categorical features (random uniform noise). The model's reaction to noise, as seen in Table3, indicates that AutoSKLearn-e and AutoSKLearn-m are more robust compared to Recipe and SmartML-m.

Time Budget	Framework	$Mean_{Succ}$	Mean	SD	Time Budget	Framework	$Mean_{Succ}$	Mean	SD
10 Min	**ATM**	0.664	0.886	0.126	60 Min	**ATM**	0.700	**0.886**	**0.132**
	AutoWeka	0.724	0.842	0.165		**AutoWeka**	0.743	0.835	0.167
	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

Table 4: $Mean_{Succ}$, Mean and standard deviation of the predictive performance of AutoML frameworks per time budget. Bold entries highlight highest $Mean_{Succ}$, mean and lowest standard deviation

These metrics collectively provide a comprehensive view of the strengths and weaknesses of various AutoML frameworks, guiding users in selecting the most appropriate tool for their specific needs and constraints.

4.5. Performance Evaluation of Different Design Decisions

In this section, we aim to investigate the effect of various design choices such as the time budget (see Section 4.5.1), the search space size (see Section 4.5.2), meta-learning (see Section 4.5.3), and ensembling (see Section 4.5.4) on the performance of different AutoML frameworks over diverse time budgets. The reported performance of each framework in each experiment is based on the average of 10 runs.

4.5.1. Impact of Time Budget

Optimizing the time budget is a critical and complex task in AutoML, as it necessitates a balance between the available computational resources and the desired level of performance. This task requires careful consideration of the trade-offs between generalization and overfitting of the AutoML frameworks. To investigate the influence of the time budget on the performance of various AutoML frameworks, we analyzed the speed at which they can generate ML pipelines and their ability to consistently improve performance with more time. We evaluated each framework's performance on successful runs under four different time budgets: 10, 30, 60, and 240 minutes. Table 4 reports the mean (`Mean`) and standard deviation (`SD`) of performance for all successful runs at each time budget. Furthermore, we present the mean predictive performance weighted by the percentage of successful runs ($Mean_{succ}$).

$$Mean_{succ} = Mean \times \frac{N}{T} \tag{4.1}$$

Time Budget in minutes	Framework	Gain (g)		#datasets with			Loss (l)	
		Mean	Max	$g > 1\%$	$g \approx 0\%$	$l > 1\%$	Mean	Max
$10 \rightarrow 30$	**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
$30 \rightarrow 60$	**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	-5.7
	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
$60 \rightarrow 240$	**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

Table 5: 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.

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

The findings indicate that for the time budgets of 10 and 240 minutes, `Auto-SKlearn` and `AutoSKlearn-e` exhibit similar $Mean_{succ}$ values. However, `AutoSKlearn-e` stands out with the highest $Mean_{succ}$ across other time frames. On the other hand, `Recipe` records the lowest mean performance and $Mean_{succ}$ for all time budgets, as detailed in Table 4. It's noteworthy that the performance of the genetic-based frameworks, specifically `Recipe` and `TPOT`, shows improvement over time, as reflected in their increasing $Mean_{succ}$ values. Figures 31, 32, 33, 34, 35, 36, 37, 38, 39, and 40 in Appendix A.7 illustrate the effect of increasing time budgets on the performance of each AutoML framework across 100 datasets.

Contrary to common assumptions, extending the time budgets does not always result in enhanced performance, as evidenced in Table 5. The table reports either a gain (g) or loss (l) in the predictive performance of the frameworks with increased time budgets. This variation is calculated based on the mean and maximum predictive performance change across all datasets that either

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	**0.034**		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	0.013	**0.018**
	240	60	0.005	**0.042**		240	60	0.006	0.129
TPOT	30	10	0.009	0.117	AutoSKLearn-v	30	10	0.005	0.175
	60	10	0.008	0.388		60	10	0.008	**0.001**
	60	30	0.001	0.428		60	30	0.003	0.088
	240	10	0.013	**0.016**		240	10	0.005	**0.000**
	240	30	0.006	**0.035**		240	30	0.000	**0.040**
	240	60	0.004	**0.008**		240	60	-0.003	0.099
Recipe	30	10	0.014	0.866	AutoSKLearn-e	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	AutoSKLearn-m	30	10	0.004	0.156
	60	10	-0.007	0.254		60	10	0.006	0.105
	60	30	-0.008	0.499		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	SmartML-e	30	10	0.008	0.521
	60	10	0.009	0.832		60	10	0.003	0.589
	60	30	0.009	0.597		60	30	-0.004	0.672
	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

Table 6: Wilcoxon test p-values for all the AutoML frameworks over different time budgets. Bold entries highlight significant difference.

improved or declined. Upon extending the time budget from 10 to 30 minutes, `Recipe` demonstrates the highest average gain of 19.3 on 2 datasets, followed closely by `SmartML-m`, while `AutoSKlearn` shows a more modest mean gain of 3.5 across 17 datasets. Interestingly, `Recipe` features the fewest datasets with either performance improvement or degradation upon time budget extension. `AutoSKlearn-v` records the highest number of datasets showing performance enhancement when the time budget is increased from 30 to 60 minutes and from 60 to 240 minutes. Meanwhile, `AutoSKlearn-e` shows improvement over the most significant number of datasets when extending the time from 10 to 30 minutes. In contrast, `ATM` experiences the most substantial performance drop across the highest number of datasets when the time budget is increased from 10 to 30 minutes and then from 30 to 60 minutes.

The statistical significance of the average performance difference resulting from an increase in time budget is assessed through the Wilcoxon signed-rank test, as presented in Table 6. The impact of time budget extension varies across different frameworks. Notably, the frameworks AutoSKlearn-m, Recipe, ATM, SmartML-m, and SmartML-e do not exhibit significant performance differences. However, most versions of AutoSKlearn, AutoWeka, and TPOT reveal statistically significant differences. These findings suggest that end-users must carefully weigh the trade-off between time budget and performance for benchmark frameworks, depending on their specific objectives.

The time budget for running AutoML frameworks is a critical factor that impacts their performance and applicability. Given the diverse characteristics of datasets, it is essential to establish guidelines for setting appropriate time budgets automatically. By analyzing the number of instances, features, and class distribu-

tion of a dataset, we can estimate a suitable time budget that balances computational efficiency and model performance. For instance, datasets with a large number of features (e.g., Amazon and AP_Breast_Lung) may require extended time budgets to allow for comprehensive feature selection and model training. Conversely, datasets with fewer features (e.g., jungle_chess_2pcs and ipums_la_99-small) can achieve optimal performance with shorter time budgets. This approach ensures that the time budget allocated for each dataset is aligned with its complexity and computational demands.

4.5.2. Impact of the Size of Search Space

The search space is a crucial aspect of exploring the structural paradigm of various optimization methods. However, designing an effective search space is a challenging task. Figure 7 provides an overview of the most commonly used ML models across different AutoML frameworks. The analysis of the best-performing models reveals that no single ML algorithm dominates all AutoML frameworks. Nevertheless, tree-based models are the most frequent across all frameworks for all time budgets. Specifically, the returned pipelines by `AutoWeka`, `AutoSKLearn-v`, and `SmartML-m` indicate that the random forest is the most commonly used classifier, as shown in Figures 7(a), 7(c), and 7(e). The most frequently used classifiers for `AutoSKlearn-m`, `TPOT`, and `Recipe` are gradient boosting, as depicted in Figures 7(d), 7(f), and 7(g), respectively. In contrast, to optimize the time budget, `ATM` restricts its default search space to only three classifiers, namely, k-nearest neighbours, decision tree, and logistic regression. The decision tree is the most frequently used one, as shown in Figure 7(b).

To address the time-bounded optimization challenge in AutoML, it is crucial to define and efficiently navigate the search space for optimal ML pipelines. However, these search spaces are often arbitrarily chosen, sometimes leading to inefficient searches and suboptimal results as noted in Zoller and Hutter[109]. In this context, we explore the effectiveness of a budget allocation strategy as an additional design consideration for AutoML frameworks. This strategy involves employing a static portfolio, as discussed by Kotthoff et al. [60], which comprises a set of configurations that can handle a wide range of datasets while minimizing failure risks on new tasks. For this study, a portfolio comprising the top three classifiers (support vector machine, random forest, and decision tree), supported by all AutoML frameworks and demonstrated to be effective across 100 datasets, is utilized. We refer to this portfolio of the three classifiers as 3*C*. The predictive performance of using this portfolio (3*C*) is compared against the full classifier set (denoted as *FC*) in AutoML frameworks that allow search space configuration, namely `ATM`, `AutoSKlearn`, and `TPOT`. The comparison, based on a 30-minute time budget, is summarized in Figure 8.

In `AutoSKlearn` (Figure 8(a)), the *FC* outperforms 3*C* on 28 datasets, with an average predictive performance gain of 3.3%. Conversely, 3*C* outperforms

(a) AutoWeka (b) ATM

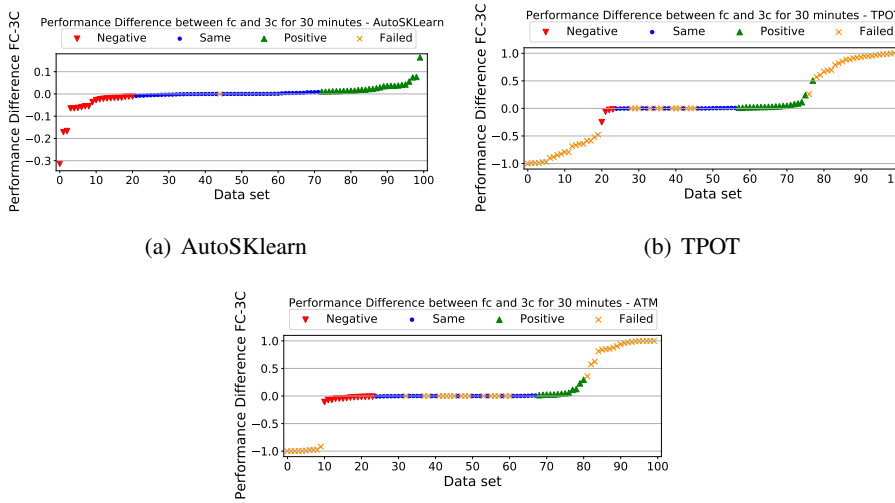
(c) AutoSKlearn-v (d) AutoSKlearn-m

(e) SmartML-m (f) TPOT

(g) Recipe

Figure 7: The frequency of using different machine learning models by the different AutoML frameworks.

FC on 21 datasets by 5.9%. This variation is due to the framework's focus on tuning historically well-performing classifiers. The strategy focuses the search on promising areas and ignores less relevant ones. On 50 datasets, the performances



(a) AutoSKlearn

(b) TPOT

(c) ATM

Figure 8: 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 3*C* search space, yellow markers on the left represent failed runs with *FC* but successful with 3*C*, yellow markers on the right represent failed runs with 3*C* but successful with *FC*, and yellow markers in the middle represent failed runs with both *FC* and 3*C*.

of *FC* and 3*C* are similar, with less than 1% difference in predictive performance.

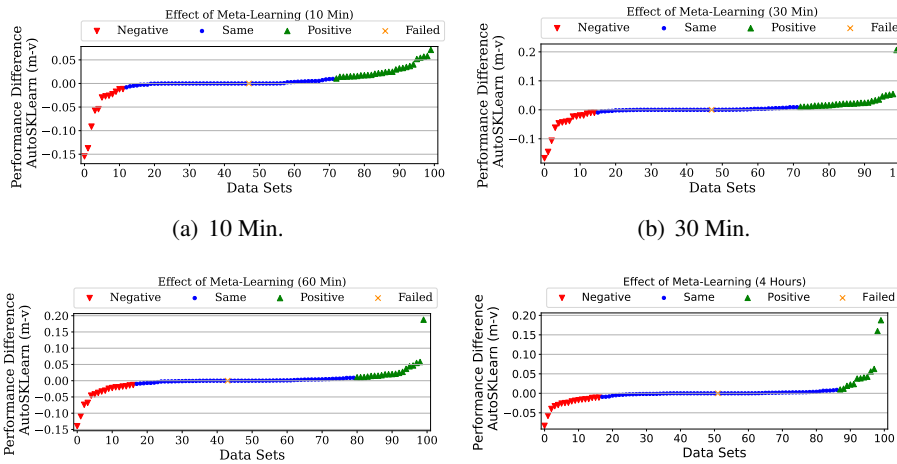
For `TPOT` (Figure 8(b)), 23 datasets failed under 3*C*, while 20 failed under *FC*. Both configurations were unsuccessful for 12 datasets. Where successful, *FC* outperformed 3*C* on 21 datasets, with an average performance boost of 9.6%. However, 3*C* was superior on six datasets, with an average performance increase of 8.8%. The performance was comparable on 18 datasets.

Regarding `ATM` (Figure 8(c)), 3*C* outperformed *FC* on 17 datasets with a 4% average improvement, while *FC* was better on 15 datasets, with a 9.3% average improvement. The performance of both configurations was comparable on 22 datasets. Notably, *FC* failed on 19 datasets where 3*C* succeeded, and vice versa for 10 datasets.

A Wilcoxon signed-rank test was conducted to evaluate the statistical significance of performance differences between *FC* and 3*C* across all datasets. For `TPOT`, the test confirms a statistically significant difference with over 95% confidence (p-value=0.003). However, no significant performance difference was found between the two search spaces for `AutoSKlearn` and `ATM`.

4.5.3. Impact of Meta-learning

Meta-learning refers to the process of acquiring knowledge from past experiences gained by utilizing diverse learning algorithms on various machine learning tasks [101]. This approach aims to reduce the time required for learning new tasks. In this study, we examine the impact of meta-learning on the performance of AutoML frameworks. `AutoSKlearn` is currently the only framework that supports meta-learning configuration. Furthermore, we explore the correlation between the characteristics of different datasets and the enhancement achieved by utilizing either the vanilla version or the meta-learning version of `AutoSKlearn`.



(a) 10 Min.

(b) 30 Min.

(c) 60 Min.

(d) 240 Min.

Figure 9: The impact of meta-learning over all time budgets. Green markers represent better performance with `AutoSKlearn-m`, blue markers represent comparable performance with a difference less than 1%, red markers represent better performance using `AutoSKlearn-v`, and yellow markers represent failed runs with both runs with both FC and 3C.

The AutoSKLearn tool leverages a meta-learning mechanism that relies on a knowledge base containing meta-features of datasets and the best-performing pipelines on these datasets. The tool uses a variety of meta-features, including statistical, information-theoretic, and simple features, and stores them along with the best-performing pipelines in an offline phase for each of the 140 datasets in the OpenML repository. In an online phase, when a new dataset is entered, the tool extracts its meta-features and searches for the most similar datasets in the knowledge base. It then returns the top k best-performing pipelines on these similar datasets, which can act as a warm start for the Bayesian optimization algorithm used in the optimization process.

In order to evaluate the effectiveness of the meta-learning mechanism, we conducted a comparison between `AutoSKlearn-v` and `AutoSKlearn-m` on 100 datasets using different time budgets. The results, as shown in Figure 9, indi-

Time Budget	Framework	Predictive Performance		Performance Gain			#datasets with gain > 1%
		Mean	SD	Min	Mean	Max	
10	AutoSKlearn-m	0.864	0.153	1.1%	2.9%	7.1%	28
	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
	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
	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
	AutoSKlearn-v	0.867	0.156	1.1%	2.7%	8.4%	20

Table 7: 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.

cate that utilizing meta-learning does not necessarily lead to better performance. On average, the vanilla and meta-learning versions exhibited very similar performance across the four different time budgets. Specifically, both versions performed comparably on 64, 55, 65, and 69 datasets for 10 minutes, 30 minutes, 60 minutes, and 240 minutes, respectively. Table 7 provides a summary of the performance of `AutoSKlearn-m` and `AutoSKlearn-v`, as well as the number of datasets that showed improvement in performance when `AutoSKlearn-m` was used instead of `AutoSKlearn-v` under different time budgets. The improvement achieved by using the meta-learning version decreases as the time budget increases. For instance, while 28 datasets showed improvement with the use of meta-learning under a 30-minute budget, this number decreased to 14 for a 240-minute budget as shown in Table 7.

To evaluate the performance difference between the vanilla version and the meta-learning version, we conducted a statistical test using Wilcoxon. Based on the results, it was found that the impact of meta-learning is only statistically significant for the smallest time budget of 10 minutes. This result was obtained with more than 95

We investigate how the characteristics of datasets are related to the improvement achieved by using the meta-learning version of `AutoSKlearn` across different time budgets. Our model utilizes the meta-features of datasets, which are their characteristics, to predict whether meta-learning can enhance the performance. We have labeled each dataset as either `Class 1` if using meta-learning improves performance over the vanilla version or `Class 0` otherwise. We have used a total of 42 meta-features from the literature, including simple, information-theoretic, and statistical meta-features [50, 77], such as statistics about the number of data points, features, and classes, as well as data skewness and entropy of the targets. You can find all meta-features listed in Appendix A.5, Table 19. We have built a decision tree classifier of depth 4 using the meta-feature variables as predictors. We have chosen the decision tree classifier due to its interpretable nature, allowing rules to be derived from a root-leaf path in the tree. We have extracted information from the knowledge base for our 100 datasets and used it to

fit the model. When given a new dataset, we calculate its meta-features and use the decision tree model to determine whether meta-learning is likely to improve performance or not. Our model achieves Recall = 0.85 and F1 Score = 0.85. The rules for `Class 1` and `Class 0` can be represented as follows, where $\wedge$ is the logical AND:

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

It is evident from the extracted rules that the number of features p, the percentage of the minority class to the number of instances ($min(\frac{c}{n})$), and noisiness of data (*noise-signal ratio*) are critical features for making predictions. More information about these metafeatures could be found in Table 19.

4.5.4. Impact of Ensembling

Ensembling, a technique well-documented in the work of Dietterich [16], refers to the process of integrating multiple base Machine Learning (ML) models to enhance predictive performance for a specific task. This method integrates various models using several techniques, including straightforward voting (or averaging), weighted voting, bagging, and boosting, as detailed by Dietterich [16]. This study aims to evaluate the impact of ensembling on the effectiveness of AutoML frameworks that permit the toggling of post-processing ensemble features, particularly focusing on `AutoSKlearn` and `SmartML-m`. We also explore the possible correlation between dataset characteristics and the performance enhancements derived from using either the standard (vanilla) version or the ensembled version of these AutoML frameworks.

During the optimization phase, `AutoSKlearn` and `SmartML` retain an array of generated models rather than exclusively preserving the best-performing model. These models are later utilized in a post-processing stage to construct an ensemble model. This method of automatic ensemble creation avoids the reliance on a singular hyperparameter configuration, thereby strengthening the model's resilience against overfitting. `AutoSKlearn` adopts the ensemble selection methodology introduced by Caruana et al.[12], while `SmartML` implements a majority voting system as proposed by Lam[64]. The ensemble selection strategy operates on a greedy algorithm, initiating with an empty ensemble and incrementally incorporating base models to enhance the ensemble's performance in the validation phase. This approach employs uniform weighting, but interestingly allows for model repetitions. Conversely, majority voting is a more straightforward approach, grounded in democratic principles where the most frequently occurring

Time Budget	Framework	Predictive Performance		Performance Gain			#datasets with gain > 1%
		Mean	SD	Min	Mean	Max	
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

Table 8: Performance comparison between vanilla/base version vs ensembling version of `AutoSKlearn` and `SmartML` different time budgets.

class prediction prevails. For our analysis, `AutoSKlearn` and `SmartML` are configured to utilize 50 and 5 base models, respectively, in their ensemble setups.

To evaluate the effectiveness of ensembling, we conducted a comparative analysis of the mean performances of the standard (vanilla) and ensembled versions of `AutoSKlearn` and `SmartML` across various time budgets. This comparison is detailed in Table 8 and further elaborated with comprehensive performance comparisons across all datasets and time budgets in Figures 29 and 30 found in Appendix A.6.

AutoSKlearn: Our analysis indicates that employing ensembling does not uniformly lead to superior performance when compared to the vanilla version of `AutoSKlearn`. Nonetheless, the ensembled version, `AutoSKlearn-e`, exhibits a notable mean performance improvement of 3.9%, 3.2%, 4.7%, and 8.0% across 24, 32, 25, and 24 datasets, respectively, for time budgets of 10, 30, 60, and 240 minutes. These results are summarized in Table 8. The Wilcoxon statistical test was applied to ascertain the statistical significance of the observed performance differences between `AutoSKlearn-e` and `AutoSKlearn-v`. The test results affirm that ensembling contributes to performance enhancement with a statistical confidence level exceeding 95% (p value < 0.05) across all four time budgets, reaching near 99% when combining all time budgets.

SmartML: The ensembling version of `SmartML`, denoted as `SmartML-e`, demonstrates a marginal yet consistent improvement over its meta-learning counterpart, `SmartML-m`. This improvement is quantified as average performance gains of 12.6%, 12.5%, 11.2%, and 10.2% on 30, 33, 28, and 28 datasets for time budgets of 10, 30, 60, and 240 minutes, respectively, as presented in Table 8. To determine the significance of these performance differences, we employed the Wilcoxon statistical test. The outcomes of this test confirm that `SmartML-e` sig-

nificantly enhances performance, with a confidence level exceeding 95% (p value < 0.05) for each of the four time budgets.

We conducted an investigation to discern the correlation between dataset characteristics and the performance enhancement realized through employing the ensembling version of `AutoSKlearn` across varying time budgets. This study mirrors the methodology employed in Section 4.5.3, where we utilized a decision tree with a depth of 3. This tree processes the meta-features of 100 datasets to predict the likelihood of performance improvement (categorized as `Class 1`) or lack thereof (`Class 0`) when using ensembling. Thus, for any new dataset, we first calculate its meta-features and then apply our decision tree model to determine if ensembling is likely to boost its performance. The performance metrics for our `AutoSKlearn` model are as follows: a Recall of 0.70 and an F1 Score of 0.70. The decision rules for categorizing datasets into `Class 1` and `Class 0` in the context of `AutoSKlearn` are detailed as follows:

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

The accuracy of the model's predictions is significantly influenced by specific features, including the mean pairwise correlation between features ($\mu(\rho)$), the standard deviation of this pairwise correlation ($\sigma(\rho)$), the maximum ($\max(\rho)$) and minimum ($\min(\rho)$) values of this correlation, the standard deviation of the ratio between the number of instances and features ($\sigma\left(\frac{n}{p}\right)$), the minimal mutual information between features and the class ($Mutual\ inform.$), and the average number of unique categorical values in features ($\mu(\pi_i)$). More information about these metafeatures could be found in Table 19.

For the `SmartML` models, despite extensive experimentation, it was observed that none were adept at establishing a correlation between meta-features and the performance enhancement attributed to ensembling techniques.

4.6. Discussion and Summary

Our exhaustive evaluation, prioritizing successful run rates, identifies `Auto-SKlearn-e` and `AutoSKlearn` as leading performers, with `Recipe` at the

lower spectrum. Significantly, AutoSklearn registers the most successful executions across diverse time budgets, demonstrating notable enhancements on many datasets with increased time allocation. This study highlights the diminishing influence of meta-learning in extended durations, as opposed to the consistent performance elevation observed with ensembling strategies. Furthermore, AutoSKlearn, along with ATM and TPOT, exhibit exceptional proficiency in multi-class classification tasks, especially in datasets with high instance-to-feature ratios, where ATM particularly shines.

The importance of training efficiency is accentuated in our findings, especially in resource-constrained or rapid model development scenarios. Frameworks like ATM and Recipe emerge as superior choices due to their efficiency and stability, making them ideal for consistent model performance. Robustness to noise is another critical factor, with AutoSKLearn-e and AutoSKLearn-m displaying impressive resilience, rendering them suitable for unpredictable, noisy data environments.

Distinct performance variations among AutoSKlearn versions across different iterations, particularly in datasets with a greater number of features than instances, highlight the significant role of data preprocessing in influencing performance variations. The challenge of over-fitting in comprehensive systematic searches and the necessity for more efficient prevention mechanisms are underscored. Additionally, the analysis propels further research in automated feature engineering, a largely under-explored domain in most AutoML frameworks, despite its potential to profoundly impact overall performance.

5. FEATURE ENGINEERING IN AUTOMATED MACHINE LEARNING

5.1. Introduction

Feature engineering is a critical step in the machine learning pipeline, often determining the upper bounds of a model's performance. Despite its significance, it remains one of the most challenging and time-consuming aspects of machine learning [37]. Automated Machine Learning (AutoML) frameworks have revolutionized many aspects of the ML pipeline, yet the automation of feature engineering still presents unique challenges and opportunities [41, 80]. In this chapter, we delve into the role of feature engineering within AutoML, exploring its impact through specific experiments and discussing the merits and limitations of integrating automated feature engineering into the AutoML pipeline [27, 52]. We begin by presenting an experiment that highlights the influence of different feature engineering preprocessors on the predictive performance of AutoML-generated models. This experiment serves to underscore the significant role that feature engineering plays, even when other components of the ML pipeline are held constant. Subsequently, we experiment integrating automated FE tool (autoFeat) [44] into the AutoML pipeline.

