UNIVERSITY OF TARTU
Faculty of Science and Technology
Institute of Computer Science
Computer Science Curriculum

Kayahan Kaya

Improving Automated Feature Engineering Using Meta-Learning Based Techniques

Master's Thesis (30 ECTS)

Supervisor(s): Radwa El Shawi, Associate Professor
Hassan Eldeeb, PhD Candidate

Tartu 2022

Improving Automated Feature Engineering Using Meta-learning Based Techniques

Abstract:

Building well-performing machine learning pipelines requires the use of feature engineering. However, building highly predictive features takes time and requires subject-matter expertise. Although automated feature engineering research has recently gained a lot of attention from both academia and industry, the scalability and efficiency of the current methods and tools are still essentially subpar. To this end, we proposed meta-learning techniques to improve the performance of two automated machine learning frameworks; BigFeat and AutoFeat. Extensive experiments were conducted on 17 and 10 datasets for Bigfeat and AutoFeat, respectively. The results show that the proposed meta-learning techniques achieved an average improvement of F1-Score = 1.51% on BigFeat and an average improvement of F1-Score = 1.11% on AutoFeat.

Keywords: Feature Engineering, Automated Machine Learning, Meta-learning

CERCS: P170 - Computer science, numerical analysis, systems, control

Automatiseeritud Tunnnuste Väljatöötamise Täiustamine Metaõpe Meetodite Alusel

Lühikokkuvõte:

Hästi toimivate masinõppe konveierprotsesside loomine nõuab tunnuste väljatöötamist. Väga heade ennustustulemustega tunnuste väljatöötamine võtab aga aega ja nõuab erialaseid teadmisi. Kuigi automatiseeritud tunnuste väljatöötamisega seotud uuringud on viimasel ajal pälvinud palju tähelepanu nii akadeemiliste ringkondade kui ka tööstuse poolt, on praeguste meetodite ja tööriistade skaleeritavus ja tõhusus siiski sisuliselt kehvad. Selleks pakuti lõputöös välja metaõppetehnikaid kahe automatiseeritud masinõpperaamistiku jõudluse parandamiseks: BigFeat ja AutoFeat. Laiaulatuslikud katsed viidi läbi 17 Bigfeati ja 10 AutoFeati andmekogumiga. Tulemused näitavad, et pakutud metaõppe meetodid parandasid BigFeat andmete puhul F1-skoori keskmiselt 1,51% ja AutoFeati andmete puhul keskmiselt 1,11%.

Võtmesõnad:

Tunnuste Väljatöötamine, Automatiseeritud Masinõpe, Metaõpe

CERCS: P170 - Arvutiteadus, arvutusmeetodid, süsteemid, juhtimine (automaatjuhtimisteooria)

Contents

1 Introduction

This thesis highlights the effect of meta-learning techniques on Automated Feature Engineering (AutoFE) frameworks. This introductory chapter describes the motivation for utilizing meta-learning techniques in the area of feature engineering. Then, we present a list of contributions and an outline of this dissertation.

1.1 Motivation

Nowadays, improvements in machine learning have led to advancements in fields such as computer vision and speech recognition. In classifying objects in images, image recognition systems have surpassed human ability [RDS+14]. In transcribing spoken language, speech recognition systems have outperformed human teams [XDH+16]. These impressive accomplishments have created a great deal of interest among enterprises, organizations, and governments in utilizing machine learning to solve their data science problems.

Data science is the process of deriving insights, information, and predicting models from data. A typical data science pipeline involves five steps: problem definition, data collection, feature engineering, learning model selection, and hyper-parameter tuning. It begins with the formulation of the predicted problem of interest, which is often determined by the demands of the individual or organization. The process then includes gathering and storing problem-specific data. Next, it synthesizes new features from gathered data by implying what could be used to estimate the outcome. The final step is selecting a machine learning model and hyper-parameter tuning to achieve the best performance.

Feature engineering (FE) is one of the most time-consuming and essential steps. It is crucial because the efficacy of machine learning algorithms highly depends on the characteristics of the input data. Therefore, the performance of the same machine learning model with the same parameters and even barely different input features can vary significantly [Dom12]. Creating useful features is also quite time-consuming because it requires data scientists with extensive experience in machine learning and statistics.

Individuals and organizations can derive significant benefits from the automation of feature engineering. It enables data scientists to prioritize other pipeline processes and iterate through numerous pipelines more efficiently. Frequently, it is unclear if additional time and effort should be spent constructing new features, selecting a different machine learning model, or acquiring more relevant data. However, AutoFE can eliminate a considerable amount of ambiguity and accelerate the decision-making process by estimating how much value can be obtained from the current dataset and model. In addition, this would address the lack of experienced data scientists in organizations and governments by eliminating the need for intuition and domain-specific data knowledge to generate significant features. Using AutoFE, non-experts may rapidly extract value

from their data and create a machine learning model that performs comparably better than models designed by a machine learning practitioner.

Rarely, if ever, do we begin learning new abilities from scratch. We start with abilities earned by completing similar tasks in the past, reusing methods that have worked well in the past, and focusing on what is likely worth attempting based on prior experience [LUTG17]. Learning new skills becomes easier and quicker with each new talent, requiring fewer examples and less trial-and-error. In summary, we acquire the ability to learn across tasks. Similarly, when creating machine learning models to solve a problem, we frequently rely on our experience with related tasks to make the appropriate decisions or use our (sometimes implicit) knowledge of the behavior of machine learning approaches to help us make the proper decisions.

The challenge of meta-learning is learning from previous experiences in a systematic and data-driven way. First, we must acquire meta-data describing previous learning tasks and previously trained models. They consist of the exact algorithm configurations used to train the models, such as hyper-parameter settings, pipeline configurations, and evaluation metrics, such as F-1 score, accuracy, a neural network's weights, and the task's measurable properties, also referred to as meta-features. Second, we must extract and transfer information from this metadata that leads to the search for appropriate models for new tasks.

Many studies have been done on the subject of AutoFE. However, these studies consistently generate all features during the feature generation phase and later select a subset of those features. As a result, the time and space complexity are extremely high, making it inapplicable for applications involving big data volumes or high feature dimensions. In contrast, the desired AutoFE framework should be computationally efficient and scalable for larger tasks. We believe that using meta-learning based techniques in the FE field can fill the gap between the current and desired states.

Hence, this thesis aims to integrate meta-learning techniques into the AutoFE frameworks to reduce the execution time of the framework and improve the quality of the generated features in a relatively short amount of time. We expect that integrating meta-learning methods will allow us to get a warm-start on the optimization process by determining better initial weights for sampled operators. We aim to get similar outcomes with fewer iterations and less computational power by starting weights closer to optimal levels.

1.2 Contribution

The following points highlight the contribution of the thesis:

- We analyzed many different meta-features to determine the best features to characterize datasets and previous tasks.
- We proposed meta-learning techniques to improve the performance of two automated feature engineering frameworks, including BigFeat and AutoFeat.
- We evaluated the performance of the proposed techniques on different datasets.

1.3 Outline

The outline of the thesis is as follows: Section 2 discusses the related work in the domain of AutoML, Automated Feature Engineering, and Meta-learning Techniques in AutoML. Section 3 shows the proposed methodology of this thesis in 2 parts; Recommendation Engine for BigFeat and Recommendation Engine for AutoFeat. Section 4 presents the performance evaluation of meta-learning based BigFeat and the performance evaluation of meta-learning based AutoFeat. Lastly, section 5 concludes the previous sections and provides a brief overview of future work.

2 Background

In recent years, a substantial amount of work has been conducted in the field of Automated Machine Learning. The following is a summary of the relevant studies and literature.

2.1 Automated Machine Learning (AutoML)

As machine learning becomes more popular, many solutions to support the AutoML [HKV19] domain are being created. Such development is intended to reduce the time needed to build well-performing machine learning pipelines. More specifically, AutoML aims to find an algorithm with the best set of hyperparameters to achieve good performance.

Numerous AutoML techniques have been created to automate the process of algorithm selection and hyper-parameter tuning problem (CASH). The objective of the CASH problem is to find the algorithm and hyper-parameter combination that optimizes the performance of a pipeline for a given task based on a given metric.

The most widely used AutoML tools are presented below in short.

2.1.1 Supervised AutoML

This section shows different AutoML frameworks for supervised learning problems such as regression and classification.

TPOT Tree-based Pipeline Optimization Tool, TPOT, is an open-source Python project built on top of Scikit-Learn [OM16]. Figure 1 shows an example of a TPOT pipeline. It is based on genetic programming and explores a variety of feature preprocessing and learning algorithm pipelines. TPOT is a supervised learning tool that uses 150 SciKit-Learn algorithms, including preprocessing approaches, to perform classification. The initial generation generates 100 tree-based pipelines, which are then optimized using the Python library DEAP. Twenty pipelines are chosen to mutate and create new pipeline generations. Through cross-over, each pipeline generates five additional pipelines. The algorithm runs for 100 generations, with each generation updating a pareto front of non-dominated solutions.

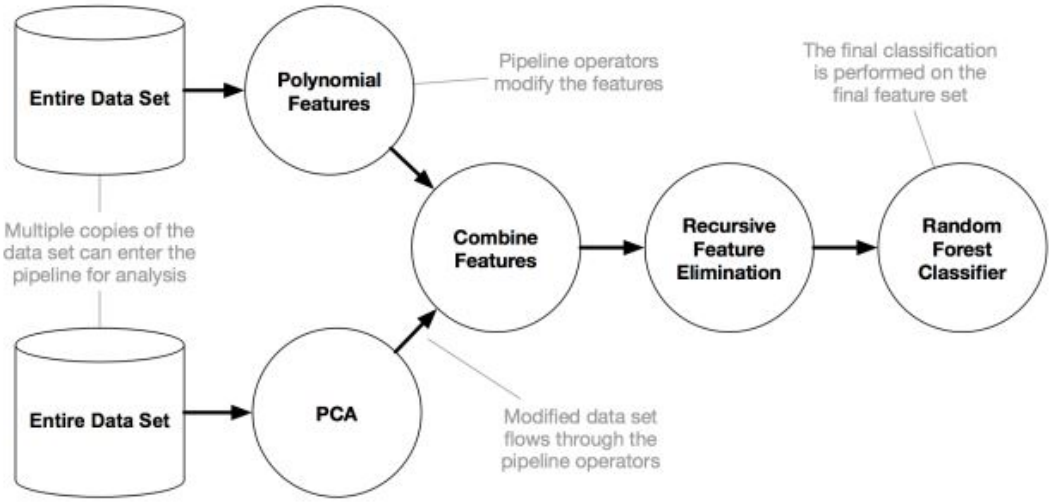

Figure 1. TPOT Pipeline [OM16].

Auto-Sklearn Auto-Sklearn is implemented on top of Scikit-Learn, a popular Python machine learning package [FEF+20]. It employs SMAC [BBE+20] as Bayesian optimization for hyperparameter search and then selects the most likely hyper-parameter configuration to give the best performance. The generated model is evaluated, and the probabilistic model is updated with new data. This procedure continues throughout the entire process. The AutoSklearn pipeline consists of data preprocessing (imputation, one-hot encoding, resealing) and feature preprocessing. It generates an ensemble of previously discovered, high-performing models at the end of the process. Also, initialization of algorithm selection and hyperparameter tuning is done through meta-learning, but it is only available for classification at the moment.

AutoWEKA AutoWEKA [THHL12] is one of the most widely used AutoML tools. It is based on WEKA and aims to make using the software as simple as possible. It uses Bayesian optimization for hyperparameter tuning, similar to AutoSklearn. It initially just supported classification, but with version 2.0, it also supports regression tasks. AutoWEKA offers a large number of classifiers, which is not applicable for other AutoML frameworks. It also has a graphical user interface (GUI) and a command line. It is one of the most user-friendly frameworks, so it is straightforward to start building machine learning models with minimal knowledge.

