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

Shota Amashukeli

Toward Automatic Construction of Machine Learning Pipelines

Master's Thesis (30 ECTS)

Supervisor(s): Radwa Elshawi, PhD
Hassan Eldeeb, MSc

Tartu 2021

Toward Automatic Construction of Machine Learning Pipelines

Abstract:
The rapid increase in popularity and demand for machine learning solutions has resulted in rising of the automated machine learning (AutoML) field. AutoML aims to automate the process of building machine learning pipelines by optimizing each component. Most of the current automated machine learning frameworks focus on automating the algorithm selection and hyper-parameter optimization problem with a limited focus on automating the feature engineering which is a key value-adding step that aims to construct informative features automatically and reduce manual labor for building well-performing machine learning pipelines. In addition, most of the current automated machine learning frameworks generate pipelines without human intervention. In practice, completely excluding the human from the loop creates several limitations. For example, most of these approaches ignore the user-preferences on defining or controlling the search space which consequently can impact the acceptance of the returned models by the end-users. The contribution of this thesis is twofold: 1) We design and implement *iSmartML*, an interactive visualization tool that supports users in controlling the search space of AutoML and analyzing and explaining the results. 2) We design and implement *BigFeat*, a scalable automated feature engineering tool.

Keywords:
Machine learning, AutoML, feature engineering

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

Automaatse masinõppe torujuhtmete ehitamise suunas

Lühikokkuvõte:
Masinõppelahenduste populaarsuse ja nõudluse kiire kasvu tulemuseks on automatiseeritud masinõppe (AutoML) välja kasv. AutoMLi eesmärk on automatiseerida masinõppe torujuhtmete loomise protsess, optimeerides iga komponenti. Enamik praegustest automatiseeritud masinõppe raamistikest keskenduvad algoritmide valiku ja hüperparameetrite probleemi automatiseerimisele, keskendudes piiratud määral tunnusehõive automatiseerimisele, mis on peamine lisaväärtust andev samm, mille eesmärk on automaatselt informatiivseid funktsioone koostada ja vähendada käsitsi tööd heakorratiste ehitamiseks masinõppe torujuhtmete sooritamine. Lisaks toodavad enamus praegustest automatiseeritud masinõppe raamistikest torujuhtmeid ilma inimese sekkumiseta. Praktikas loob inimese täielik väljajätmine kontuurist mitu piirangut. Näiteks ignoreeritakse enamikus nendest lähenemistest kasutaja eelistusi otsinguruumi määratlemisel või kontrollimisel, mis võib seega mõjutada tagastatavate mudelite aktsepteerimist lõppkasutajate poolt. Selle lõputöö panus on kahekordne: 1) kujundame ja rakendame interaktiivse visualiseerimise tööriista *iSmartML*, mis toetab kasutajaid AutoML-i otsinguruumi juhtimisel ning tulemuste analüüsimisel ja selgitamisel. 2) Kujundame ja juurutame skaleeritava

automatiseeritud funktsioonide väljatöötamise tööriista *BigFeat*.

Võtmesõnad:
Masinõpe, AutoML, tunnusehõive

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

Contents

1 Introduction

Throughout the last decade, as applications of machine learning in the practical world increases at a high rate, the number of dedicated data scientists to work on these problems can not scale fast enough to keep up with the demand[Col17]. This naturally raises the need for automation in machine learning to either reduce the workload and time required from data scientists to open up possibilities to domain experts to effectively use machine learning models without in-depth knowledge in the field. As a result, recently, Automated Machine Learning (AutoML) has risen to address the above-mentioned need. Several AutoML approaches (e.g., Auto-Weka[THHLB13], Auto-Sklearn[FEF$^+$20], TPOT[OM16]) have been developed to automate the Combined Algorithm Selection and Hyper-parameter Optimization (CASH).

CASH problem aims to find the algorithm and hyper-parameter combination that optimizes the performance of a pipeline for a particular task based on a provided metric. For dataset D consisting of training set D_{train} and validation set D_{valid}, alongside machine learning algorithm set **A**, we need to find A^*, seen in Equation 1. A^* being a hyper-parameter tuned configuration of machine learning algorithm $A \in \mathbf{A}$.

$$A^* = arg\min_{A \in \mathbf{A}} L(A, D_{train}, D_{valid}) \tag{1}$$

L is a loss function of algorithm A trained on D_{train} and evaluated D_{valid}, for a specific configuration. The main limiting factor of CASH problem solutions is the amount of time required to produce meaningful results. Thus, producing the best possible results in a limited time budget is particularly important.

1.1 Interactive Automated Machine Learning

While existing AutoML tools perform well when solving the CASH problem without any human involvement, it is still important to effectively include a human in the loop and provide options to interact with the optimization process easily. For example, domain experts might decide that they need only interpretable machine learning models for their application (e.g., medical field) and thus need to limit search space to such machine learning models, even if a different model produces better results based on metric used by an AutoML tool. With the increased importance of interpretability of machine learning models[RSG16], it also falls upon the user to understand the model and use appropriate interpretability tools if necessary. This further raises a barrier for domain experts who are unfamiliar with these tools, while it requires extra time for data scientists to perform it.

iSmartML To address the above-mentioned challenges, we introduce a user-guided AutoML framework iSmartML.

- iSmartML provides users with a configuration control panel that lets users easily modify and configure parts of the AutoML process without technical knowledge or familiarity with integrated tools. Thus, making AutoML more accessible for users such as domain experts of their respective fields.
- A recommendation engine that uses a meta-learning-based approach to provide recommendations to help to configure the framework for best results.
- A monitoring panel provides information on explored search space and best performing pipelines while the search process is in progress.
- A logging system keeps track of previously executed tasks and allows reusing existing results for search space exploration, reducing required time to produce new results.
- The framework provides an explanation module, helping users understand and interpret the model by providing additional information and plots.

1.2 Automated Feature Engineering

Feature Engineering is a crucial step in creating machine learning pipelines, as well-engineered features can significantly improve the model's performance[HTF09]. However, feature engineering is a very time-consuming process. It can also be one of the more frustrating parts of constructing an ml pipeline, as it often requires a trial-and-error approach from data scientists. The increased demand of data scientists further emphasizes the need for AutoML and the inclusion of feature engineering steps in AutoML pipelines. However, most popular AutoML pipeline construction tools [SZL+20] [HPR19] only include some feature prepossessing (e.g., encoding, imputation), feature selection, model selection, and hyperparameter optimization. They leave out the construction of new features from their pipeline, which is an important part of any machine learning pipeline.

In order to fill this gap, automated feature engineering tools have been developed [HPR19] [SZL+20] [KV15], however we believe that these approaches are not effective enough for real-world applications, either due to lack of scalability on large datasets [HPR19] or interpretability of the generated features [KV15].

BigFeat To create an automated feature engineering tool that meets the requirements for real-world applications, we have developed BigFeat, a scalable and interpretable Bayesian Automated feature engineering framework for High-Dimensional Data. Contributions of BigFeat can be summarized as follows:

- Dynamically generated features - Allowing to produce features of any depth at any iteration, without a need to store features of lower depths in the memory. Thus, avoiding the need for larger memory size for deep features.

- A stability-based feature selection mechanism was used by aggregating the results of multiple feature selection steps on the subsets of candidate features while keeping computation time low.
- Sampling features and operators during the generation with weights allow exploiting more likely candidates while exploring with lower probabilities.

Document Outline The remainder of this paper is organized as follows: Section 2 reviews the related work, Section 3 covers interactive automated machine learning and iSmartML in detail. Section 4 focuses on automated feature engineering and BigFeat. Section 5 Concludes the paper.

2 Related Work

A significant amount of work has been done in the field of AutoML during the last few years. In the following, we review the related work and literature.

2.1 Optimization techniques

2.1.1 Blackbox optimization techniques

Grid and Random Search Grid search is one of the most simple and widely used black-box optimization techniques. During the search process, all possible combinations of parameters defined by the user are evaluated to select the best configuration[FH19] . While it provides great results once the entire process is finished, in practice, it is often infeasible due to the curse of dimensionality and requiring an increasingly larger amount of time to produce results.