5.2. Experiment on Feature Engineering Preprocessors

The impact of feature engineering preprocessors in automated machine learning (AutoML) is crucial, particularly in datasets with unique characteristics. In our analysis, we paid special attention to datasets showing significant performance variations across different learning settings. These datasets were subjected to repeated runs, each with three configurations. Notably, many of these datasets either had significantly more features than instances or a very small number of instances, which is common in certain domains like medical datasets; see Table 9.

5.2.1. Variance in Pipeline Performance

We observed that the generated pipelines varied greatly in accuracy in each iteration. For example, in the `sonar`, six unique pipelines selected five different classifiers. The importance of the learners' hyper-parameters cannot be understated, as their impact on accuracy varies and is highly dependent on dataset characteristics [24]. Regularization parameters, for instance, are particularly crucial in datasets with fewer instances than features, to prevent overfitting.

5.2.2. Feature Engineering in AutoML

`AutoSklearn`'s pipeline structure includes data preprocessing, feature preprocessing, and classifier components. It evaluates several options for each stage and

Dataset	#Feat.	#Inst.	A	Accuracy		
				1st run	2nd run	3rd run
10 minutes						
sonar	61	208	m	0.885	0.827	0.788
			v	0.846	0.827	0.712
bodies	4703	64	m	0.875	0.875	0.75
			v	0.938	0.938	0.875
tumors_C	7130	60	m	0.666	0.666	0.60
			v	0.666	0.466	0.466
micro-mass	1301	517	m	0.881	0.839	0.811
			v	0.947	0.867	0.832
GCM	160064	190	e	0.604	0.604	0.583
			v	0.792	0.708	0.646
30 minutes						
stemmed	3722	64	m	0.812	0.812	0.75
			v	0.875	0.875	0.812
lymphoma	4027	45	m	0.812	0.812	0.75
			v	0.875	0.875	0.812
rsctc2010_3	22278	95	m	0.958	0.875	0.875
			v	1.0	0.833	0.75
240 minutes						
CovPokElec	65	1.4 M	m	0.954	0.888	0.62
			v	0.80	0.572	0.504

Table 9: A sample of the datasets' characteristics and results from the repeated (special) runs. 'm', 'e', 'v' stands for the version of AutoSklearn (A).

chooses the one that maximizes validation accuracy. This highlights the significant role of feature engineering. Even when the same classifier is used, pipelines with different feature preprocessors exhibit varying performance. For instance, in the dbworld-bodies (bodies) dataset, the lda classifier paired with different preprocessors resulted in accuracies of 93.8% and 75% for different versions of AutoSklearn. The two ML pipelines are identical, except for the feature preprocessor where nystroem_sampler for the first and None for the second. Similarly, the Gaussian naive Bayes classifier showed a performance variation between 100% and 83.3% accuracy depending on the feature preprocessor used. Despite the importance of feature engineering, it is often the most time-consuming phase and is not adequately automated by current AutoML tools, including AutoSklearn [99].

The proper feature engineering phase can transform the feature space into a linearly separable space, allowing even simple classifiers to achieve high accuracy. In contrast, inadequate or incorrect feature engineering can hinder the performance of even the most efficient classifiers. This is particularly evident in datasets with raw pixel features, where performance oscillates based on the selected preprocessors.

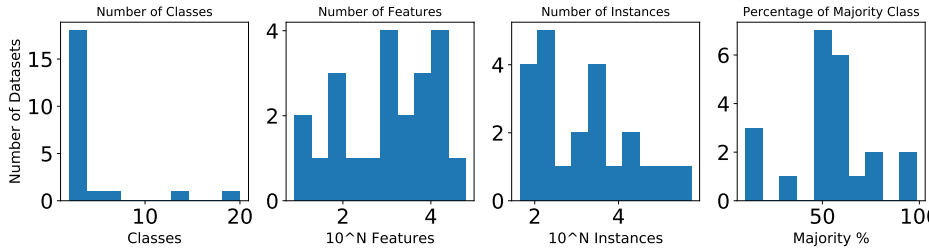


Figure 10: Histogram of the main characteristics of the 22 datasets.

5.3. Integrating Automated Feature Engineering into AutoML

5.3.1. Experiment Design

The design of the experiments aims to evaluate the efficacy of dividing the pipeline generation process into two distinct phases: feature representation enhancement and model selection with hyperparameter tuning. This is executed through three different experimental setups:

1. **Combined Setup**: In this configuration, `AutoFeat` is first applied for automated feature engineering. Following this, an `AutoML` framework is utilized for model selection and hyperparameter tuning. This setup allows us to assess the impact of `AutoFeat` on the overall performance of the `AutoML` framework.
2. **AutoML Setup**: This is the standard approach where the `AutoML` framework is responsible for generating the entire pipeline, including feature engineering. This setup serves as a control to determine how well `AutoML` frameworks handle feature engineering on their own.
3. **Baseline**: In this setup, the `AutoML` framework is used exclusively for model selection and hyperparameter tuning, without any feature engineering preprocessors. This baseline setup helps to highlight the importance of feature engineering by comparing it with the other two setups.

These setups were implemented on a high-performance Linux-based machine with 256 GB of RAM and 64 vCPUs to ensure sufficient computational resources for extensive evaluations. Notably, the time taken by `AutoFeat` in the Combined Setup is accounted for by adding it to the `AutoML` time budget in the other configurations. This ensures a fair comparison of the time efficiency across different setups.

5.3.2. Datasets

The experiments utilized twenty-two datasets, carefully selected to represent a broad range of characteristics, such as the number of instances, the number of features, and the class imbalance. This diverse selection aims to uncover any performance variations attributable to these characteristics.

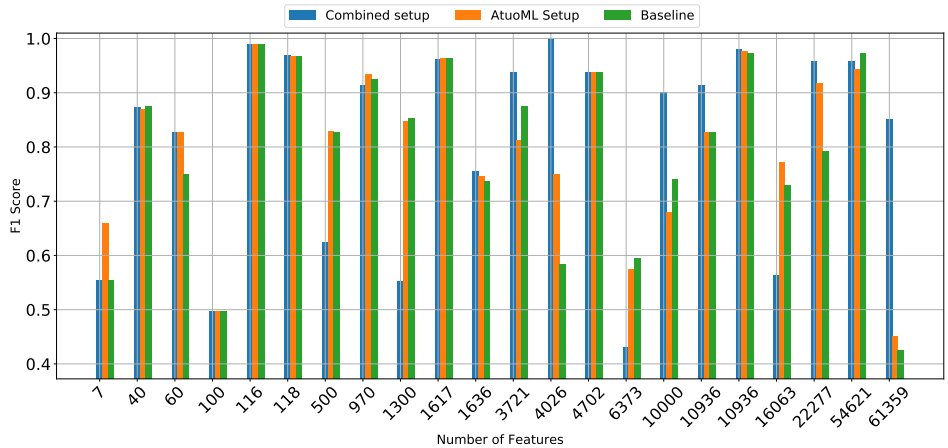


Figure 11: Setups of combing `AutoFeat` with `Auto-Sklearn`. Each dataset denoted by number of features of raw data.

Instance Count: Datasets were chosen to include a wide spectrum of instance counts, from a few hundred to tens of thousands. This variety helps in understanding how the size of the dataset affects the performance of different setups.

Feature Quantity: The datasets also varied significantly in the number of features, ranging from datasets with fewer than 100 features to those with over 10,000 features. This allows for an assessment of how feature engineering and model selection processes handle high-dimensional data versus lower-dimensional data.

Class Imbalance: The prevalence of the majority class in each dataset was another key characteristic. Datasets with significant class imbalance were included to evaluate how each setup manages this common issue in machine learning tasks.

A summary of the primary characteristics of the datasets is illustrated in Figure 10. This histogram provides a visual overview of the distribution of these characteristics across the selected datasets. Detailed information about each dataset, including their specific attributes and the rationale for their inclusion, is provided in the online repository associated with this study. This repository also includes the source of each dataset, preprocessing steps taken, and the exact configuration files used in the experiments to ensure reproducibility and transparency.

5.4. Impact Assessment and Discussion

We executed `AutoFeat` with varying feature generation (`FG`) steps (0, 1, and 2), each followed by a feature selection (`FS`) step. The performance of `Auto-Sklearn`, represented by `F1 Score` across diverse datasets, is depicted in Figure 11. Here, `AutoFeat` is deployed solely for `FS`, with no `FG` steps. This setup showcased mixed results, enhancing the `F1 Score` for six datasets, while diminishing it for four others. `TPOT` presented comparable outcomes, as indicated in Figure 12.

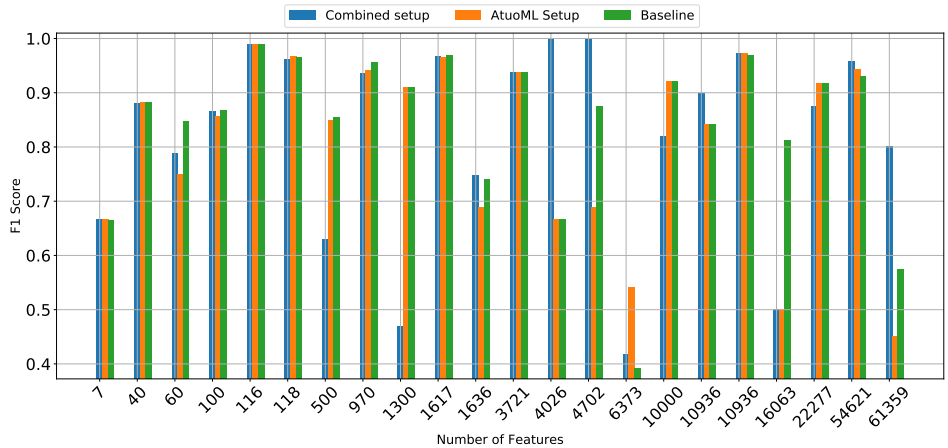


Figure 12: Setups of combing `AutoFeat` with `TPOT`.

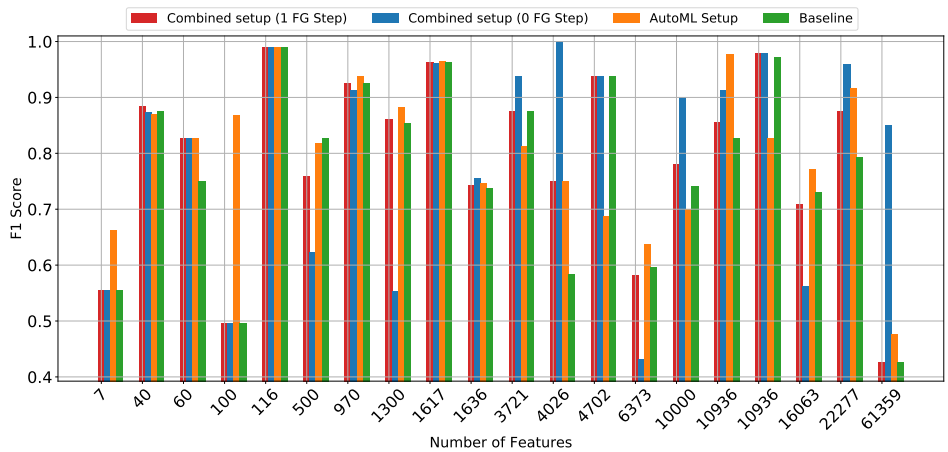


Figure 13: Impact of adding a `FG` step `AutoFeat` compared to adding more time to `Auto-Sklearn`.

The influence of an additional time allocation, equating to six times the standard budget, is illustrated in Figure 13. This compares the performance of `Auto-Sklearn` when searching for an improved pipeline against incorporating a single `FG` step with `AutoFeat`. This approach reduced significant performance drops observed in Figure 11, while marginally enhancing the sole `FS` step.

`TPOT` consistently exhibited a slight advantage over `Auto-Sklearn`, attributable to its ability to construct dynamic pipeline structures with multiple components [82]. For both frameworks, extending the time budget correlated with performance improvements. Although the combined setup generally surpassed the `AutoML`-only approach within identical time constraints, it fell short when competing against `AutoML` with extended budgets. This indicates that current `AutoML` methods scale more effectively with larger time allocations compared to

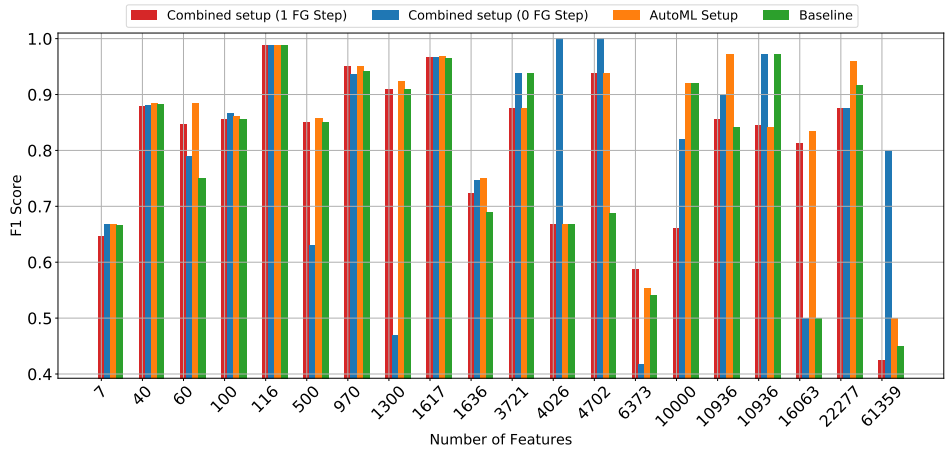


Figure 14: Impact of adding a FG step AutoFeat compared to adding more time to TPOT.

AutoFeat employing additional iterations. Feature generation also led to superior performance compared to solely conducting FS. Thus, deeper FE steps are posited to further enhance results.

Notably, the FE phase's technique had a more substantial impact on performance than the model selection and hyperparameter tuning methods. This is evident from the similarity in performance trends between Auto-Sklearn and TPOT.

In attempting to utilize AutoFeat with two feature generation (FG) steps, we encountered significant computational challenges, leading to unsuccessful runs within a 24-hour period for all datasets. The predominant issue was memory overflow before reaching the time limit.

Theoretically, each FG step in AutoFeat increases the number of generated features exponentially. Assuming only binary operators (such as $+, -, *, /$) are used, the number of features generated after each step can be represented as T_3N^2, where T_3 is the number of operators and N is the initial number of input features. This calculation considers every possible combination of two features per operator, leading to a memory complexity that grows double exponentially with each FG step as $O(T_3N^{2^k})$, where k denotes the number of FG steps taken.

For instance, using the tumor_C dataset, which initially occupies approximately 1.8 MB with 7130 features, the memory usage escalates dramatically with each FG step. After the first FG step, memory usage would theoretically increase to about 12.8 GB, as the number of features would expand to 7130^2. Continuing to the second FG step, memory requirements would surge to an impractical 652 PB due to the increase to 7130^3 features, not accounting for the multiplier k.

The time complexity, similarly, is $O(T_3N^{2^k})$. This means that even with a moderate increase in the number of steps to $k = 2$, as suggested by the developers of AutoFeat, the processing time becomes excessively long. This scalability

issue underscores a critical challenge and highlights a significant opportunity for enhancing AutoFeat's performance, particularly in managing its exponential growth in resource demands with additional FG steps.

To mitigate these challenges, it is essential to explore optimization techniques for the AutoFeat framework. This could involve reducing the number of feature combinations generated per FG step, implementing more efficient data structures to manage memory usage, or leveraging distributed computing environments to handle large-scale computations. By addressing these issues, we can significantly improve the scalability and efficiency of automated feature engineering processes, making them more applicable to a broader range of real-world datasets and scenarios.

5.5. Conclusion

In our exploration, we assessed the effectiveness of integrating automated feature engineering (FE) into AutoML frameworks, as an alternative to the traditional trial-and-error method commonly employed in their FE phase. Our findings suggest that solely relying on feature selection does not markedly improve pipeline performance. However, the introduction of a single feature generation step offers slight enhancement. Despite this, the practical application of current automated FE tools like AutoFeat struggles to create complex features for real-world datasets beyond one generation step. As we continue to address the challenges of big data, there is an undeniable need for distributed automated FE frameworks that can address the shortcomings of existing methods.

6. BIGFEAT: SCALABLE AND INTERPRETABLE AUTOMATED FEATURE ENGINEERING

6.1. Introduction

In this chapter, we introduce BigFeat, a novel framework aimed at enhancing the efficiency of machine learning pipelines. BigFeat comprises two primary modules: BigFeat-FE for advanced feature engineering, and BigFeat-AutoML for effective algorithm selection and hyperparameter optimization. Key contributions of this chapter are:

- BigFeat-FE adopts a dynamic feature generation approach, utilizing computation tree logic for scalable and efficient iteration. It also incorporates a sophisticated feature selection process for identifying robust and informative features.
- Employing a meta-learning strategy, BigFeat-FE significantly boosts the framework's performance, focusing on optimization efficiency.
- BigFeat-AutoML presents a comprehensive and interpretable machine learning pipeline, utilizing a random search methodology for navigating the hyperparameter landscape of interpretable models.
- Comparative experiments with BigFeat-FE against leading frameworks like AutoFeat and SAFE, using benchmark datasets, demonstrate BigFeat's superior performance, leading in most datasets.
- BigFeat-AutoML shows a notable performance edge over TPOT and AutoSklearn, marking a significant advancement in the field.
- For transparency and repeatability, we provide open access to BigFeat's source code and detailed study results[1].

6.2. Design and Implementation of BigFeat

6.2.1. Overview

An overview of BigFeat is provided in Algorithm 1 and Figure 15. BigFeat introduces two modes of feature engineering (FE) operation: BigFeat-FE, which leverages meta-learning for enhanced performance, and BigFeat-vanilla, a simpler version without meta-learning. Additionally, BigFeat-AutoML is designed for the automated selection of algorithms and hyperparameter optimization, employing a random search approach within a space of interpretable machine learning models.

In its structure, as outlined in Algorithm 1 and illustrated in Figure 15, BigFeat-vanilla operates iteratively. Each iteration consists of two main stages: the generation of features, followed by their selection. The total number of iterations is

[1] `https://github.com/DataSystemsGroupUT/BigFeat`

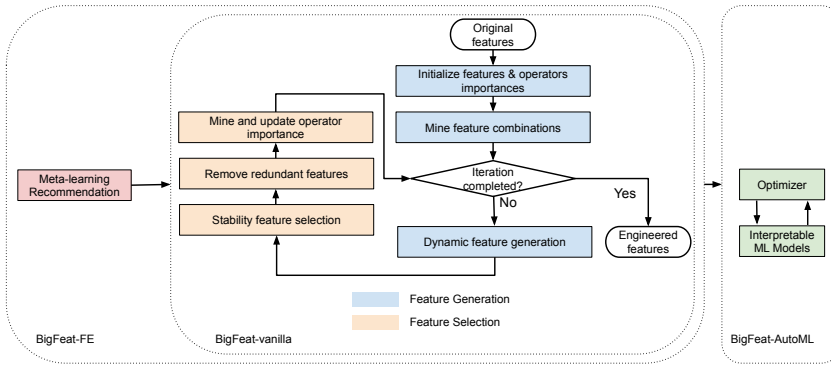


Figure 15: Flowchart of BigFeat.

dependent on the available resources, such as the time budget and computational power at hand.

The process of generating all potential features from the base feature set using various operators leads to a significant expansion of the feature space. This makes the exhaustive search method impractical due to scalability issues. To address this, BigFeat employs tree-based models to analyze the relationships among the base features, represented as X_{train}. These models calculate an importance score for each feature based on information gain, prioritizing features with higher scores in the generation process. A set of predefined operators is then applied to the original features, creating an extended feature set $\mathbb{X}_{train}$. The importance of each operator, reflecting its historical effectiveness in generating meaningful features, influences their application. The final comprehensive feature set is formed by merging $\mathbb{X}_{train}$ with the original set X_{train}.

Given the substantial size of the combined feature set $\mathbb{X}_{train} \cup X_{train}$, a proficient selection mechanism is implemented to maintain a limited subset of N features in each iteration. This approach focuses on identifying the most informative, stable, and non-redundant N features. Initially, the importance score is used to select the most impactful features on the target label. Then, to remove redundant features providing similar information, the Pearson correlation coefficient is applied. Lastly, the importance scores for operators are revised according to their representation in the chosen features.

The components of BigFeat-vanilla, including feature generation and selection, are elaborated in Sections 6.2.2 and 6.2.3. In contrast, the specifics of the meta-learning aspect within BigFeat-FE are detailed in Section 6.2.4. Furthermore, Section 6.2.5 is dedicated to presenting the elements and functioning of BigFeat-AutoML.

6.2.2. Feature Generation

During the feature generation stage, the objective is to create a set of informative features using the existing base feature set (X_{train}). The goal is to select the most promising N features in each iteration, while also ensuring that none of them are

Algorithm 1: Pseudo-code of BigFeat

Inputs : $\mathscr{D}_{train}$; Operators $\mathscr{O}$; Batch size of generated features K; Number of original features N; Number of iterations $Iterations$.

Output: feature generation function (Ψ)

$I^o \leftarrow$ Initialize the operator importance score;
$I^f \leftarrow$ Initialize the feature importance score;
Mine feature combinations **in parallel** and compute matrix C (Equation 6.4);
$iterator \leftarrow 0$;
while $iterator < Iterations$ **do**
    $k \leftarrow 0$; $\mathbb{X}_{train} \leftarrow \phi$;
    $height \leftarrow samplefrom[1:4]$;
    while $k < (K \times N)$ ***in parallel*** **do**
        $\mathbb{X}_{train} \leftarrow \mathbb{X}_{train} \cup$ Generate a candidate feature $\mathbb{X}^k_{train}$ with height = $height$ (See Algorithm 2);
        $k \leftarrow k+1$;
    end
    $\mathbb{X}^*_{train} \leftarrow$ Select the most important features from $\mathbb{X}_{train} \cup X_{train}$ and remove redundant features (See section 6.2.3);
    Update the operators importance scores I^o;
    $iterator \leftarrow iterator + 1$;
end
The feature generation function (Ψ) is obtained from the selected features ($\mathbb{X}^*_{train}$);
$A^*, \lambda_* \leftarrow$ find_best_model$(A, \Psi(\mathscr{D}_{train}))$
return Ψ, A^*, λ^*;

overlooked.

The search space (S) consists of the possible combinations of original features X_{train} and operators $\mathscr{O}$. The size of this search space can be determined as follows:

$$\mid S \mid = \sum_{i=1}^{N} \binom{N}{i} \times \mid \mathscr{O}_i \mid \tag{6.1}$$

where $\binom{N}{i}$ is the number of ways to select i features from N features, and $\mid \mathscr{O}_i \mid$ is the number of i-ary operators. For example, for i equals 2, the size of this search space portion equals to the number of ways to select two features out of N features $\binom{N}{2}$ multiplied by the number of binary operators we have in our portfolio ($\mid \mathscr{O}_2 \mid$). $\mid S \mid$ is the summation overall $i < N$. It is worth mentioning that, for simplicity, we counted $\mathscr{O}_i(X^i, X^j)$ and $\mathscr{O}_i(X^j, X^i)$ as one while calculating $\mid S \mid$.

In general, the feature generation stage consists of the following steps:

Initializing the importance score of features and operators. In order to evaluate the significance of each base feature and operator in our portfolio, we compute

an importance score. Firstly, we train a tree-based model on the training dataset D_{train}, which then scores the importance of each base feature X_{train} based on the information gain (IG) value. Next, we normalize these scores to construct the feature importance vector I^f.

$$I^f = \begin{pmatrix} I_1^f & I_2^f & \cdots & I_N^f \end{pmatrix} \tag{6.2}$$

where I_i^f is the importance of feature X^i.