Auto-Net Auto-Net [MKF+] is an AutoML framework for tuning neural networks based on SMAC optimization, built on PyTorch. Auto-Net's first version is built in Auto-Sklearn to take advantage of some of Auto-Sklearn's machine learning pipeline components, such as preprocessing. Only fully-connected feed-forward neural networks were considered in the first version of Auto-Net as they were applied on a range of datasets. Auto-net uses the Lasagne Python deep learning library [DSR+15] to access deep learning algorithms. Vanilla stochastic gradient descent, stochastic gradient descent with momentum, Adadelta [Zei12], Adam [KB14], and Adagrad [DHS11] are some of the algorithms included in Auto-Net for tuning neural network weights.

2.1.2 Unsupervised AutoML

The popularity of AutoML has contributed to the development of many frameworks that are primarily focused on supervised problems. This is mainly because there are no ground truths for real-world datasets, causing obscure objective function to optimize. [PDK20]. However, only a few methods have been developed to assist inexperienced practitioners in selecting the best-performing algorithms for unsupervised learning tasks. The problem of unsupervised AutoML is formally defined as follows:

Let $\mathcal{D}$ represent a group of n data sets $\mathcal{D} = \{D_1, D_2, \ldots, D_n\}$. Also, let $F(D) = \{f_1, \ldots, f_k\}$ represent a collection of k meta-features extracted from dataset D. Let $\mathcal{A} = \{A_1, \ldots, A_m\}$ be a group of clustering algorithms, and let Λ_i represent the search space of the hyper-parameters for the algorithm A_i. Finally, let $L\left(A_i(\lambda), D\right)$ represent the loss of A_i with hyperparameters $\lambda \in \Lambda_i$ on D. The objective of the CASH problem is to discover the algorithm and hyperparameter values that minimize this loss:

$$A^*, \lambda^* = \underset{A_i \in \mathcal{A}, \lambda \in \Lambda_i}{\arg\min}\, L\left(A_i(\lambda), D\right) \tag{1}$$

The following part will cover several AutoML frameworks that address unsupervised task-related problems.

AutoML4Clust AutoML4Clust [FBTS21] integrates the algorithm selection and hyperparameter tuning to address the problem. It is built on top of Scikit-learn and only supports K-means, MiniBatch K-means, and k-Medoids, which are all k-center clustering algorithms. AutoML4Clust only tunes the number of clusters k over a fixed search space for hyperparameter tuning, ensuring that the maximum k value is proportional to the number of instances in the dataset. Calinski-Harabasz, Davies-Bouldin Index, and Silhouette are three internal metrics that the user can use for evaluation. To obtain high-performing configurations rapidly, AutoML4Clust employs a Bayesian optimization, Hyperband, or BOHB.

AutoClust AutoClust [PDK20] is an end-to-end system for selecting automatic clustering algorithms that rely on meta-learning and cluster validity indices. It solves the problem by combining meta-learning with Bayesian optimization techniques. The framework presents a method for hyperparameter tuning of clustering algorithms that takes advantage of a novel optimization criterion: regression of cluster validity indices. Empirical testing of the methodology utilizing many real-world data sets confirms the framework's effectiveness over 24 state-of-the-art methods. K-means, OPTICS, Affinity Propagation, Birch, DBSCAN, Spectral, Agglomerated and MeanShift are among the eight clustering methods considered in the framework.

cSmartML cSmartML [ELS21] provides a solution that addresses algorithm selection and hyperparameter tuning as a single problem. The four main phases of the framework are the input phase, algorithm and metric selection, hyper-parameter optimization, and computing the output and updating the knowledge base. The user uploads the dataset and sets the time budget limit for the metric selection, algorithm selection, and hyper-parameter tuning processes in the input phase. Meta-feature extraction and meta-learning recommendation are the two primary components of the algorithm and metric selection phase.

cSmartML assesses a collection of meta-features on eight clustering methods, including KMeans, DBSCAN, OPTICS, Birch, Spectral, Agglomerated, Affinity Propagation, and MeanShift. Twelve internal indices were utilized to evaluate the clustering solutions of the analyzed algorithms. Each dataset was clustered utilizing each of the eight clustering methods with different hyper-parameter values and assessed with every possible combination of the three clustering functions to choose the best configurations.. It uses Spearman's rank correlation to compute the correlation between configuration rankings derived through external validation with Normalized Mutual Information (NMI) to determine the validity of the multi-objective rankings. For each dataset, cSmartML stores the clustering technique and the three validity indices with the highest correlation coefficients for meta-learning purposes. cSmartML considers both main hyperparameters, such as maximum iteration, n-init MeanShift bandwidth, and n-clusters, as well as conditional hyperparameters such as gamma, xi and gamma.

Autocluster Autocluster [LLT21] contains Clustering-oriented Meta-feature Extraction (CME) and Multi-CVIs Clustering Ensemble Construction (MC^2EC). CME captures the meta-features of spatial randomization and the different learning aspects of clustering algorithms to improve meta-learning. MC^2EC develops a collaborative mechanism based on clustering ensemble to balance the measuring criterion of various CVIs, such as Calinski-Harabasz Index (CHI) and Davies-Bouldin Index (DBI), and one non-center-based representative, the Silhouette Coefficient (SC), to build a more suitable clustering model for given datasets.

2.2 Automated Feature Engineering

Feature engineering is a technique for creating and choosing features from a raw dataset. It is well known that the quality of the features has a significant impact on the success of the machine learning pipeline [HTF09]. However, developing meaningful features is time-consuming and requires highly competent data scientists with robust machine learning and statistics backgrounds to extract significant findings from raw data. Additionally, it necessitates the use of human intuition and experience. Furthermore, as domain knowledge is critical in this stage, senior data scientists have claimed that FE is the most time-consuming. On the other hand, the amount of data scientists required to support the expanding number of machine learning applications is unsustainable. As a result, manually performing FE for these applications becomes implausible. The problem of AutoFE is formally defined as follows:

Assume a machine learning model that consists of:

- A dataset that consists of input-output set. Let $x \in X$ be a single record of the dataset with M features. $\left\{\boldsymbol{x}^{(1)}, \cdots, \boldsymbol{x}^{(M)}\right\}$. Let $y \in Y$ be the label of the dataset. Training data that has N records can be shown as $\mathcal{D}_{\text{train}} = \{X_{train}, Y_{train}\} = \{(\boldsymbol{x}_1, y_1), \cdots, (\boldsymbol{x}_N, y_N)\}$.Likewise, testing and validation data can be shown as $\mathcal{D}_{\text{valid}}$ and $\mathcal{D}_{\text{test}}$.

- A machine learning model $\mathcal{A}$ builds a predictive function $\mathcal{F} : X \rightarrow Y$ that takes validation and training data as an input and returns a predicted label as an output.

- An error function $\mathcal{E}$ which calculates the loss of the function $\mathcal{F}$, based on the original label y.

Feature engineering aims to find a feature generation function: $\Psi : X \rightarrow Z$ that produces a new feature $z \in Z$ depending on the original features $\boldsymbol{x} \in X$, by applying collection of k operators $\mathcal{O} = \{o_1, \cdots, o_k\}$, therefore the machine learning model $\mathcal{A}$ can construct a function $\mathcal{F}$ which reduce the error function $\mathcal{E}$. More theoretically, we wish to identify the feature generation function that minimizes the error $\mathcal{E}$ of the predictive function $\mathcal{F}$:

$$\Psi^* = \arg\min_{\Psi} \mathcal{L}\left(\mathcal{F}\left(\Psi\left(X_{test}\right)\right), Y_{test}\right) \tag{2}$$

2.2.1 Deep Learning based approach

Deep learning (DL) approaches have been used on various data types, including tabular, text, video streaming, and audio [BCV13]. The capability of deep learning models to learn expressive representations from the original input data is well-known. However, this way of learning insightful data characteristics has some disadvantages in practice. First, training high-performing deep learning models needs a massive amount of data, which is not always feasible across a wide variety of application areas. Second, training deep learning models requires tuning a large number of hyperparameters, which takes time and requires a large amount of computational resources. Finally, since the features obtained from deep learning models are not interpretable, machine learning practitioners have difficulty believing models based on them. While tremendous progress has been achieved in terms of improving the interpretability of neural networks, this is still inadequate for critical domains in practice, such as applications in healthcare [ESAMS19].

2.2.2 Feature Generation & Feature Selection Based Approach

AutoFE techniques often integrate both feature generation and feature selection phases and and utilize one of two main techniques for creating informative representations from the raw data [KSS17a].

Generate-and-select-based approach builds an extensive feature pool and then chooses features [KMP17]. This technique has performance and scalability limitations due to conducting feature selection on a large number of features produced by continuously applying all transformers.

Iterative-constructive-based approach iteratively builds features by assessing the improvement gain obtained by combining the new features [KSS17b]. While this technique is more scalable than the generate-and-select-based approach, it may remove some potential features in the early stages of the feature engineering pipeline. Thus, it may prevent sophisticated features from being learned by the model. The majority of current FE frameworks use the iterative-constructive-based technique.

FICUS FICUS [MR02] employs beam search to find the space of possible features, generating new features using "constructor functions" (e.g., applying transformations to the original features). The search for superior features in FICUS is guided by heuristic measurements such as information gain in a decision tree and other evaluation metrics.

AutoFeat AutoFeat is a Python module that automates the feature engineering process [HPR20]. It goes through the following stages: feature generation and feature selection. Nonlinear operators are applied to each feature during the feature generation step. These operators are $x^2, x^3, 1/x, |x|, 2^x, log(x), \sqrt{x}, \exp(x), sin(x), cos(x)$. and

then combining them with different operators ($+, *, -$). These processes can be repeated many times to build deeper and more sophisticated features. However, since the number of features increases exponentially with each iteration, a large number of iterations is impractical in terms of time and computational resources. In the second phase, AutoFeat conducts feature selection to reduce the number of features to a more realistic size. Also, feature selection is accomplished in two stages. First, created features are prioritized according to their correlation with the target residual, and only the features with the highest correlation are selected for the second stage. Subsets of features which are chosen in the previous step and random permutations of all features are then used to train the Lasso LARS regression model. Then the regression coefficient with the maximum value among the random features is used as a threshold. Only features with regression coefficients greater than the threshold are retained for final selection. Besides, the feature selection stage can be performed numerous times to increase the reliability of the results.

SAFE Scalable Automatic Feature Engineering is a feature engineering library that automates the generation and selection of features [SZL+20]. The first feature relations are mined during the feature generation step to reduce the range of possible feature combinations. Therefore, the training and validation sets are used to train the XGBoost model. The model tracks the split features and combinations of split features on the same tree path. The tool is based on the assumption that split features that exist on the same tree path are more likely to produce good combinations than split features that do not appear on the same tree path or, non-split features. The information gain ratio is used to rate the features and feature combinations. The features with the highest information gain ratio are chosen to generate the features. Feature selection is made in three stages after candidate features are generated. Then, features with low predictive power are eliminated based on their information values. The Pearson correlation between candidate feature pairs is then used to remove redundant features. If a pair's absolute correlation is greater than 0.8, the pair with the lowest information value is deleted. The final process involves training the XGBoost model on the remaining features and selecting the most important ones.

BigFeat BigFeat is another automated feature engineering framework implemented in python. It is an iterative process where each process consists of 2 stages: feature generation and feature selection. Using all combinations of base features with predefined operators can cause a combinatorial explosion. To overcome this problem, BigFeat uses tree-based and linear models to calculate an importance score for each feature to mine the relationship between the base features. This score is determined by the information gain of tree-based models and the coefficients of linear models so that features with higher importance scores have a greater probability of being chosen. The original features are then applied to a set of predetermined operators to form new features. To avoid using all predefined operators with equal frequency to new feature combinations, BigFeat assigns each operator an importance score. This score is higher for operators which frequently

contributed to creating powerful features in prior iterations. Then BigFeat combines base features and generated features, but still, the number of features is enormous. That's why BigFeat keeps only N number of features in each iteration. The main aim of the feature selection phase is to identify the most informative N features while deleting redundant ones. Therefore, BigFeat uses the importance score to select the most informative features that have significant impact on the label. The redundant features are then eliminated using the Pearson correlation coefficient. Then it updates the importance score of operators depending on the frequency with which each operator appeared in the selected features.