Random search attempts to solve the issues of the grid search by randomly evaluating configurations from defined search space until a specified budget is exhausted. Random search has been shown to generally produce better results than grid search in a limited time budget [BB12]. While the simplicity of both grid and random search means that they lack advantages of many more advanced techniques, it also means that they are straightforward to parallelize and scale on a cluster of workers.

Bayesian Optimization Bayesian Optimization attempts to minimize the function by constructing a surrogate model to approximate the objective function[SLA12] . Choice of a surrogate function is an important aspect of Bayesian optimization, as its choice defines how well the function can be approximated. The Gaussian process is usually used as a surrogate model in most applications. Another significant decision to be made when using Bayesian optimization is the acquisition function to be used. The acquisition function dictates how the next points should be chosen from the surrogate model for the next evaluation. After a new point is evaluated on an objective function, the surrogate model is updated respectively[SSW$^+$15]. Pseudocode for Bayesian optimization is given in Algorithm 1

Algorithm 1: Bayesian optimization

```
Initilize surogate funtion ;
while budget not exhausted do
    Select new point X from the surrogate function using the acquisition
     function;
    Evaluate point X on objective function;
    Update the surrogate function with the value of the objective function at X;
end
```

Genetic Algorithms Genetic algorithms are an optimization techniques family based on natural selection [Whi94]. The genetic algorithm starts by generating the initial population of chromosomes(configurations). Then, for each generation, the best performing set of chromosomes are selected based on the objective function and the fitness function. The new generation of the population is generated by performing recombination and mutation on selected chromosomes. During the recombination phase, the members of the selected population are selected as pairs at random, and crossover is performed by exchanging the genes(parameters) of the chromosomes. At the mutation stage, genes of chromosomes are randomly modified based on some probability (usually lower than 1%) in order to introduce diversity in the new population. This process is repeated for every new generation until a termination criterion has been met.

Simulated Annealing Simulated Annealing is the process of finding the minimum of a function by simulating the process of physical annealing[BT$^+$93], which includes repeatedly heating up and cooling down the metal. Until the termination condition is met, simulated annealing chooses a new solution and accepts it if it is better than the old one. Otherwise, it will still accept the solution with some probability. The probability depends on the temperature parameter. Higher temperature means that the algorithm is more likely to accept less optimal solutions. Similar to physical annealing, in Simulated annealing, the temperature is slowly decreased using one of the temperature reduction rules. This approach encourages more exploration and reduces the chances that the algorithm will be stuck on a local minimum.

2.1.2 Multi fidelity optimization techniques

Successive Halving Successive Halving is a multi-armed bandit-based optimization algorithm [JT16]. The algorithm starts with predefined n configurations and budget B. On the first iteration, the budget is uniformly distributed between all configurations, each having allocated B/n resources. Then the best performing half of the configurations are kept, and the other half is discarded. During the next iteration, the budget is similarly

distributed between the remaining configurations. This process is repeated until only one configuration remains, giving best-performing configurations exponentially more budget as iterations increase(Figure 1). However, for the fixed budget, it remains up to the user to select appropriate n and thus the budget of each configuration B/n.

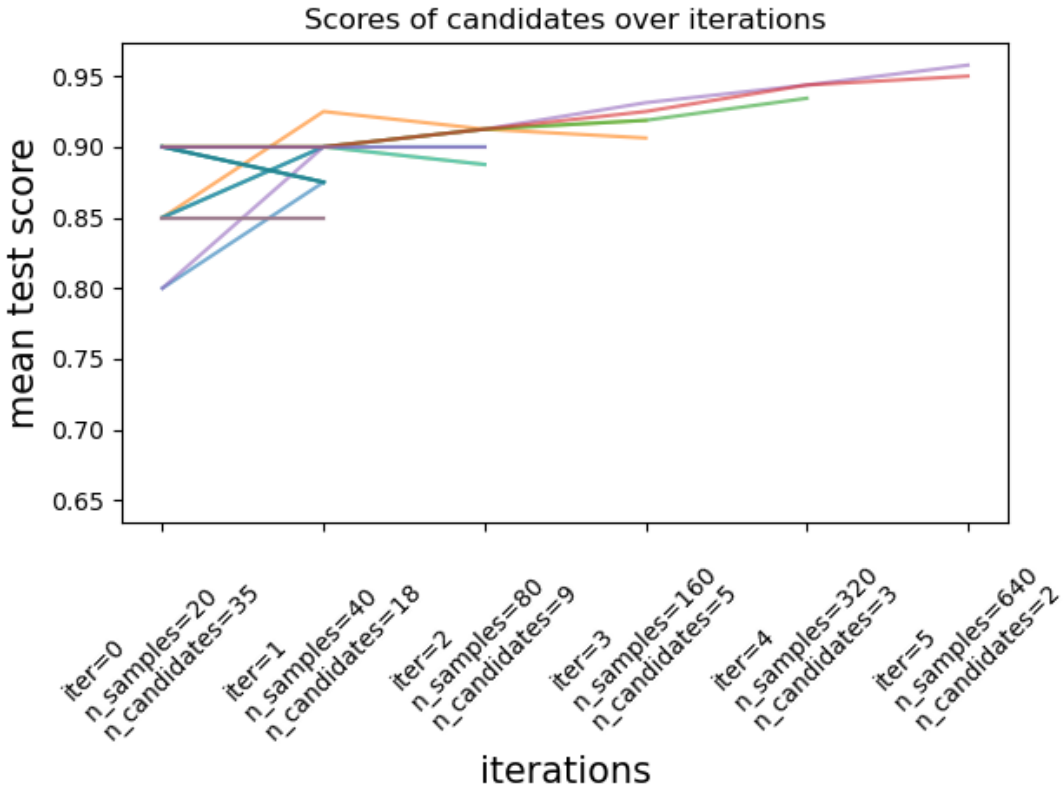

Figure 1. Performance of candidates over iterations using succesive halvig [sci]

Hyper-Band Hyper-Band is an extension of the Successive Halving algorithm[LJD+17], which attempts to address the trade-off between n and the allocated budget for each configuration. It is essentially performing a grid search by running Successive Halving on different values of n, while B is fixed for each iteration. Hyper-Band depends on two parameters, R - maximum resources allocated to each configuration and η - ratio of discarded configurations during each iteration of the Successive Halving. For each run of Hyper-Band, $S_{max} + 1$ values of n are considered for the search, where $S_{max} = \lfloor log_\eta(R) \rfloor$. Thus each run of Hyper-Band takes $(S_{max} + 1)B$ budget, but in return, solves the issue of selecting best n values.

2.2 Meta-Learning Techniques

Meta-Learning is a process of using previous experiences(meta-data) to improve the process of learning new tasks. Meta-data, among other things, can be pipeline configurations, evaluation results, and characteristics of the previous tasks. Meta-learning techniques can be divided into three categories: *learning from previous evaluation*, *learning from task properties*, and *learning from previous models*.

2.2.1 Leaning from Previous Evaluation

Let **P** be set of all previous evaluations $P_{i,j} = (\theta_i, t_j)$, for a previous task $t_j \in T$ and learning algorithm configuration $\theta \in \Theta$. Θ being a configuration space including all hyper-parameter configurations and pipeline components. Given $\mathbf{P}_{new}$(a set of evaluations $P_{i,new}$) for task t_{new}, we want to train meta-learner L on $\mathbf{P} \cup \mathbf{P}_{new}$ to predict recommended configuration Θ^*_{new} [Van18].

Relative Landmarks Relative Landmarks $RL_{a,b,j} = P_{a,j} - P_{b,j}$ similarity between configurations θ_a and θ_b for task t_j [FP01]. One way to use relative landmarks is to start with some initial θ_{best} (e.g. globally best configuration) and during each iteration select θ_c which outperforms θ_{best} on all similar tasks. The tasks are considered similar if the relative landmarks are similar between them. The selected configuration θ_c is evaluated on a task t_{new} to produce $P_{c,new}$. Next, task similarities are updated for new configuration and the process is repeated. [LBV12].