To initialize operator importance scores, a uniform distribution is used, as no initial preference exists for any operator. The operator importance vector, denoted as I^o, is initialized as follows:

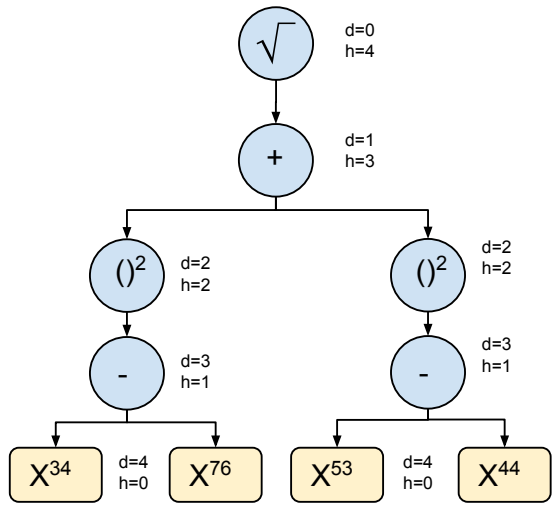
$$I^o = \begin{pmatrix} I_1^o & I_2^o & \cdots & I_{|\mathscr{O}|}^o \end{pmatrix} = \begin{pmatrix} 1 & 1 & \cdots & 1 \end{pmatrix} \tag{6.3}$$

Mine feature combinations. As we've previously mentioned, due to the vastness of the search space, our focus is on generating expressive features. This involves giving more opportunities to features that are hypothetically expressive, while also making sure that potential features with the capacity to become expressive in future iterations are not overlooked. Therefore, our automated feature engineering framework is based on two fundamental assumptions: 1) new features generated from split features of a tree-based model have the potential to be more efficient than features derived from non-split features, and 2) new features generated from split features along the same path within a tree-based model are likely to be more expressive than those originating from split features along different paths. To follow these assumptions, we assign higher probabilities for selecting split features from the same paths together. We keep a matrix, denoted as C, which keeps track of the frequency of occurrences for each pair of split features sharing the same path (as outlined in Algorithm 1, line 3):

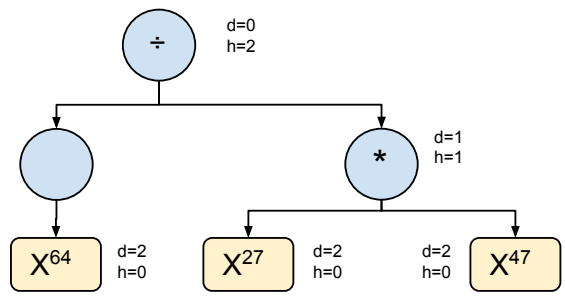
$$C = \begin{pmatrix} C_{1,1} & C_{1,2} & \cdots & C_{1,N} \\ C_{2,1} & C_{2,2} & \cdots & C_{2,N} \\ \vdots & \vdots & \ddots & \vdots \\ C_{N,1} & C_{N,2} & \cdots & C_{N,N} \end{pmatrix} \tag{6.4}$$

In a tree-based trained model, $C_{i,j}$ represents the number of times X^i and X^j occur together in the same path. Initially, $C_{i,j}$ is set to 1 and then incremented by 1 for each path in which both X^i and X^j appear together.

Dynamic Feature Generation. BigFeat-FE uses computation trees to address the scalability challenge. These trees generate KN features per iteration, where K is much smaller than N. Each internal node in the computation tree represents a unique computational state, while leaf nodes represent base features. Edges indicate transitions to potential computations. Figure 16 shows some examples of computation trees. The depth of a node in the computation tree is the length of



(a) Euclidean Distance



(b) Price per square meter

Figure 16: Examples of computation trees to generate two features; (a) Euclidean distance between two points (X^{34}, X^{53}) and (X^{76}, X^{44}), and (b) price per meter square given the price X^{64}, length X^{27} and width X^{47}. Each internal node in each tree corresponds to an operator, and each leaf node corresponds to a base feature.

Algorithm 2: Dynamic Feature Generation

Inputs : $\mathscr{D}_{train}$; Operators $\mathscr{O}$; Operators importance vector I^o; Feature Importance vector I^f; computational tree height *height*; Selected Features = ϕ.

Output: Generated Feature $\mathbb{X}^i$.

```
if height = 0 then
    return a sampled feature from X_train with a probability of selection
     proportional to its importance score in I^f;
else
    o ← sample a new (n)-ary operator from O with probability of
     selection proportional to its importance score in I^o;
    for each operand of (n)-ary operator do
        New Feature ← Dynamic Feature Generation(..., height − 1,
         Selected Features, ...);
        Selected Features ← Selected Features ∪ New Feature;
    end
    X^i = the result of applying o on Selected Features;
    return X^i;
end
```

the path from the root to that node. The tree's height, denoted as *height*, is the maximum distance between the root and any leaf node. The *height* is determined through random sampling, with a preference for simpler and shallower trees.

The method initiates the computation tree using a depth-first strategy, commencing with a singular root node. This node is indicative of an operator chosen from the portfolio, selected based on the corresponding importance score from I^o. The process advances by extending a branch to its maximum depth. When constructing nodes at a *height* greater than zero, operators are selected, whereas at *height* equal to zero, features are chosen. These selections are informed by their respective importance scores from I^o and I^f. The tree's development proceeds with similar expansions across additional branches, followed by a sequential return. The tree's structure is such that the number of children nodes at a given level directly relates to the nature of their parent node at the preceding level. For instance, a node connected to an n-ary operator will give rise to n offspring in the next level. As the process unfolds, we refine the feature importance vector I^f, enhancing the likelihood of features that frequently coexist in the same path within the initial tree-based model. This refinement involves multiplying the I^f vector by $C_{i,:}$, with i representing the index of the currently selected feature. The dynamic feature generation phase's algorithmic flow is meticulously described in Algorithm 2.

6.2.3. Feature Selection

After each iteration, our process generates a new set of KN features in addition to the existing base features. However, not all of these features may be equally significant. To identify the most representative N features efficiently, we follow a three-step feature selection technique. Firstly, we sort the features based on their importance, which is determined by their stability and predictive performance. Secondly, we focus on selecting the features that are not only important but also non-redundant. Finally, our approach involves updating the operator importance vector to reflect the contribution of operators to the generation of important features in previous iterations.

Stability feature selection. To select the optimum N features out of a group of $(K+1)N$ options, trying all possible combinations is the most effective method. However, this approach is computationally expensive. On the other hand, selecting features individually without considering their interdependencies can lead to suboptimal choices. To achieve a balance between performance and efficiency, we use a stability selection mechanism as an approximation to the optimal technique [75, 66]. We train α tree-based models, with each model being trained on a subset of the total number of features and instances. These subsets are sampled with replacement, including both features and instances. We then determine the importance of each feature by using the information gain from the α tree-based models, which is then normalized, aggregated, and sorted in descending order. This sorting process allows us to select the most important and non-redundant N features.

Remove redundant features. We use the Pearson correlation method [69] to measure the correlation between every pair of candidate features within $\mathbb{X}_{train} \cup X_{train}$. The correlation coefficient's magnitude indicates the strength of the correlation between these features. We set a threshold for the Pearson correlation, which we call η. If the correlation coefficient between any pair of features within $\mathbb{X}_{train} \cup X_{train}$ exceeds the η threshold, we keep only the features with higher importance scores and discard the others. Finally, we select the most important and uncorrelated set of N features, which we store within $\mathbb{X}^*_{train}$.

Mine and update operator importance. After each iteration, we calculate the importance score of each operator based on their contribution to the generation of features in $\mathbb{X}train^*$. The score of each operator is incremented based on the number of times it contributes to the creation of generated features in $\mathbb{X}train^*$. The operator importance vector I^o at iteration c is updated by this process.

$$I^o \leftarrow ((c-1)I^o + \overline{I^o})/c \tag{6.5}$$

Where $\overline{I^o}$ is the newly calculated importance vector I^o. All operators that appear in forming generated features in $\mathbb{X}^*_{train}$ are incremented by their count of occurrences and then normalized.

Name	Formula	Rationale	Additional Variants
Nr instances	n	Speed, Scalability	n, $log(n)$, $\frac{p}{n}$,X,$(n-X)$
Nr features	p	Curse of dimensionality	p,$log(p)$, Nr|min|μ|$\sigma(\pi_i)$
Nr classes	c	Complexity, imbalance	c,min|max|$\sigma(\frac{c}{n})$
Nr missing values	m	Imputation effects	m, $\frac{m}{n}$
Skewness	$\frac{E(X-\mu_X)^3}{\sigma_X^3}$	Feature normality	min,max,μ,σ,q_1,q_3
Kurtosis	$\frac{E(X-\mu_X)^4}{\sigma_X^4}$	Feature normality	min,max,μ,σ,q_1,q_3
Correlation	$\rho_{X_1X_2}$	Feature interdependence	min,max,μ,σ
Class probability	$P(\mathtt{C})$	Class distribution	min,max,μ,σ
Class entropy	$H(\mathtt{C})$	Class imbalance	$H(\mathtt{C})/\mu(MI(\mathtt{C},\mathtt{X}))$
Norm. entropy	$\frac{H(\mathtt{X})}{log_2 n}$	Feature informativeness	min,max,μ,σ
Mutual inform.	$MI(\mathtt{C},\mathtt{X})$	Feature importance	min,max,μ,σ
Equiv. nr. feats	$\frac{H(C)}{MI(C,X)}$	Intrinsic dimensionality	
Noise-signal ratio	$\frac{H(X)-MI(C,X)}{MI(C,X)}$	Noisiness of data	
Nr nodes, leaves	$\|\eta\|$,$\|\psi\|$	Concept complexity	$\mu\|\sigma$(Tree depth), $\mu\|\sigma$(Nodes per(attr, inst, level))
Branch length		Concept complexity	μ,σ
Leaves corroboration		Concept complexity	μ,σ
Leaves homogeneity		Class complexity	μ,σ
Nodes per feature	$\|\eta_X\|$	Feature importance	min,max,μ,σ
Leaves per class	$\frac{\|\psi_c\|}{\|\psi\|}$	Class complexity	μ,σ
Nodes per level		Concept complexity	μ,σ
Leaves agreement	$\frac{n_{\psi_i}}{n}$	Class separability	min,max,μ,σ
Landmarker(1NN)	$P(\theta_{1NN},t_j)$	Data sparsity	$\mu\|\sigma$, $\mu\|\sigma$(eliteNN)
Landmarker(Tree)	$P(\theta_{Tree},t_j)$	Data separability	$\mu\|\sigma$(bestNode, randomNode,worestNode)
Landmarker(Lin)	$P(\theta_{Lin},t_j)$	Linear separability	$\mu\|\sigma$(Lin.Disciminant)
Landmarker(NB)	$P(\theta_{NB},t_j)$	Feature independence	μ,σ

Table 10: Overview of the used meta-features. Groups from top to bottom: statistical and information-theoretic, model-based, and landmarkers. Continuous features X and target Y have mean μ_X, stdev σ_X, variance σ_X^2. Categorical features $\mathtt{X}$ and class $\mathtt{C}$ have categorical values π_i, conditional probabilities $\pi_{i|j}$, joint probabilities $\pi_{i,j}$, marginal probabilities $\pi_{i+} = \sum_j \pi_{ij}$, entropy $H(\mathtt{X}) = -\sum_i \pi_{i+} log_2(\pi_{i+})$ [101].

6.2.4. Meta-Learning for Optimizing Operator Importance Vector in BigFeat-FE

We propose a method called BigFeat-FE that utilizes meta-learning to warmstart the operator importance vector. Our approach involves identifying instantiations of operator importance vectors that are more likely to play a significant role in generating important features for new datasets. To accomplish this, we employ meta-learning by gathering performance data and extracting a set of efficiently computable meta-features from a pool of diverse datasets. These meta-features encapsulate key dataset characteristics. The concept of warmstarting optimization through meta-learning has been demonstrated in previous works [25, 27, 20].

Algorithm 3: Offline phase of BigFeat-FE

Inputs : $\mathscr{D}_{train}$; Operators $\mathscr{O}$.
Output: Operators models $Models$.

$Catalog \leftarrow \{\}$;
for *each dataset in Datasets* **do**
  BigFeat-v($dataset$, ...);
  best $I^o \leftarrow$ final I^o;
  $I^m \leftarrow$ extract_metafeatures($dataset$);
  $Catalog \leftarrow Catalog \cup (I^m$, best $I^o)$;
end
$Models \leftarrow \{\}$;
for *each o in* $\mathscr{O}$ **do**
  $\widehat{Catalog} \leftarrow$ drop_other_operators($Catalog$, o);
  $model \leftarrow$ TPOT($\widehat{Catalog}, I^o$);
  $Models \leftarrow Models \cup (o, model)$;
end
return $Models$;

Meta-features Extraction. Certainly, here is the modified text with the scientific tone and the latex format:

As per Brazdil et al. [11], there exist three primary categories of metafeatures that are used to characterize datasets. These categories are:

1. **Statistical and Information-Theoretic Metafeatures:** These metafeatures are derived from statistical and information-theoretic properties of the dataset. They include features such as the number of classes (C), the number of features (F), the ratio of examples to features ($\frac{N}{F}$), the degree of correlation between features and the target concept, and the average class entropy (H). This approach is widely used in literature and has consistently yielded positive results [1, 20, 29].
2. **Model-based metafeatures:** This approach involves creating a decision tree from the dataset and capturing properties of the tree, such as the number of nodes per feature, maximum tree depth, tree shape, and tree imbalance. These properties collectively form a set of metafeatures [6, 83].
3. **Landmarkers:** Another approach involves using insights gained from the performance of a selection of simple and fast learners, each with distinct learning mechanisms. The accuracy of these reference models, often referred to as 'landmarkers,' serves to characterize a dataset and pinpoint specific areas where each type of learner excels as an expert [86, 5].

To characterize datasets, we implemented a total of 104 meta-features shown in Table10.

Selecting good instantiations. Our meta-learning approach involves two phases: Offline and Online. During the Offline phase, we evaluate a set of meta-features

Classifier	Hyper-parameter	Range
rf	n_estimator	[10, 50, 100, 250]
	max_depth	[None, 5, 10, 20]
	class_weight	[None, {0:1,1:5}, {0:1,1:10}, {0:1,1:25}]
lr	C	[10^-2, 10^-1, 10^0, 10^1, 10^2]
	penalty	['l1', 'l2']
	class_weight	[None, {0:1,1:5}, {0:1,1:10}, {0:1,1:25}]
dt	max_depth	[5,10,25,None]
	min_samples_split	[2,5,10]
	class_weight	None, {0:1,1:5}, {0:1,1:10}, {0:1,1:25}]
gb	n_estimators	[10, 50, 100, 250]
	max_depth	[5, 10, 20]

Table 11: Search space of BigFeat-AutoML

for each dataset in our knowledge-base, which consists of 176 real and synthetic datasets. We then use BigFeat-Vanilla to report and store the learned operator importance vector after running it for 5 iterations. For each operator, we train a meta-model on the meta-features and the importance score associated with that operator. This meta-model predicts the importance score of that specific operator when applied to a given dataset. To train the meta-models, we use TPOT, a powerful automated machine learning framework, which identifies the best-performing machine learning model for the given operator.

When we are given a new dataset D, during the online phase, we compute its meta-features and obtain the importance scores for all operators from all meta-models. These importance scores are used as an instantiation for the operator importance vector for the dataset D.

6.2.5. BigFeat-AutoML

BigFeat-AutoML is a framework that helps with the selection of models and tuning of hyperparameters for classification tasks. It is built on top of the popular Python ML package, Scikit-Learn, and utilizes random search for algorithm selection and hyperparameter tuning. BigFeat-AutoML provides a variety of interpretable models such as Random Forest, Logistic Regression, Decision Tree, and Gradient Boosting to create a comprehensive machine learning pipeline. To explore the space of possible classification pipelines efficiently, we need to be able to identify the hyperparameters that describe the recommended classification algorithm. Table 11 shows the supported classification algorithms along with their main hyperparameters.

6.2.6. Time Complexity Analysis

In this section, we delve into the time complexity of the BigFeat-FE framework and compare it with the complexities of AutoFeat and SAFE. Our analysis is

predicated on the worst-case scenario using the default parameters set for each framework.

BigFeat-FE. For BigFeat-FE, we simplify the analysis by assuming the exclusive use of binary operators and the employment of Random Forest for determining feature importance and combinations. The complexity for training a Random Forest classifier in the feature initialization phase is $O(N_1N_2D_1M)$, where N_1 is the number of trees, N_2 is the number of features per tree, D_1 is the depth of each tree, and M is the number of instances. The initialization of operator importance via meta-learning, which is computationally intensive primarily due to the extraction of landmarking features using Decision Tree models, has a complexity of $O(F_1N_2D_1M)$ with F_1 representing the total number of meta-features extracted.

The dynamic feature generation phase is characterized by a complexity of $O(I_1KN2^{D_2}C)$, where I_1 denotes the iteration count, K is a scaling factor, D_2 is the maximal feature generation depth, and C represents the cost of applying an operator to M instances for a feature. Stability-based feature selection adds further complexity of $O(\alpha N_1N_2D_1M)$, where α is the number of models, while the evaluation of each operator's contribution to generating significant features and updating operator importance scores (I^o) requires a complexity of $O(T_2N)$, with T_2 being the total number of operators.

Additional computational overhead is incurred when removing redundant features through high-correlation checks within the computational tree, calculated at $O(T_1^{D_2}N)$, where T_1 is the operator category count. Constants in this analysis such as $\alpha, K, D_1, D_2, F_1, T_1$, and T_2 establish the baseline overhead, witAutoFeath α, I_1 and N_1 being user-defined, typically set to 5, 7, and 100 respectively, treating I_1 and 2^{D_2} effectively as constants.

$$\begin{aligned} O(BigFeat\text{-}vanilla) &= (\alpha+2)O(N_1N_2D_1M) + O(I_1KN2^{D_2}C) \\ &\quad + O(T_2N) + O(T_1^{D_2}N) \\ &= O(N_2M) + O(NM) + (2)O(N) \\ &= O(NM) \end{aligned} \tag{6.6}$$

$$\begin{aligned} O(BigFeat\text{-}meta) &= (\alpha+2)O(N_1N_2D_1M) + O(F_1N_2D_1M) \\ &\quad + O(I_1KN2^{D_2}C) + O(T_2N) + O(T_1^{D_2}N) \\ &= (2)O(N_2M) + O(NM) + (2)O(N) \\ &= O(NM) \end{aligned} \tag{6.7}$$

It is crucial to acknowledge that the prior complexity analysis does not include the advantages offered by BigFeat-FE's inherent parallelization capabilities. These parallel processing techniques substantially enhance the framework's efficiency beyond what a purely sequential approach would achieve. Despite the linear complexity outlined, the parallel execution within BigFeat-FE enables optimal use of

computational resources, thereby accelerating processing speed and amplifying scalability.

AutoFeat [44]. AutoFeat is designed to iteratively enhance the feature set of a dataset through its unique feature generation and selection mechanisms, which have considerable implications on its computational complexity. In practical terms, AutoFeat performs several iterations where each iteration consists of a feature generation (`FG`) step before it applies a single final feature selection stage.

During the `FG` step, AutoFeat employs various operators (transformations) to create new features. Specifically, if we assume the usage of only binary operators (such as $+, -, *, /$), the number of feature combinations generated from an initial set of N input features is N^2, considering that each new feature is derived by applying an operator to a pair of existing features. Therefore, after each `FG` step, the total number of new features produced can be approximated as T_3N^2, where T_3 is the count of different operators used. Consequently, if this process is iterated over k iterations, the growth of features follows a double exponential trend with respect to k, which is each iteration exponentiates the number of features produced in the previous iteration.

In terms of computational complexity, each feature generation phase exhibits a time/space complexity of $O(T_3MN^{2^k})$, reflecting both the number of operations performed and the increased space required to store these features. Here, M denotes the number of data instances. This growth in complexity is significant because it not only involves the operations for generating features but also the storage needed to hold these expanded feature sets throughout the process.

The subsequent feature selection phase in AutoFeat involves employing logistic regression to evaluate and retain the most informative features. This phase also carries a complexity of $O(T_3MN^{2^k})$ due to the need to process an exponentially growing number of features against the dataset's instances. The logistic regression model must assess each new feature generated in the previous `FG` steps, further compounding the complexity with each iteration.

Therefore, the total complexity for running AutoFeat for k iterations combines these factors, leading to a time and space complexity that is double exponential in nature, fundamentally $O(T_3MN^{2^k})$. This complexity underscores the computational and memory-intensive nature of AutoFeat, particularly as the number of iterations increases. Such complexity is a critical consideration when deploying AutoFeat in environments where computational resources are constrained or where rapid feature engineering is required.

$$\begin{aligned} O(AutoFeat) &= O(T_3MN^{2^k}) + O(T_3MN^{2^k}) \\ &= O(MN^{2^k}) + O(MN^{2^k}) \\ &= O(MN^{2^k}) \end{aligned} \tag{6.8}$$

SAFE [93]. In its feature generation phase, SAFE leverages the capabilities of an XGBoost classifier to pinpoint and prioritize the most informative feature combinations. These top-rated combinations are then used to create new features. The subsequent selection phase in SAFE involves a comprehensive process to detect and eliminate redundant features, assessing correlations between each feature pair generated earlier. As a result, the time and space complexity of SAFE is presented as:

$$O(SAFE) = O(NM^2) \tag{6.9}$$

This comparison indicates that BigFeat-FE achieves linear complexity, contrasting with AutoFeat's double exponential complexity and SAFE's quadratic complexity.

6.3. Experimental Setup

We utilized two external benchmarking frameworks, [8] and [9], for our evaluations to avoid bias towards specific dataset characteristics. The datasets used from these benchmarks contain a wide array of datasets, with instances ranging from 200 to 96,320 and features from 4 to 16,063, detailed in Table 12. We made sure that the test datasets do not intersect with the training datasets to prevent any data leakage, and we carefully selected datasets that have minimal or zero data cleaning to avoid any potential bias from the data cleaning stage as it is not the focus of our contribution. For simplicity in BigFeat-FE, only basic operators like arithmetic (addition, subtraction, multiplication, division) and square functions were used. Our CPU-based experimental environment, with an Ubuntu 18.04 LTS system, 64-core Intel Processor @ 2.00GHz and 240 GB RAM, facilitated our experiments. F1-Score metric was the chosen performance indicator. In BigFeat's feature selection, both random forest and LightGBM were employed to determine feature importance. Initially, feature importance in BigFeat-FE was ascertained using a random forest model. We benchmarked BigFeat-FE and BigFeat-vanilla against AutoFeat and SAFE, and BigFeat-AutoML against AutoSklearn and TPOT. The experiments adhered to a 10-minute time budget for all AutoML frameworks, with a 24-hour threshold for unsuccessful automated FE runs. For BigFeat-FE/BigFeat-vanilla, parameters were set as follows: $\eta = 0.8, k = 10, \alpha = 5, Iterations = 7$.

For further information regarding the autoML and automated FE frameworks, you can refer to Sections 3.3 and 3.5.

Dataset	Instances	Features	Labels
madelon	2600	500	2
gina	3153	970	2
arcene	200	1000	2
nomao	34465	118	2
kc1	2109	21	2
eeg-eye	14980	14	2
credit-g	1000	20	2
sonar	208	60	2
shuttle	58000	9	7
blood	784	4	2
amazon_employee	32769	9	2
australian	690	14	2
Christine	5418	1636	2
GCM	190	16063	14
micro-mass	571	1300	20
numerai28	96320	21	2
phoneme	5404	5	2
isolet	7797	618	26
steel-plates-fault	1941	28	7
texture	5500	41	11

Table 12: The information of the benchmark datasets.