2.2.3 Reinforcement Learning based approach

Some AutoFE frameworks utilize reinforcement learning techniques. The authors of [GS10] defined the selection of a learning algorithm as a reinforcement learning problem and employed Monte Carlo tree search to solve it. The challenge of selecting a subset of accessible features is described as a single-player game, with states corresponding to all possible subsets of features and actions corresponding to selecting a feature and putting it into the subset.

2.2.4 Meta-learning based approach

Meta-learning approaches are used in several AutoFE frameworks. ExploreKit [KSS17b] evaluates and selects the most promising features based on meta-learning. Another approach used meta-learning to find potentially valuable transformations for numerical features [KST17]. Since meta-features do not take into account the relation between features, they perform best when unary transformations are used.

2.3 Meta-learning in AutoML

Meta-learning, also known as learning to learn, is the discipline of methodically examining how various learning algorithms perform on a large variety of tasks and then leveraging this experience to learn new tasks quickly [HKV19]. This method not only substantially accelerates and improves the development of machine learning pipelines but also enables us to replace hand-engineered techniques with new data-driven techniques.

2.3.1 Warm-Starting Optimization from Similar Tasks

Meta-features are suitable for evaluating task similarity and initiating optimization techniques based on ideal task configurations. This approach is analogous to how machine learning practitioners begin a manual search for high-quality models, given their familiarity with similar tasks [HKV19].

Beginning a genetic search algorithm with promising configurations in the search space can accelerate convergence to a good solution. Gomes et al. [GPS$^+$10] suggest starting configurations by selecting the k most similar previous tasks t_j using the L1 distance between vectors $m_{(t_j)}$ and $m_{(t_new)}$, where each $m_{(t_j)}$ consists of 17 meta-features. For each of the k most similar tasks, the best configuration is determined using t_{new} and utilized to initiate a genetic search algorithm.

Feurer et al. [FSH14] provide a straightforward technique that warm-starts Bayesian optimization by ranking previous tasks t_j similar to [GPS$^+$10], but contains 46 meta-features. To begin the surrogate model with a warm-start, the t best configurations for the d most similar tasks are used. They searched across more hyperparameters than previous research, which also included preprocessing processes.

2.3.2 Learning from Task Properties

One of the primary metadata sources is the characterizations (meta-features) of the current task. Every task $t_j \in T$ can be expressed as a vector of $m(t_j) = (m_{j,1}, \ldots, m_{j,K})$ with K meta-features $m_{j,k} \in M$. This can be utilized to construct a measure of task similarity, such as the Euclidean distance between $m(t_i)$ and $m(t_j)$, so that information can be transferred from the tasks that are most similar to the new task t_{new}. Furthermore, we can build a meta-learner L on previously made evaluations P to suggest configurations for a given new task t_{new}.

Meta-features Table 0 presents a summary of the most frequently used meta-features, along with a brief explanation of why they are predictive of model performance. To construct a meta-feature vector $m(t_j)$, one should choose and analyze these meta-features further. Many researches on OpenML metadata have demonstrated that the most appropriate meta-features depend heavily on the project itself [BAAB17]. Numerous meta-features are calculated on single features or sets of features which must be aggregated by statistical operators such as min, avg, and max [KH00]. When calculating the similarity of the task, it is essential to standardize whole sets of meta-features by applying feature selection techniques or utilizing dimension reduction techniques (e.g., PCA).

2.3.3 Meta-model

The meta-model recommends the best configuration for a specific task by training a model on the meta-features and configurations of the task [Van18]. Although the model selection highly depends on the meta-features utilized, tree-based ensemble models have been found to perform the best overall [KH01]. Certain meta-models have been successful in giving rankings of the top K configurations. By creating a regressor model, meta-models can also be used to predict performance or required time for a configuration

on a specific task. This enables users to analyze specific configurations and save resources by eliminating configurations with poor performance or requiring long-time execution.

Name	Formula	Rationale	Variants
Nr instances	n	Speed, Scalability	p/n, $log(n)$, $log(n/p)$
Nr features	p	Curse of dimensionality	$log(p)$, % categorical
Nr classes	c	Complexity, imbalance	ratio min/maj class
Nr missing values	m	Imputation effects	% missing
Nr outliers	o	Data noisiness	o/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,ρ_{XY}
Covariance	$cov_{X_1X_2}$	Feature interdependence	min,max,μ,σ,cov_{XY}
Concentration	$\tau_{X_1X_2}$	Feature interdependence	min,max,μ,σ,τ_{XY}
Sparsity	sparsity(X)	Degree of discreteness	min,max,μ,σ
Gravity	gravity(X)	Inter-class dispersion	
ANOVA p-value	$p_{val_{X_1X_2}}$	Feature redundancy	$p_{val_{XY}}$
Coeff. of variation	$\frac{\sigma_Y}{\mu_Y}$	Variation in target	
PCA ρ_{λ_1}	$\sqrt{\frac{\lambda_1}{1+\lambda_1}}$	Variance in first PC	$\frac{\lambda_1}{\sum_i \lambda_i}$
PCA skewness		Skewness of first PC	PCA kurtosis
PCA 95%	$\frac{dim_{95\%var}}{p}$	Intrinsic dimensionality	
Class probability	$P(\mathrm{C})$	Class distribution	min,max,μ,σ
Class entropy	$H(\mathrm{C})$	Class imbalance	
Norm. entropy	$\frac{H(\mathrm{X})}{log_2 n}$	Feature informativeness	min,max,μ,σ
Mutual inform.	$MI(\mathrm{C}, \mathrm{X})$	Feature importance	min,max,μ,σ
Uncertainty coeff.	$\frac{MI(\mathrm{C},\mathrm{X})}{H(\mathrm{C})}$	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	
Fisher's discrimin.	$\frac{(\mu_{c1}-\mu_{c2})^2}{\sigma_{c1}^2-\sigma_{c2}^2}$	Separability classes c_1, c_2	See
Volume of overlap		Class distribution overlap	See
Concept variation		Task complexity	See
Data consistency		Data quality	See
Nr nodes, leaves	$\lvert\eta\rvert$, $\lvert\psi\rvert$	Concept complexity	Tree depth
Branch length		Concept complexity	min,max,μ,σ
Nodes per feature	$\lvert\eta_X\rvert$	Feature importance	min,max,μ,σ
Leaves per class	$\frac{\lvert\psi_c\rvert}{\lvert\psi\rvert}$	Class complexity	min,max,μ,σ
Leaves agreement	$\frac{n_{\psi_i}}{n}$	Class separability	min,max,μ,σ
Information gain		Feature importance	min,max,μ,σ, gini

Table 0. Summary of the mainly used meta-features [HKV19].

3 Proposed Methodology

3.1 Recommendation Engine for BigFeat

The proposed method includes two phases. In the offline phase, we extract a set of meta-features from a large number of datasets obtained from openML [VvRBT13] to characterize these datasets. Then we compute the operator importance vector for these datasets with BigFeat to build a meta-learning space. In the online phase, given a new dataset, we extract the same meta-features as in the offline phase and get a recommended operator importance vector for this dataset. Figure 2 shows the architecture of the proposed method.

The details of each phase are explained in the next section.

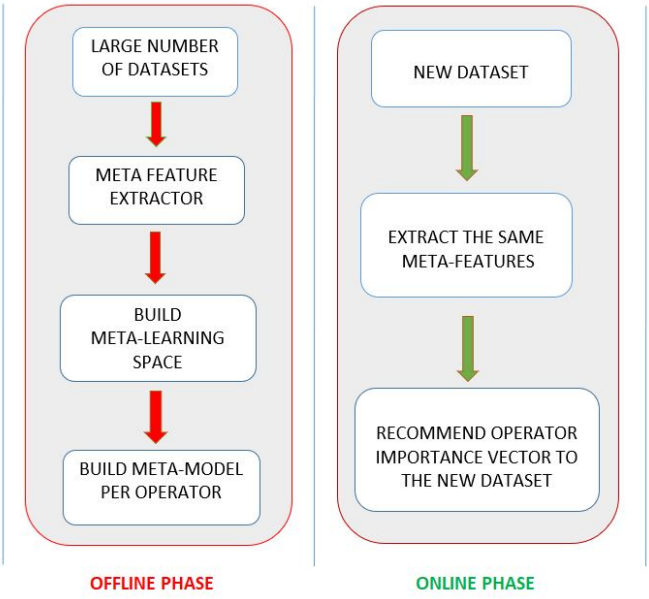

Figure 2. The architecture of the proposed method.

3.1.1 Offline Phase

The offline phase involves extracting the meta-features from the datasets and building the larger meta-learning space, which includes the extracted meta-features of a large number of datasets along with their operator importance vector generated by BigFeat.

Dataset In this study, we utilize 197 datasets from the OpenML project [VvRBT13], covering many different domains. The datasets contain various attributes, between 500 and 1,000,000 data points, and are well balanced. There are multiple target classes ranging from 2 to 48.

Meta-feature Extraction In many meta-learning systems, each dataset represents a single metadata point. Therefore, it is essential to provide a growing list of various datasets. Regardless of the content of the data, the metadata gives information about the dataset. In general, metadata represents the nature of the dataset through structural features such as the dataset's mean, number of instances, variance, prediction accuracy, median, standard deviation, number of occurrences, and number of attributes. Different meta-learning systems may employ different dataset characteristics and performance metrics. There are hundreds of different metadata types, and no proven method for identifying the set would produce the best results. In this study, we extracted different types of meta-features to characterize these datasets more precisely, as listed below.

Landmarking meta-features Landmarking is a method for characterizing datasets based on the performance of many learning algorithms, as opposed to model-based meta-features, which extract information from learning models. Table 1 illustrates the description of landmarking meta-features. In addition to those metadata extracted by the pymfe package [ASR+20] implemented in Python, one more meta-feature is included in the list, which is the accuracy of KNN (n_neighbors=1).

Meta-features	**Description**
$bestNode$	Decision tree model on most informative attribute
$eliteNN$	The accuracy of the elite-Nearest Neighbor
$linearDiscr$	The accuracy of the Linear Discriminant
$naiveBayes$	The accuracy of naive bayes
$oneNN$	The accuracy one-Nearest Neighbor
$randomNode$	Decision tree model on random informative attribute
$worstNode$	Decision tree model on least informative attribute
KNN	The accuracy of the KNN model

Table 1. Description of Landmarking meta-features [RGS+18].

Model-based Meta-features (Tree-based) We calculated model-based meta-features using pymfe package. These meta-features are extracted data from a predictive learning model, specifically a Decision Tree model (DT). They characterize dataset based on the complexity of the induced model, which for DT, it can be the number of nodes, dept of the tree, and the number of leaves. Table 2 shows the description of tree-based meta-features.

Meta-features	**Description**
$leaves$	Number of leaves
$leavesBranch$	The number of distinct paths for each leaves
$leavesCorrob$	The proportion of training instances to the leaf
$leavesHomo$	The distribution of the leaves in the tree
$leavesPerClass$	The proportion of leaves to the classes
$snodes$	The number of nodes
$nodesPerAttr$	The proportion of nodes per attribute
$nodesPerInst$	the proportion of nodes per instance
$nodesPerLevel$	The number of nodes per level
$nodesRepeated$	The number of repeated nodes
$treeDepth$	Depth of each node and leaf
$treeImbalance$	The degree of imbalance in the tree
$treeShape$	The shape of the tree
$varImportance$	The importance of each attribute

Table 2. Description of Tree-based meta-features [RGS$^+$18].

Combined Meta-features We followed a previous Elshawi et al. [BSPS21] study to determine the combined meta-features in this thesis. These meta-features consist of information-theoretic meta-features, statistical meta-features, and simple meta-features, as seen in Table 3. Additionally, some meta-features are discarded from the scope as they are landmarking-related.