Surrogate Models Another way to to use results from previous tasks is using surrogate model $s_j(\theta_i) = P_{i,j}$ which is trained for all previous tasks t, on all evaluated configurations **P**. In this case, similarity between task t_j and new task t_{new} can be measured using the error between $P_{i,new}$ and $s_j(\theta_i)$.

Warm-Started Multi-task Learning It is also possible to combine surrogate models $s_j(\theta_i)$ for previous tasks using feed-forward neural network $NN(\theta_i)$ in order to learn joint task representation and predict Pi, new [Bis06]. Another approach is to use the Gaussian Process model for Bayesian optimization and learn relationships between the tasks [SSA13].

2.2.2 Learning from Task Properties

Meta-Features One way to gain insights on potential configurations for a particular task is using its meat-features as the data source [Van18]. A task $t_j \in T$ can be described with meta features vector $m(t_j) = (m_{j,1}, m_{j,2}...m_{j,K})$. Some of the most used meta-features are: number of instances, number of features, statistical features , landmarks ,

information theoretic features[EMS19]. Meta features $m(t_{new}$ for new task t_{new} can be used to find distance to an existing task $m(t_j)$. By finding k most similar tasks to t_{new}, best configurations can be selected for each of these tasks and for warm-starting a best configuration search for t_{new} [GPS$^+$12].

Meta-Models The Meta-models approach relies on training a model on the task's meta-features and configurations in order to recommend the best configuration for a particular task [Van18]. Although, choice of model Highly depends on the particular set of meta-features used, overall, tree-based ensemble models have shown to perform the best[AM01] [KI02]. Particular meta-models have been effective at producing rankings of top-K configurations as well. Another use of meta-models is to predict performance or required time for a configuration on a particular task by building a regressor model. This allows users to check certain configurations and save resources by eliminating configurations with particularly long-time requirements or poor performance. Linear regression, SVM, and MultiLayer Perceptron-based meta-learners have successfully been used for this task [BK01] [RSG$^+$14] [DGC18].

2.2.3 Learning from Previous Models

Transfer Learning Transfer learning uses previously trained models as a starting point for building models for a new task t_{new}. Transfer learning has been used for different models such as kernel methods and parametric Bayesian models. However, it has been most successful when applied to neural networks in particular due to its usefulness in terms of both the network's architecture and parameters[Van18]. In computer vision, transfer learning has been successfully used with Convolutional Neural Networks after training it on image datasets such as ImageNet [KSH12]

2.3 Overview on AutoML frameworks

Below is the overview of AutoML tools focusing on traditional machine learning.

AutoSklearn Autosklearn is a python library based on scikit-learn estimators. It uses Bayesian optimization for hyperparameter search, which fits a model learning to maximize performance based on hyper-parameters and chooses the hyper-parameter configuration most likely to produce the best model[FH19]. The produced model is evaluated, and the probabilistic model is updated with new data, which then repeats the entire process. AautoSklearn uses SMAC[LEF$^+$17], which implements Bayesian optimization. AutoSklearn pipeline includes data pre-possessing (one-hot encoding, balancing, resealing, imputation) and feature preprocessing. At the final stage, it produces an ensemble of previously found well-performing models. Meta-learning is used for initialization but is only available for classification at the time.

TPOT Tree-based Pipeline Optimization Tool (TPOT) is also a python library built on top of Scikit-Learn[OM16]. TPOT uses genetic algorithms for hyper-parameter optimization. Tree-based pipelines consist of operators which can perform feature prepossessing, feature selection, and classification/regression. As shown in [figure 3, it is possible for operators to make changes to separate copies of the data and combine the results. For pipeline optimization, TPOT randomly selects 100 pipelines, evaluates them, and selects the top 20. Then 5 copies of each pipeline are created, and 5 % of them are randomly crossover with other offspring, while 90 % of the remaining offspring is randomly mutated. Then the process of evaluating, selecting, and mutating is repeated over and over for 100 generations. Finally, the best performing pipeline is selected from the population as the final result. Population size, offspring size, and the number of generations can be modified by the user as desired for the trade-off between the time taken and model performance. An example of TPOT pipeline is shown in Figure 2

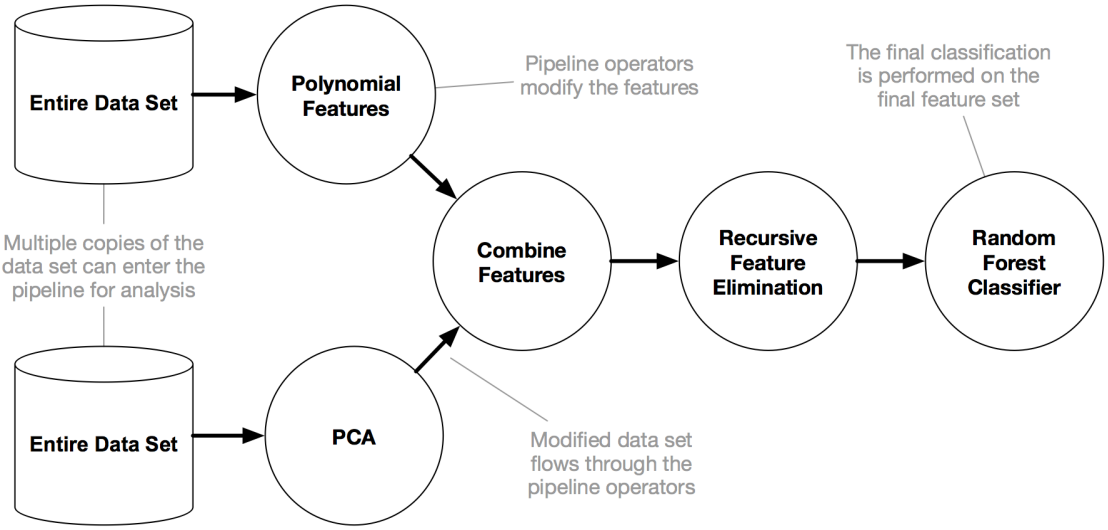

Figure 2. An example of TPOT pipeline[OM16]

SmartML SmartML is a meta-learning-based AutoML framework for R [MS19]. It uses the knowledge base of meta-features of all previously trained datasets to provide the best fitting pipeline for a new dataset based on its meta-features. Results based on the knowledge base are used as the initial configuration for Bayesian optimization using SMAC [LEF$^+$17]. After the optimization is done, the best fitting models are selected as a final result. Generating the ensemble model of top-scoring configurations is also possible. The knowledge base is dynamically updated as the new datasets are submitted for optimization.

AutoWeka AutoWEKA is one of the most popular AutoML tools available. It is based on WEKA and tries to make using the software almost entirely automatic[THHLB13]. For hyperparameter optimization, it uses Bayesian optimization based on SMAC[LEF$^+$17], similar to AutoSklearn[FKE$^+$19]. While initially only supporting classification, it provides support for regression tasks as well since version 2.0. Overall, AutoWEKA supports many classifiers(30 as of version 2.6), which is not the case for many other AutoML tools. It also provides a GUI interface along with a command line. It is one of the more user-friendly and flexible frameworks out there, making it very simple to start making machine learning models with minimum knowledge.

2.4 Overview on Automated Feature Engineering Frameworks

AutoFeat Autofeat is a python library for automated feature engineering[HPR19]. It performs two phases: feature generation and feature selection. During the feature selection phase, non-linear operators are applied to each feature. These operators are: $log(x), \sqrt{x}, , 1/x, x^2, x^3, |x|, exp(x), 2^x, sin(x), cos(x)$. Additionally, all features are combined pairwise with the following operators: $(+, -, *)$. These steps can be repeated multiple times to generate deeper and more complex features. However, given that number of features grows exponentially after each iteration, a high number of iterations is infeasible both in terms of memory and computational time required. It has double explanation complexity in both cases - n^{2^i} features are generated after $i - th$ iteration, where n is the number of original features. In the second phase, Autofeat performs feature selection to reduce many features to a more reasonable amount. Feature selection is also performed in two stages. First, generated features are ranked by correlation with the target residual, and only the highest-ranking features are considered for the second stage. In the second stage, the Lasso LARS regression model is trained on subsets of previously selected features alongside random permutations of all features. Next, the highest regression coefficient among the random features is selected as a threshold. To perform the final selection, only features with regression coefficients higher than the threshold are kept.