6.4. Results

6.4.1. Predictive Performance of BigFeat for FE

We evaluated the performance of BigFeat-vanilla and BigFeat-FE by analyzing the original features and those generated by AutoFeat, SAFE, BigFeat-vanilla, and BigFeat-FE across eight state-of-the-art classification algorithms. These algorithms include AdaBoost (AB), Decision Tree (DT), Extremely Randomized Trees (ET), k-Nearest Neighbors (kNN), Logistic Regression (LR), Multi-Layered Perceptron (MLP), Random Forest (RF), and Gradient Boosting (GBoost), which we implemented using their default hyperparameter settings from the scikit-learn library [85]. We reported the mean value of the performance from 10 independent runs, and for each classifier, we considered five distinct configurations: original raw features (ORG), engineered features derived from AutoFeat (AF), engineered features derived from SAFE (SF), engineered features derived from BigFeat-vanilla (BF-v), and engineered features obtained through the BigFeat-FE (BF-FE). SF Our results demonstrate that BigFeat-FE outperforms the most competitive baseline, SAFE, across 19 out of 20 datasets. It achieved an average improvement of 4.71% compared to SAFE. Additionally, BigFeat-FE surpassed both the ORG and AF configurations across most datasets, with notable average improvements of 5.66% and 8.65% over the ORG and AF configurations,

DS	CLF	ORG	AF	SF	BF-v	BF-m	DS	CLF	ORG	AF	SF	BF-v	BF-m
eeg-eye	LR	**97.25**	97.23	97.23	97.23	97.23	gina	LR	79.88	79.88	86.00	89.27	**89.92**
	AB	97.23	97.23	97.23	97.23	**97.24**		AB	86.45	86.45	86.00	89.27	**89.92**
	DT	**97.23**	96.19	95.86	96.31	96.2		DT	85.17	83.35	84	86.94	**87.59**
	ET	96.25	97.07	**97.32**	96.89	97.11		ET	**94.58**	93.86	94.00	91.91	92.56
	KNN	97.08	97.05	97.08	**97.16**	97.06		KNN	82.57	82.57	83.00	91.70	**92.35**
	MLP	97.05	96.49	94.26	97.24	**97.27**		MLP	86.15	86.99	88.00	94.79	**95.44**
	RF	96.81	97.42	**97.45**	97.13	97.4		RF	92.72	**93.11**	92.00	92.15	92.80
	GB	97.23	**97.26**	97.24	97.25	**97.26**		GB	92.42	92.42	93.00	92.88	**93.53**
	Avg	97.02	96.99	96.71	97.06	**97.10**		Avg	87.49	87.33	88.25	91.11	**91.76**
madelon	LR	**85.94**	55.95	66.42	68.45	84.53	kc1	LR	32.56	37.21	25	**38.46**	33.33
	AB	56.24	62.19	73.13	**81.78**	79.59		AB	37.21	34.57	36.36	26.37	**39.08**
	DT	62.19	75.4	74.86	77.19	**77.37**		DT	34.57	33.61	38.71	**40.94**	40.65
	ET	77.29	70.24	80.07	**89.06**	87.89		ET	36.97	41.76	**43.3**	40.43	42.22
	KNN	72.97	72.01	84.17	87.4	**89.25**		KNN	**47.31**	29.55	26.67	36.17	46.43
	MLP	72.01	56.3	56.3	80.23	**81.99**		MLP	29.55	**41.51**	8.96	36.78	36.14
	RF	30.21	73	81.63	87.6	**87.96**		RF	14.93	41.3	**45.83**	40.86	38
	GB	67.63	76.15	84.45	85.94	**87.92**		GB	26.32	36.14	**46.32**	32.56	41.76
	Avg	65.56	67.66	75.13	82.21	**84.56**		Avg	32.43	36.96	33.89	36.57	**39.70**
arcene	LR	68.18	63.39	**82.61**	81.82	81.74	shuttle	LR	96.73	86.82	56.62	96.77	**97.13**
	AB	**80.95**	61.75	79.07	71.11	69.41		AB	87.93	87.93	75.82	89.84	**89.9**
	DT	**68.29**	61.86	59.09	63.64	67.26		DT	99.97	99.97	78.7	**100**	99.99
	ET	60.87	62.29	**76.19**	68.18	72.73		ET	99.97	**99.99**	84.74	99.96	99.95
	KNN	71.43	61.1	66.67	**82.93**	80.95		KNN	99.88	98.94	79.09	**99.93**	99.92
	MLP	**86.96**	60.26	55.56	70.97	83.26		MLP	**99.99**	97.03	68.94	99.96	99.97
	RF	70.97	63.91	69.77	68.18	**73.91**		RF	**99.97**	**99.97**	83.44	99.96	99.96
	GB	**74.42**	63.74	71.11	68.18	70.41		GB	**100**	**100**	78.7	99.99	**100**
	Avg	72.76	62.29	70.01	71.88	**74.96**		Avg	98.06	96.33	75.76	98.3	**98.35**
credit-g	LR	88.28	58.91	**89.04**	**89.04**	88.88	nomao	LR	**96.98**	92.95	93.46	96.3	96.42
	AB	89.8	87.59	88.44	89.47	**89.86**		AB	93.2	96.16	96.07	96.09	**96.34**
	DT	**87.59**	82.67	81.63	80	85.03		DT	96.16	**96.4**	96.23	96.31	96.21
	ET	84.35	89.04	**92**	88.59	89.8		ET	96.52	**97.98**	97.87	97.83	97.86
	KNN	**92**	70.89	85.52	85.71	91.92		KNN	**97.91**	94.77	94.54	97.13	97
	MLP	72.96	69.35	81.12	89.19	**90.74**		MLP	94.96	94.53	94.37	97.29	**97.41**
	RF	78.83	88.44	**89.19**	88.89	88.89		RF	95.37	**97.86**	97.66	97.71	97.75
	GB	73.62	**90.54**	88.89	88.28	**90.54**		GB	93.82	96.89	96.9	**96.98**	96.94
	Avg	83.43	79.68	86.98	87.4	**89.46**		Avg	95.62	95.94	95.89	96.96	**96.99**
blood	LR	**48.57**	24.49	42.11	20.83	21.38	sonar	LR	**76.72**	0	72.69	55.26	75.06
	AB	24.49	51.61	48.57	50	**53.76**		AB	43.56	70.18	68.84	71.87	**72.29**
	DT	**51.61**	44.8	34.78	47.89	48.44		DT	70.18	77.67	63.19	**78.5**	76.66
	ET	39.44	42.9	41.18	41.79	**44.78**		ET	77.41	85.57	72.46	85.58	**88.36**
	KNN	42.25	46.9	**53.13**	52.31	51.48		KNN	**85.67**	53.38	68.34	77.96	76.77
	MLP	**54.55**	40	52.78	36.36	40.48		MLP	**79.48**	44.41	69.88	74.76	71.71
	RF	28	46.4	43.48	**50.75**	48.48		RF	67.8	83.58	72.83	84.49	**88.51**
	GB	4.76	43.8	48.48	48.57	**53.56**		GB	74.05	74.83	73.01	76.72	**79.06**
	Avg	36.71	42.61	**45.56**	43.56	45.30		Avg	71.86	61.20	70.16	75.64	**78.55**
Amazon_employee	LR	**97.25**	97.23	97.23	97.23	97.23	Micro-mass	LR	**96.02**	-	-	94.81	94.94
	AB	97.23	97.23	97.23	97.23	**97.24**		AB	14.18	-	-	**19.22**	18.15
	DT	**97.23**	96.19	95.86	96.31	96.2		DT	**86.94**	-	-	86.7	86.48
	ET	96.25	97.07	**97.32**	96.89	97.11		ET	93.89	-	-	**94.78**	93.76
	KNN	97.08	97.05	97.08	**97.16**	97.06		KNN	88.74	-	-	**92.55**	91.22
	MLP	97.05	96.49	94.26	97.24	**97.27**		MLP	97.28	-	-	96.02	**97.96**
	RF	96.81	97.42	**97.45**	97.13	97.4		RF	92.55	-	-	93.76	**94.94**
	GB	97.23	**97.26**	97.24	97.25	**97.26**		GB	86.46	-	-	**92.05**	91.23
	Avg	97.02	96.99	96.71	97.06	**97.10**		Avg	82.01	-	-	**83.74**	83.59

DS	CLF	ORG	AF	SF	BF-v	BF-m	DS	CLF	ORG	AF	SF	BF-v	BF-m
Christine	LR	**72.98**	-	72.69	67.19	69.5	Phoneme	LR	**76.72**	0	-	55.26	75.06
	AB	68.34	-	68.84	**71.46**	69.89		AB	43.56	70.18	-	71.87	**72.29**
	DT	**70.58**	-	63.19	65.31	63.09		DT	70.18	77.67	-	**78.5**	76.66
	ET	63.38	-	72.46	71.81	**73.03**		ET	77.41	85.57	-	85.58	**88.36**
	KNN	**74.05**	-	68.34	72.44	73.38		KNN	**85.67**	53.38	-	77.96	76.77
	MLP	70.59	-	69.88	68.57	**71.48**		MLP	**79.48**	44.41	-	74.76	71.71
	RF	0	-	**72.83**	71.53	72.37		RF	67.8	83.58	-	84.49	**88.51**
	GB	74	-	73.01	72.98	**74.54**		GB	74.05	74.83	-	76.72	**79.06**
	Avg	61.74	-	70.16	70.16	**70.91**		Avg	71.86	61.20	-	75.64	**78.55**
Australian	LR	83.87	65.6	**90.77**	89.6	90.41	isolet	LR	87.87	70.6	**94.77**	92.6	92.41
	AB	86.18	85	86.4	85.25	**87.3**		AB	90.18	87	89.4	90.25	**90.3**
	DT	**85.04**	79.4	77.69	78.63	83.62		DT	**90.04**	81.4	82.69	83.63	87.62
	ET	78.99	86	88	83.33	**88.9**		ET	83.99	90	92	87.33	**93.9**
	KNN	**87.8**	55.9	79.69	**87.8**	86.1		KNN	89.8	57.9	82.69	89.1	**91.8**
	MLP	67.77	36	77.05	**91.94**	88.1		MLP	69.77	38	80.05	**92.94**	91.1
	RF	78.26	86.9	87.8	86.18	**88**		RF	83.26	88.9	89.8	89.18	**91**
	GB	63.64	81.7	**85.25**	83.87	84.1		GB	66.64	85.7	**89.25**	85.87	86.1
	Avg	78.94	72.06	84.08	85.83	**87.07**		Avg	82.69	74.94	87.58	88.86	**90.53**
GCM	LR	**76.4**	32.5	-	74.74	75.25	texture	LR	**95.25**	93.23	94.23	94.23	95.23
	AB	**27.57**	18.3	-	22.92	23.6		AB	95.23	92.23	93.23	92.23	**97.24**
	DT	40.26	45.8	-	49.19	**55.51**		DT	92.23	92.19	90.86	91.31	**94.2**
	ET	64.28	62.4	-	**69.47**	65.79		ET	91.25	95.07	**95.32**	92.89	95.11
	KNN	**54.72**	27.7	-	50.85	53		KNN	93.08	95.05	95.08	**95.16**	95.06
	MLP	0.53	40.3	-	64.41	**77.11**		MLP	94.05	91.49	89.26	94.24	**96.27**
	RF	63.77	**71.9**	-	66.88	65.88		RF	93.81	93.42	93.45	94.13	**95.4**
	GB	63.49	61.3	-	59.42	**72.25**		GB	95.23	95.26	93.24	95.25	**97.26**
	Avg	48.88	45.03	-	57.24	**61.05**		Avg	93.77	93.49	93.08	93.68	**95.72**
Numerai28	LR	**57.08**	56.9	56.67	56.22	56.22	steel-plates-fault	LR	**76.72**	45.41	42.56	54.26	76.06
	AB	**56.9**	55.7	55.78	55.66	55.32		AB	42.56	70.18	72.18	**73.87**	73.29
	DT	**55.7**	50.16	50.83	50.29	55.57		DT	71.18	79.67	78.67	**80.5**	77.66
	ET	50.49	**51.92**	51.52	51.01	51.2		ET	79.41	84.57	86.57	85.58	**90.36**
	KNN	51.76	**51.84**	51.16	51.32	51.04		KNN	**84.67**	53.38	52.38	79.96	78.77
	MLP	51.84	3.92	**60.97**	51.55	56.93		MLP	**79.48**	46.41	45.41	74.76	72.71
	RF	**65.98**	52	51.38	51.31	64.63		RF	67.8	85.58	84.58	83.49	**89.51**
	GB	57.17	56.96	**57.31**	57.08	57.1		GB	73.05	74.83	75.83	77.72	**80.06**
	Avg	55.87	47.43	54.45	53.06	**56.00**		Avg	71.86	67.50	67.27	76.27	**79.80**

Table 13: Performance comparison of BigFeat (BF), original base features (ORG), and AutoFeat (AF) on different classifiers across different datasets. The configuration that surpasses all other configurations is presented in **bold** while the configuration with the lowest performance is underlined. Failed runs are presented using hyphens (-).

DS	CLF	RFG	AFS	BF	DS	CLF	RFG	AFS	BF
eeg-eye	LR	69.93	70.10	**70.23**	gina	LR	87.28	**90.06**	89.27
	AB	76.33	71.65	**77.89**		AB	87.11	88.50	**89.27**
	DT	83.09	77.99	**83.89**		DT	85.17	85.47	**86.94**
	ET	92.95	87.42	**92.58**		ET	93.86	92.35	**91.91**
	KNN	83.95	80.79	**84.60**		KNN	91.46	91.24	**91.70**
	MLP	68.32	69.87	**68.57**		MLP	94.32	**95.63**	94.79
	RF	91.42	85.92	**92.04**		RF	91.28	91.30	**92.15**
	GB	84.18	77.62	**84.41**		GB	92.79	92.51	**92.88**
	Avg	81.27	77.67	**81.78**		Avg	90.41	90.88	**91.11**

DS	CLF	RFG	AFS	BF	DS	CLF	RFG	AFS	BF
madelon	LR	55.01	**65.35**	63.97	kc1	LR	**28.77**	23.94	27.59
	AB	58.25	73.90	**78.28**		AB	21.66	27.03	**33.73**
	DT	70.26	**77.91**	73.79		DT	37.97	35.78	**46.53**
	ET	68.59	**86.89**	85.61		ET	37.93	38.27	**48.04**
	KNN	60.89	**86.09**	83.05		KNN	28.57	30.38	**31.90**
	MLP	60.89	**79.30**	74.22		MLP	32.05	20.90	**34.15**
	RF	67.56	84.56	**85.31**		RF	35.15	35.58	**40.94**
	GB	70.77	81.50	**83.81**		GB	21.79	30.38	**40.96**
	Avg	64.03	**79.44**	78.51		Avg	30.49	30.28	**37.98**
arcene	LR	83.33	**86.21**	83.64	shuttle	LR	91.44	**94.54**	93.60
	AB	**83.02**	76.67	77.19		AB	96.01	**96.71**	96.18
	DT	68.97	56.14	**76.67**		DT	99.96	99.80	**99.97**
	ET	77.19	79.31	**85.19**		ET	**99.94**	99.79	99.92
	KNN	76.67	74.58	**84.75**		KNN	99.82	99.74	**99.84**
	MLP	76.19	69.23	**83.64**		MLP	99.63	99.34	**99.66**
	RF	77.19	75.86	**78.57**		RF	**99.94**	99.80	99.92
	GB	60.71	77.42	**80.70**		GB	99.96	99.76	**99.97**
	Avg	75.41	74.43	**81.29**		Avg	98.34	**98.69**	98.63
credit-g	LR	80.79	**84.48**	81.03	nomao	LR	91.53	90.92	**91.61**
	AB	81.24	**85.94**	81.97		AB	91.18	90.74	**91.37**
	DT	78.02	**83.92**	76.03		DT	90.42	89.67	**90.61**
	ET	82.93	**86.29**	82.57		ET	**94.90**	94.04	94.73
	KNN	81.95	**85.38**	82.38		KNN	92.51	92.04	**92.51**
	MLP	81.20	**84.77**	81.36		MLP	93.13	91.72	**93.28**
	RF	82.63	**86.07**	83.56		RF	94.84	93.98	**94.56**
	GB	83.23	**85.12**	83.97		GB	92.87	92.08	**93.18**
	Avg	81.50	**85.25**	81.61		Avg	92.67	91.90	**92.73**
blood	LR	85.11	84.71	**85.11**	sonar	LR	75.00	78.38	**84.21**
	AB	82.29	**84.60**	83.33		AB	81.58	76.32	**84.62**
	DT	77.84	**83.71**	79.79		DT	74.36	70.42	**77.92**
	ET	81.96	**84.16**	83.29		ET	**91.14**	78.48	82.50
	KNN	83.25	81.93	**83.38**		KNN	82.93	76.92	**85.71**
	MLP	84.13	**84.96**	84.54		MLP	81.08	80.56	**87.67**
	RF	82.05	82.44	**83.67**		RF	82.05	76.92	**82.50**
	GB	83.03	84.29	**84.56**		GB	78.95	78.95	**84.21**
	Avg	82.46	**83.85**	83.46		Avg	80.89	77.12	**83.67**
Amazon_employee	LR	95.83	95.99	**97.23**	Australian	LR	88.56	87.7	**90.41**
	AB	95.78	96.52	**97.24**		AB	85.67	87.1	**87.3**
	DT	**96.38**	94.40	96.2		DT	83.62	**84.1**	83.62
	ET	**97.72**	95.08	97.11		ET	**89.78**	89.1	88.9
	KNN	**97.27**	96.08	97.06		KNN	85.55	85.9	**86.1**
	MLP	94.81	94.49	**97.27**		MLP	**88.79**	88.2	88.1
	RF	94.78	95.29	**97.4**		RF	85.81	85.8	**88**
	GB	**97.44**	95.59	97.26		GB	82.64	**85.1**	84.1
	Avg	96.25	95.43	**97.10**		Avg	86.30	86.62	**87.07**
Christine	LR	**69.72**	67.57	69.5	GCM	LR	72.44	74.6	**75.25**
	AB	69.26	67.73	**69.89**		AB	22.76	23.3	**23.6**
	DT	**63.72**	62.22	63.09		DT	**55.72**	53.3	55.51
	ET	70.26	71.10	**73.03**		ET	65.47	65.7	**65.79**
	KNN	71.65	70.77	**73.38**		KNN	50.90	52.2	**53**
	MLP	69.83	69.58	**71.48**		MLP	76.29	**77.3**	77.11
	RF	72.10	70.53	**72.37**		RF	**65.92**	63.8	65.88
	GB	**75.31**	72.86	74.54		GB	69.66	69.9	**72.25**
	Avg	70.23	69.04	**70.91**		Avg	59.90	60.01	**61.05**

DS	CLF	RFG	AFS	BF	DS	CLF	RFG	AFS	BF
Micro-mass	LR	93.01	**95.25**	94.94	Numerai28	LR	56.15	55.7	**56.22**
	AB	**19.12**	15.87	18.15		AB	54.36	54.7	**55.32**
	DT	85.63	84.87	**86.48**		DT	**55.68**	54.4	55.57
	ET	91.74	92.96	**93.76**		ET	48.62	49.5	**51.2**
	KNN	88.74	88.36	**91.22**		KNN	49.10	49.1	**51.04**
	MLP	**98.27**	97.44	97.96		MLP	55.38	56.1	**56.93**
	RF	**95.78**	95.55	94.94		RF	64.12	61.7	**64.63**
	GB	88.87	88.59	**91.23**		GB	55.48	**57.1**	57.1
	Avg	82.64	82.36	**83.58**		Avg	54.86	54.79	**56.00**
Phoneme	LR	72.23	72.61	**75.06**	isolet	LR	89.42	**92.9**	92.41
	AB	**72.75**	72.17	72.29		AB	**90.66**	88.3	90.3
	DT	74.61	75.78	**76.66**		DT	**87.84**	86.4	87.62
	ET	86.39	87.06	**88.36**		ET	**94.66**	93.7	93.9
	KNN	76.96	**77.47**	76.77		KNN	90.71	89.9	**91.8**
	MLP	69.90	71.46	**71.71**		MLP	90.11	**91.4**	91.1
	RF	87.57	87.44	**88.51**		RF	88.35	**91.5**	91
	GB	76.84	**79.90**	79.06		GB	**86.38**	85.2	86.1
	Avg	77.16	77.99	**78.55**		Avg	89.77	89.92	**90.53**
steel-plates-fault	LR	74.40	73.77	**76.06**	texture	LR	95.16	95.0	**95.23**
	AB	71.11	72.97	**73.29**		AB	95.73	96.1	**97.24**
	DT	**78.43**	75.85	77.66		DT	**94.78**	92.6	94.2
	ET	88.62	87.47	**90.36**		ET	**95.22**	93.0	95.11
	KNN	**79.64**	76.44	78.77		KNN	93.71	**96.0**	95.06
	MLP	71.26	71.87	**72.71**		MLP	94.41	96.2	**96.27**
	RF	**89.96**	89.33	89.51		RF	94.36	92.7	**95.4**
	GB	77.72	**80.13**	80.06		GB	**97.34**	96.0	97.26
	Avg	78.89	78.48	**79.80**		Avg	95.09	94.72	**95.72**

Table 14: Performance comparison of the feature generation and selection techniques of Bigfeat-vanilla/BigFeat-FE to random feature generation (RFG) and ANOVA feature selection (AFS) on different classifiers. The configuration that surpasses all other configurations is presented in **bold** while the configuration with the lowest performance is underlined.

respectively, as detailed in Table 13. These findings highlight the effectiveness of BigFeat-FE in improving classification outcomes.

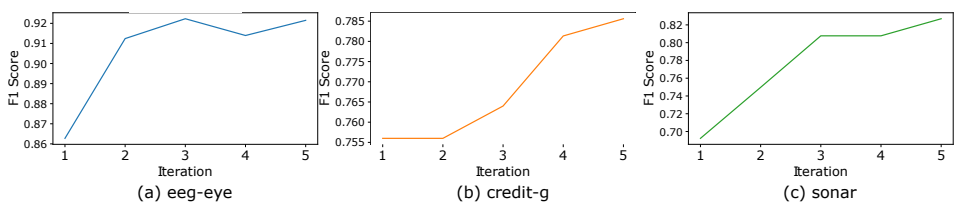


Figure 17: Performance Improvements over Iterations.

6.4.2. Feature Generation

We conducted a performance evaluation to measure the effectiveness of the dynamic feature generation technique integrated into the various execution options within the BigFeat framework. To do so, we compared the performance of features generated by BigFeat-vanilla to those generated randomly, using the same selection technique as in BigFeat-vanilla (referred to as BF). The performance of

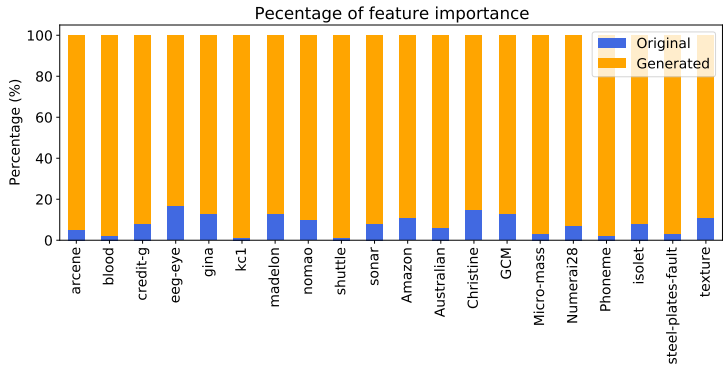


Figure 18: Contribution of generated and original feature set

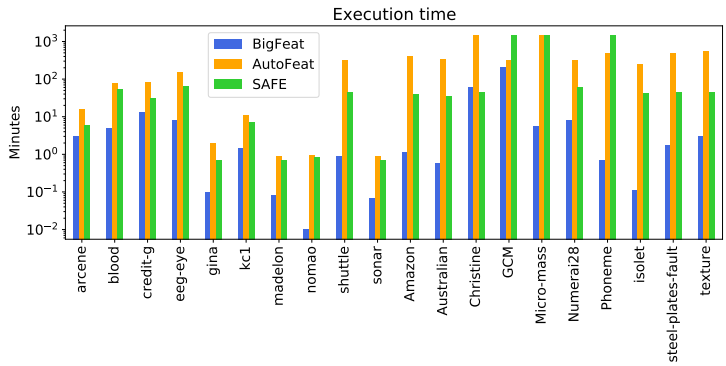


Figure 19: Execution time of BigFeat v.s. AutoFeat. Y-axis is log-scaled for presentation purposes.

different classification algorithms based on BigFeat-vanilla and RFG features is presented in Table 14. We used all algorithms with their default hyperparameters as provided by scikit-learn. The results indicate that features generated by BigFeat-vanilla are superior, showcasing an average improvement of 2.19% across all classifiers and datasets, compared to RFG, as shown in Table 14. This highlights the remarkable effectiveness of BigFeat's dynamic feature generation approach in enhancing classification outcomes.

6.4.3. Feature Selection

We conducted a comparative analysis to evaluate the effectiveness of the stability-based feature selection method used in BigFeat's diverse execution options. Specifically, we compared the performance of the feature subsets selected by BigFeat-vanilla with those chosen by ANOVA feature selection (AFS), using the same feature generation technique as BigFeat-vanilla across various classifiers. AFS identifies the top K features based on the ANOVA F-value between labels and features. The results of our comparison showed that the features derived from the stability-based feature selection technique consistently outperformed those selected by AFS. They exhibited an average improvement of 1.73% across all

classifiers and datasets, as demonstrated in Table 14.

6.4.4. Feature Importance

To evaluate the significance of the features produced by the different execution options provided by BigFeat, we follow a specific assessment procedure. For BigFeat-vanilla, we closely observe the N selected features and compare the proportion of combined feature importance derived from the base features to that of the generated features. To be more precise, we train Random Forest and Light Gradient Boosting models on the set that comprises the initial N base features, along with the KN generated features obtained from BigFeat-vanilla. We then calculate feature importance scores and select the top N most important features. Figure 18 demonstrates the fraction of base features (shown in orange) and generated features (shown in blue) that are deemed important among the top-ranked N features. Remarkably, across all datasets, the proportion of generated features within the top-ranked N exceeds 80%, as shown in Figure 18. This compelling outcome strongly validates that the generated features significantly enhance the overall importance and usefulness of the feature set.