Combined $\cup$ *Tree-based Meta-features* In order to extract characteristics of the datasets more precisely, tree-based and combined meta-features are concatenated. The more information can be found in Table 2 and Table 3.

Combined $\cup$ *Landmarking Meta-features* In order to extract characteristics of the datasets more precisely, landmarking and combined meta-features are concatenated. The more information can be found in Table 1 and Table 3.

Landmarking $\cup$ *Tree-based Meta-features* In order to extract characteristics of the datasets more precisely, tree-based and landmarking meta-features are concatenated. The more information can be found in Table 1 and Table 2.

Meta-features	**Description**
$nr\_classes$	Number of classes
$nr\_instances$	Number of instances
$log\_nr\_instances$	Logarithm of the number of instances
$missing\_val$	Number of missing values
$ratio\_missing\_val$	Ratio of missing values
$nr\_numerical\_features$	Number of numerical features
$nr\_categorical\_features$	Number of categorical features
$labels\_sum$	Sum of the target classes
$labels\_mean$	Mean of the target classes
$labels\_std$	Standard deviation of the target classes
$dataset\_ratio$	Ration of number of features over number of instances
$skew\_min$	Minimum value of skewness
$skew\_std$	Standard deviation of skewness
$skew\_mean$	Mean value of skewness
$skew\_q1$	First quintile of the skewness
$skew\_q3$	Third quintile of the skewness
$skew\_max$	Maximum value of the skewness
$kurtosis\_min$	Minimum value of kurtosis
$kurtosis\_mean$	Mean value of kurtosis
$kurtosis\_q1$	First quintile of the kurtosis
$kurtosis\_q3$	Third quintile of the kurtosis
$kurtosis\_max$	Maximum value of kurtosis
rho_min	Minimum value of Spearman's rank correlation coefficient
rho_max	Maximum value of Spearman's rank correlation coefficient
rho_mean	Mean value of Spearman's rank correlation coefficient
rho_std	Standard deviation of Spearman's rank correlation coefficient
$class\_entropy$	Entropy of the target values
$prob\_min$	Minimum probability of the target values
$prob\_mean$	Mean probability of the target values
$prob\_std$	Standard deviation of the target values
$prob\_max$	Maximum probability of the target values
$norm\_entropy\_min$	Minimum value of norm attribute entropy
$norm\_entropy\_mean$	Mean value of norm attribute entropy
$norm\_entropy\_std$	Standard deviation of norm attribute entropy
$norm\_entropy\_max$	Maximum value of norm attribute entropy
mi_min	Minimum mutual information of the dataset
mi_mean	Mean mutual information of the dataset
mi_std	Standard deviation of mutual information of the dataset
mi_max	Maximum mutual information of the dataset
$equiv\_nr\_feat$	The ratio of $class\_entropy$ over mi_mean
$noise\_signal\_ratio$	Noise signal ratio of dataset

Table 3. Description of Combined meta-features.

Building Meta-Learning Space The primary objective of this phase is to build a meta-learning space. The meta-learning space consists of the meta-features of the previously stated datasets and their corresponding operator importance vectors, which are obtained from BigFeat. Table 4 shows a list of the operators used in BigFeat. In order to determine the optimal operator importance vector for each dataset, we ran seven different configurations of BigFeat on 180 datasets. We analyzed the performance of those datasets on different machine learning models. For each dataset, we selected the operator importance vector of the configuration that has the best average performance score on this dataset. Figure 3 illustrates the established meta-learning space.

Operator	**Type**
$multiply$	Binary operator
add	Binary operator
$subtract$	Binary operator
$absolute$	Unary operator
$square$	Unary operator

Table 4. List of operators used in BigFeat.

DATASET	EXTRACTED METADATA				OPERATOR IMPORTANCE				
	Statistical,Landmarking,Model-Based Info				Multiply	Abs	Square	Subtract	Add
abalone	...	...	...	...	0.1875	0.28125	0.114583	0.177083	0.239583
eye_movements	...	...	...	...	0.174757	0.398058	0.165049	0.126214	0.135922
JapaneseVowels	...	...	...	...	0.111111	0.462963	0.222222	0.12963	0.074074
heart-c	...	...	...	...	0.156677	0.296142	0.224926	0.156083	0.166172
adult	...	...	...	...	0.215517	0.241379	0.204023	0.175287	0.163793

Figure 3. The established meta-learning space.

Building Meta-model Per Operator After the meta-learning space is ready, a tree-based pipeline optimization tool known as TPOT is used to determine the meta-model per operator. At the end of this process, we obtained five different meta-models for each operator, as can be seen in Figure 4.

```
models={'multiply':make_pipeline(MaxAbsScaler(),
    RBFSampler(gamma=0.2),
    RandomForestRegressor(bootstrap=True, max_features=0.7 min_samples_leaf=13, min_samples_split=16, n_estimators=100)),
        'square':make_pipeline(
    MaxAbsScaler(),
    PCA(iterated_power=1, svd_solver="randomized"),
    RidgeCV()),
        'add':make_pipeline(
    PCA(iterated_power=10, svd_solver="randomized"),
    RandomForestRegressor(bootstrap=True, max_features=0.45, min_samples_leaf=8, min_samples_split=17, n_estimators=100)),
        'subtract':KNeighborsRegressor(n_neighbors=10, p=2, weights="distance"),
        'abs': make_pipeline(
    PolynomialFeatures(degree=2, include_bias=False, interaction_only=False),
    AdaBoostRegressor(learning_rate=0.01, loss="square", n_estimators=100))}
```

Figure 4. Suggested meta-models per operator by TPOT.

3.1.2 Online Phase

Given a new dataset, the proposed system extracts the same meta-features from the dataset as in the offline phase. Then the meta-model suggests an operator importance vector for this dataset. BigFeat starts using this vector while performing the feature engineering process.

Weighted Average Given a new dataset, the proposed system extracts the same meta-features from the dataset as in the offline phase. Then the meta-model suggests weighted operator importance vectors that come from 3 different meta-learning techniques (Combined, Tree-based, and Model-based). The contributions (weights) of these three techniques to the operator importance vector was calculated based on their performance on the previously analyzed 180 datasets.

3.2 Recommendation Engine for AutoFeat

AutoFeat system requires vast computational power, and most of the time, it is not feasible to use for larger datasets. In order to improve the execution time and performance of this framework, we remove the feature selection mechanism, which takes enormous time for larger datasets, and propose a novel method for evaluating the usefulness of a generated feature using machine learning. Figure 5 illustrates the architecture of proposed approach. Our approach describes the interactions between the generated features and the dataset, as well as various characteristics of the dataset itself.

The proposed methodology consists of two phases: The offline phase involves the preparation of the meta-learning space by considering the characteristics of both datasets and generated features and the usefulness label of the generated features. In the online phase, the meta-model leverages the metadata and evaluates the generated feature as either useful or not.

The following section describes each phase in depth.

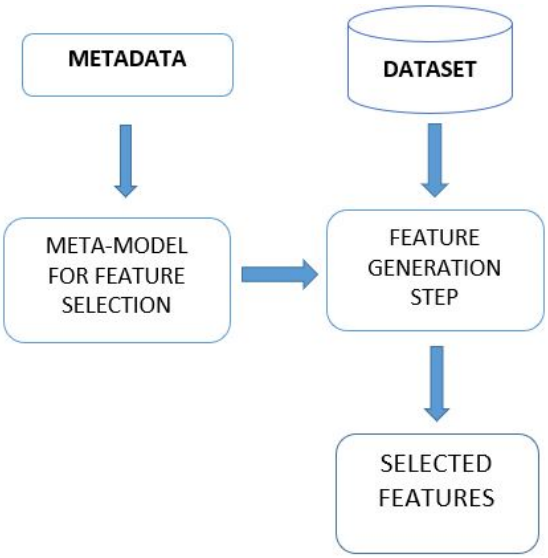

Figure 5. The architecture of proposed method for AutoFeat

3.2.1 Offline Phase

The offline phase involves extracting the meta-features from the dataset and building the larger meta-learning space, which includes the extracted characteristics of both the dataset and each generated feature, as well as the usefulness label of the generated feature.

Dataset In this study, we use 40 datasets from the OpenML project, covering many different domains. These datasets are well balanced and contain a number of instances ranging from 200 to 58,000, while the number of features goes between 4 to 1,000. There are multiple target classes ranging from 2 to 10.

Meta-feature Extraction We extracted statistical and information-theoretic meta-features using the pymfe package to build metadata for the AutoFeat system. These meta-features illustrate information-theoretic metrics and data distribution, such as central tendency and dispersion. Table 5 shows the description about meta-features.

Building Meta-learning Space and Meta-model This process consists of three phases: first, AutoFeat is used to generate candidate features for each training dataset. This process produces a large list of candidate features that can be examined. In the second phase, meta-features are generated for each candidate feature from $F_{Cand} \cup F_{ORG}$, where F_{ORG} represents the original feature of the dataset and F_{Cand} denotes generated feature by AutoFeat. This analysis is performed independently for each training dataset. The generated meta-features are later utilized to build the training data for the meta-model. In the third stage, labels ("good" or "bad") are assigned to each candidate feature of the datasets. A RandomForestClassifier is utilized to calculate the error reduction for $F_{Cand} \cup F_{ORG}$. F_{Cand} is labeled as "good" if the error rate decreases by more than the threshold and "bad" otherwise. Figure 6 shows the established meta-learning space. On top of the established meta-learning space, XGBoost is used as a meta-model.

DATASET	EXTRACTED METADATA				Usefulness Score
abalone_Feature1	...	...	...	...	1
abalone_Feature2	...	...	...	...	0
abalone_Feature3	...	...	...	...	1
abalone_Feature4	...	...	...	...	0
GCM_Feature1	...	...	...	...	0
GCM_Feature2	...	...	...	...	1
GCM_Feature3	...	...	...	...	1
GCM_Feature4	...	...	...	...	0
GCM_Feature5	...	...	...	...	1

Figure 6. The established meta-learning space.

Meta-features	Description
$canCor$	Canonical correlations between the predictive attributes and the class
cor	Absolute attributes correlation
cov	Attributes covariance
$nrDisc$	The number of discriminant values
$eigenvalues$	Eigenvalues of the covariance matrix
$gMean$	Geometric mean of attributes
$hMean$	Harmonic mean of attributes
$iqRange$	Interquartile range of attributes
$kurtosis$	Kurtosis of attributes
mad	Median absolute deviation of attributes
max	Maximum value of attributes
$mean$	Mean value of attributes
$median$	Median value of attributes
min	Minimum value of attributes
$nrCorAttr$	Number of attributes pairs with high correlation
$nrNorm$	Number of attributes with normal distribution
$nrOutliers$	Number of attributes with outliers values
$range$	Range of Attributes
sd	Standard deviation of the attributes
$sdRatio$	Statistic test for homogeneity of covariances
$skewness$	Skewness of attributes
$tMean$	Trimmed mean of attributes
var	Attributes variance
$wLambda$	Wilks Lambda
$attrEnt$	Attributes entropy
$classEnt$	Class entropy
$eqNumAttr$	Equivalent number of attributes
$jointEnt$	Joint Entropy of attributes and classes
$mutInf$	Mutual information of attributes and classes
$nsRatio$	Noisiness of attributes

Table 5. Description of meta-features used by AutoFeat [RGS$^+$18].

3.2.2 Online Phase

Given a new dataset, Autofeat generates all possible features, and then the system extracts the same meta-features for each candidate feature as in the offline phase. The meta-model predicts the usefulness label for each candidate feature. If the feature is predicted as "good," the system keeps it as a generated feature; if not, the system discards it.

4 Evaluation

This section covers the evaluation of three main experiments and the experimental setup. The first experiment was conducted to see the impact of using different meta-features and select the one with the best predictive performance for the BigFeat. The second experiment illustrates the performance benchmarks of different AutoFE frameworks, including BigFeat, AutoFeat, and SAFE. The third experiment shows the evaluation of predictive performance and execution time of the AutoFeat.