The feature selection step can also be repeated multiple times to improve the robustness of the results.

SAFE SAFE(Scalable Automatic Feature Engineering) is another automated feature engineering library[SZL$^+$20]. Similar to AutoFeat, SAFE performs feature generation and feature selection steps. At the feature generation stage, first feature relations are mined to narrow the set of possible feature combinations. Therefore, the XGBoost model is trained on the training and validation sets. Using the model, the split features are tracked, alongside the combinations of split features on the same tree path. The tool is based on hypothesizing that split features that appear on the same tree path are

more likely to generate good combinations than split features not in the same path or non-split features. The features and feature combinations are then further ranked based on the information gain ratio. Next, features with the highest information gain ratio are selected to generate the features. Once candidate features are generated, feature selection is performed in three stages. First, features with low predictive power are discarded based on their information values. Next, redundant features are removed by calculating the Pearson correlation between candidate feature pairs. If the absolute value of the correlation of a pair is higher than 0.8, one with a lower information value is discarded. During the final phase, XGBoost model is trained on the remaining features, and the most important features are selected.

FeatureTools Featuretools is an automated feature engineering framework based on Deep Feature Synthesis [KV15] algorithm. It is designed for relational data and relies on creating new features by aggregating or transforming original features.

3 Interactive Automated machine learning

iSmartML is an interactive, user-guided automated machine learning tool, which guides users throughout the configuration phase and helps interpret the produced results. It provides backends of different AuotML tools : *Auto-Sklearn*, *TPOT* and *D-SmartML* [AEEHES20].

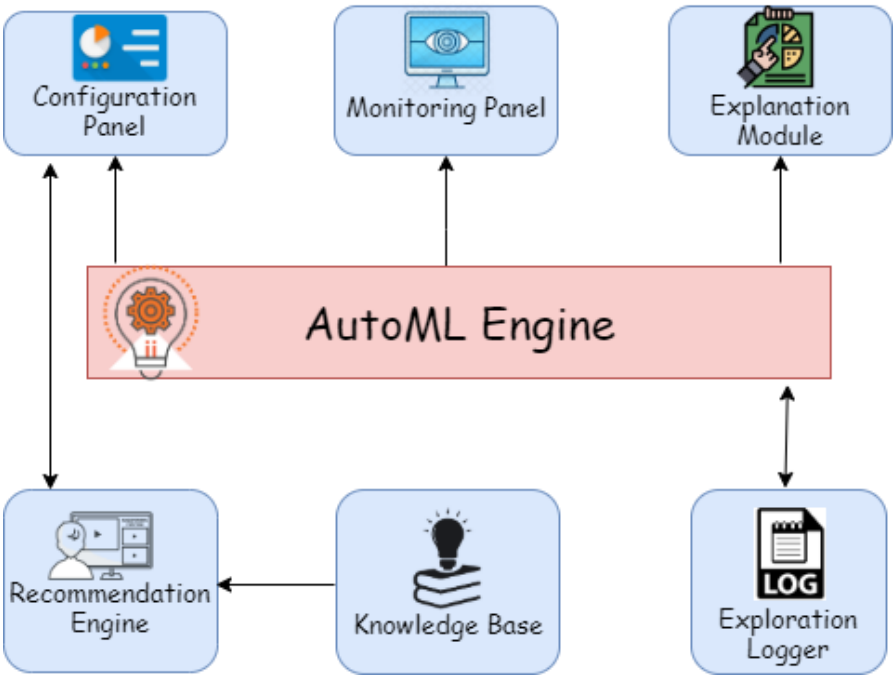

Figure 3. iSmartML architehcture.

3.1 iSmartML Architecture

iSmartML architecture and components can be seen in Figure 3. *Configuration panel* lets users configure their task and search space for the AutoML engine. The *recommendation engine* provides recommendations for aspects of configuration space, which are integrated within the *configuration panel* to guide users. The *monitoring panel* is used to keep users updated on the status of the optimization process and provide the best performing models. The *explanation module* helps users interpret the pipelines produced by the AutoML engine by providing additional information and plots for each one of them. Additionally, the *exploration logger* keeps track of explored search space results for every task(if agreed by the user) and allows the user to resume the search if the same task is re-run.

3.2 Configuration Control Panel

iSmartML provides users with a user-friendly web-based control panel for configuring all aspects of AutoML pipeline search while providing guidance during each step. The user selects task type (classification or regression), AutoML engine, and input data at the first stage. Next, the user is presented with a list of features in the dataset alongside their distributions and statistical information. The user chooses the desired target feature to be predicted and can discard any features they might consider unhelpful due to the presented information or prior knowledge. Then configuring metric used for configuration and option to use SMOTE oversampling technique for unbalanced target features(for classification tasks). Next, the user can define search space for the AutoML engine, where the user is presented with recommendations generated by the meta-learning model, as seen in Figure 4. Finally, the time budget is specified by the user, and the optimization process begins.

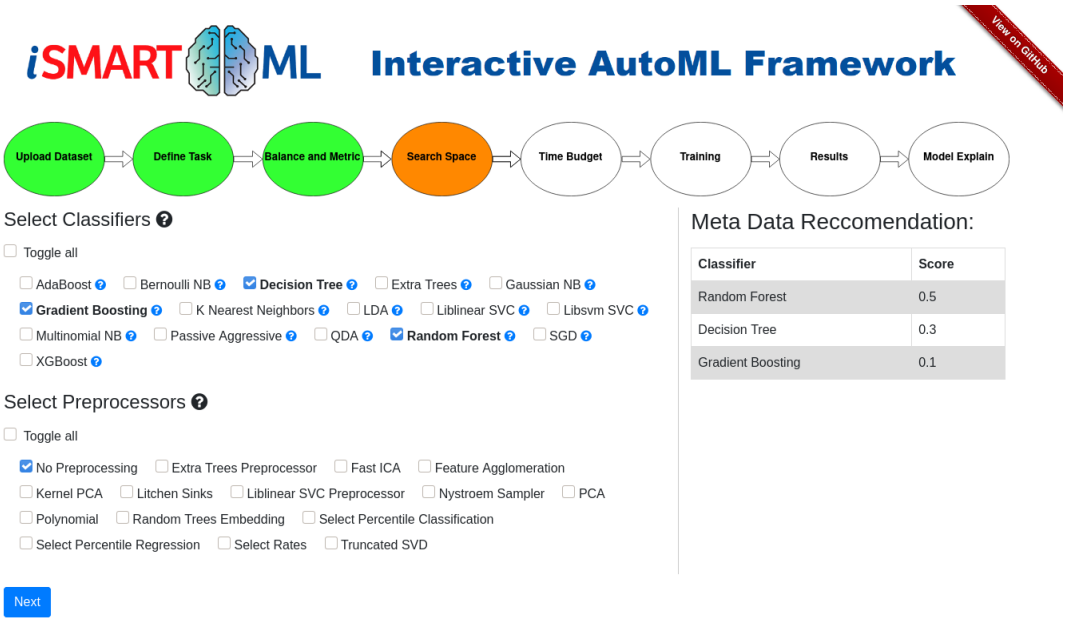

Figure 4. Search space configuration for iSmartML.