6.4.5. Performance Improvements over Iterations

We have conducted experiments to determine whether the performance of our machine learning model can be improved over several iterations. To achieve this, we trained the model using `Auto-Sklearn` on the features generated from BigFeat-vanilla over five iterations and recorded the performance of the best-performing model for each dataset, as illustrated in Figure 17. Our results revealed that the performance improved significantly during the early iterations but gradually stabilized after a few iterations. The reason for this is that most useful combinations are explored in the early phase, and this has a positive impact on the predictive performance. In the later iterations, there is less room for improvement, and the generated feature combination has had a slight impact on the performance.

6.4.6. Scalability of BigFeat-vanilla and BigFeat-FE

BigFeat-FE and BigFeat-vanilla exhibit linear complexity concerning the original base feature's size. To evaluate scalability, we analyzed the execution time of BigFeat-FE, BigFeat-vanilla, AutoFeat, and SAFE's default configurations. Figure 19 depicts the results, which indicate that BigFeat is notably more scalable, with an average execution time of 20 times faster than AutoFeat, and 14 times faster than SAFE.

6.4.7. Meta-Learning Evaluation in the BigFeat-FE

We performed a detailed analysis to comprehensively evaluate the significance of the meta-learning phase in BigFeat-FE and to determine the effectiveness of the meta-features used for data representation. We compared the performance of

BigFeat-FE using four different sets of features: (1) BF-StatLandMod, which includes statistical, information-theoretic, landmarkers, and model-based features; (2) BF-Stat, which only includes statistical features; (3) BF-Land, which only includes landmarkers metafeatures; and (4) BF-Mod, which only includes model-based metafeatures. This examination allowed us to understand the contributions of each feature category to the performance of BigFeat-FE, providing valuable insights into their respective significance within the meta-learning phase. Our analysis showed that BF-StatLandMod consistently improved the performance of BigFeat-FE across all ten datasets, with an average difference of 1.83% (refer to Table 13). This highlights the essential role of BF-StatLandMod features in enhancing the effectiveness of BigFeat-FE. Additionally, the Nemenyi signed-rank test confirmed the statistically significant superiority of BigFeat-FE's performance based on BF-StatLandMod over BF-Stat, BF-Land, and BF-Mod (refer to Figure 20). We conducted the Nemenyi test to evaluate the significance of performance differences between BigFeat-FE, BigFeat-vanilla, SAFE, AutoFeat, and ORG. The results of the Nemenyi signed-rank test confirmed the statistically significant superiority of BigFeat-FE's performance (refer to Figure 21).

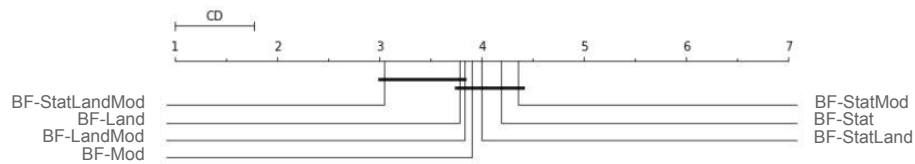


Figure 20: Nemenyi Test ($\alpha = 0.05$) of the performance of meta-learning-based techniques for warm starting including BF-StatLandMod, BF-Land, BF-LandMod, and BF-Mod

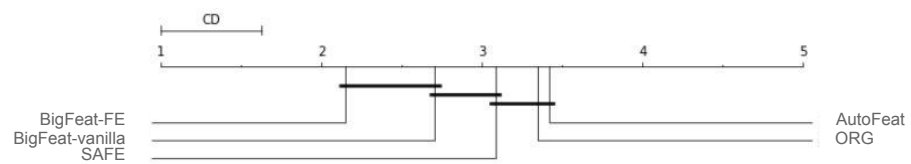


Figure 21: Nemenyi Test ($\alpha = 0.05$) of the performance of BigFeat-FE, BigFeat-vanilla, SAFE AutoFeat, and ORG.

6.5. Evaluating BigFeat-AutoML

We conducted a comparative analysis of BigFeat-AutoML, TPOT, and AutoSKLearn, which are popular AutoML tools, to evaluate the effectiveness of BigFeat-AutoML as a complete end-to-end AutoML solution. We have summarized the performance of BigFeat-AutoML and the baseline methods in Table 15. Our results indicate that BigFeat-AutoML outperforms TPOT on 15 datasets and AutoSKLearn on 16 datasets. The average performance difference is shown in Table 15, which highlights BigFeat-AutoML's significant edge over AutoSKLearn

and TPOT, with an average improvement of 2.25% and 0.74%, respectively. To determine the significance of these performance differences, we conducted the Nemenyi test, comparing BigFeat-AutoML to Auto-Sklearn and TPOT. The Nemenyi signed-rank test confirmed the statistically significant superiority of BigFeat-AutoML's performance, as shown in Figure 21.

Dataset	AutoSKLearn	TPOT	BigFea AutoML
arcene	80.00	90.00	90.00
blood	79.13	78.07	**80.40**
credit-g	**76.40**	73.60	75.35
eeg-eye	92.31	**97.52**	92.47
gina	92.65	94.68	**94.99**
kc1	84.47	86.74	**87.32**
madelon	75.85	84.92	**88.81**
nomao	96.79	96.24	**96.87**
shuttle	**99.98**	99.96	99.94
sonar	75.00	**84.61**	84.29
Amazon_employee	**99.53**	97.88	98.94
Australian	88.07	88.15	**89.92**
Christine	**72.93**	72.35	72.56
GCM	63.19	**63.63**	62.69
Micro-mass	90.00	90.59	**91.71**
Numerai28	57.01	**58.61**	58.57
Phoneme	78.57	78.57	**81.30**
isolet	93.09	90.69	**93.31**
steel-plates-fault	82.37	80.87	**82.73**
texture	97.21	97.14	**97.38**
Avg	83.73	85.24	**85.98**

Table 15: Classification performance BigFeat AutoML against AutoSKLearn and TPOT. The configuration that surpasses all other configurations is presented in **bold** while the configuration with the lowest performance is underlined.

6.6. Interpretability

As interpretability is one of the main features of BigFeat, it is critical for each generated feature to be viewed as an aggregation of a set of mathematical operations over the base features. Therefore, BigFeat easily map the generated features to the original features using a simple mathematical formula. For example, the Euclidean distance feature in Figure 16(a) is expressed as:
$\mathbb{X}^i = \sqrt{(X^{34} - X^{76})^2 - (X^{53} - X^{44})^2}$ and the price per meter feature in Figure 16(b) is expressed as $\mathbb{X}^i = X^{64} \div (X^{27} \times X^{47})$. These simple expression could be easily explained and mapped to the original features, especially those generated from shallow computational trees.

7. CONCLUSION

7.1. Summary of Key Findings

In this thesis, we undertake an exhaustive evaluation of multiple Automated Machine Learning (AutoML) frameworks, providing a detailed understanding of their performance characteristics across a broad range of datasets. This includes an in-depth analysis of crucial design decisions such as time allocation, search space optimization, the application of meta-learning, and ensemble construction strategies.

A significant portion of the thesis is dedicated to exploring the transformative impact of Automated Feature Engineering (Automated FE) on the effectiveness of AutoML frameworks. Through empirical evaluation, we demonstrate that integrating Automated FE tools like AutoFeat into AutoML systems like Auto-Sklearn and TPOT notably enhances their predictive performance. Moreover, the limitations of existing Automated FE tools are critically examined, highlighting the need for further advancements in this field.

The thesis introduces BigFeat, a comprehensive and scalable framework designed to streamline and optimize both the feature engineering and AutoML phases. BigFeat consists of two key components: BigFeat-FE for feature engineering and BigFeat-AutoML for algorithm selection and hyperparameter tuning. Our experiments reveal that BigFeat-FE significantly outperforms existing Automated FE frameworks like AutoFeat and SAutomated FE in various datasets, offering an average performance improvement of 8.65% over AutoFeat and 4.71% over SAFE. Additionally, BigFeat-AutoML shows substantial gains over renowned AutoML frameworks like TPOT and AutoSklearn, marking an average improvement of 0.74% and 2.25%, respectively. The linear complexity and execution speed of BigFeat further affirm its scalability, averaging 20 times faster than AutoFeat, and 14 times over SAFE.

In conclusion, this thesis underscores the critical synergy between AutoML and Automated FE, suggesting that the integration of these two components is key to unlocking higher levels of efficiency and effectiveness in machine learning applications.

7.2. Contributions of the Research

This thesis contributes to the field of Automated Machine Learning (AutoML) and Automated Feature Engineering (AFE) in several significant ways:

- Provides a comprehensive evaluation and comparison of multiple AutoML frameworks, analyzing their performance across various datasets and investigating the impact of design decisions such as time budget, search space, meta-learning, and ensemble construction.

- Empirically evaluates the integration of Automated Feature Engineering tools, particularly AutoFeat, into AutoML systems like Auto-Sklearn and TPOT, demonstrating notable enhancements in predictive performance.
- Introduces BigFeat, a scalable and interpretable framework that optimizes both feature engineering and AutoML phases. BigFeat-FE surpasses existingAutomated FE frameworks in performance, and BigFeat-AutoML shows significant improvements over renowned AutoML frameworks.
- Highlights the synergy between AutoML andAutomated FE, underscoring the importance of their integration for achieving greater efficiency and effectiveness in machine learning applications.

7.3. Limitations and Areas for Future Research

While this thesis advances the understanding of AutoML andAutomated FE, it has certain limitations, which open avenues for future research:

- The evaluation of AutoML frameworks may not cover all possible real-world scenarios. Future studies could explore these frameworks in more diverse and complex environments.
- The integration of AutoFeat into AutoML systems was limited to specific frameworks. Extending this integration to a broader range of systems could yield more insights.
- BigFeat-FE is a feature engineering approach that achieves linear time and space complexity by optimizing the feature generation and selection phases. However, it may not always produce optimal features as the use of information gain as a feature importance criterion in the generation phase may not accurately represent feature importance for every machine learning model. To overcome this limitation and ensure interpretability, BigFeat-AutoML has constrained its machine learning search space to three decision tree-based models and a single linear model.
- In the feature selection phase, BigFeat is designed to check the similarity between parent features and their offspring, which are features generated from the parent features. This approach significantly reduces the number of comparisons required, as compared to comparing each generated/original pair of features. However, for datasets with a large number of similar features, this approach may not be optimal and could lead to redundancy in the selected features.
- The focus on supervised learning in the current work leaves the area of unsupervised learning in AutoML relatively unexplored. Future research should delve into this area, addressing challenges unique to unsupervised learning tasks.
- In the age of big data, this research can be broadened to explore various data characteristics. For example, for rapidly evolving data, BigFeat should be

flexible and continually learn to respond to shifts in data distribution, trends, and patterns. A real-time solution is necessary to tackle this challenge.

7.4. Conclusion

In this dissertation, we have undertaken a comprehensive investigation of Automated Machine Learning (AutoML) frameworks, with particular emphasis on feature engineering and its impact on predictive performance. Our studies focused on evaluating and comparing various frameworks, including Auto-SKlearn, AutoSKlearn, and Recipe.

Key insights from our research include:

- **Performance Analysis:** Auto-SKlearn-e and AutoSKlearn emerged as top performers across different time budgets, while Recipe lagged. The efficacy of meta-learning strategies diminished over longer time budgets, yet ensembling techniques consistently improved performance.
- **Critical Role of Feature Engineering:** The research underscored the paramount importance of feature engineering in influencing predictive performance. Variations in preprocessing choices, especially in datasets with fewer instances than features, significantly impacted results. This finding suggests a potential for enhancing AutoML frameworks with robust feature engineering capabilities.
- **Framework Limitations and Overfitting:** Many frameworks exceeded their designated time budgets, leading to terminated runs. The study also highlighted the risk of overfitting associated with extensive search processes within AutoML tools, calling for more efficient mechanisms to mitigate this challenge.
- **Future Directions:** The research opens new avenues for improvement in AutoML, particularly in the area of feature engineering. Developing novel techniques to prevent overfitting and to enhance the overall performance of AutoML pipelines through improved feature engineering is a promising direction for future work.

This dissertation contributes to a deeper understanding of the AutoML frameworks, emphasizing the crucial role of feature engineering, and sets a foundation for future advancements in the field of automated machine learning.

BIBLIOGRAPHY

[1] David W Aha. "Generalizing from case studies: A case study". In: *Machine Learning Proceedings 1992*. Elsevier, 1992, pp. 1–10.

[2] Rahman Ali, Sungyoung Lee, and Tae Choong Chung. "Accurate multi-criteria decision making methodology for recommending machine learning algorithm". In: *Expert Systems with Applications* 71 (2017), pp. 257–278.

[3] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. "Neural machine translation by jointly learning to align and translate". In: *arXiv preprint arXiv:1409.0473* (2014).

[4] Yoshua Bengio, Aaron Courville, and Pascal Vincent. "Representation learning: A review and new perspectives". In: *IEEE transactions on pattern analysis and machine intelligence* 35.8 (2013), pp. 1798–1828.

[5] Hilan Bensusan and Christophe Giraud-Carrier. "Discovering task neighbourhoods through landmark learning performances". In: *European Conference on Principles of Data Mining and Knowledge Discovery*. Springer. 2000, pp. 325–330.

[6] Hilan Bensusan, Christophe G Giraud-Carrier, and Claire Julia Kennedy. "A Higher-order Approach to Meta-learning." In: *ILP Work-in-progress reports* 35 (2000), p. 44.

[7] James Bergstra, Daniel Yamins, and David Daniel Cox. "Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures". In: *The Journal of Machine Learning Research* (2013).

[8] Bernd Bischl et al. "OpenML benchmarking suites and the OpenML100". In: *arXiv preprint arXiv: 1708.03731* (2017).

[9] Bernd Bischl et al. "OpenML Benchmarking Suites". In: *arXiv:1708.03731 v2 [stat.ML]* (2019).

[10] Christopher M Bishop and Nasser M Nasrabadi. *Pattern recognition and machine learning*. Vol. 4. 4. Springer, 2006.

[11] Pavel Brazdil et al. *Metalearning: Applications to data mining*. Springer Science & Business Media, 2008.

[12] Rich Caruana et al. "Ensemble selection from libraries of models". In: *ICML*. 2004.

[13] Ciro Castiello, Giovanna Castellano, and Anna Maria Fanelli. "Meta-data: Characterization of input features for meta-learning". In: *International Conference on Modeling Decisions for Artificial Intelligence*. Springer. 2005, pp. 457–468.

[14] M. Chen, S. Mao, and Y. Liu. "Big data: A survey". In: *Mobile Networks and Applications* 19.2 (2012), pp. 171–209.

[15] Xiangning Chen et al. “Neural feature search: A neural architecture for automated feature engineering”. In: *2019 IEEE International Conference on Data Mining (ICDM)*. IEEE. 2019, pp. 71–80.

[16] Thomas G Dietterich. “Ensemble methods in machine learning”. In: *International workshop on multiple classifier systems*. Springer. 2000, pp. 1–15.

[17] Finale Doshi-Velez and Been Kim. “Towards a rigorous science of interpretable machine learning”. In: *arXiv preprint arXiv:1702.08608* (2017).

[18] Radwa El Shawi, Youssef Sherif, and Sherif Sakr. “Towards Automated Concept-based Decision TreeExplanations for CNNs.” In: *EDBT*. 2021, pp. 379–384.

[19] Radwa Elshawi, Mouaz H Al-Mallah, and Sherif Sakr. “On the interpretability of machine learning-based model for predicting hypertension”. In: *BMC medical informatics and decision making* 19.1 (2019), pp. 1–32.

[20] Radwa ElShawi and Sherif Sakr. “TPE-AutoClust: A Tree-based Pipline Ensemble Framework for Automated Clustering”. In: *2022 IEEE International Conference on Data Mining Workshops (ICDMW)*. IEEE. 2022, pp. 1144–1153.

[21] Nick Erickson et al. *Autogluon-tabular: Robust and accurate automl for structured data*. 2020.

[22] Stefan Falkner, Aaron Klein, and Frank Hutter. “Bohb: Robust and efficient hyperparameter optimization at scale”. In: *arXiv preprint arXiv:1807.01774* (2018).

[23] Wei Fan et al. “Generalized and heuristic-free feature construction for improved accuracy”. In: *Proceedings of the 2010 SIAM International Conference on Data Mining*. SIAM. 2010, pp. 629–640.

[24] Matthias Feurer and Frank Hutter. “Hyperparameter optimization”. In: *Automated Machine Learning*. Springer, Cham, 2019, pp. 3–33.

[25] Matthias Feurer, Jost Springenberg, and Frank Hutter. “Initializing bayesian hyperparameter optimization via meta-learning”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 29. 1. 2015.

[26] Matthias Feurer et al. “Auto-sklearn 2.0: The next generation”. In: *arXiv preprint arXiv:2007.04074* 24 (2020).

[27] Matthias Feurer et al. “Efficient and robust automated machine learning”. In: *Advances in neural information processing systems*. Vol. 28. 2015, pp. 2962–2970.

[28] Brian Fitzgerald. “Software Crisis 2.0”. In: *Computer* 45.4 (2012).

[29] Joao Gama and Pavel Brazdil. “Characterization of classification algorithms”. In: *Portuguese Conference on Artificial Intelligence*. Springer. 1995, pp. 189–200.

[30] A. Gandomi and M. Haider. “Beyond the hype: Big data concepts, methods, and analytics”. In: *International Journal of Information Management* 35.2 (2015), pp. 137–144.

[31] Romaric Gaudel and Michele Sebag. “Feature selection as a one-player game”. In: 2010.

[32] Pieter Gijsbers and Joaquin Vanschoren. “GAMA: A General Automated Machine Learning Assistant”. In: *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer. 2020, pp. 560–564.

[33] Pieter Gijsbers et al. *AMLB: an AutoML Benchmark*. 2022.

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

[35] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT Press, 2016.

[36] Isabelle Guyon and André Elisseeff. “An introduction to variable and feature selection”. In: *Journal of machine learning research* 3.Mar (2003), pp. 1157–1182.

[37] Isabelle Guyon et al. “Analysis of the AutoML Challenge Series”. In: *Automated Machine Learning* (2019), p. 177.

[38] J. Han, M. Kamber, and J. Pei. *Data Mining: Concepts and Techniques*. Elsevier, 2011.

[39] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The elements of statistical learning: data mining, inference, and prediction*. Springer Science & Business Media, 2009.

[40] Kaiming He et al. “Deep residual learning for image recognition”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

[41] Xin He, Kaiyong Zhao, and Xiaowen Chu. “AutoML: A Survey of the State-of-the-Art”. In: *arXiv preprint arXiv:1908.00709* (2019).

[42] James Heaton. *Deep learning and neural networks: An overview for newcomers*. Createspace Independent Publishing Platform, 2017.

[43] Geoffrey Hinton et al. “Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups”. In: *IEEE Signal Processing Magazine* 29.6 (2012), pp. 82–97.

[44] Franziska Horn, Robert Pack, and Michael Rieger. “The autofeat python library for automated feature engineering and selection”. In: *arXiv preprint arXiv:1901.07329* (2019), pp. 111–120.

[45] Franziska Horn, Robert Pack, and Michael Rieger. “The autofeat python library for automated feature engineering and selection”. In: *Machine Learning and Knowledge Discovery in Databases: International Workshops of ECML PKDD 2019, Würzburg, Germany, September 16–20, 2019, Proceedings, Part I*. Springer. 2020, pp. 111–120.

[46] Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. “Sequential model-based optimization for general algorithm configuration”. In: *International conference on learning and intelligent optimization*. Springer. 2011, pp. 507–523.

[47] International Data Corporation. *Worldwide semiannual cognitive/artificial intelligence systems spending guide.* Tech. rep. 2017.

[48] Haifeng Jin, Qingquan Song, and Xia Hu. “Auto-keras: An efficient neural architecture search system”. In: *ACM KDD*. 2019.

[49] Michael I Jordan and Tom M Mitchell. “Machine learning: Trends, perspectives, and prospects”. In: *Science* 349.6245 (2015), pp. 255–260.

[50] Alexandros Kalousis. “Algorithm selection via meta-learning”. PhD thesis. University of Geneva, 2002.

[51] James Max Kanter and Kalyan Veeramachaneni. “Deep feature synthesis: Towards automating data science endeavors”. In: *2015 IEEE International Conference on Data Science and Advanced Analytics, DSAA 2015, Paris, France, October 19-21, 2015*. IEEE. 2015, pp. 1–10.

[52] Gilad Katz, Eui Chul Richard Shin, and Dawn Song. “Explorekit: Automatic feature generation and selection”. In: *2016 IEEE 16th International Conference on Data Mining (ICDM)*. IEEE. 2016, pp. 979–984.

[53] Ambika Kaul, Saket Maheshwary, and Vikram Pudi. “Autolearn—Automated feature generation and selection”. In: *2017 IEEE International Conference on data mining (ICDM)*. IEEE. 2017, pp. 217–226.

[54] Samina Khalid, Tehmina Khalil, and Shamila Nasreen. “A survey of feature selection and feature extraction techniques in machine learning”. In: *2014 science and information conference*. IEEE. 2014, pp. 372–378.

[55] Udayan Khurana, Horst Samulowitz, and Deepak Turaga. “Feature engineering for predictive modeling using reinforcement learning”. In: *arXiv preprint arXiv:1709.07150* (2017).

[56] Udayan Khurana, Horst Samulowitz, and Deepak Turaga. “Feature engineering for predictive modeling using reinforcement learning”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 32. 1. 2018.

[57] Udayan Khurana et al. “Cognito: Automated feature engineering for supervised learning”. In: *2016 IEEE 16th International Conference on Data Mining Workshops (ICDMW)*. IEEE. 2016, pp. 1304–1307.

[58] Aaron Klein et al. “Robo: A flexible and robust bayesian optimization framework in python”. In: *NIPS 2017 Bayesian Optimization Workshop*. 2017.

[59] Brent Komer, James Bergstra, and Chris Eliasmith. “Hyperopt-sklearn: automatic hyperparameter configuration for scikit-learn”. In: *ICML workshop on AutoML*. Vol. 9. Citeseer. 2014, pp. 2825–2830.

[60] Lars Kotthoff. “Algorithm selection for combinatorial search problems: A survey”. In: *Data mining and constraint programming*. Springer, 2016, pp. 149–190.

[61] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. “Imagenet classification with deep convolutional neural networks”. In: *Advances in neural information processing systems*. 2012, pp. 1097–1105.

[62] Hoang Thanh Lam et al. “Automated data science for relational data”. In: *2021 IEEE 37th International Conference on Data Engineering (ICDE)*. IEEE. 2021, pp. 2689–2692.

[63] Hoang Thanh Lam et al. “Neural feature learning from relational database”. In: *arXiv preprint arXiv:1801.05372* (2018).

[64] Louisa Lam and SY Suen. “Application of majority voting to pattern recognition: an analysis of its behavior and performance”. In: *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans* 27.5 (1997), pp. 553–568.

[65] J Richard Landis and Gary G Koch. “An application of hierarchical kappa-type statistics in the assessment of majority agreement among multiple observers”. In: *Biometrics* (1977), pp. 363–374.

[66] Tilman Lange et al. “Stability-based model selection”. In: *NIPS*. Vol. 15. 2002.

[67] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. “Deep learning”. In: *nature* 521.7553 (2015), p. 436.

[68] Erin LeDell and Sebastien Poirier. “H2O AutoML: Scalable Automatic Machine Learning”. In: *7th ICML Workshop on Automated Machine Learning (AutoML)* (July 2020). URL: https://www.automl.org/wp-content/uploads/2020/07/AutoML_2020_paper_61.pdf.

[69] Joseph Lee Rodgers and W Alan Nicewander. “Thirteen ways to look at the correlation coefficient”. In: *The American Statistician* 42.1 (1988), pp. 59–66.

[70] Chenxi Liu et al. “Progressive neural architecture search”. In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2018, pp. 19–34.

[71] Guoshan Lu et al. “Catch: Collaborative Feature Set Search for Automated Feature Engineering”. In: *Proceedings of the ACM Web Conference 2023*. 2023, pp. 1886–1896.

[72] Mohamed Maher and Sherif Sakr. “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*. 2019.

[73] Shaul Markovitch and Dan Rosenstein. “Feature generation using general constructor functions”. In: *Machine Learning* 49.1 (2002), pp. 59–98.

[74] S. Marsland. *Machine Learning: An Algorithmic Perspective*. CRC Press, 2015.

[75] Nicolai Meinshausen and Peter Bühlmann. "Stability selection". In: *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 72.4 (2010), pp. 417–473.

[76] Donald Michie, David J Spiegelhalter, and Charles C Taylor. *Machine learning, neural and statistical classification*. 1994.

[77] Tom Mitchell et al. "Machine learning". In: *Annual review of computer science* 4.1 (1990), pp. 417–433.

[78] Grégoire Montavon, Wojciech Samek, and Klaus-Robert Müller. "Methods for interpreting and understanding deep neural networks". In: *Digital Signal Processing* 73 (2018), pp. 1–15.