4.1 Experimental Setup

4.1.1 Dataset

We utilized 17 datasets with varying numbers of instances and attributes from OpenML. As indicated in Table 6, the number of rows ranges from 190 to 96,300, whereas the number of features varies from 4 to 16.000.

Dataset	Instances	Features	Labels
amazon_employee_access	32770	9	2
ap_omentum_ovary	275	10936	2
arcene	200	10001	2
australian	690	15	2
blood-transfusion-service-center	6748	5	2
christine	5418	1637	2
credit-g	1000	21	2
egg-eye-state	14980	15	2
GCM	190	16064	14
gina	3153	971	2
kc1	2109	22	2
madelon	2600	501	2
micro-mass	360	1301	10
nomao	34465	119	2
numerai28	96321	21	2
phoneme	5404	6	2
shuttle	58000	10	7

Table 6. The information of the benchmark datasets.

4.1.2 Hardware Configuration

We did our studies on a CPU environment that runs on Ubuntu 18.04 LTS with a 64-core Intel Processor @ 2.00GHz and 240 GB of RAM.

4.1.3 Software Configuration

We used eight different state-of-the-art machine learning algorithms. All the algorithms are used with their default parameters, which are defined in scikit-learn [PVG+11]. We

chose F1-Score to evaluate the effectiveness of these designed ML pipelines.

Throughout the experiment "Performance Benchmarks of Meta-features in FE", we analyzed different types of meta-features that were extracted from datasets. In order to choose the best meta-features, we compared the performance result of the BigFeat with different meta-features. For simplicity, we used a few operators, including arithmetic addition, subtraction, absolute, multiplication, and square.

- LMRKS- Landmarking meta-features extracted from datasets.
- CMBD- Combined meta-features extracted from datasets.
- T_based- Tree-based meta-features extracted from datasets.
- CMBD ∪ LMRKS- Concatenating landmarking and combined meta-features.
- CMBD ∪ T_Based- Combining tree-based and combined meta-features.
- LMRKS ∪ T_based- Combining landmarking and tree-based meta-features.
- W_AVG- BigFeat uses a weighted operator importance vector that comes from 3 different meta-learning techniques (tree_based, landmarking, and combined).

In the second experiment "Benchmark of Automated Feature Engineering Frameworks", the best-performing meta-features are integrated into BigFeat and compared with other feature engineering frameworks, including SAFE and AutoFeat. The default configuration parameters are used for AutoFeat and SAFE in each execution. The following configuration parameters are used for BigFeat: n=0.9, k = 10, $\alpha = 5, Iterations = 5, IterationHeight(Depth)list = [1, 1, 1, 1, 1, 1, 2]$.

- ORG- Datasets without feature engineering.
- BigFeat- Feature engineering performed by BigFeat.
- AutoFeat- Feature engineering performed by AutoFeat.
- BigFeat_M- Feature engineering performed by meta-learning based BigFeat.
- SAFE- Feature engineering performed by SAFE.

In the third experiment "Performance Evaluation of Meta-learning Based AutoFeat", we make a comparison of performance and execution time between AutoFeat and meta-learning based AutoFeat over ten datasets. For simplicity, in each execution, while $featen g\_steps = 1$ and $featsel\_runs = 5$ configuration parameters are used for the AutoFeat, $featen g\_steps = 1$ and $featsel\_runs = 0$ are used for meta-learning based AutoFeat.

- AutoFeat- Feature engineering performed by AutoFeat.
- AutoFeat_M- Feature engineering performed by meta-learning based AutoFeat.

4.2 Performance Evaluation of Meta-learning Based BigFeat

4.2.1 Performance Benchmarks of Meta-features in FE

The primary purpose of this experiment is to find the meta-features that enable BigFeat to achieve the best performance evaluation score. For that, we evaluate the performance of generated features from the BigFeat with seven different meta-features on eight different classification models, which are Decision Tree (DT), Multi-Layered Perceptron (MLP), Random Forest (RF), k-Nearest Neighbors (kNN), Logistic Regression(LR), Extremely randomized Trees (ET), AdaBoost (AB), and gradient boosting (GB).

Table 10 in Appendix shows the performance of eight different types of classification algorithms for all seventeen datasets. The results show that the BigFeat with W_AVG outperforms the BigFeat with other meta-features on most datasets. It has the best average F-1 Score on 9 datasets out of 17. According to this experiment, the W_AVG based meta-learning technique is selected and integrated into BigFeat for further analysis.

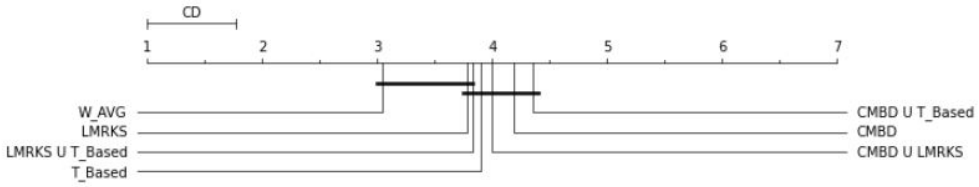

Figure 7. Nemenyi Test ($\alpha = 0.05$).

We performed the Nemenyi test over the performance results obtained in Table 10 to see whether the results were significant or not. As can be seen in Figure 7, W_AVG achieves the best results compared to the others, but we also noticed that while the results of W_AVG is significantly better than T_based, CMBD $\cup$ LMRKS, CMBD $\cup$ T_Based, CMDB, it is not significantly different than LMRKS, LMRKS $\cup$ T_based.

4.2.2 Benchmark of Automated Feature Engineering Frameworks

In this experiment, We compared the performance results of meta-learning based BigFeat, BigFeat without meta-learning, SAFE, and AutoFeat on the state-of-the-art learning algorithms, which are Decision Tree (DT), Multi-Layered Perceptron (MLP), Random Forest (RF), k-Nearest Neighbors (kNN), Logistic Regression(LR), Extremely randomized Trees (ET), AdaBoost (AB), and gradient boosting (GB).

Table 7 illustrates the performance of eight classification algorithms across all seventeen datasets for which feature engineering was performed using the indicated FE frameworks. The results demonstrate that the BigFeat_M system surpasses ORG, BigFeat, AutoFeat, and SAFE systems on most datasets, with an average improvement of 5.62 % , 1.51 %, 9.02 %, and 4.46 % over ORG, BigFeat, AutoFeat, and SAFE systems, respectively.

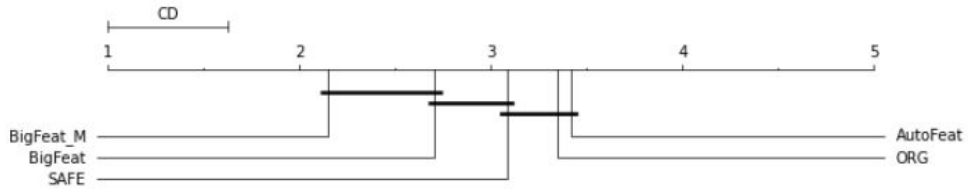

Figure 8. Nemenyi Test ($\alpha = 0.05$).

Figure 8 shows the Nemenyi test's result over the performance evaluation of FE frameworks. Based on the figure, we can conclude that BigFeat_M outperforms all the other frameworks. It also illustrates that while the performance result of BigFeat_M is not significantly different than BigFeat, it is significantly different than ORG, SAFE, and AutoFeat.

DATASETS	CLF	ORG	BigFeat	BigFeat_M	AutoFeat	SAFE
Amazon_employee	LR	**97.25**	97.23	97.23	97.23	97.23
	AB	97.23	97.23	**97.24**	97.23	97.23
	DT	**97.23**	96.31	96.2	96.19	95.86
	ET	96.25	96.89	97.11	97.07	**97.32**
	KNN	97.08	**97.16**	97.06	97.05	97.08
	MLP	97.05	97.24	**97.27**	96.49	94.26
	RF	96.81	97.13	97.4	97.42	**97.45**
	GB	97.23	97.25	**97.26**	**97.26**	97.24
	Avg	97.02	97.06	**97.10**	96.99	96.71
Ap_omentum	LR	89.13	89.13	**91.11**	90.1	89.66
	AB	88.89	87.91	90.32	85.39	**90.91**
	DT	85.39	85.06	84.09	80	**90.91**
	ET	76.32	88.89	87.41	89.36	**90.32**
	KNN	**88.42**	86.05	87.91	83.95	86.36
	MLP	83.95	89.89	86.96	21.28	**88.64**
	RF	86.6	90.11	**91.23**	88.17	88.42
	GB	88.42	89.13	**92.31**	92.13	89.89
	Avg	85.89	88.27	88.92	78.80	**89.39**
Arcene	LR	68.18	81.82	81.74	–	**82.61**
	AB	**80.95**	71.11	69.41	–	79.07
	DT	**68.29**	63.64	67.26	–	59.09
	ET	60.87	68.18	72.73	–	**76.19**
	KNN	71.43	**82.93**	80.95	–	66.67
	MLP	**86.96**	70.97	83.26	–	55.56
	RF	70.97	68.18	**73.91**	–	69.77
	GB	**74.42**	68.18	70.41	–	71.11
	Avg	72.76	71.88	**74.96**	–	70.01
Australian	LR	83.87	89.6	90.41	65.6	**90.77**
	AB	86.18	85.25	**87.3**	85	86.4
	DT	**85.04**	78.63	83.62	79.4	77.69
	ET	78.99	83.33	**88.9**	86	88
	KNN	**87.8**	**87.8**	86.1	55.9	79.69
	MLP	67.77	**91.94**	88.1	36	77.05
	RF	78.26	86.18	**88**	86.9	87.8
	GB	63.64	83.87	84.1	81.7	**85.25**
	Avg	78.94	85.83	**87.07**	72.06	84.08
Blood-transfusion	LR	**48.57**	20.83	21.38	24.49	42.11
	AB	24.49	50	**53.76**	51.61	48.57
	DT	**51.61**	47.89	48.44	44.8	34.78
	ET	39.44	41.79	**44.78**	42.9	41.18
	KNN	42.25	52.31	51.48	46.9	**53.13**
	MLP	54.55	36.36	40.48	40	**52.78**
	RF	28	**50.75**	48.48	46.4	43.48
	GB	4.76	48.57	**53.56**	43.8	48.48
	Avg	36.71	43.56	45.30	42.61	**45.56**

Table 7. Performance comparison of BigFeat (BF), original base features (ORG), Meta-learning based BigFeat (BigFeat_M), AutoFeat, and SAFE on different classifiers across different datasets. The system outperforms all other frameworks is presented in **bold**. '–' represents the failure during the execution.