3.3 Monitoring Panel

Once the optimization process begins, users are presented with the results screen, which is periodically updated with the new result. Thus, users can monitor search progress and the status of the current best models while the search is still going on. This is very useful

for running the optimization process for a long time giving users an option to stop early by analyzing the results. For example, if the results are not improving significantly, the user might decide to stop the process and start a new process with only well-performing parts of the pipeline. The Results page provides information for each trained pipeline's performance and its components. In addition, for each estimator model(e.g., Random Forst, SVM), the user is provided with scatter plots showing the model's performance with different parameters, as shown in Figure 5.

Figure 5. Results page for iSmartML.

3.4 Explanation Module

Interpreting and understanding a model trained by an AutoML tool is particularly critical in certain domains. Therefore, the user can open an explanation panel for any trained pipeline at any point on the training or results page. As seen in Figure 6. The explanation panel provides detailed information on every aspect of the pipeline through the *Parameters* page. It provides an evaluation of the model by different metrics (For classification: Accuracy, Recall, Precision, F1), as well as the confusion matrix for classification tasks.

Additionally, a feature importance graph is shown for models that provide it. To help the user interpret the model, partial dependence and feature interaction plot can be displayed with specified features and labels using the *PDPbox*[pdp] toolbox [Sau]. Finally, the user can actually download the pipeline objects directly from this page without the need to rebuild them manually.

3.5 Implementation

iSmartML is implemented on top of the Python framework Flask[Gri18]. It provides a web interface for the user. Pandas[M$^+$10] and Numpy[HMvdW$^+$20] libraries are used for loading and prepossessing the data. Auto-sklaern[FKE$^+$19], TPOT[OM16] and D-Smartml[AEEHES20] provide AutoML engines. PDPBox library is used for producing model interpretability plots, and the imbalanced-learn[LNA17] library is used for oversampling with SMOTE. iSmartML can be run locally or be hosted through a web server. For quick and easy setup, *Docker* image is also available.

3.6 Meta Learning-Based Recommendation Engine

iSmartML provides recommendations for the user throughout the process of search space configuration. It includes two meta-models based on meta-features to recommend the best classifiers for each submitted task, along with recommended minimum time budget to be used. These recommendations provide a starting point for the users, who have no prior knowledge about what models might perform well and the appropriate time for the task.

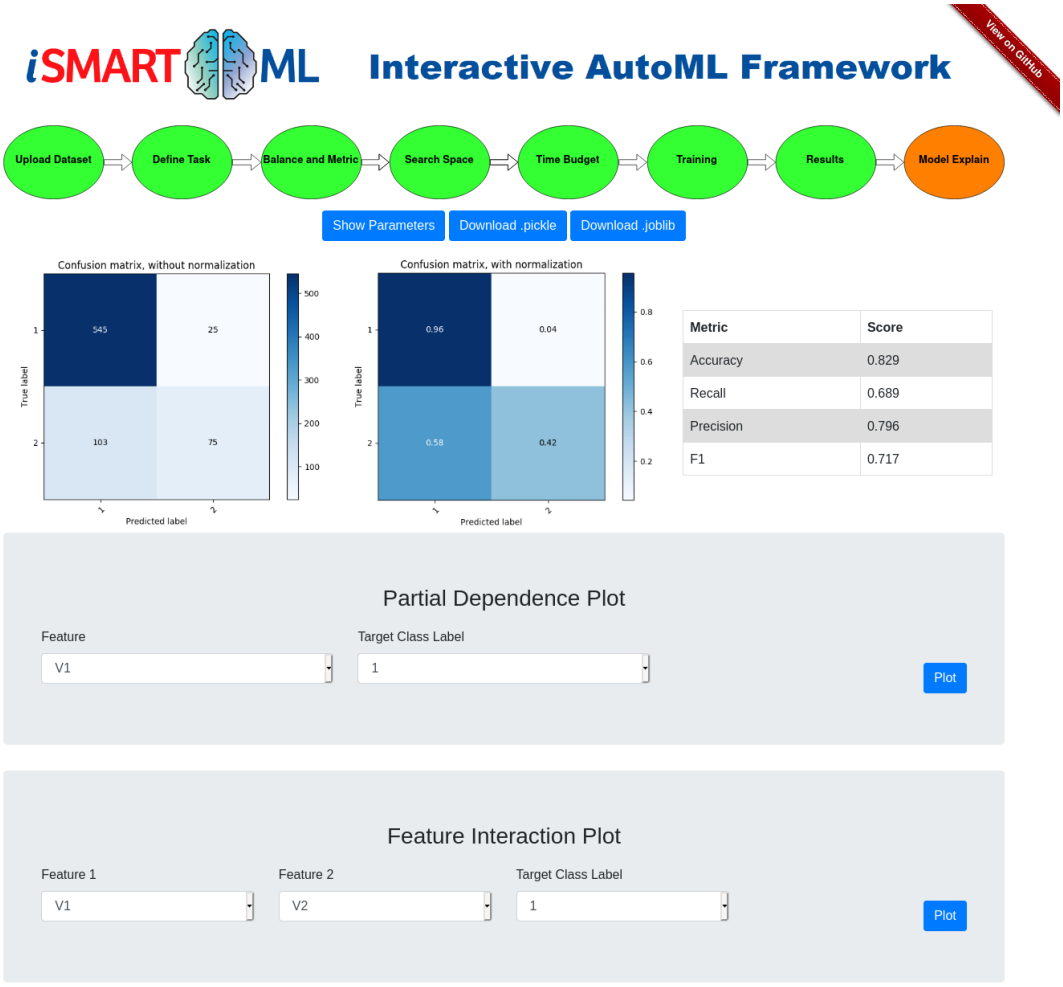

Figure 6. Model explenation page for iSmartML.

4 Automated Feature Engineering

Automated feature engineering aims to produce transform feature set into a new, more informative, and better performing one. More specifically, it can be defined in the following way: Given dataset D, split into training D_{train} and validation D_{valid} sets, alongside feature engineering function Ψ. X_{train} and Y_{train} being features and labels of set D_{train} and X_{valid} and Y_{valid} being features and labels of set D_{valid} .

As shown in Equation 2, the goal is to find Ψ^* , particular configuration of Ψ, that minimizes L. L is a loss function for machine learning algorithm A trained on $(\Phi^*(X_{train}), Y_{train})$ and evaluated on $(\Phi^*(X_{valid}), Y_{valid})$.

$$\Psi^* = arg\min_{\Phi} L(A, (\Phi(X_{train}), Y_{train}), (\Phi^*(X_{valid}), Y_{valid})) \quad (2)$$

4.1 BigFeat

BigFeat relies on generating features by performing operations on input features and selecting the most important features among them. In case of unary operators: x^2,x^3, e^x, $\sqrt{x}, |x|$, operator is applied to a feature to transform it into a new one. For binary operators, new features are produced by combining the input features by applying the operator, e.g. $x_1 + x_2$, $x_1 * x_2$. More complex features can be generated by applying a new operator again to generated features. BigFeat is an iterative feature generation tool. It starts with *the Initialisation* stage where, original feature importances and feature combinations(Section 4.1.1) are calculated, alongside the initial operator importance. Next, candidate features are generated in the *Feature Generation* stage. Finally, the highest-ranking features are selected in *the Feature Selection* stage, and operator importances are updated. *Feature Generation* and *Feature Selection* stages are repeated for each iteration. Pseudocode for BigFeat is shown in Algorithm 2.

Algorithm 2: BigFeat

```
Result: Generated Features
I^f ←feaute importances;
C ←feaute combinations;
I^o ←(1,1,··· , 1);
for iteration in iteratoins do
    Generate n × k features;
    Select up to n features;
    Update operator importances I^o;
end
```

4.1.1 Initialization Stage

Operator Importance As shown in Equation3, operator Importance vector I^o is initialized uniformly as we have no information about each operator's effect on the current task.

$$I^o = (I^o_1, I^o_2, \cdots, I^o_N) = (1, 1, \cdots, 1) \tag{3}$$

Feature Importance The next step is to initialize a Feature importance vector I^f. We fit a machine learning algorithm on the training data. This machine learning model generates importance values for each feature, which is used to populate I^f. Given that it is not feasible to generate every feature combination, the feature importance vector is used as weights when sampling input features for each operator.