[79] K-R Muller et al. "An introduction to kernel-based learning algorithms". In: *IEEE transactions on neural networks* 12.2 (2001), pp. 181–201.

[80] Fatemeh Nargesian et al. "Learning Feature Engineering for Classification." In: *Ijcai*. Vol. 17. 2017, pp. 2529–2535.

[81] Randal S Olson and Jason H Moore. "TPOT: A tree-based pipeline optimization tool for automating machine learning". In: *Automated Machine Learning*. Springer, 2019, pp. 151–160.

[82] Randal S. Olson and Jason H. Moore. "TPOT: A Tree-based Pipeline Optimization Tool for Automating Machine Learning". In: *Proceedings of the Workshop on Automatic Machine Learning*. Ed. by Frank Hutter, Lars Kotthoff, and Joaquin Vanschoren. Vol. 64. Proceedings of Machine Learning Research. New York, New York, USA: PMLR, June 2016, pp. 66–74. URL: http://proceedings.mlr.press/v64/olson_tpot_2016.html.

[83] YPPAF Pavel and Brazdil2 Carlos Soares. "Decision tree-based data characterization for meta-learning". In: *IDDM-2002* 111 (2002).

[84] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[85] Fabian Pedregosa et al. "Scikit-learn: Machine learning in Python". In: *Journal of machine learning research* 12.Oct (2011), pp. 2825–2830.

[86] Bernhard Pfahringer, Hilan Bensusan, and Christophe G Giraud-Carrier. "Meta-Learning by Landmarking Various Learning Algorithms." In: *ICML*. Citeseer. 2000, pp. 743–750.

[87] Selwyn Piramuthu and Riyaz T Sikora. "Iterative feature construction for improving inductive learning algorithms". In: *Expert Systems with Applications* 36.2 (2009), pp. 3401–3406.

[88] Aleksandra Płońska and Piotr Płoński. *MLJAR: State-of-the-art Automated Machine Learning Framework for Tabular Data. Version 0.10.3*. Łapy, Poland, 2021. URL: https://github.com/mljar/mljar-supervised.

[89] Stuart J Russell and Peter Norvig. *Artificial intelligence a modern approach*. London, 2010.

[90] Alex GC de Sá et al. "RECIPE: a grammar-based framework for automatically evolving classification pipelines". In: *European Conference on Genetic Programming*. Springer. 2017, pp. 246–261.

[91] John Schulman et al. "Proximal policy optimization algorithms". In: *arXiv preprint arXiv:1707.06347* (2017).

[92] Radwa El Shawi, Mohamed Maher, and Sherif Sakr. "Automated Machine Learning: State-of-The-Art and Open Challenges". In: *CoRR* abs/1906.02287 (2019). arXiv: 1906.02287. URL: http://arxiv.org/abs/1906.02287.

[93] Qitao Shi et al. "Safe: Scalable automatic feature engineering framework for industrial tasks". In: *2020 IEEE 36th International Conference on Data Engineering (ICDE)*. IEEE. 2020, pp. 1645–1656.

[94] Micah J Smith et al. "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*. 2020, pp. 785–800.

[95] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[96] Thomas Swearingen et al. "ATM: A distributed, collaborative, scalable system for automated machine learning". In: *2017 IEEE International Conference on Big Data (Big Data)*. 2017.

[97] Christian Szegedy et al. "Rethinking the inception architecture for computer vision". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 2818–2826.

[98] Chris Thornton et al. "Auto-WEKA: Combined Selection and Hyperparameter Optimization of Classification Algorithms". In: *KDD* (2012).

[99] Anh Truong et al. "Towards automated machine learning: Evaluation and comparison of automl approaches and tools". In: *arXiv preprint arXiv:1908.05557* (2019), pp. 1471–1479.

[100] Anton Vakhrushev et al. *LightAutoML: AutoML Solution for a Large Financial Services Ecosystem*. 2021.

[101] Joaquin Vanschoren. "Meta-learning: A survey". In: *arXiv preprint arXiv:1810.03548* (2018).

[102] Joaquin Vanschoren et al. "OpenML: Networked Science in Machine Learning". In: *SIGKDD Explorations* 15.2 (2013), pp. 49–60. DOI: 10.1145/2641190.2641198. URL: http://doi.acm.org/10.1145/2641190.2641198.

[103] Chi Wang et al. "FLAML: A fast and lightweight automl library". In: *Proceedings of Machine Learning and Systems* 3 (2021), pp. 434–447.

[104] Fei-Yue Wang et al. “Where does AlphaGo go: From church-turing thesis to AlphaGo thesis and beyond”. In: *IEEE/CAA Journal of Automatica Sinica* 3.2 (2016), pp. 113–120.

[105] Christopher JCH Watkins and Peter Dayan. “Q-learning”. In: *Machine learning* 8 (1992), pp. 279–292.

[106] Jianyu Zhang et al. “Automatic feature engineering by deep reinforcement learning”. In: *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*. 2019, pp. 2312–2314.

[107] Guanghui Zhu et al. “DIFER: differentiable automated feature engineering”. In: *International Conference on Automated Machine Learning*. PMLR. 2022, pp. 17–1.

[108] X. Zhu and A.B. Goldberg. *Introduction to Semi-supervised Learning*. Synthesis lectures on artificial intelligence and machine learning. Morgan & Claypool, 2009. ISBN: 9781598295474. URL: https://books.google.ee/books?id=c_haJrQ0ScAC.

[109] Marc-André Zöller and Marco F Huber. “Benchmark and survey of automated machine learning frameworks”. In: *Journal of artificial intelligence research* 70 (2021), pp. 409–472.

[110] Marc-André Zöller and Marco F Huber. “Survey on Automated Machine Learning”. In: *arXiv preprint arXiv:1904.12054* (2019).

[111] Albert Y Zomaya and Sherif Sakr. *Handbook of big data technologies*. Springer, 2017.

[112] Barret Zoph and Quoc V Le. “Neural architecture search with reinforcement learning”. In: *arXiv preprint arXiv:1611.01578* (2016).

Appendix A. AUTOML BENCHMARK

A.1. Evaluated Datasets

Table 16 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_9_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
hiva_agnostic (1039)	1618	4229	2	0	0	3,32
ipums_la_99-small (378)	60	8844	9	0	0	6,64
jm1 (1053)	22	10885	2	0	0	1,71
jungle_chess_2pcs (40997)	45	4704	3	0	0	6,64
KDDCup99 (1113)	40	494020	23	0	0	6,64
kin8nm (189)	9	8192	2	0	1	2,41
leukemia (1104)	7130	72	2	0	1	0,47
lymphoma_2classes (1101)	4027	45	2	0	1	2,81
MagicTelescope (1120)	11	19020	2	0	0	0,34
mfeat-pixel (20)	241	2000	2	0	0	1,00
mnist_784 (554)	720	70000	10	0	0	1,53
openml_phpJNxH0q (15)	10	699	2	0	0	1,00
page-blocks (30)	11	5473	2	0	1	3,60
php0FyS2T (1492)	65	1600	100	0	0	0,22
php3CTpvq (1509)	5	149332	22	0	0	0,97
php5OMDBD (40971)	23	1000	30	0	6	1,16
php5s7Ep8 (40982)	28	1941	7	0	0	1,86
php7KLval (1547)	21	1000	2	0	0	0,59
phpB0xrNj (300)	618	7797	26	0	0	1,58
phpbL6t4U (1476)	129	13910	6	0	1	0,11
phpchCuL5 (40966)	81	1080	8	0	0	1,71
phpCsX3fx (1491)	65	1600	100	0	1	0,48
phpdo58hj (1562)	4703	64	2	0	0	3,32
phpdReP6S (1487)	73	2534	2	0	0	2,30
phpEZ030X (1561)	3722	64	2	0	0	2,48
phpfLuQE4 (1485)	501	2600	2	0	0	1,00
phpfrJpBS (1568)	9	12958	4	0	0	1,00
phpGReJjU (40985)	4	45781	20	0	1	4,70
phpGUrE90 (1494)	42	1055	2	0	22	1,00
phphQEck0 (1502)	4	245057	2	0	1	1,00
phpHyLSNF (1515)	1083	571	20	0	26	0,48
phpkIxskf (1461)	17	45211	2	0	1	2,19
phpmcGu2X (1468)	857	1080	9	0	1	0,94
phpmPOD5A (4135)	10	32769	2	50	0	0,71
phpn1jVwe (310)	7	11183	2	0	0	1,57
phpN4gaxw (1477)	130	13910	6	0	0	0,16
phpNevWWL (40477)	27	2800	5	0	0	1,71
phpoOxxNn (1493)	65	1599	100	0	9	1,72
phpoW7Dbi (1566)	101	1212	2	0	0	2,55
phpPbCMyg (1475)	52	6118	6	0	0	2,55
phprAeXmK (4535)	42	299285	2	0	1	0,94

phpSZJq5T (1514)	1088	360	10	0	1	4,70
phptd5jYj (1501)	37	5100	2	0	1	2,64
phpTJRsqa (40498)	257	1593	10	0	0	0,32
phpvcoG8S (1169)	12	4898	7	0	9	0,52
phpVeNa5j (1497)	8	539383	2	0	1	0,98
phpvtdNPU (1079)	25	5456	4	0	0	4,25
phpWfYmlu (1496)	21	7400	2	0	9	0,79
phpxijhaP (1507)	22278	95	5	0	0	0,96
phpYLeydd (4538)	21	7400	2	0	0	3,32
phpZrCzJR (40900)	33	9873	5	0	0	1,22
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 16: 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.

A.2. Framework and Source Code

Table 17 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

Table 17: Source code repositories for all used AutoML frameworks

A.3. Cut-off time Budget

We tested the cutt-off timeouts of 4 and 8 hours on 14 randomly selected datasets. Table 18 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.

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

A.4. General Performance Evaluation

22,23,24 show the average performance of all frameworks for time budgets 10, 30, and 60 minutes compared the average performance of the baseline. 5(b),26,27,28 show the AutoML framwworks' 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.

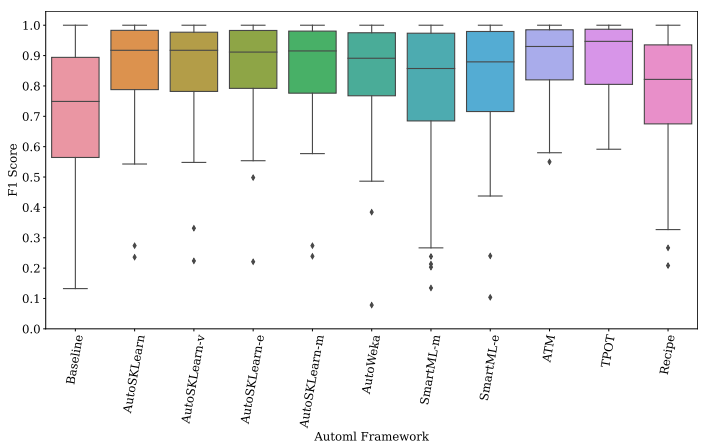


Figure 22: Average performance of all frameworks (10 Min) compared to the baseline.

A.5. Impact of Meta Learning

Table 19 lists a total of 42 meta-features including simple, information-theoretic and statistical meta-features.

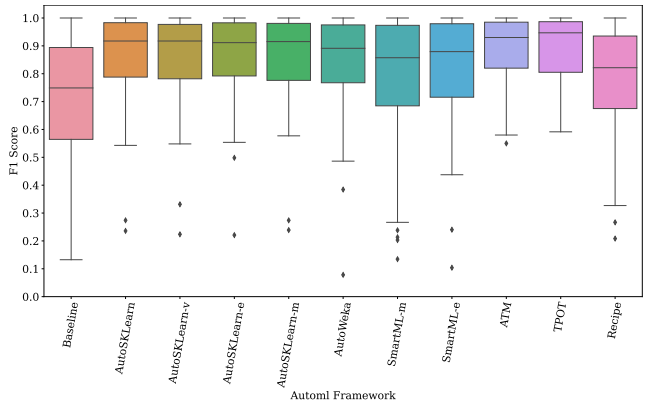


Figure 23: Average performance of all frameworks (30 Min) compared to the baseline.

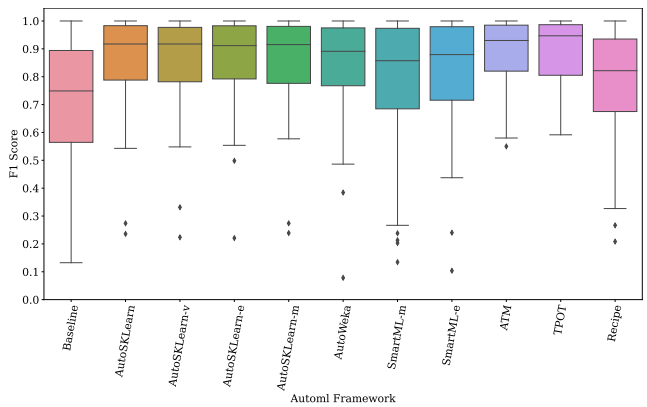


Figure 24: Average performance of all frameworks (60 Min) compared to the baseline.

A.6. Impact of Ensembling

Figures 29 and 30 shows the performance difference between the ensembling version and the vanilla/base version of `AutoSKlearn` and `SmartML`, respectively over 10, 30, 60

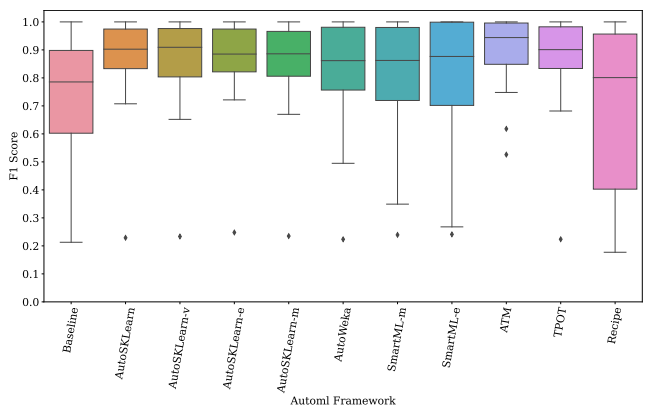


Figure 25: Performance of the final pipeline for datasets with large number of features and small number of instances.

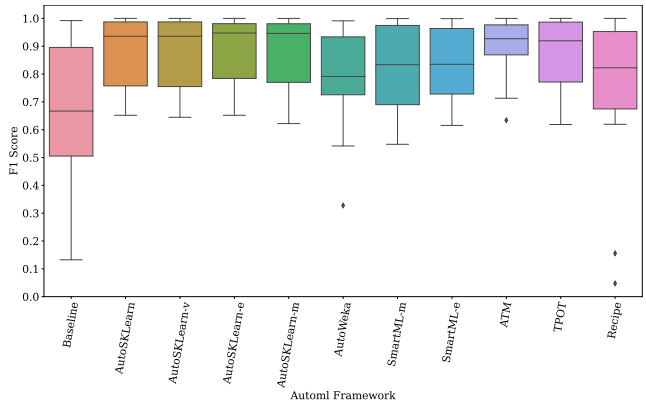


Figure 26: Performance of the final pipeline for datasets with large number of features and large number of instances.

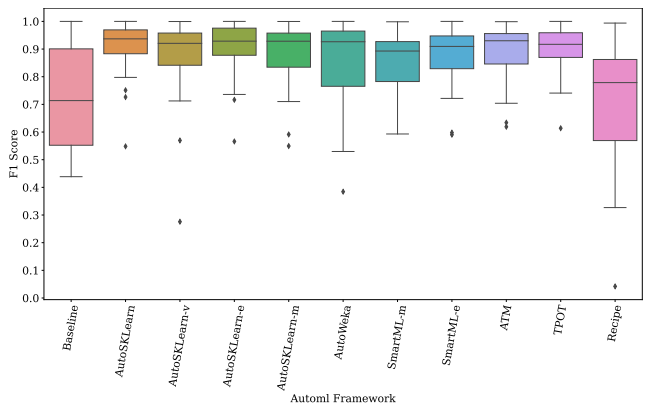


Figure 27: Performance of the final pipeline for datasets with small number of features and small number of instances.

and 240 minutes time budgets.

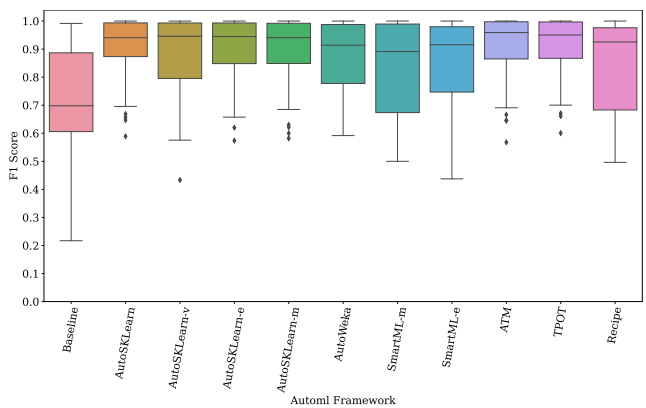
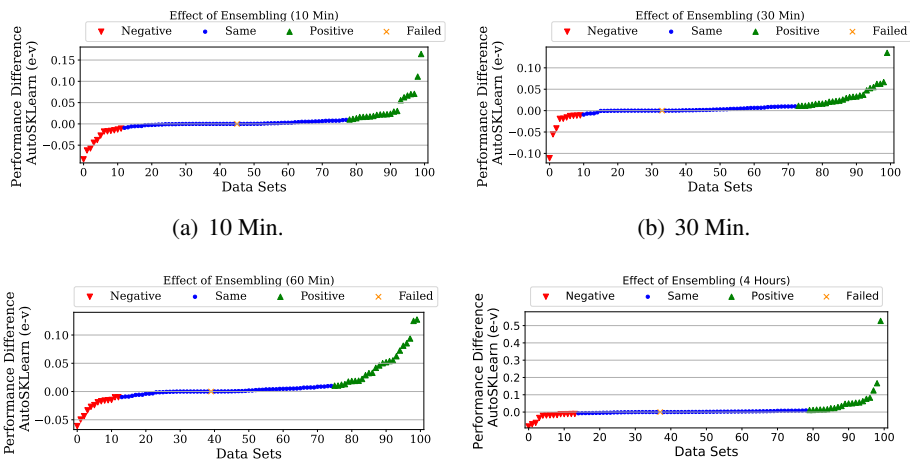


Figure 28: Performance of the final pipeline for datasets with small number of features and large number of instances.

Name	Formula	Rationale	Additional Variants
Nr instances	n	Speed, Scalability [76]	p/n, $log(n)$
Nr features	p	Curse of dimensionality [76]	$log(p)$, Nr/μ/$\sigma(\pi_i)$
Nr classes	c	Complexity, imbalance [76]	min/max/$\sigma(\frac{c}{n})$
Nr missing values	m	Imputation effects [50]	
Skewness	$\frac{E(X-\mu_X)^3}{\sigma_X^3}$	Feature normality [76]	min,max,μ,σ,q_1,q_3
Kurtosis	$\frac{E(X-\mu_X)^4}{\sigma_X^4}$	Feature normality [76]	min,max,μ,σ,q_1,q_3
Correlation	$\rho_{X_1X_2}$	Feature interdependence [76]	min,max,μ,σ
Class entropy	$H(\mathtt{C})$	Class imbalance [76]	$H(\mathtt{C})/\mu(MI(\mathtt{C},\mathtt{X}))$
Norm. entropy	$\frac{H(\mathtt{X})}{log_2 n}$	Feature informativeness [13]	min,max,μ,σ
Mutual inform.	$MI(\mathtt{C},\mathtt{X})$	Feature importance [76]	min,max,μ,σ
Noise-signal ratio	$\frac{\overline{H(X)}-\overline{MI(C,X)}}{\overline{MI(C,X)}}$	Noisiness of data [76]	

Table 19: Overview of the used meta-features. Groups from top to bottom: simple, statistical, and information-theoretic. Continuous features X and target Y have mean μ_X, stdev σ_X, variance σ_X^2. Categorical features $\mathtt{X}$ and class $\mathtt{C}$ have categorical values π_i, conditional probabilities $\pi_{i|j}$, joint probabilities $\pi_{i,j}$, marginal probabilities $\pi_{i+} = \sum_j \pi_{ij}$, entropy $H(\mathtt{X}) = -\sum_i \pi_{i+} log_2(\pi_{i+})$. [101]



(a) 10 Min.

(b) 30 Min.

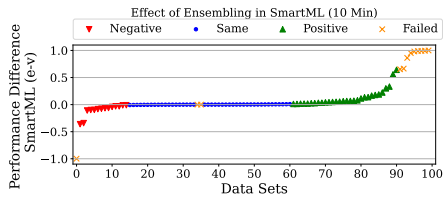
(c) 60 Min.

(d) 240 Min.

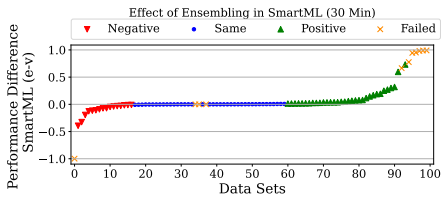
Figure 29: The performance difference between the `AutoSKlearn-e` and `AutoSKlearn-v` over different time budgets. Green markers represent better performance with `AutoSKlearn-e`, blue markers represent comparable performance with a difference less than 1%, red markers represent better performance with `AutoSKlearn-v`, and yellow markers represent failed runs on both versions.

A.7. Impact of time budget

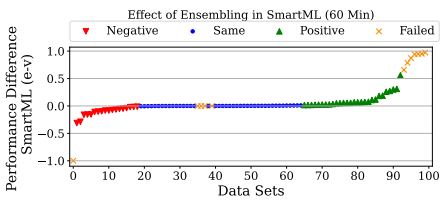
31,32,33,34,35,36,37,38,39,40 show the impact of increasing the time budget on the performance of the all the AutoML frameworks considered in this work.



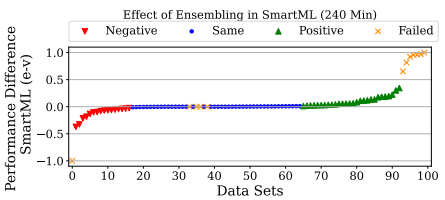
(a) 10 Min.



(b) 30 Min.

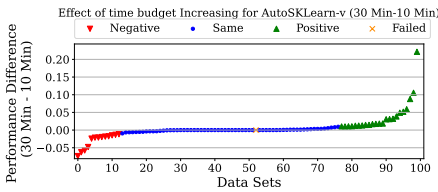


(c) 60 Min.

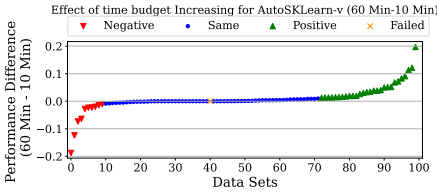


(d) 240 Min.

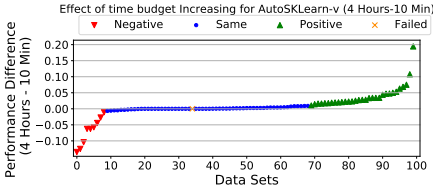
Figure 30: The performance difference between the `SmartML-m` and `SmartML-e`. Green markers represent better performance with `SmartML-e`, blue markers represent comparable performance with a difference less than 1%, red markers represent better performance with `SmartML`, yellow markers on the right 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`.



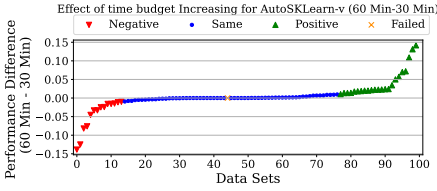
(a) 10-30 Min.



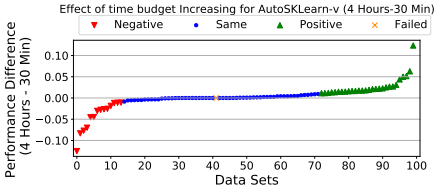
(b) 10-60 Min.



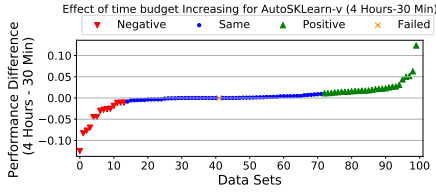
(c) 10-240 Min.



(d) 30-60 Min.

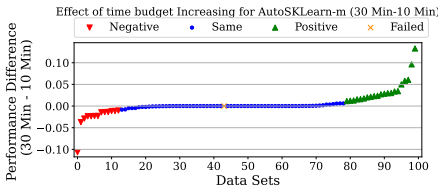


(e) 30-240 Min.