DATASETS	CLF	ORG	BigFeat	BigFeat_M	AutoFeat	SAFE
	LR	**72.98**	67.19	69.5	–	72.69
	AB	68.34	**71.46**	69.89	–	68.84
	DT	**70.58**	65.31	63.09	–	63.19
Christine	ET	63.38	71.81	**73.03**	–	72.46
	KNN	**74.05**	72.44	73.38	–	68.34
	MLP	70.59	68.57	**71.48**	–	69.88
	RF	0	71.53	72.37	–	**72.83**
	GB	74	72.98	**74.54**	–	73.01
	Avg	61.74	70.16	**70.91**	–	70.16
	LR	88.28	**89.04**	88.88	58.91	**89.04**
	AB	89.8	89.47	**89.86**	87.59	88.44
	DT	**87.59**	80	85.03	82.67	81.63
Credit-g	ET	84.35	88.59	89.8	89.04	**92**
	KNN	**92**	85.71	91.92	70.89	85.52
	MLP	72.96	89.19	**90.74**	69.35	81.12
	RF	78.83	88.89	88.89	88.44	**89.19**
	GB	73.62	88.28	**90.54**	**90.54**	88.89
	Avg	83.43	87.40	**89.46**	79.68	86.98
	LR	**82.27**	60.12	62.56	41.31	46.99
	AB	54.85	74.62	**75.49**	69.29	73.33
	DT	69.29	83.99	84.05	83.08	**84.3**
Egg-eye-state	ET	82.59	**95.11**	94.81	94.29	93.53
	KNN	94.39	**96.29**	96.22	62.26	83.56
	MLP	**95.8**	89.07	94.96	62.71	66.82
	RF	81.58	**93.51**	92.44	92.52	92.11
	GB	78.51	82.27	**85.35**	77.37	80.95
	Avg	79.91	84.37	**85.74**	72.85	77.70
	LR	**76.4**	74.74	75.25	32.5	–
	AB	**27.57**	22.92	23.6	18.3	–
	DT	40.26	49.19	**55.51**	45.8	–
GCM	ET	64.28	**69.47**	65.79	62.4	–
	KNN	**54.72**	50.85	53	27.7	–
	MLP	0.53	64.41	**77.11**	40.3	–
	RF	63.77	66.88	65.88	**71.9**	–
	GB	63.49	59.42	**72.25**	61.3	–
	Avg	48.88	57.24	**61.05**	45.03	–
	LR	**91.54**	91.21	90.24	77.85	86
	AB	77.85	**89.59**	88.7	83.85	86
	DT	83.85	86.37	**88.41**	85.11	84
Gina	ET	83.97	93.33	92.38	93.88	**94**
	KNN	**93.9**	92.8	92.89	83.36	83
	MLP	83.36	95.08	**95.24**	84.85	88
	RF	87.5	91.71	91.18	**92.87**	92
	GB	90.39	91.54	90.94	91.51	**93**
	Avg	86.55	**91.45**	91.25	86.66	88.25
	LR	32.56	**38.46**	33.33	37.21	25
	AB	37.21	26.37	**39.08**	34.57	36.36
	DT	34.57	**40.94**	40.65	33.61	38.71
Kc1	ET	36.97	40.43	42.22	41.76	**43.3**
	KNN	**47.31**	36.17	46.43	29.55	26.67
	MLP	29.55	36.78	36.14	**41.51**	8.96
	RF	14.93	40.86	38	41.3	**45.83**
	GB	26.32	32.56	41.76	36.14	**46.32**
	Avg	32.43	36.57	**39.70**	36.96	33.89

Table 7. Continued.

DATASETS	CLF	ORG	BigFeat	BigFeat_M	AutoFeat	SAFE
Madelon	LR	**85.94**	68.45	84.53	55.95	66.42
	AB	56.24	**81.78**	79.59	62.19	73.13
	DT	62.19	77.19	**77.37**	75.4	74.86
	ET	77.29	**89.06**	87.89	70.24	80.07
	KNN	72.97	87.4	**89.25**	72.01	84.17
	MLP	72.01	80.23	**81.99**	56.3	56.3
	RF	30.21	87.6	**87.96**	73	81.63
	GB	67.63	85.94	**87.92**	76.15	84.45
	Avg	65.56	82.21	**84.56**	67.66	75.13
Micro-mass	LR	**96.02**	94.81	94.94	–	–
	AB	14.18	**19.22**	18.15	–	–
	DT	**86.94**	86.7	86.48	–	–
	ET	93.89	**94.78**	93.76	–	–
	KNN	88.74	**92.55**	91.22	–	–
	MLP	97.28	96.02	**97.96**	–	–
	RF	92.55	93.76	**94.94**	–	–
	GB	86.46	**92.05**	91.23	–	–
	Avg	82.01	**83.74**	83.59	–	–
Nomao	LR	**96.98**	96.3	96.42	92.95	93.46
	AB	93.2	96.09	**96.34**	96.16	96.07
	DT	96.16	96.31	96.21	**96.4**	96.23
	ET	96.52	97.83	97.86	97.98	**97.87**
	KNN	**97.91**	97.13	97	94.77	94.54
	MLP	94.96	97.29	**97.41**	94.53	94.37
	RF	95.37	97.71	**97.75**	97.86	97.66
	GB	93.82	**96.98**	96.94	96.89	96.9
	Avg	95.62	96.96	**96.99**	95.94	95.89
Numerai28	LR	**57.08**	56.22	56.22	56.9	56.67
	AB	**56.9**	55.66	55.32	55.7	55.78
	DT	**55.7**	50.29	55.57	50.16	50.83
	ET	50.49	51.01	51.2	51.92	**51.52**
	KNN	**51.76**	51.32	51.04	51.84	51.16
	MLP	51.84	51.55	56.93	3.92	**60.97**
	RF	**65.98**	51.31	64.63	52	51.38
	GB	57.17	57.08	57.1	56.96	**57.31**
	Avg	55.87	53.06	**56.00**	47.43	54.45
Phoneme	LR	**76.72**	55.26	75.06	0	–
	AB	43.56	71.87	**72.29**	70.18	–
	DT	70.18	**78.5**	76.66	77.67	–
	ET	77.41	85.58	**88.36**	85.57	–
	KNN	**85.67**	77.96	76.77	53.38	–
	MLP	**79.48**	74.76	71.71	44.41	–
	RF	67.8	84.49	**88.51**	83.58	–
	GB	74.05	76.72	**79.06**	74.83	–
	Avg	71.86	75.64	**78.55**	61.20	–
Shuttle	LR	96.73	96.77	**97.13**	86.82	56.62
	AB	87.93	89.84	**89.9**	87.93	75.82
	DT	99.97	**100**	99.99	99.97	78.7
	ET	99.97	99.96	99.95	**99.99**	84.74
	KNN	99.88	**99.93**	99.92	98.94	79.09
	MLP	**99.99**	99.96	99.97	97.03	68.94
	RF	**99.97**	99.96	99.96	**99.97**	83.44
	GB	**100**	99.99	**100**	**100**	78.7
	Avg	98.06	98.30	**98.35**	96.33	75.76

Table 7. Continued.

4.3 Performance Evaluation of Meta-feature Based AutoFeat

In this experiment, we compared the predictive performance and execution time of meta-learning based AutoFeat and AutoFeat on the state-of-the-art machine learning models (Decision Tree (DT), Multi-Layered Perceptron (MLP), Random Forest (RF), k-Nearest Neighbors (kNN), Logistic Regression(LR), Extremely randomized Trees (ET), AdaBoost (AB), and gradient boosting (GB)) over ten datasets.

Table 9 shows the performance of eight classification algorithms across all ten datasets for which feature engineering was performed using the FE frameworks specified. The results demonstrate that AutoFeat_M outperforms AutoFeat on eight out of ten datasets, with an average improvement of 1.1 %.

Datasets	AutoFeat	AutoFeat_M
Blood-transfusion	**2.915**	4.07
Australian	**12.18**	16.517
Egg-eye-state	466.59	**30.79**
Kc-1	**40.23**	59.51
Numerai28	287.36	**8.76**
Credit-g	**8.378**	14.972
Amazon-employee	87.324	**19.67**
Gina	1254.57	**65.08**
Shuttle	6494	**253.583**
Phoneme	12.35	**2.877**
Average	866.59	**47.58**

Table 8. Run time (seconds) of AutoFeat and AutoFeat_M across different datasets. The system with the lower execution time is presented in **bold**.

We performed Wilcoxon test ($\alpha = 0.05$) on the performance evaluation of AutoFeat. The calculated p-value (0.0169) implies that the performance results of AutoFeat and AutoFeat_M are statistically significant.

Table 8 shows the execution time of the two systems. While AutoFeat_M has lower run-time in six datasets, AutoFeat has four datasets. We also noticed that when the number of instances and attributes is relatively smaller, AutoFeat runs faster than AutoFeat_M; when the number of attributes and instances gets bigger, AutoFeat_M runs faster. This is because if the dataset is too large, Autofeat generates a lot of features in the feature-generation phase, and selecting informative features out of them requires more time in the feature-selection phase.

DATASETS	CLF	AutoFeat	AutoFeat_M	DATASETS	CLF	AutoFeat	AutoFeat_M
	LR	24.49	**30.19**		LR	**58.91**	56.90
	AB	51.61	51.61		AB	87.59	87.59
	DT	37.14	**40.00**		DT	83.56	83.56
	ET	41.18	**44.44**		ET	**90.54**	89.80
Blood-transfusion	KNN	**46.87**	41.27	Credit-g	KNN	70.89	**74.68**
	MLP	28.00	**29.41**		MLP	64.00	**69.01**
	RF	**50.00**	46.38		RF	89.80	89.80
	GB	43.75	43.75		GB	90.54	90.54
	Avg	40.38	**40.88**		Avg	79.48	**80.24**
	LR	65.59	**68.75**		LR	97.23	97.23
	AB	85.04	85.04		AB	97.23	97.23
	DT	**77.05**	75.21		DT	**96.23**	96.11
	ET	86.18	**89.43**		ET	97.07	**97.14**
Australian	KNN	55.86	**60.16**	Amazon_employee	KNN	97.05	**97.15**
	MLP	**74.07**	62.50		MLP	97.23	97.23
	RF	86.89	**88.89**		RF	97.39	**97.50**
	GB	80.99	**81.67**		GB	97.26	97.26
	Avg	76.46	76.46		Avg	97.09	**97.11**
	LR	44.10	**55.02**		LR	77.85	**80.48**
	AB	69.29	69.29		AB	83.85	83.85
	DT	**82.64**	82.42		DT	82.82	**84.29**
	ET	93.62	**93.94**		ET	93.09	**94.56**
Egg-eye-state	KNN	73.72	**84.57**	gina	KNN	**83.36**	82.55
	MLP	64.14	**68.66**		MLP	**87.39**	86.77
	RF	**92.68**	91.64		RF	92.74	**93.55**
	GB	77.37	**77.42**		GB	91.51	91.51
	Avg	74.70	**77.87**		Avg	86.58	**87.20**
	LR	**37.21**	27.85		LR	86.92	**96.65**
	AB	34.57	34.57		AB	87.93	87.93
	DT	36.07	**36.67**		DT	**99.97**	99.96
	ET	43.18	**43.96**		ET	**99.98**	99.96
Kc-1	KNN	**29.55**	23.16	Shuttle	KNN	98.94	**99.87**
	MLP	12.50	**35.97**		MLP	98.08	**99.96**
	RF	40.43	**46.94**		RF	**99.99**	99.97
	GB	36.14	36.14		GB	99.99	**100.00**
	Avg	33.71	**35.66**		Avg	96.48	**98.04**
	LR	56.90	56.90		LR	0.00	0.00
	AB	55.70	55.70		AB	70.18	70.18
	DT	50.23	**50.41**		DT	76.25	**78.63**
	ET	**51.69**	51.36		ET	83.03	**85.39**
Numerai28	KNN	51.84	51.84	Phoneme	KNN	53.38	53.38
	MLP	3.28	**62.76**		MLP	**44.56**	2.46
	RF	**52.39**	51.21		RF	83.20	**83.63**
	GB	56.96	56.96		GB	74.96	74.96
	Avg	47.37	**54.64**		Avg	**60.70**	56.08

Table 9. Performance comparison of AutoFeat and AutoFeat_M on different classifiers across different datasets. The system outperforms all other frameworks is presented in **bold**.

5 Conclusion

The focus of this study was to apply a meta-learning based approach to the field of FE to improve the predictive performance and execution time of the frameworks, namely BigFeat and AutoFeat.

We investigated seven different meta-learning techniques to warm-start an AutoFE framework (e.g., BigFeat), helping it boost its predictive performance. Our analysis shows that the weighted average technique outperforms the other meta-learning techniques. We integrated the meta-features used in this technique into BigFeat and compared the predictive performance results with different AutoFE frameworks (e.g., AutoFeat). The results show that warm starting BigFeat (BigFeat_M) has significantly improved the predictive performance of eight different classifiers. BigFeat_M has statistically significant performance difference compared to the state-of-the-art frameworks (SAFE, AutoFeat).

We integrated the meta-learning technique into AutoFeat to improve the framework's predictive performance and execution time by building a novel meta-learning based approach for feature selection. We compared the results of AutoFeat_M to AutoFeat on eight classifiers using ten different datasets. The results demonstrate that integrating the developed meta-learning technique has significantly improved the predictive performance of eight classifiers and the average execution time compared to AutoFeat.