Feature Combinations While the feature importance vector provides information on how important individual features are, it provides no insight on what feature combinations will produce good results. We follow the hypothesis [SZL+20] that split features that appear on the same path of a decision tree are more likely to generate informative features when combined. Feature combinations matrix C, as shown in Equation 4, is a $n \times n$ matrix, where n is the number of original features.

$$C = \begin{pmatrix} c_{1,1} & c_{1,2} & \cdots & c_{1,N} \\ c_{2,1} & c_{2,2} & \cdots & c_{2,N} \\ \vdots & \vdots & \ddots & \vdots \\ c_{N,1} & c_{N,2} & \cdots & c_{N,N} \end{pmatrix} \tag{4}$$

Both rows and columns of C represent the same n original features. Every time features a and b appear on the same tree path, element $C_{a,b}$ is incremented by one. Thus, row C_a represents frequencies of each feature appearing in the same path as a.

4.1.2 Feature Generation Stage

Once feature importances and initial feature combinations are extracted, we generated kn new features during each iteration, where N is the number of original features, and k is a user-defined parameter for controlling the relative size of intermediate features.

Feature generation for depth 1 is performed in the following way. First, BigFeat selects an operator by sampling randomly from the set of operators O, with weights vector I^o. Next, the first input feature for this operator is selected. Similarly, this selection is weighted by feature importance vector I^f. If the selected operator was urinary, a new feature is generated by applying this operator to the selected feature. In the case of binary operators, we multiply element-wise a row matching the first feature from the feature combination matrix C by feature importance vector I^f.

The produced vector is used as weights for sampling the second feature from f. Then we produce a new feature by applying the operator to the selected features. To generate deeper features, this method is applied recursively. For depth larger than 1, features to be used for generation result from applying the same algorithm with lower depth. Feature generation algorithm can be seen in more detail in Algorithm 3.

Algorithm 3: Dynamic Feature Generation

Input : Feature importance - I^f, Operator importance - I^o, Feature Combinations - C, Depth - D, Set of features - X Selected base features - S ($\emptyset$)
Output : New feature - F^*

if *D = 0* **then**
  $\hat{I^f} \leftarrow I^f$;
  for *feature f in S* **do**
    $\hat{I^f} \leftarrow C(f)$;
  end
  F^* ←Sample feature with $\hat{I^f}$
  $S \leftarrow S \cup F^{*;}$
else
  $D \leftarrow D - 1$;
  $F \leftarrow \emptyset$;
  o ←Sample operator with weights I^o
  n ←number of input features for operator O;
  for *i in n* **do**
    F_n ←recursively call itself with:I^f, I^o, D, X;
    $F \leftarrow F \cup F_n$;
  end
  F^* ←Apply operator o on features in F;
end

4.1.3 Feature Selection Stage

Feature Rankings With KN new features generated, we need a way to select informative features among them accurately. To achieve this, we perform stability-based feature selection[MB10] on generated features. Given stability parameter α, we sample one-third of features and one-third of instances to create subsets of the data. We train a model on each of these subsets and aggregate feature importances from all runs. The feature importances are sorted in descending order, and top n features are selected.

Redundant Features Although the important features are targeted, we want to avoid selecting highly correlated and redundant features to produce the most useful feature set. We start by going through each feature - in order of their rankings - and calculate the Pearson correlation between it and already selected features. If the absolute value of the correlation coefficient is lower than threshold η, we select the feature. This process is repeated until n features are generated, or all features are exhausted.

Operator Importance Update With the generated set reduced to best n features, the weights for operator importance need to be updated for the next iteration. First, we go through each selected feature and count the number of appearances for each operator in selected features to produce the operator importance vector for the next iteration $\hat{I^o}$. Finally, a weighted average with the previous operator importance vector I^o is calculated based on the current iteration, and I^o is updated. The detailed process is described in Algorithm 4.

Algorithm 4: Operator Importance Update

Input : Operator importance - I^o, Selected candidate features - F, Current iteration - c

Output : Operator importance - I^o

$\hat{I^o} \leftarrow (0, 0, \ldots, 0)$;
for *feature f in F* **do**
  $O_f \leftarrow$applied operators for feature f;
  for *operator i in* O_f **do**
    $\hat{I^o} \leftarrow +1$;
  end
end
normalize $\hat{I^o}$;
$I^o \leftarrow ((c-1))I^o + \hat{I^o})/c$;

4.2 Empirical Evaluation

BigFeat's performance was evaluated on datasets from OpenML [VvRBT13] as part of the automated machine learning pipeline. We measured the performance of AutoML tools with and without automated feature engineering tools. Experiments were done with using AutoML tools rather than fixed models for two reasons. First, given that AutoML tools do not include feature generation, it was important to evaluate the effect of such a process in AutoML pipeline as a whole. The second reason being that as we attempt to find model agnostic features, AutoML tools provide good insights on their value due to considering a wide range of machine learning models.

Implementation BigFeat is implemented in Python and relies on the Numpy[HMvdW+20] library for data storage and numerical operations. There are two main reasons for using Python. Firstly, it provides a wide range of machine learning libraries, including AutoML tools such as Auto-Sklearn and TPOT. Secondly, Python is an excellent language for prototyping and thus allowed us to perform experiments on various hypotheses quickly. Scikit-learn[PVG+11] and LightGBM[KMF+17] libraries are used for training machine learning models to perform feature selection. BigFeat class provides a Sklearn-like API for fitting and transforming features.

4.2.1 Experimental Setup

The experiments were conducted on a machine with 64-cores and 256GB of RAM. 36 datasets were used in total for the experiment. The full list can be seen in Table 1.

Two AutoML tools, $Auto-Sklearn$, $TPOT$, were used alongside two automated feature engineering tools, $BigFeat$ and $AutoFeat$. For each dataset 8 configurations were evaluated. In particular 4 separate feature engineering setups were considered for each AutoML tool: $ORIG$, AF, $RAND$, BF.

- $ORIG$ - Original features passed to AutoML tool without additional feature generation step.
- AF - Feature engineering performed by *AutoFeat*.
- $RAND$ - Features and operators are selected randomly for feature generation. The same feature selection process is used as in *BigFeat*.
- BF - Feature engineering performed by *BigFeat* as described in 4.1.

F1 score was used for evaluation as well as optimization metric for AutoML tools. For runs using BigFeat, the following configuration was used:

- $Iterations$ - 7
- $Depth$ - For each iteration depth for generating features was used in this order: [1, 1, 1, 1, 1, 1, 2]
- η - 0.9. Correlation threshold.
- k - 10. Generated candidate size parameter. $k \times n$ features were generated during each iteration.

Dataset	Instances	Features	Labels
madelon	2600	500	2
australian	690	14	2
vehicle_sensIT	98528	100	2
gina	3153	970	2
Fashion-MNIST	70000	784	10
arcene	200	1000	2
nomao	34465	118	2
connect-4	67557	42	3
micro-mass	571	1300	20
bank-marketing	45211	16	2
kc1	2109	21	2
covertype	581012	54	7
eeg-eye-state	14980	14	2
credit-g	1000	20	2
amazon_employee_access	32769	9	2
MiniBooNE	130064	50	2
numerai28	96320	21	2
Christine	5418	1636	2
adult	48842	14	2
yeast$_m l8$	2417	117	2
kr-vs-kp	3196	36	2
hiva_agnostic	4229	1617	2
sonar	208	60	2
jungle_chess_2pcs_raw_endgame_complete	44819	6	3
dbworld-bodies	64	4702	2
ailerons	13750	40	2
cnae-9	1080	856	2
airlines	53983	7	2
phoneme	5404	5	2
mfeat-factors	2000	216	2
shuttle	58000	9	7
GCM	190	16063	14
car	1728	6	4
segment	2310	19	7
blood-transfusion-service-center	784	4	2
AP_Omentum_Ovary	275	10936	2

Table 1. Datasets used for experiments, along with the number of instances, features and unique target labels. Obtained from *OpenML*[VvRBT13]

4.2.2 Results

Evaluation results of the above-mentioned configurations can be seen in Table 2. BigFeat clearly provides the best results out of 4 configurations on 15 out of 36 datasets when evaluating using Auto-Sklearn. Additionally, it is tied to the best methods on 7 out of 21 other datasets. Finally, it outperforms AutoFeat on 21 out of 36 datasets and is tied in 5 others. The performance difference between BF and OG configurations is shown in Figure 7.