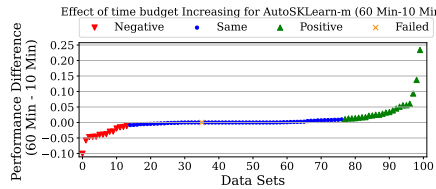


(f) 60-240 Min.

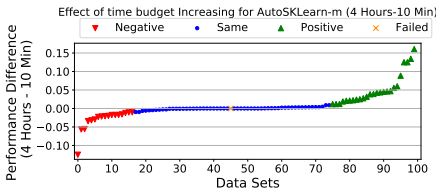
Figure 31: The impact of increasing the time budget on `AutoSKlearn-v` performance from x to y minutes (x-y). Green markers represent better performance with y time budget, blue markers means that the difference between x and y is < 1. Red markers represent better performance on x time budget.



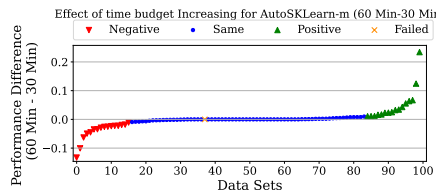
(a) 10-30 Min.



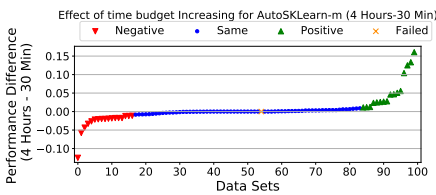
(b) 10-60 Min.



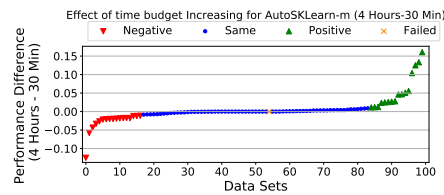
(c) 10-240 Min.



(d) 30-60 Min.

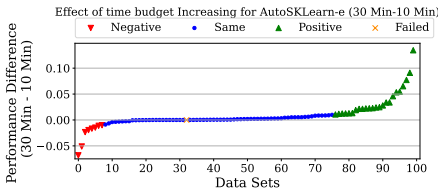


(e) 30-240 Min.

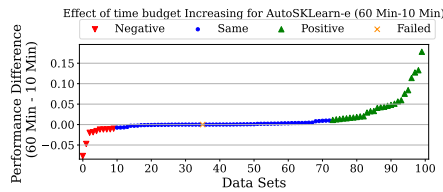


(f) 60-240 Min.

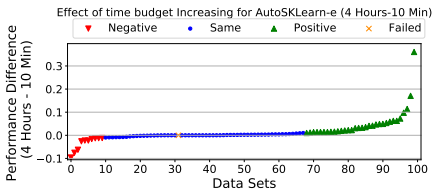
Figure 32: The impact of increasing the time budget on `AutoSKlearn-m` performance from x to y minutes (x-y). Green markers represent better performance with y time budget, blue markers means that the difference between x and y is < 1. Red markers represent better performance on x time budget.



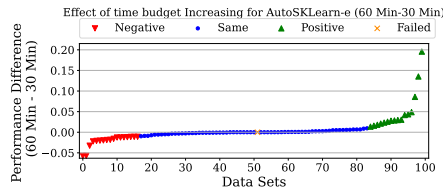
(a) 10-30 Min.



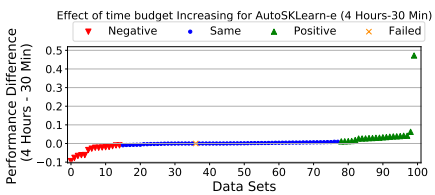
(b) 10-60 Min.



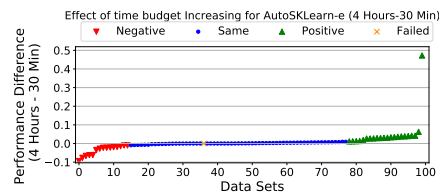
(c) 10-240 Min.



(d) 30-60 Min.

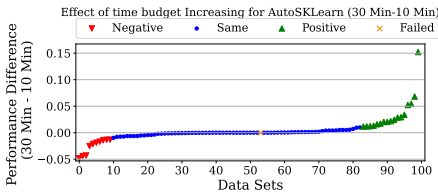


(e) 30-240 Min.

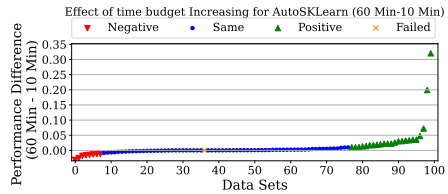


(f) 60-240 Min.

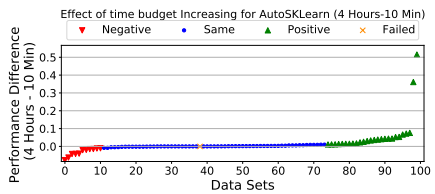
Figure 33: The impact of increasing the time budget on `AutoSKlearn-e` performance from x to y minutes (x-y). Green markers represent better performance with y time budget, blue markers means that the difference between x and y is < 1. Red markers represent better performance on x time budget.



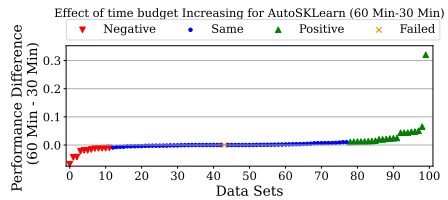
(a) 10-30 Min.



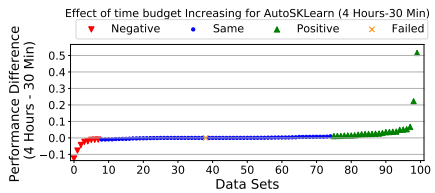
(b) 10-60 Min.



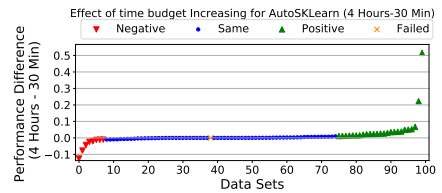
(c) 10-240 Min.



(d) 30-60 Min.

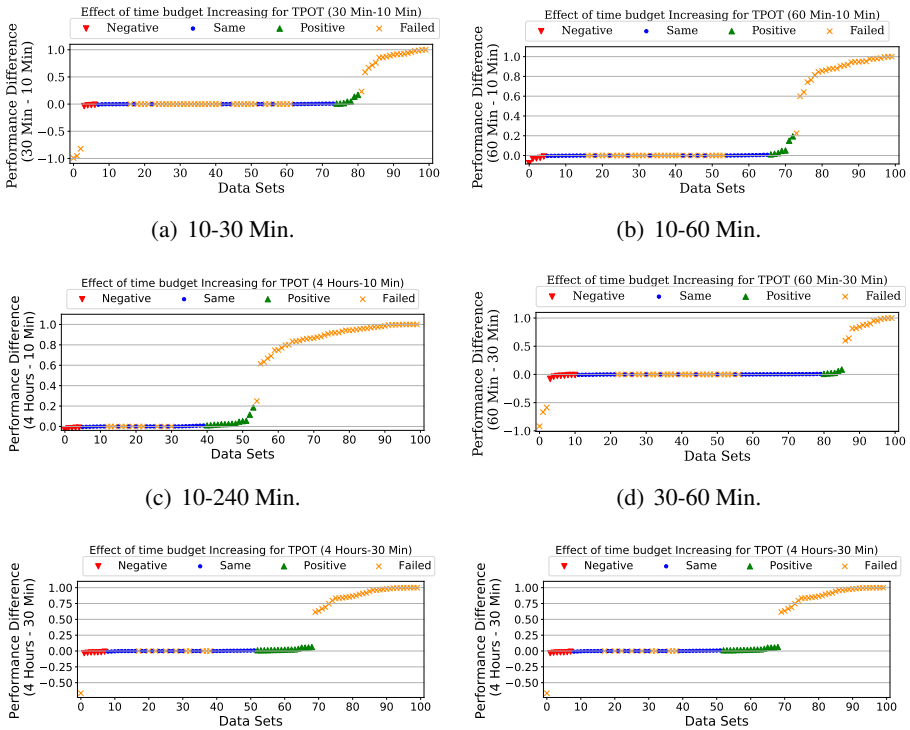


(e) 30-240 Min.



(f) 60-240 Min.

Figure 34: 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 means that the difference between x and y is < 1. Red markers represent better performance on x time budget.



(a) 10-30 Min.

(b) 10-60 Min.

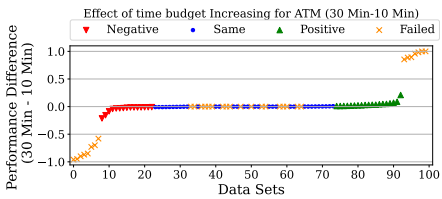
(c) 10-240 Min.

(d) 30-60 Min.

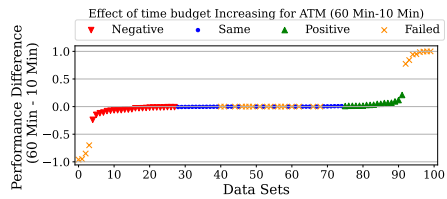
(e) 30-240 Min.

(f) 60-240 Min.

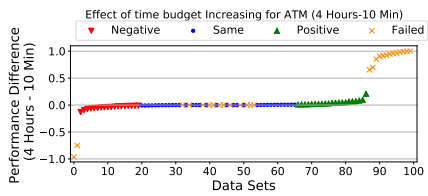
Figure 35: The impact of increasing the time budget on `TPOT` performance from x to y minutes (x-y). Green markers represent better performance with y time budget, blue markers means that the difference between x and y is < 1. Red markers represent better performance on x time budget.



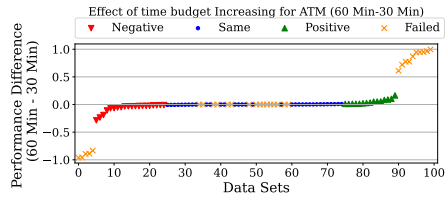
(a) 10-30 Min.



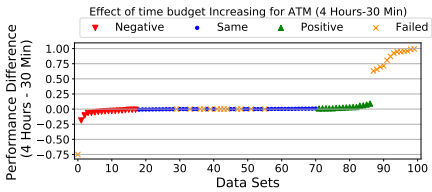
(b) 10-60 Min.



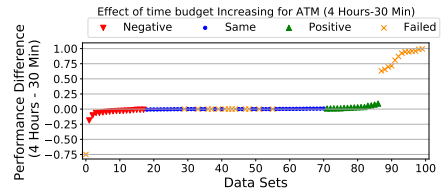
(c) 10-240 Min.



(d) 30-60 Min.

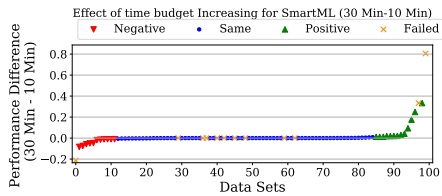


(e) 30-240 Min.

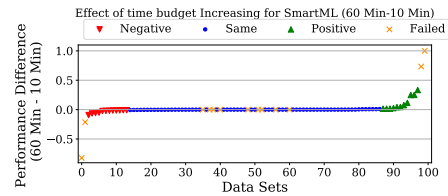


(f) 60-240 Min.

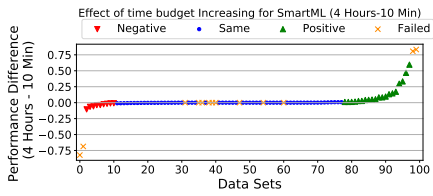
Figure 36: The impact of increasing the time budget on ATM performance from x to y minutes (x-y). Green markers represent better performance with y time budget, blue markers means that the difference between x and y is < 1. Red markers represent better performance on x time budget.



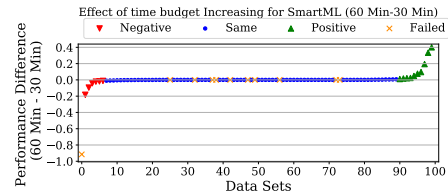
(a) 10-30 Min.



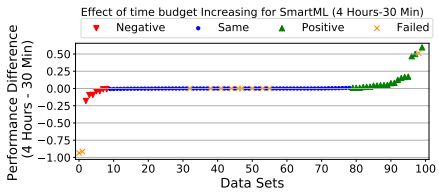
(b) 10-60 Min.



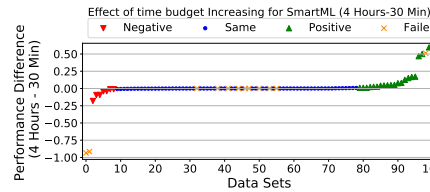
(c) 10-240 Min.



(d) 30-60 Min.

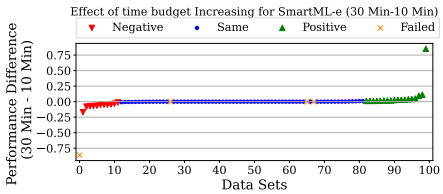


(e) 30-240 Min.

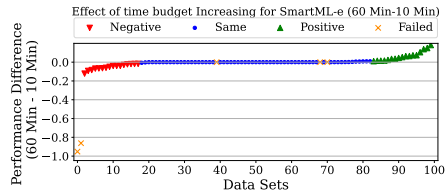


(f) 60-240 Min.

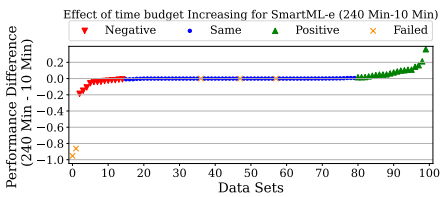
Figure 37: The impact of increasing the time budget on `SmartML-m` performance from x to y minutes (x-y). Green markers represent better performance with y time budget, blue markers means that the difference between x and y is < 1. Red markers represent better performance on x time budget.



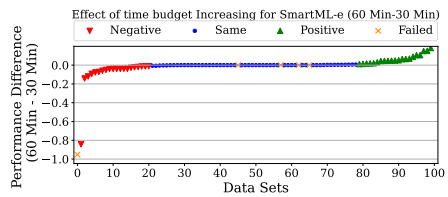
(a) 10-30 Min.



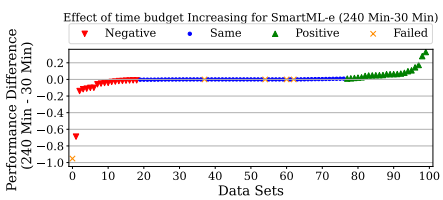
(b) 10-60 Min.



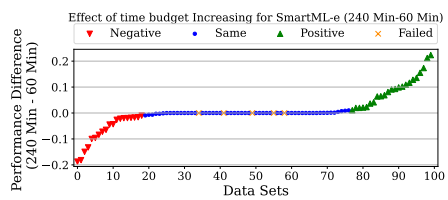
(c) 10-240 Min.



(d) 30-60 Min.

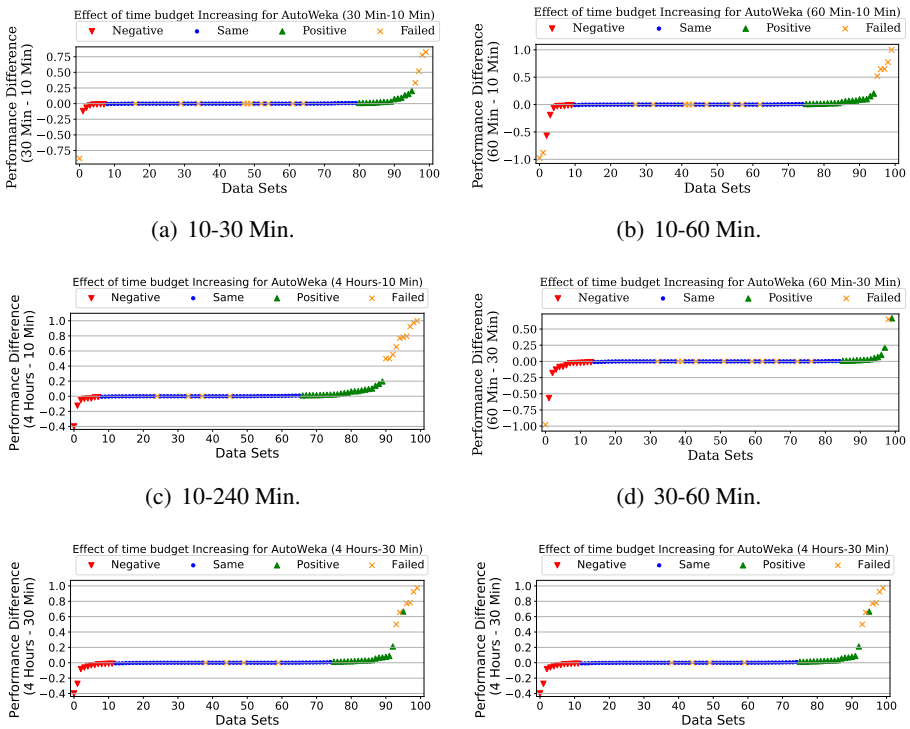


(e) 30-240 Min.



(f) 60-240 Min.

Figure 38: 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.



(a) 10-30 Min.

(b) 10-60 Min.

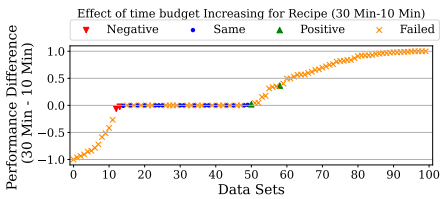
(c) 10-240 Min.

(d) 30-60 Min.

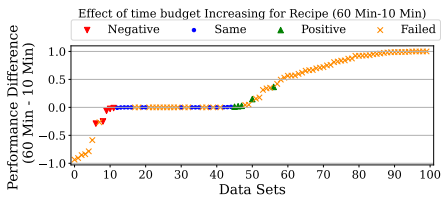
(e) 30-240 Min.

(f) 60-240 Min.

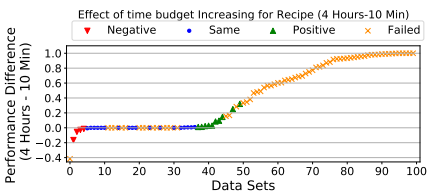
Figure 39: The impact of increasing the time budget on `AutoWeka` performance from x to y minutes (x-y). Green markers represent better performance with y time budget, blue markers means that the difference between x and y is < 1. Red markers represent better performance on x time budget.



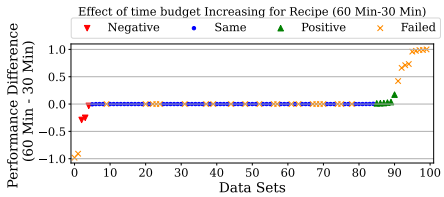
(a) 10-30 Min.



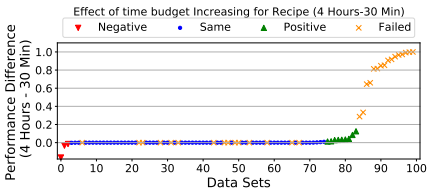
(b) 10-60 Min.



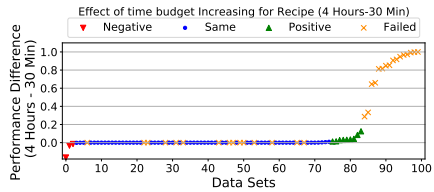
(c) 10-240 Min.



(d) 30-60 Min.



(e) 30-240 Min.



(f) 60-240 Min.

Figure 40: 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.

ACKNOWLEDGEMENTS

First and foremost, I extend my deepest gratitude to **Allah** for His unwavering guidance, wisdom, and peace, which have been the cornerstone of my journey through this research. With His generous support, I have navigated the challenges of my work and life. I also send my profound respect and blessings upon His **Prophet Muhammad**, whose life and teachings continually inspire me to pursue excellence in all aspects of my life.

I am profoundly grateful to Professor **Radwa** for her invaluable expertise, guidance, and support throughout my research journey. My heartfelt appreciation also extends to the late Professor **Sakr**, who was not only my supervisor during the first year of my PhD but also a significant influence in my early academic development. His hard work and ambition have left a lasting impact on my work and approach to research.

I am also immensely thankful to the University Institute of Computer Science, especially to Professors **Jaak**, **Heisi**, **Meelis**, **Raul**, and **Marlon** for their efforts enriching scientific environment that has been essential for my studies.

Special thanks go to my colleagues, **Shota**, **Javad**, **Kristo**, **Nashan**, and **Riccardo**, and Prof. **Awad** for their insightful discussions and steadfast moral support, and to my friends **Ali**, **Rasheed**, **Ragab**, **Salah**, **Gamal**, **Dalal**, **Feras**, **Yousef**, **H Arsalan**, **Mahmoud**, **Maher**, **Zain**, **Naveed**, **Rauno**, and **Ildar** for bringing joy and companionship to my life in Tartu. More importantly, to **Aqeel**, the brother I made during my stay in Tartu. My words can not thank you enough for your help and support.

Most importantly, I owe a debt of gratitude to my family—my parents, **Abduldjaleel** and **Karema**, and brothers, **Anan** and **Salem**. Your constant love, encouragement, and belief in me have been my greatest source of motivation and strength. This achievement is as much yours as it is mine.

SISUKOKKUVÕTE

Võimestades masinõppekonveiereid automatiseeritult tunnuste loomisega

Oleme tunnistajaks tohutule huvile tehisintellekti rakenduste vastu valitsustes, tööstustes ja teadusringkondades, mille aastast mahtu hinnatakse 12,5 miljardile USA dollarile. Selle huvi ajendiks on masinõppe (ML) ja süvaõppe (DL) tehnikate tulek ja kasvav populaarsus. Erinevatest allikatest genereeritud andmete, töötlemisvõimaluste ja ML-algoritmide suurenemine avas tee ML-i kasutuselevõtuks paljudes rakendustes. Selline olukord soodustab üha enam võimalikku *andmeteaduse kriisi*, mis on sarnane tarkvarakriisiga [28], kuna on ülioluline vajadus omada üha rohkem tugevate teadmiste ja heade kogemustega andmeteadlasi, et nad saaksid hoida end kursis igapäevaselt toodetud tohutute andmehulkade rakendamisega. Seega oleme tunnistajaks kasvavale huvile ML-konveierite ehitamise protsessi automatiseerimise vastu, kus inimese olulist protsessis saaks märkimisväärselt vähendada. Automatiseeritud masinõppe (AutoML) valdkonna uuringute eesmärk on tõhusate algoritmidega automatiseerimise abil vähendada nii arvutuskulusid kui ka inimteadmisi, mis on vajalikud ML-i töövoogude arendamiseks. Eelkõige võimaldavad AutoML-i tehnikad domeeniekspertidel ja mittetehnilistel kasutajatel ML-tehnikaid laialdaselt kasutada.

ML-i rakendamine reaalsete probleemide lahendamiseks on mitmeetapiline ja väga iteratiivne protsess. Selle eesmärk on automaatselt toota optimaalne ML-konveier (töövoog), mis maksimeerib prognoositava jõudluse andmestiku valideerimiskomplekti suhtes fikseeritud arvutuseelarve piires. Hästi toimiva masinõppekonveieri loomine on aga iteratiivne ja keeruline protsess, mis nõuab kindlat arusaamist erinevatest tehnikatest, mida saab kasutada masinõppe töövoo igas komponendis. Lisaks on tunnuste loomine (Feature Engineering, FE) üks kõige aeganõudvamaid, kuid kõige vähem automatiseeritud samme masinõppe töövoogude ehitamisel. Toorandmetest asjakohaste käsitsi loodud tunnuste avastamiseks on vaja domeeni sügavat mõistmist ja andmete uurimist.

Praktikas on AutoML-i probleemi lahendamise eesmärk valida ja häälestada määratletud otsinguruumist ML-algoritm, et saavutada (peaaegu) optimaalne jõudlus kasutaja määratud hindamismõõdiku (nt täpsus, tundlikkus, spetsiifilisus, F1-skoor) osas, kasutaja määratud otsinguprotsessi eelarve piires. Lisaks arvestavad erinevad AutoML-i raamistikud erinevaid disainiotsuseid. Näiteks `SmartML` [72] võtab automatiseeritud otsinguprotsessi toimivuse parandamiseks kasutusele *metaõppe*-põhise mehhanismi, alustades kõige lootustandvamatest klassifikaatoritest, mis varem sarnaste andmekogumitega hästi toimisid. `AutoSKlearn` [27] kasutab optimeerimisprotsessi käigus võimalust võtta ennustuste kaalutud keskmine, mis koosneb kõige paremini treenitud mudelitest. Automaatselt häälestatud mudelid (`ATM`) [96] piirab vaikeotsinguruumi ainult kolme klassifikaatoriga: otsustuspuu, K-lähimate naabrite ja logistilise regressiooniga. Sellegipoolest puudub selge arusaam erinevate AutoML-i raamistike erinevate disainiotsuste mõjust jõudlusele. Selles töös püüame vastata neljale järgmisele küsimusele:

1. Milline on ajaeelarve mõju erinevate AutoML-i raamistike jõudlusele? Kas AutoML-i raamistikud suudavad suurema ajaeelarvega tagada järjepideva jõudluse parandamise?