For future work, we plan to release new versions of the AutoFE frameworks (BigFeat, AutoFeat) to contribute to the open source community. In addition, we intend to integrate BigFeat_M and AutoFeat_M into different AutoML frameworks (e.g., Auto-Sklearn and TPOT) and analyze their performance.

6 Acknowledgements

In this section, I would like to express my gratitude to everyone who helped me throughout my time as a student at the University of Tartu. I would like to give my warmest gratitude to my supervisors Dr. Radwa El Shawi and Hassan Eldeeb who made this work possible. Their advice and guidance took me through all the steps of writing my dissertation. I am grateful to my best friend Mustafa Oztroprak who has read and commented on several sections. I am thankful to my colleague Brandon Loorits who helped with the Estonian translation. I would also like to give special appreciation to my parents Olcay Kaya, Pervin Kaya, and my sister Ezgi Kaya, for their constant support and understanding when conducting my research and writing my thesis.

References

[ASR+20] Edesio Alcobaça, Felipe Siqueira, Adriano Rivolli, Luís P. F. Garcia, Jefferson T. Oliva, and André C. P. L. F. de Carvalho. Mfe: Towards reproducible meta-feature extraction. *Journal of Machine Learning Research*, 21(111):1–5, 2020.

[BAAB17] Besim Bilalli, Alberto Abelló, and Tomàs Aluja-Banet. On the predictive power of meta-features in OpenML. *International Journal of Applied Mathematics and Computer Science*, 27(4):697–712, dec 2017.

[BBE+20] André Biedenkapp, H Furkan Bozkurt, Theresa Eimer, Frank Hutter, and Marius Lindauer. Dynamic algorithm configuration: foundation of a new meta-algorithmic framework. In *ECAI 2020*, pages 427–434. IOS Press, 2020.

[BCV13] Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation learning: A review and new perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(8):1798–1828, 2013.

[BSPS21] Mohamadjavad Bahmani, Radwa El Shawi, Nshan Potikyan, and Sherif Sakr. To tune or not to tune? an approach for recommending important hyperparameters. *CoRR*, abs/2108.13066, 2021.

[DHS11] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159, 2011.

[Dom12] Pedro Domingos. A few useful things to know about machine learning. *Communications of the ACM*, 55(10):78–87, oct 2012.

[DSR+15] Sander Dieleman, Jan Schlüter, Colin Raffel, Eben Olson, Søren Kaae Sønderby, Daniel Nouri, et al. Lasagne: First release., August 2015.

[ELS21] Radwa ElShawi, Hudson Lekunze, and Sherif Sakr. csmartml: A meta learning-based framework for automated selection and hyperparameter tuning for clustering. In *2021 IEEE International Conference on Big Data (Big Data)*, pages 1119–1126. IEEE, 2021.

[ESAMS19] Radwa El Shawi, Mouaz Al-Mallah, and Sherif Sakr. On the interpretability of machine learning-based model for predicting hypertension. *BMC Medical Informatics and Decision Making*, 19, 07 2019.

[FBTS21] Manuel Fritz, Michael Behringer, Dennis Tschechlov, and Holger Schwarz. Efficient exploratory clustering analyses in large-scale exploration processes. *The VLDB Journal*, 11 2021.

[FEF+20] Matthias Feurer, Katharina Eggensperger, Stefan Falkner, Marius Lindauer, and Frank Hutter. Auto-sklearn 2.0: The next generation. *arXiv preprint arXiv:2007.04074*, 24, 2020.

[FSH14] Matthias Feurer, Jost Tobias Springenberg, and Frank Hutter. Using meta-learning to initialize bayesian optimization of hyperparameters. In *MetaSel@ECAI*, 2014.

[GPS+10] T.A.F. Gomes, Ricardo Prudêncio, Carlos Soares, André Rossi, and Andre de Carvalho. Combining meta-learning and search techniques to svm parameter selection. pages 79 – 84, 11 2010.

[GS10] Romaric Gaudel and Michèle Sebag. Feature selection as a one-player game. *International Conference on Machine Learning*, 06 2010.

[HKV19] Frank Hutter, Lars Kotthoff, and Joaquin Vanschoren. *Automated Machine Learning: Methods, Systems, Challenges*. Springer Publishing Company, Incorporated, 1st edition, 2019.

[HPR20] Franziska Horn, Robert Pack, and Michael Rieger. *The autofeat Python Library for Automated Feature Engineering and Selection*, pages 111–120. 03 2020.

[HTF09] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction, Second Edition (Springer Series in Statistics)*. 02 2009.

[KB14] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014.

[KH00] A. Kalousis and M. Hilario. Model selection via meta-learning: a comparative study. In *Proceedings 12th IEEE Internationals Conference on Tools with Artificial Intelligence. ICTAI 2000*, 2000.

[KH01] Alexandros Kalousis and Melanie Hilario. Model selection via meta-learning: A comparative study. *International Journal on Artificial Intelligence Tools*, 10:525–554, 12 2001.

[KMP17] Ambika Kaul, Saket Maheshwary, and Vikram Pudi. Autolearn — automated feature generation and selection. In *2017 IEEE International Conference on Data Mining (ICDM)*, pages 217–226, 2017.

[KSS17a] Gilad Katz, Eui Chul Richard Shin, and Dawn Song. Proceedings - IEEE International Conference on Data Mining, ICDM, pages 979–984, United States, January 2017. Institute of Electrical and Electronics Engineers Inc.

[KSS17b] Gilad Katz, Eui Chul Richard Shin, and Dawn Song. Explorekit: Automatic feature generation and selection. Proceedings - IEEE International Conference on Data Mining, ICDM, pages 979–984, United States, January 2017. Institute of Electrical and Electronics Engineers Inc.

[KST17] Udayan Khurana, Horst Samulowitz, and Deepak Turaga. Feature engineering for predictive modeling using reinforcement learning. 09 2017.

[LLT21] Yue Liu, Shuang Li, and Wenjie Tian. *AutoCluster: Meta-learning Based Ensemble Method for Automated Unsupervised Clustering*, pages 246–258. 05 2021.

[LUTG17] Brenden M. Lake, Tomer D. Ullman, Joshua B. Tenenbaum, and Samuel J. Gershman. Building machines that learn and think like people. *Behavioral and Brain Sciences*, 40:e253, 2017.

[MKF+] Hector Mendoza, Aaron Klein, Matthias Feurer, Jost Tobias Springenberg, Matthias Urban, Michael Burkart, Max Dippel, Marius Lindauer, and Frank Hutter. Towards automatically-tuned deep neural networks. pages 135–149.

[MR02] Shaul Markovitch and Dan Rosenstein. Feature generation using general constructor functions. *Machine Learning*, 49:59–98, 10 2002.

[OM16] Randal S Olson and Jason H Moore. Tpot: A tree-based pipeline optimization tool for automating machine learning. In *Workshop on automatic machine learning*, pages 66–74. PMLR, 2016.

[PDK20] Yannis Poulakis, Christos Doulkeridis, and Dimosthenis Kyriazis. Autoclust: A framework for automated clustering based on cluster validity indices. In *2020 IEEE International Conference on Data Mining (ICDM)*, pages 1220–1225, 2020.

[PVG+11] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *Journal of machine learning research*, 12(Oct):2825–2830, 2011.

[RDS+14] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael S.

Bernstein, Alexander C. Berg, and Li Fei-Fei. Imagenet large scale visual recognition challenge. *CoRR*, abs/1409.0575, 2014.

[RGS+18] Adriano Rivolli, Luís Paulo F. Garcia, Carlos Soares, Joaquin Vanschoren, and André C. P. L. F. de Carvalho. Towards reproducible empirical research in meta-learning. *CoRR*, abs/1808.10406, 2018.

[SZL+20] Qitao Shi, Ya-Lin Zhang, Longfei Li, Xinxing Yang, Meng Li, and Jun Zhou. Safe: Scalable automatic feature engineering framework for industrial tasks. pages 1645–1656, 04 2020.

[THHL12] Chris Thornton, Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. Auto-weka: Automated selection and hyper-parameter optimization of classification algorithms. *CoRR*, abs/1208.3719, 2012.

[Van18] Joaquin Vanschoren. Meta-learning: A survey, 10 2018.

[VvRBT13] Joaquin Vanschoren, Jan N. van Rijn, Bernd Bischl, and Luis Torgo. Openml: networked science in machine learning. *SIGKDD Explorations*, 15(2):49–60, 2013.

[XDH+16] Wayne Xiong, Jasha Droppo, Xuedong Huang, Frank Seide, Mike Seltzer, Andreas Stolcke, Dong Yu, and Geoffrey Zweig. Achieving human parity in conversational speech recognition. *CoRR*, abs/1610.05256, 2016.

[Zei12] Matthew D. Zeiler. ADADELTA: an adaptive learning rate method. *CoRR*, abs/1212.5701, 2012.

Appendix

Performance Bencmarks of Meta-features in FE

DATASETS	CLF	LMRKS	CMBD	T_Based	CMBD ∪ LMRKS	CMBD ∪ T_Based	LMRKS ∪ T_Based	W_AVG
	LR	97.23	97.23	97.23	97.23	97.23	97.23	97.23
	AB	**97.24**	97.22	97.23	97.23	97.23	97.24	**97.24**
	DT	96.15	96.22	**96.35**	96.07	96.06	96	96.2
Amazon_employee	ET	97.1	97.1	**97.14**	97.01	97.11	**97.08**	97.11
	KNN	97.03	97.05	97.07	97.05	**97.08**	97.06	97.06
	MLP	97.24	**97.27**	97.25	97.25	97.24	97.25	**97.27**
	RF	**97.47**	97.36	97.39	97.35	97.44	97.42	97.4
	GB	97.27	97.29	97.3	97.24	97.27	97.27	97.26
	Avg	97.09	97.09	**97.12**	97.05	97.08	97.07	97.10
	LR	87.91	88.17	85.71	90.11	86.96	**91.3**	91.11
	AB	87.36	88.89	**93.33**	89.89	85.71	92.13	90.32
	DT	84.71	**87.06**	84.34	75.95	80	85.06	84.09
AP_Omentum	ET	87.64	86.96	87.91	88.89	82.22	**91.3**	87.41
	KNN	88.89	89.41	85.06	86.36	82.22	**89.66**	87.91
	MLP	89.13	89.13	86.96	**92.13**	91.11	86.6	86.96
	RF	89.13	90.11	91.11	91.11	85.39	**92.31**	91.23
	GB	91.3	89.66	88.89	87.91	86.67	90.11	**92.31**
	Avg	88.26	88.67	87.91	87.79	85.04	**89.81**	88.92
	LR	79.07	75	**83.72**	78.05	76.92	79.07	81.74
	AB	72.73	**74.42**	65.12	66.67	**74.42**	60	69.41
	DT	69.77	58.54	69.77	68.29	**74.42**	69.57	67.26
Arcene	ET	73.91	72.73	73.47	71.11	**76.19**	72.73	72.73
	KNN	75	79.07	77.27	74.42	80	75	**80.95**
	MLP	79.07	80	70	**76.19**	79.07	73.91	**83.26**
	RF	70.83	71.43	75	69.77	**78.05**	68.18	73.91
	GB	73.17	58.54	**77.55**	68.18	72.73	66.67	70.41
	Avg	74.19	71.22	73.99	71.59	**76.48**	70.64	74.96
	LR	89.6	**91.34**	91.06	91.06	88.52	90.32	90.41
	AB	82.35	84.03	85.25	86.18	86.89	85.95	**87.3**
	DT	82.05	74.38	81.03	84.13	82.54	**85.48**	83.62
Australian	ET	88	87.1	86.89	86.89	88.71	85.95	**88.9**
	KNN	86.18	85.95	**87.6**	**87.6**	85	85.25	86.1
	MLP	88.52	90	89.6	90.48	**91.94**	90.16	88.1
	RF	87.3	86.18	86.18	85.25	86.18	86.18	**88**
	GB	86.4	82.35	**88.71**	83.87	85	86.4	84.1
	Avg	86.30	85.17	87.04	86.93	86.85	86.96	**87.07**
	LR	20.83	20.83	20.41	**24.49**	**24.49**	24	21.38
	AB	41.94	47.62	49.23	49.18	43.75	41.94	**53.76**
	DT	39.47	41.1	41.79	40	43.24	46.15	**48.44**
Blood-transfusion	ET	42.42	**48.57**	44.12	40	42.86	38.81	**44.78**
	KNN	45.45	52.94	55.07	**55.88**	51.52	50	51.48
	MLP	**49.18**	37.74	39.29	39.29	**36.36**	36.36	40.48
	RF	52.17	**56.72**	50	47.06	48.48	46.88	**48.48**
	GB	51.52	**55.07**	51.52	54.55	49.23	45.9	53.56
	Avg	42.87	45.07	43.93	43.81	42.49	41.26	**45.30**

Table 10. Performance comparison of BigFeat with different meta-features on different classifiers across different datasets.The system outperforms all other meta-features is presented in **bold**.