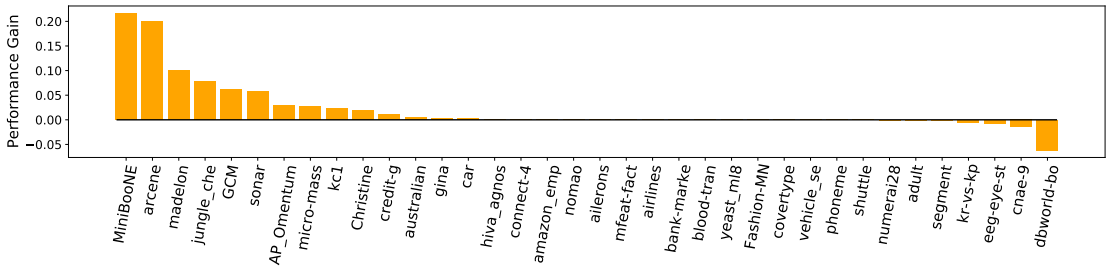

Figure 7. Difference between F1 scores of the BF and ORIG configurations, using Auto-Sklearn

Execution Time Considering that time budget is generally the limiting factor for AutoML tools, the low execution time of automated feature engineering is also critical. As seen in Figure 9, BigFeat is significantly faster than AutoFeat, due to avoiding exhaustive feature generation, yet produces better results in most cases.

Selected Feature Importance To evaluate the importance of generated features, we measure the feature importance ratio between generated and original features, among finally selected n features. Figure 8 shows that generated features contribute significantly more compared to selected original features.

Dataset	Auto-Sklearn				TPOT			
	ORIG	AF	RAND	BF	ORIG	AF	RAND	BF
madelon	75.85	75.85	82.29	**85.85**	**84.92**	84.31	81.02	84.62
australian	87.28	84.39	84.15	**87.86**	85.55	85.55	84.11	**87.86**
vehicle_sensIT	49.62	49.62	47.96	49.62	77.5	49.62	86.6	**87.19**
gina	92.65	92.65	90.52	**92.9**	**94.68**	92.9	91.31	94.17
Fashion-MNIST	9.85	-	8.27	9.85	=	-	=	=
arcene	80.0	76.0	97.29	**100.0**	90.0	90.0	91.77	**100.0**
nomao	96.79	96.8	95.4	**96.88**	**96.24**	96.12	96.04	96.22
connect-4	80.4	**81.05**	77.49	80.57	83.19	79.31	**83.59**	79.5
micro-mass	87.41	85.31	88.55	**90.21**	**90.91**	89.51	90.06	85.31
bank-marketing	90.28	90.28	87.49	90.28	90.65	**90.81**	90.15	90.68
kc1	84.47	85.42	86.06	**86.74**	86.74	86.36	84.46	**88.26**
covertype	36.44	-	35.08	36.44	=	-	=	=
eeg-eye-state	92.31	92.31	88.64	91.51	**97.52**	95.51	95.38	91.96
credit-g	76.4	76.0	76.61	**77.6**	73.6	77.6	76.04	77.6
amazon_employee_access	94.53	94.53	94.21	**94.64**	**95.11**	95.01	94.33	94.89
MiniBooNE	71.38	71.38	69.28	**93.1**	**91.91**	71.38	90.86	91.34
numerai28	51.83	**51.85**	49.37	51.71	51.26	51.33	51.78	**51.99**
Christine	73.21	74.39	72.23	**75.2**	71.22	**74.32**	73.93	73.65
adult	**87.09**	86.98	86.99	86.95	**87.32**	87.09	86.14	87.09
yeast_ml8	98.84	98.84	97.74	98.84	98.84	98.84	97.27	98.84
kr-vs-kp	99.25	99.25	97.0	98.75	99.5	99.5	96.45	98.62
hiva_agnostic	96.6	96.6	96.2	**96.79**	96.88	96.41	93.82	96.88
sonar	75.0	76.92	76.81	**80.77**	84.62	80.77	84.14	84.62
jungle_chess_2pcs_raw_endgame_complete	82.23	82.37	86.37	**89.97**	86.1	86.49	**87.43**	86.51
dbworld-bodies	93.75	93.75	73.03	87.5	93.75	93.75	85.37	87.5
ailerons	87.46	**88.48**	87.11	87.52	87.93	**88.39**	87.83	87.93
cnae-9	95.56	**95.93**	90.81	94.07	95.19	95.19	94.62	91.11
airlines	55.46	55.46	53.42	55.46	57.31	64.69	**66.28**	55.46
phoneme	89.64	89.12	86.16	89.64	89.49	89.71	88.09	**91.04**
mfeat-factors	96.4	**97.2**	95.61	96.4	96.2	**97.2**	96.69	96.6
shuttle	99.98	**99.99**	99.51	99.98	99.96	99.97	99.79	**99.98**
GCM	75.0	72.92	64.07	**81.25**	68.75	68.75	70.34	**75.14**
car	94.21	94.44	90.54	94.44	99.07	**99.31**	97.78	96.99
segment	**97.58**	96.71	96.67	97.4	**98.79**	97.58	97.29	97.75
blood-transfusion-service-center	79.14	78.61	78.56	79.14	78.07	77.01	74.91	**80.21**
AP_Omentum_Ovary	76.81	76.81	**81.37**	79.71	79.71	**85.51**	83.14	84.06

Table 2. F1 scores for classification experiments for all datasets and configurations. Bold entries highlight best performing configuration for each AutoML tool. '-' denote failure of a run due to automated feature engineering tool. '=' denotes failure of a run due to AutoML tool

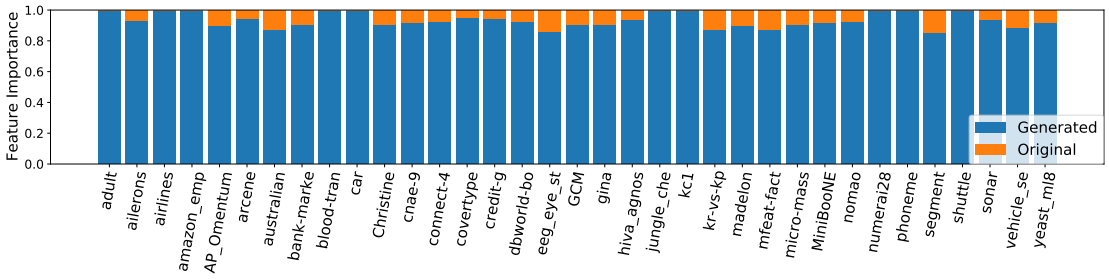

Figure 8. Contribution of generated and original feature importances among final selection

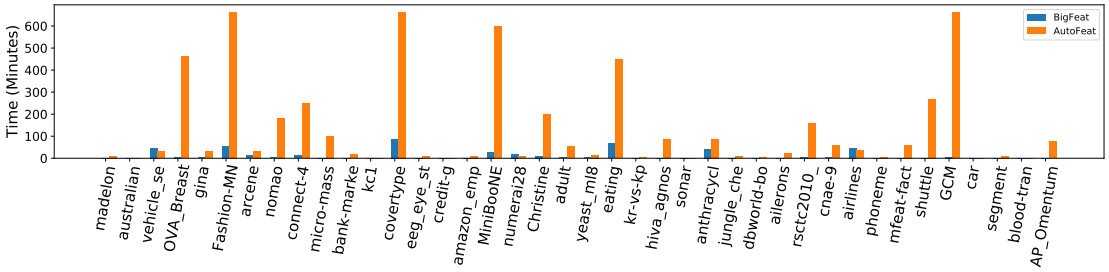

Figure 9. Execution times of BigFeat and AutoFeat.