2. Millist mõju avaldab AutoML-i raamistiku otsinguruumi suurus jõudlusele? Kuidas mõjutab otsinguruumi piiramine etteantud portfelliga ennustavat jõudlust?

3. Kas metaõpe annab alati järjepideva jõudluse paranemise erinevate ajaeelarvetega? Kas andmekogumite omaduste ja AutoML-i raamistiku metaõppeversiooni kasutamisest põhjustatud paranemise vahel on seos?

4. Kas ansambli ehitamine annab erinevate ajaeelarvete puhul paremaid tulemusi kui üksikud õppijad? Kas andmestike omaduste ja AutoML-i raamistiku komplekteeriva versiooni kasutamisest põhjustatud paranemise vahel on seos?

Nendele küsimustele vastamiseks esitame kuue populaarse AutoML-i raamistiku, nimelt AutoWeka, AutoSKlearni, TPOT, retsepti, ATM-i ja SmartML-i jõudlusnäitajate põhjaliku hinnangu ja võrdluse 100 andmekogumis väljakujunenud AutoML-i etalonkomplektidest. Meie eksperimentaalne hindamine võtab selle võrdlemisel arvesse erinevaid aspekte, sealhulgas mitmete disainiotsuste tulemuslikkuse mõju nagu näiteks ajaeelarve, otsinguruumi suurus, metaõpe ja ansambli ehitus.

FE on masinõppe töövoos üks väärtuslikumaid faase. Kuid meie teadmiste kohaselt lahendavad praegused avatud lähtekoodiga AutoML raamistikud seda kõrgel abstraktsioonitasemel. Need raamistikud [98, 26, 82] toodavad põhifunktsioonide eelprotsessoreid, rakendades selliseid tehnikaid nagu peakomponentide analüüs (PCA) ja lineaarne diskriminantanalüüs (LDA), ilma tuletatud funktsioone loomata.

Teisest küljest järgivad enamik nüüdisaegseid automaatseid FE tehnikaid, erinevalt AutoML tööriistadest, kasulike tunnuste saamiseks ühte kolmest peamisest lähenemisviisist. Esimese lähenemisviisi puhul, milleks on *loo-ja-vali*, luuakse ammendav tunnuste kogum ja valik tehakse kogu tunnuste hulga pealt [73, 87, 23, 53]. Teise lähenemisviisi korral kasutatakse FE probleemi lahendamiseks stiimulõppe tehnikat, võttes kasutusele Monte-Carlo puuotsingu [31], Q-learning [55] või sügava Q-õppe [106]. Kolmanda lähenemisviisi korral koolitatakse metaõppel põhinevat algoritmi eelnevalt üle andmekogumite kogumi, enne kui seda kasutatakse algoritmi jooksutamise faasis, et soovitada sisendandmestiku paljulubavaid tunnuseid [52, 55, 79].

Hindame empiiriliselt automatiseeritud funktsioonide ekstraheerimise tööriista (AutoFeat) integreerimise mõju kahte automatiseeritud masinõpperaamistikku, nimelt Auto-Sklearni ja TPOT-i ning nende ennustavale jõudlusele. Lisaks arutame AutoFeati piiranguid millega tuleb tegeleda, et parandada automaatsete masinõpperaamistike ennustavat jõudlust päriselu andmekogumites.

Arvestades AutoFEati ja muude automatiseeritud FE-tööriistade piiranguid, sealhulgas praktilise skaleeritavuse ja tõhususe puudumist, tutvustame BigFeati – skaleeritavat ja tõlgendatavat raamistikku, mis muudab masinõppe konveieri kriitilised etapid sujuvamaks: tunnuste loomine, mudelite valik ja hüperparameetrite häälestamine. BigFeat pakub kahte täitmisvalikut: eraldiseisva FE raamistikuna, mida tähistatakse kui BigFeat-FE, ja kui AutoML-i raamistikku, mida nimetatakse BigFeat-AutoML-iks. BigFeat-FE optimeerib sisendtunnuste kvaliteeti, eesmärgiga maksimeerida ennustusvõimet. See kasutab dünaamilist tunnuste loomise ja valiku mehhanismi, mis süstemaatiliselt loob komplekti väljendusrikastest tunnustest. Need tunnused mitte ainult ei paranda ennustusvõimet, vaid seavad prioriteediks ka tõlgendatavuse.

BigFeat-FE kasutab optimeerimisprotsessi soe-käivituseks ka metaõppe tehnikat, mille tulemuseks on oluline üldine jõudluse kasv. BigFeat-AutoML, mis on kohandatud algoritmi valimiseks ja hüperparameetrite häälestamiseks, kasutab tõlgendatavate mudelite ruumis juhuslikku otsingumeetodit. Viisime läbi ulatuslikud katsed ja tulemused näitavad, et BigFeat-FE ületab järjekindlalt tipptasemel automatiseeritud funktsioonide projekteerimisraamistikke, nagu AutoFeat ja SAFE, paljudes andmekogumites, saavutades

keskmise jõudluse paranemise 8.65% võrrelduna raamistikuga AutoFeat ja 4.71% võrrelduna SAFE-ga.

Lisaks näitab BigFeat-AutoML võrreldes TPOT-i ja AutoSKlearniga olulist ennustusvõime paranemist, keskmiselt vastavalt 0.74% võrreldes TPOT-ga ja 2.25% võrreldes AutoSKlearniga. Samuti kinnitab BigFeati skaleeritavust selle lineaarne ajaline ja ruumiline keerukus ning kiire täitmisaeg, olles keskmiselt 20 korda kiirem kui AutoFeat ja 14 korda kiirem kui SAFE.

Järeldus

Selles väitekirjas oleme läbi viinud põhjaliku AutoML-i raamistike uurimise, pöörates erilist tähelepanu funktsioonide projekteerimisele ja selle mõjule ennustavale jõudlusele. Meie uuringud keskendusid erinevate raamistike, sealhulgas Auto-SKlearn, AutoSKlearn ja Recipe, hindamisele ja võrdlemisele.

Meie uurimistöö peamised tulemused hõlmavad järgmist:

- **Jõudluse analüüs:** Auto-SKlearn-e ja AutoSKlearn saavutasid parimad tulemused, samas kui Recipe jäi maha. Metaõppe strateegiate tõhusus vähenes pikema ajaeelarvega, kuid kombineerimistehnikad parandasid järjekindlalt jõudlust.
- **Tunnuste loomise kriitiline roll:** Uuring rõhutas tunnuste loomise ülimat tähtsust ennustusvõime mõjutamisel. Eeltöötluse valikute erinevused, eriti andmekogumite puhul, milles esinemisjuhte on vähem kui tunnuseid, mõjutasid tulemusi märkimisväärselt. See leid viitab võimalusele täiustada AutoML-i raamistikke tugevate tunnuste loomise võimalustega.
- **Raamistiku piirangud ja ülesobitus:** Paljud raamistikud ületasid oma määratud ajaeelarveid, mis tõi kaasa töö lõpetamise. Uuring rõhutas ka ülesobitamise ohtu, mis on seotud ulatuslike otsinguprotsessidega AutoML-i tööriistades, nõudes tõhusamaid mehhanisme selle väljakutse leevendamiseks.
- **Tulevikujuhised:** Uuring avab uusi võimalusi AutoML-i täiustamiseks, eriti tunnuste loomise valdkonnas. Uudsete tehnikate väljatöötamine, et vältida ülesobitust ja parandada AutoML-i töövoogude üldist jõudlust täiustatud tunnuste loomise abil, on paljutõotav suund tulevaseks tööks.

See väitekiri aitab kaasa AutoML-i raamistike nüansside sügavamale mõistmisele, rõhutades olulist rolli tunnuste loomisel ja loob aluse tulevastele edusammudele automatiseeritud masinõppe valdkonnas.

CURRICULUM VITAE

Personal data

Name:	Hassan Abdulgaleel Hassan Salim Eldeeb
Date of Birth:	25.08.1990
Citizenship:	Egypt
Current Position:	Junior Research Fellow of Data Science
Contact:	ha.eldeeb@gmail.com

Education

2019–2024	University of Tartu, Estonia, Ph.D. in Computer Science
2013–2018	University of Tanta, Egypt, MSc. in Computer Engineering
2008–2013	University of Tanta, Egypt, BEng. in Computer Engineering

Employment

2019–2024	Junior Research Fellow of Data Science, University of Tartu
2021–2022	Sr. Data/ML Scientist, MindTitan, Estonia
2015–2029	Teaching Assistant, Univerity of Tanta, Egypt
2017–2019	R&D Engineer, Wireless Research Center (Uber Technologies Inc Project), Egypt
2016–2017	R&D Engineer, InnoTech, Egypt
2011–2012	Co-founder and Marketing Manage, RecycloBekia, Egypt

Teaching

2021–2023	Explainable Automated Machine Learning: Teaching and Co-creating the materials
2020–2022	Data Engineering: Teaching and Co-creating the materials
2020–2022	Big Data Management: Teaching
2016–2019	Machine Learning: Teaching and Co-creating the materials
2017–2019	Neural Networks: Teaching and Co-creating the materials

Scientific work

Main fields of interest:

- Automated Machine Learning
- Explainable Machine Learning
- Automated Algorithm Design
- Computational Optimization

ELULOOKIRJELDUS

Isikuandmed

Nimi:	Hassan Abdulgaleel Hassan Salim Eldeeb
Sünnikuupäev:	25.08.1990
Kodakondsus:	Egiptus
Praegune ametikoht:	Andmeteaduse nooremteadur
Kontakt:	ha.eldeeb@gmail.com

Haridus

2019–2024	Tartu Ülikool, Eesti, Ph.D. arvutiteaduses
2013–2018	Tanta Ülikool, Egiptus, MSc arvutitehnika alal
2008–2013	Tanta Ülikool, Egiptus, BEng arvutitehnika alal

Teenistuskäik

2019–2024	Andmeteaduse nooremteadur, Tartu Ülikool
2021–2022	Vanem andme-/ML teadlane, MindTitan, Eesti
2015–2029	Õppeassistent, Tanta Ülikool, Egiptus
2017–2019	insener, Wireless Research Center (Uber Technologies Inc projekt), Egiptus
2016–2017	R&D insener, InnoTech, Egiptus
2011–2012	Kaasasutaja ja turundusjuht, RecycloBekia, Egiptus

Õpetamine

2021–2023	Seletatav automatiseeritud masinõpe: õpetamine ja materjali kaasloomine
2020–2022	Andmetehnika: õpetamine ja materjali kaasloomine
2020–2022	Suurandmete haldamine: õpetamine
2016–2019	Masinõpe: õpetamine ja materjali kaasloomine
2017–2019	Neurovõrgud: õpetamine ja materjali kaasloomine

Teadustöö

Peamised huvivaldkonnad:

- Automaatne masinõpe
- Seletatav masinõpe
- Automaatne algoritmide disain
- Arvutuslik optimeerimine

DISSERTATIONES INFORMATICAE PREVIOUSLY PUBLISHED IN DISSERTATIONES MATHEMATICAE UNIVERSITATIS TARTUENSIS

19. **Helger Lipmaa.** Secure and efficient time-stamping systems. Tartu, 1999, 56 p.
22. **Kaili Müürisep.** Eesti keele arvutigrammatika: süntaks. Tartu, 2000, 107 lk.
23. **Varmo Vene.** Categorical programming with inductive and coinductive types. Tartu, 2000, 116 p.
24. **Olga Sokratova.** Ω-rings, their flat and projective acts with some applications. Tartu, 2000, 120 p.
27. **Tiina Puolakainen.** Eesti keele arvutigrammatika: morfoloogiline ühestamine. Tartu, 2001, 138 lk.
29. **Jan Villemson.** Size-efficient interval time stamps. Tartu, 2002, 82 p.
45. **Kristo Heero.** Path planning and learning strategies for mobile robots in dynamic partially unknown environments. Tartu 2006, 123 p.
49. **Härmel Nestra.** Iteratively defined transfinite trace semantics and program slicing with respect to them. Tartu 2006, 116 p.
53. **Marina Issakova.** Solving of linear equations, linear inequalities and systems of linear equations in interactive learning environment. Tartu 2007, 170 p.
55. **Kaarel Kaljurand.** Attempto controlled English as a Semantic Web language. Tartu 2007, 162 p.
56. **Mart Anton.** Mechanical modeling of IPMC actuators at large deformations. Tartu 2008, 123 p.
59. **Reimo Palm.** Numerical Comparison of Regularization Algorithms for Solving Ill-Posed Problems. Tartu 2010, 105 p.
61. **Jüri Reimand.** Functional analysis of gene lists, networks and regulatory systems. Tartu 2010, 153 p.
62. **Ahti Peder.** Superpositional Graphs and Finding the Description of Structure by Counting Method. Tartu 2010, 87 p.
64. **Vesal Vojdani.** Static Data Race Analysis of Heap-Manipulating C Programs. Tartu 2010, 137 p.
66. **Mark Fišel.** Optimizing Statistical Machine Translation via Input Modification. Tartu 2011, 104 p.
67. **Margus Niitsoo**. Black-box Oracle Separation Techniques with Applications in Time-stamping. Tartu 2011, 174 p.
71. **Siim Karus.** Maintainability of XML Transformations. Tartu 2011, 142 p.
72. **Margus Treumuth.** A Framework for Asynchronous Dialogue Systems: Concepts, Issues and Design Aspects. Tartu 2011, 95 p.
73. **Dmitri Lepp.** Solving simplification problems in the domain of exponents, monomials and polynomials in interactive learning environment T-algebra. Tartu 2011, 202 p.

74. **Meelis Kull.** Statistical enrichment analysis in algorithms for studying gene regulation. Tartu 2011, 151 p.
77. **Bingsheng Zhang.** Efficient cryptographic protocols for secure and private remote databases. Tartu 2011, 206 p.
78. **Reina Uba.** Merging business process models. Tartu 2011, 166 p.
79. **Uuno Puus.** Structural performance as a success factor in software development projects – Estonian experience. Tartu 2012, 106 p.
81. **Georg Singer.** Web search engines and complex information needs. Tartu 2012, 218 p.
83. **Dan Bogdanov.** Sharemind: programmable secure computations with practical applications. Tartu 2013, 191 p.
84. **Jevgeni Kabanov.** Towards a more productive Java EE ecosystem. Tartu 2013, 151 p.
87. **Margus Freudenthal.** Simpl: A toolkit for Domain-Specific Language development in enterprise information systems. Tartu, 2013, 151 p.
90. **Raivo Kolde.** Methods for re-using public gene expression data. Tartu, 2014, 121 p.
91. **Vladimir Šor.** Statistical Approach for Memory Leak Detection in Java Applications. Tartu, 2014, 155 p.
92. **Naved Ahmed.** Deriving Security Requirements from Business Process Models. Tartu, 2014, 171 p.
94. **Liina Kamm.** Privacy-preserving statistical analysis using secure multi-party computation. Tartu, 2015, 201 p.
100. **Abel Armas Cervantes.** Diagnosing Behavioral Differences between Business Process Models. Tartu, 2015, 193 p.
101. **Fredrik Milani.** On Sub-Processes, Process Variation and their Interplay: An Integrated Divide-and-Conquer Method for Modeling Business Processes with Variation. Tartu, 2015, 164 p.
102. **Huber Raul Flores Macario.** Service-Oriented and Evidence-aware Mobile Cloud Computing. Tartu, 2015, 163 p.
103. **Tauno Metsalu.** Statistical analysis of multivariate data in bioinformatics. Tartu, 2016, 197 p.
104. **Riivo Talviste.** Applying Secure Multi-party Computation in Practice. Tartu, 2016, 144 p.
108. **Siim Orasmaa.** Explorations of the Problem of Broad-coverage and General Domain Event Analysis: The Estonian Experience. Tartu, 2016, 186 p.
109. **Prastudy Mungkas Fauzi.** Efficient Non-interactive Zero-knowledge Protocols in the CRS Model. Tartu, 2017, 193 p.
110. **Pelle Jakovits.** Adapting Scientific Computing Algorithms to Distributed Computing Frameworks. Tartu, 2017, 168 p.
111. **Anna Leontjeva.** Using Generative Models to Combine Static and Sequential Features for Classification. Tartu, 2017, 167 p.
112. **Mozhgan Pourmoradnasseri.** Some Problems Related to Extensions of Polytopes. Tartu, 2017, 168 p.

113. **Jaak Randmets.** Programming Languages for Secure Multi-party Computation Application Development. Tartu, 2017, 172 p.
114. **Alisa Pankova.** Efficient Multiparty Computation Secure against Covert and Active Adversaries. Tartu, 2017, 316 p.
116. **Toomas Saarsen.** On the Structure and Use of Process Models and Their Interplay. Tartu, 2017, 123 p.
121. **Kristjan Korjus.** Analyzing EEG Data and Improving Data Partitioning for Machine Learning Algorithms. Tartu, 2017, 106 p.
122. **Eno Tõnisson.** Differences between Expected Answers and the Answers Offered by Computer Algebra Systems to School Mathematics Equations. Tartu, 2017, 195 p.

DISSERTATIONES INFORMATICAE UNIVERSITATIS TARTUENSIS

1. **Abdullah Makkeh**. Applications of Optimization in Some Complex Systems. Tartu 2018, 179 p.
2. **Riivo Kikas**. Analysis of Issue and Dependency Management in Open-Source Software Projects. Tartu 2018, 115 p.
3. **Ehsan Ebrahimi**. Post-Quantum Security in the Presence of Superposition Queries. Tartu 2018, 200 p.
4. **Ilya Verenich**. Explainable Predictive Monitoring of Temporal Measures of Business Processes. Tartu 2019, 151 p.
5. **Yauhen Yakimenka**. Failure Structures of Message-Passing Algorithms in Erasure Decoding and Compressed Sensing. Tartu 2019, 134 p.
6. **Irene Teinemaa**. Predictive and Prescriptive Monitoring of Business Process Outcomes. Tartu 2019, 196 p.
7. **Mohan Liyanage.** A Framework for Mobile Web of Things. Tartu 2019, 131 p.
8. **Toomas Krips.** Improving performance of secure real-number operations. Tartu 2019, 146 p.
9. **Vijayachitra Modhukur.** Profiling of DNA methylation patterns as biomarkers of human disease. Tartu 2019, 134 p.
10. **Elena Sügis.** Integration Methods for Heterogeneous Biological Data. Tartu 2019, 250 p.
11. **Tõnis Tasa.** Bioinformatics Approaches in Personalised Pharmacotherapy. Tartu 2019, 150 p.
12. **Sulev Reisberg.** Developing Computational Solutions for Personalized Medicine. Tartu 2019, 126 p.
13. **Huishi Yin.** Using a Kano-like Model to Facilitate Open Innovation in Requirements Engineering. Tartu 2019, 129 p.
14. **Faiz Ali Shah.** Extracting Information from App Reviews to Facilitate Software Development Activities. Tartu 2020, 149 p.
15. **Adriano Augusto**. Accurate and Efficient Discovery of Process Models from Event Logs. Tartu 2020, 194 p.
16. **Karim Baghery.** Reducing Trust and Improving Security in zk-SNARKs and Commitments. Tartu 2020, 245 p.
17. **Behzad Abdolmaleki.** On Succinct Non-Interactive Zero-Knowledge Protocols Under Weaker Trust Assumptions. Tartu 2020, 209 p.
18. **Janno Siim.** Non-Interactive Shuffle Arguments. Tartu 2020, 154 p.
19. **Ilya Kuzovkin.** Understanding Information Processing in Human Brain by Interpreting Machine Learning Models. Tartu 2020, 149 p.
20. **Orlenys López Pintado.** Collaborative Business Process Execution on the Blockchain: The Caterpillar System. Tartu 2020, 170 p.
21. **Ardi Tampuu.** Neural Networks for Analyzing Biological Data. Tartu 2020, 152 p.

22. **Madis Vasser.** Testing a Computational Theory of Brain Functioning with Virtual Reality. Tartu 2020, 106 p.
23. **Ljubov Jaanuska.** Haar Wavelet Method for Vibration Analysis of Beams and Parameter Quantification. Tartu 2021, 192 p.
24. **Arnis Parsovs.** Estonian Electronic Identity Card and its Security Challenges. Tartu 2021, 214 p.
25. **Kaido Lepik.** Inferring causality between transcriptome and complex traits. Tartu 2021, 224 p.
26. **Tauno Palts.** A Model for Assessing Computational Thinking Skills. Tartu 2021, 134 p.
27. **Liis Kolberg.** Developing and applying bioinformatics tools for gene expression data interpretation. Tartu 2021, 195 p.
28. **Dmytro Fishman.** Developing a data analysis pipeline for automated protein profiling in immunology. Tartu 2021, 155 p.
29. **Ivo Kubjas.** Algebraic Approaches to Problems Arising in Decentralized Systems. Tartu 2021, 120 p.
30. **Hina Anwar.** Towards Greener Software Engineering Using Software Analytics. Tartu 2021, 186 p.
31. **Veronika Plotnikova.** FIN-DM: A Data Mining Process for the Financial Services. Tartu 2021, 197 p.
32. **Manuel Camargo.** Automated Discovery of Business Process Simulation Models From Event Logs: A Hybrid Process Mining and Deep Learning Approach. Tartu 2021, 130 p.
33. **Volodymyr Leno.** Robotic Process Mining: Accelerating the Adoption of Robotic Process Automation. Tartu 2021, 119 p.
34. **Kristjan Krips.** Privacy and Coercion-Resistance in Voting. Tartu 2022, 173 p.
35. **Elizaveta Yankovskaya.** Quality Estimation through Attention. Tartu 2022, 115 p.
36. **Mubashar Iqbal.** Reference Framework for Managing Security Risks Using Blockchain. Tartu 2022, 203 p.
37. **Jakob Mass.** Process Management for Internet of Mobile Things. Tartu 2022, 151 p.
38. **Gamal Elkoumy.** Privacy-Enhancing Technologies for Business Process Mining. Tartu 2022, 135 p.
39. **Lidia Feklistova.** Learners of an Introductory Programming MOOC: Background Variables, Engagement Patterns and Performance. Tartu 2022, 151 p.
40. **Mohamed Ragab.** Bench-Ranking: A Prescriptive Analysis Approach for Large Knowledge Graphs Query Workloads. Tartu 2022, 158 p.
41. **Mohammad Anagreh.** Privacy-Preserving Parallel Computations for Graph Problems. Tartu 2023, 181 p.
42. **Rahul Goel.** Mining Social Well-being Using Mobile Data. Tartu 2023, 104 p.

43. **Anti Ingel.** Algorithms using information theory: classification in brain-computer interfaces and characterising reinforcement-learning agents. Tartu 2023, 142 p.
44. **Shakshi Sharma.** Fighting Misinformation in the Digital Age: A Comprehensive Strategy for Characterizing, Identifying, and Mitigating Misinformation on Online Social Media Platforms. Tartu 2023, 158 p.
45. **Kristiina Rahkema.** Quality Analysis of iOS Applications with Focus on Maintainability and Security Aspects. Tartu 2023, 182 p.
46. **Ivan Slobozhan.** Studying Online Social Media Engagement in CIS Countries during Protests, Mass Demonstrations and War. Tartu 2023, 81 p.
47. **Nurlan Kerimov.** Building a catalogue of molecular quantitative trait loci to interpret complex trait associations. Tartu 2023, 248 p.
48. **Pavlo Tertychnyi.** Machine Learning Methods for Anti-Money Laundering Monitoring. Tartu 2023, 117 p.
49. **Abasi-amefon Obot Affia.** A Framework and Teaching Approach for IoT Security Risk Management. Tartu 2023, 180 p.
50. **Raimond-Hendrik Tunnel.** Video Game Design and Development Bachelor's Curriculum for Estonia. Tartu 2024, 137 p.
51. **Ahto Salumets.** Bioinformatics analysis of various aspects in immunology. Tartu 2024, 198 p.
52. **Mohammed Abdulhameed Shaif Ali.** Deep Learning Methods for Cell Microscopy Image Analysis. Tartu 2024, 143 p.
53. **Pille Pullonen-Raudvere**. Foundations of Efficient and Secure Algorithm Development for Secure Multiparty Computation. Tartu 2024, 265 p.
54. **Marili Rõõm.** Multiple approaches to learners' success and factors affecting it in computer programming MOOCs. Tartu 2024, 170 p.
55. **Shivananda Rangappa Poojara.** Design and Orchestration of Scalable, Event-Driven Serverless Data Pipelines for Internet of Things (IoT) Applications. Tartu 2024, 172 p.