DATASETS	CLF	LMRKS	CMBD	T_Based	CMBD ∪ LMRKS	CMBD ∪ T_Based	LMRKS ∪ T_Based	W_AVG
	LR	67.7	67.2	69.44	67.25	68.23	68.42	**69.5**
	AB	69.64	68.23	70.48	69.24	68.89	**70.92**	69.89
	DT	63.14	58.79	**65.48**	62.94	61.35	64.08	63.09
Christine	ET	72.82	71.3	71.65	72.33	71.67	72.03	**73.03**
	KNN	71.77	72.59	72.38	72.44	72.01	73.1	**73.38**
	MLP	69.8	68.95	68.67	68.32	69	69.51	**71.48**
	RF	71.99	70.92	71.58	**72.6**	70.95	71.22	72.37
	GB	73.83	72.25	72.48	72.29	72.3	73.22	**74.54**
	Avg	70.09	68.78	70.27	69.68	69.30	70.31	**70.91**
	LR	**90.28**	89.04	89.19	88.44	87.5	87.5	88.88
	AB	86.84	83.92	86.9	85.52	78.87	81.12	**89.86**
	DT	83.56	77.46	81.05	81.08	78.83	82.86	**85.03**
Credit-g	ET	**90.54**	**90.54**	88.89	89.8	89.66	89.66	89.8
	KNN	90.67	88.28	89.93	91.89	86.71	91.16	**91.92**
	MLP	90.54	89.19	88.44	89.8	88.44	88.44	**90.74**
	RF	**91.28**	87.67	88.11	**87.5**	88.73	87.5	88.89
	GB	89.93	87.07	87.5	85.11	85.92	89.66	**90.54**
	Avg	89.21	86.65	87.50	87.39	85.58	87.24	**89.46**
	LR	61.42	60.05	61.62	55.7	62.52	58.57	**62.56**
	AB	**75.91**	75.18	74.46	74.25	74.59	74.17	75.49
	DT	83.59	**85.4**	83.7	84.38	82.32	83.68	84.05
Egg-eye-state	ET	**95.91**	94.83	95.27	94.96	94.65	94.83	94.81
	KNN	**96.44**	96.09	95.64	95.81	95.67	96.02	96.22
	MLP	90.23	90.42	89.37	87.97	88.91	89.3	**94.96**
	RF	**94.13**	93.87	92.95	92.36	92.73	93.28	92.44
	GB	82.57	82.76	81.18	81.44	83.12	81.94	**85.35**
	Avg	85.03	84.83	84.27	83.36	84.31	83.97	**85.74**
	LR	74.82	75.79	75.26	75.25	77.13	**77.54**	75.25
	AB	20.83	22.99	23.96	22.99	22.92	**26.55**	23.6
	DT	48.36	46.58	45.47	**56.54**	45.13	49.02	55.51
GCM	ET	65.37	69.5	62.69	69.01	**70.61**	65.87	65.79
	KNN	52.1	**55.43**	46.78	56.27	48.31	47.72	53
	MLP	22.66	39.02	23.28	22.68	23.81	2.81	**77.11**
	RF	69.72	74.04	59.11	71.97	68.68	**77.37**	65.88
	GB	50.28	68.19	62.96	67.19	**75.07**	48.59	72.25
	Avg	50.52	56.44	49.94	55.24	53.96	49.43	**61.05**
	LR	89.7	90.05	89.75	89.56	90.12	**91.19**	90.24
	AB	90	88.66	87.33	88.33	**91.12**	88.74	88.7
	DT	85.47	86.04	86.49	86.62	85.43	86.65	**88.41**
Gina	ET	91.95	93.01	92.08	91.88	**93.29**	92.83	92.38
	KNN	91.72	92	91	92.07	92.87	92.75	**92.89**
	MLP	94.91	94.37	95.03	94.55	**95.38**	95.33	95.24
	RF	91.38	91.68	91.09	90.52	**92.18**	91.5	91.18
	GB	91.83	91.72	90.64	91.85	**93.11**	92.26	90.94
	Avg	90.87	90.94	90.43	90.67	**91.69**	91.41	91.25
	LR	37.5	35	37.5	35.9	37.04	**37.97**	33.33
	AB	38.2	31.11	**39.22**	33.33	30.95	23.91	39.08
	DT	40	**45.16**	44.26	40.63	36.07	44.12	40.65
Kc1	ET	39.53	40.45	**45.45**	40.91	43.48	38.71	42.22
	KNN	35.79	40.43	27.91	38.71	35.79	38.38	**46.43**
	MLP	**40.48**	36.78	38.55	37.5	38.1	35.29	36.14
	RF	46.67	40.82	**47.31**	46.32	42.55	39.18	38
	GB	**47.31**	39.13	40.86	38.2	42.35	40.43	41.76
	Avg	**40.69**	38.61	40.13	38.94	38.29	37.25	39.70

Table 10. Continued.

DATASETS	CLF	LMRKS	CMBD	T_Based	CMBD ∪ LMRKS	CMBD ∪ T_Based	LMRKS ∪ T_Based	W_AVG
	LR	67.57	63.5	68.07	67.3	65.9	68.94	**84.53**
	AB	79.85	77.28	**81.78**	80.78	78.48	80.47	79.59
	DT	75.61	74.85	78.26	**78.67**	77.82	76.34	77.37
Madelon	ET	87.62	87.22	**89.02**	88.97	86.74	87.04	87.89
	KNN	**89.62**	86.42	88.58	89.49	84.37	87.33	89.25
	MLP	80.72	76.5	81.15	81.19	77.67	78.87	**81.99**
	RF	86.96	87.12	87.69	87.81	87.4	87.74	**87.96**
	GB	86.36	85.18	85.71	86.76	86.32	85.17	**87.92**
	Avg	81.79	79.76	82.53	82.62	80.59	81.49	**84.56**
	LR	96.02	95.14	96.33	**97.42**	93.85	93.56	94.94
	AB	16.59	6.41	**6.68**	15.71	15.92	6.68	**18.15**
	DT	80.95	88.29	86.29	89.03	85.31	**92.52**	86.48
Micro-mass	ET	**96.16**	94.94	**92.09**	94.78	92.03	93.48	93.76
	KNN	90.16	**92.55**	90.14	91.54	86.72	89.67	91.22
	MLP	96.16	97.23	97.42	97.42	94.98	96.16	**97.96**
	RF	90.75	89.89	92.09	94.78	87.01	92.08	**94.94**
	GB	90.34	88.07	88.67	90.36	**91.63**	87.77	91.23
	Avg	82.14	81.57	81.21	**83.88**	80.93	81.49	83.59
	LR	96.52	96.43	**96.58**	96.43	96.22	96.54	96.42
	AB	96.14	96.3	**96.42**	96.28	96.31	96.2	96.34
	DT	96.28	95.91	96.07	**96.31**	96.21	96.19	96.21
Nomao	ET	97.72	97.7	97.65	97.78	97.84	97.83	**97.86**
	KNN	97.14	97.05	96.87	97.13	96.96	**97.19**	97
	MLP	97.31	**97.51**	97.17	97.47	97.15	97.35	97.41
	RF	97.66	**97.77**	97.63	97.69	97.72	97.75	97.75
	GB	**97.04**	96.85	96.82	96.99	96.89	97.01	96.94
	Avg	96.98	96.94	96.90	**97.01**	96.91	**97.01**	96.99
	LR	**56.77**	56.38	55.83	55.88	55.77	56.13	56.22
	AB	55.17	55.93	55.37	**56.24**	55.71	56	55.32
	DT	50.22	50.28	51.31	50.23	50	51.06	**55.57**
Numerai28	ET	**51.87**	51.08	51.25	51.76	51.46	51.51	51.2
	KNN	51.12	51.17	51.72	50.8	**52.08**	51.54	51.04
	MLP	56.35	56.42	54.34	55.68	53.25	53.11	**56.93**
	RF	51.97	51.68	51.68	51.77	51.94	51.34	**64.63**
	GB	56.8	56.78	56.45	57.22	56.4	**57.41**	57.1
	Avg	53.78	53.72	53.49	53.70	53.33	53.51	**56.00**
	LR	56.04	54.93	56.54	57.84	54.74	55.58	**75.06**
	AB	72.49	71.82	73.7	71.69	72.79	**75.24**	72.29
	DT	77.67	76.63	78.68	**78.8**	76.45	78.04	76.66
Phoneme	ET	84.98	85.95	84.59	86.13	85.25	85.85	**88.36**
	KNN	78.1	**79.55**	78.1	77.98	77.27	79.08	76.77
	MLP	73.1	73.99	**76.18**	73.39	73.85	74.51	71.71
	RF	83.52	85.25	83.33	84.07	85.53	84.88	**88.51**
	GB	76	77.83	79.67	76.72	78.62	**79.74**	79.06
	Avg	75.24	75.74	76.35	75.83	75.56	76.62	**78.55**
	LR	**97.58**	96.8	96.97	96.89	96.96	97.17	97.13
	AB	89.84	89.84	**90.25**	89.84	89.84	**89.84**	89.9
	DT	**100**	99.99	99.99	99.99	99.99	99.99	99.99
Shuttle	ET	**99.96**	99.95	99.94	99.95	99.95	**99.96**	99.95
	KNN	99.92	99.92	99.92	**99.94**	**99.94**	99.93	99.92
	MLP	99.96	**99.97**	**99.97**	99.96	99.95	**99.97**	**99.97**
	RF	99.96	99.96	99.96	99.96	99.96	99.96	99.96
	GB	99.99	99.98	99.98	99.99	99.99	**100**	**100**
	Avg	**98.40**	98.30	98.37	98.32	98.32	98.35	98.35

Table 10. Continued.

II. Licence

Non-exclusive licence to reproduce thesis and make thesis public

I, **Kayahan Kaya**,
(author's name)

1. herewith grant the University of Tartu a free permit (non-exclusive licence) to

   reproduce, for the purpose of preservation, including for adding to the DSpace digital archives until the expiry of the term of copyright,

   Improving Automated Feature Engineering Using Meta-learning Based Techniques,
   (title of thesis)

   supervised by Radwa El Shawi and Hassan Eldeeb.
   (supervisor's name)

2. I grant the University of Tartu a permit to make the work specified in p. 1 available to the public via the web environment of the University of Tartu, including via the DSpace digital archives, under the Creative Commons licence CC BY NC ND 3.0, which allows, by giving appropriate credit to the author, to reproduce, distribute the work and communicate it to the public, and prohibits the creation of derivative works and any commercial use of the work until the expiry of the term of copyright.

3. I am aware of the fact that the author retains the rights specified in p. 1 and 2.

4. I certify that granting the non-exclusive licence does not infringe other persons' intellectual property rights or rights arising from the personal data protection legislation.

Kayahan Kaya
08/08/2022