5 Conclusion

With the increased interest in machine learning, Automated machine learning has also become more important. In this thesis, we first introduced an interactive user-guided AutoML framework, iSmartML. It integrates various tools and methods to provide an easily accessible and customizable way to produce AutoML pipelines. Secondly, we provide an automated feature engineering tool, BigFeat, to address the lack of feature extraction methods in existing AutoML pipelines. BigFeat has the ability to generate deep features by combining existing features using different operators while keeping constant memory usage. The tool was evaluated on a set of 36 datasets as a part of the AutoML pipeline by integrating it with existing AutoML tools. BigFeat outperformed existing automated engineering solutions on tested datasets while requiring significantly fewer resources.

As future work, we believe that integrating meta-learning methods will allow us to warm start the optimization process by finding better initial weights for operators to be sampled. By starting weights closer to optimal values, we expect to achieve similar results in fewer iterations with less computational power. Additionally, BigFeat can be integrated as a feature engineering step into iSmartML to simplify the entire AutoML pipeline for the end-user.

References

[AEEHES20] Ahmed Abd Elrahman, Mohamed El Helw, Radwa Elshawi, and Sherif Sakr. D-smartml: A distributed automated machine learning framework. In *2020 IEEE 40th International Conference on Distributed Computing Systems (ICDCS)*, pages 1215–1218. IEEE, 2020.

[AM01] Kalousis Alexandros and Hilario Melanie. Model selection via meta-learning: a comparative study. *International Journal on Artificial Intelligence Tools*, 10(04):525–554, 2001.

[BB12] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of machine learning research*, 13(2), 2012.

[Bis06] Christopher M Bishop. *Pattern recognition and machine learning*. springer, 2006.

[BK01] Hilan Bensusan and Alexandros Kalousis. Estimating the predictive accuracy of a classifier. In *European Conference on Machine Learning*, pages 25–36. Springer, 2001.

[BT+93] Dimitris Bertsimas, John Tsitsiklis, et al. Simulated annealing. *Statistical science*, 8(1):10–15, 1993.

[Col17] Louis Columbus. Ibm predicts demand for data scientists will soar 28% by 2020. *IBM White Paper*, 2017.

[DGC18] Casey Davis and Christophe Giraud-Carrier. Annotative experts for hyperparameter selection. In *AutoML Workshop at ICML*, 2018.

[EMS19] Radwa Elshawi, Mohamed Maher, and Sherif Sakr. Automated machine learning: State-of-the-art and open challenges. *arXiv preprint arXiv:1906.02287*, 2019.

[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*, 2020.

[FH19] Matthias Feurer and Frank Hutter. Hyperparameter optimization. In *Automated Machine Learning*, pages 3–33. Springer, Cham, 2019.

[FKE+19] Matthias Feurer, Aaron Klein, Katharina Eggensperger, Jost Tobias Springenberg, Manuel Blum, and Frank Hutter. Auto-sklearn: efficient and robust automated machine learning. In *Automated Machine Learning*, pages 113–134. Springer, Cham, 2019.

[FP01] Johannes Fürnkranz and Johann Petrak. An evaluation of landmarking variants. In *Working Notes of the ECML/PKDD 2000 Workshop on Integrating Aspects of Data Mining, Decision Support and Meta-Learning*, pages 57–68, 2001.

[GPS+12] Taciana AF Gomes, Ricardo BC Prudêncio, Carlos Soares, André LD Rossi, and André Carvalho. Combining meta-learning and search techniques to select parameters for support vector machines. *Neurocomputing*, 75(1):3–13, 2012.

[Gri18] Miguel Grinberg. *Flask web development: developing web applications with python*. " O'Reilly Media, Inc.", 2018.

[HMvdW+20] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020.

[HPR19] Franziska Horn, Robert Pack, and Michael Rieger. The autofeat python library for automated feature engineering and selection. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 111–120. Springer, 2019.

[HTF09] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The elements of statistical learning: data mining, inference, and prediction*. Springer Science & Business Media, 2009.

[JT16] Kevin Jamieson and Ameet Talwalkar. Non-stochastic best arm identification and hyperparameter optimization. In *Artificial Intelligence and Statistics*, pages 240–248. PMLR, 2016.

[KI02] Christian Köpf and Ioannis Iglezakis. Combination of task description strategies and case base properties for meta-learning. In *Proceedings of the 2nd international workshop on integration and collaboration aspects of data mining, decision support and meta-learning*, pages 65–76, 2002.

[KMF+17] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. Lightgbm: A highly efficient gradient boosting decision tree. *Advances in neural information processing systems*, 30:3146–3154, 2017.

[KSH12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25:1097–1105, 2012.

[KV15] 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)*, pages 1–10. IEEE, 2015.

[LBV12] Rui Leite, Pavel Brazdil, and Joaquin Vanschoren. Selecting classification algorithms with active testing. In *International workshop on machine learning and data mining in pattern recognition*, pages 117–131. Springer, 2012.

[LEF+17] Marius Lindauer, Katharina Eggensperger, Matthias Feurer, Stefan Falkner, André Biedenkapp, and Frank Hutter. Smac v3: Algorithm configuration in python. *URL https://github. com/automl/SMAC3*, 2017.

[LJD+17] Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. *The Journal of Machine Learning Research*, 18(1):6765–6816, 2017.

[LNA17] Guillaume Lemaître, Fernando Nogueira, and Christos K. Aridas. Imbalanced-learn: A python toolbox to tackle the curse of imbalanced datasets in machine learning. *Journal of Machine Learning Research*, 18(17):1–5, 2017.

[M+10] Wes McKinney et al. Data structures for statistical computing in python. In *Proceedings of the 9th Python in Science Conference*, volume 445, pages 51–56. Austin, TX, 2010.

[MB10] Nicolai Meinshausen and Peter Bühlmann. Stability selection. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 72(4):417–473, 2010.

[MS19] 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.

[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.

[pdp] Pdpbox¶.

[PVG+11] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[RSG+14] Matthias Reif, Faisal Shafait, Markus Goldstein, Thomas Breuel, and Andreas Dengel. Automatic classifier selection for non-experts. *Pattern Analysis and Applications*, 17(1):83–96, 2014.

[RSG16] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. Model-agnostic interpretability of machine learning. *arXiv preprint arXiv:1606.05386*, 2016.

[Sau] SauceCat. Saucecat/pdpbox.

[sci] Successive halving iterations.

[SLA12] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical bayesian optimization of machine learning algorithms. *arXiv preprint arXiv:1206.2944*, 2012.

[SSA13] Kevin Swersky, Jasper Snoek, and Ryan Prescott Adams. Multi-task bayesian optimization. 2013.

[SSW+15] Bobak Shahriari, Kevin Swersky, Ziyu Wang, Ryan P Adams, and Nando De Freitas. Taking the human out of the loop: A review of bayesian optimization. *Proceedings of the IEEE*, 104(1):148–175, 2015.

[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. In *2020 IEEE 36th International Conference on Data Engineering (ICDE)*, pages 1645–1656. IEEE, 2020.

[THHLB13] Chris Thornton, Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. Auto-weka: Combined selection and hyperparameter optimization of classification algorithms. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 847–855, 2013.

[Van18] Joaquin Vanschoren. Meta-learning: A survey. *arXiv preprint arXiv:1810.03548*, 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.

[Whi94] Darrell Whitley. A genetic algorithm tutorial. *Statistics and computing*, 4(2):65–85, 1994.

Appendix

I. Licence

Non-exclusive licence to reproduce thesis and make thesis public

I, **Shota Amashukeli**,
*(*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,

   Toward Automatic Construction of Machine Learning Pipelines,
   *(*title of thesis)

   supervised by Radwa Elshawi 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.

Shota Amashukeli
14/05/2021