Automated Feature Engineering for Automated Machine Learning

Casper de Winter, Flavius Frasincar, Bart de Peuter, Vladyslav Matsiiako, Enzo Ido, Jasmijn Klinkhamer

Erasmus University Rotterdam, P.O. Box 1738, NL-3000 DR Rotterdam, the Netherlands

Abstract

Automated feature engineering (AutoFE) and automated machine learning (AutoML) can both be used in a machine learning project to improve the efficiency of a data scientist. In recent years, different algorithms have been developed for both sub fields independently of each other. In this study, the use of AutoFE in combination with AutoML has been evaluated for the first time to determine if AutoFE can increase the model accuracy, while not increasing the computation time. A data fusion meta-learning approach was extended, generalized, and applied to an AutoFE method and then further combined with a pre-existing AutoML method. In the meta-learning approach, more than 150 online data sets were used to create models that recommended the best operator to apply to a certain feature. Using twelve evaluation data sets, we show that combining AutoFE and AutoML is indeed valuable. The accuracy measure used was increased on average by 0.54% compared to using AutoML alone. For multiple data sets, the use of AutoFE significantly outperformed a strategy in which no feature engineering was done, while in the remaining data sets it never significantly performed worse. Therefore, it can be concluded that it is beneficial to combine this computationally efficient AutoFE method with AutoML.

Keywords: automated feature engineering, automated machine learning, feature construction, feature selection, meta-learning

1. Introduction

Machine learning (ML) is one of the most booming subfields of computer science and data analytics nowadays [1]. As companies gather more data, the potential of data-driven projects becomes more evident.

Email addresses: casperdewinter@hotmail.com (Casper de Winter), frasincar@ese.eur.nl (Flavius Frasincar), bart@building-blocks.nl (Bart de Peuter), matsiiako@gmail.com (Vladyslav Matsiiako), 534787ei@eur.nl (Enzo Ido), 584059jk@eur.nl (Jasmijn Klinkhamer)

Machine learning is a broad concept encompassing many algorithms that discover meaningful results by learning and using information captured in the data. In this paper, we focus on supervised *classification* problems, which are ML problems that use data to explain or predict a discrete target variable. In general, a machine learning project can be illustrated using the ML life cycle, which is shown in Figure 1 [2].

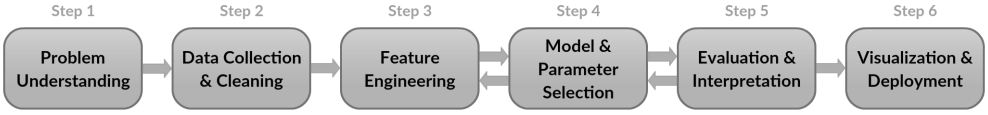


Figure 1: The machine learning life cycle.

In this paper, we limit our main focus to step 3 from Figure 1, on general and automatic algorithms for *feature engineering*, in relation to steps 4 and 5. It is assumed that the output of step 2 in the ML life cycle is one data table with raw data. Then, feature engineering can be used to enrich the raw data in step 3 by using or removing the original features to create new and potentially more useful ones. This can largely improve the performance of the *machine learning model* which is selected, trained, and evaluated in the next steps [3]. The well-known concept "Garbage In, Garbage Out" is also related to this phenomenon [4]. With ML model we denote any trained *machine learning algorithm* or sequence of algorithms. Usually, the three steps of the ML life cycle, namely feature engineering, model selection, and model evaluation, are repeated in manual machine learning until the researcher is satisfied with the results and a final model is obtained.

In recent years, more focus has been given to the automation of both the feature engineering and the modeling part of the ML life cycle. This is due to the increasing shortage of skilled data scientists [5, 6] and, specifically for feature engineering, due to the large time spent on this stage by a researcher (50% to 70% of the total project time) [7, 8]. Furthermore, manual feature engineering can be tedious, limited by creativity, problem- and data-specific such that useful relations and features can easily be missed out in practice [9]. The automated fields of research are called *automated feature engineering* (AutoFE) and *automated machine learning* (AutoML). The relationship between AutoFE and AutoML can easily be captured by the following equation: $model = AutoML(AutoFE(data))$, where the resulting *model* after applying $AutoML$ and $AutoFE$ is at least as good as the *model* obtained after applying $AutoML$ on the *data*.

Contrary to AutoFE, there already exists several well-integrated methods for AutoML [10, 11, 12] that have been applied in practice [13]. The independent potential of both subfields has been shown with different

time-consuming algorithms, but a feature engineering algorithm has been rarely combined with AutoML to efficiently improve the output produced by AutoML [3, 14, 15, 16].

Most AutoFE algorithms have only been tested with fixed, standard ML algorithms, for example only with a standard random forest. However, a feature set which performs well on one ML model does not necessarily perform well on another model [17]. Furthermore, most existing algorithms are time consuming, implying that there is currently no promising, general, and time-efficient AutoFE method available. The long computation time also applies to AutoML algorithms which usually need several hours to find a good-performing ML model. Hence, adding all feature engineering possibilities to the AutoML search space could further increase the computation time, making it an inefficient solution. In this paper, we would, therefore, like to answer the following research question:

Can automated feature engineering be used to increase the accuracy of the general automated machine learning process for supervised classification tasks, while not increasing the computation time?

For clarification, feature engineering can be seen as a combination of *feature construction* and *feature selection*. In feature construction, new features are created as functions of the original features. Feature selection is used to guarantee that only the most relevant features are constructed or kept, and that irrelevant and redundant ones are removed from the data [18]. Certain topics are not in the scope of feature engineering in AutoFE. Feature extraction, which includes principal component analysis (PCA) and other techniques, is not considered a part of developed feature engineering algorithms. It is included as a possible preprocessing algorithm in AutoML. Data integration, imputation, and outlier removal are also out of the scope of our research. AutoFE should not be confused with automated data integration, which combines multiple related data tables into a single one [19]. For this research, the data is assumed to be a single, complete data set with a discrete target variable.

Main contributions. Previously, it was stated that AutoFE and AutoML have never been researched together. Finding out if and how these two fields of research can be combined in a valuable manner is, therefore, one of the main contributions of this paper. Another significant contribution of this work is the extension of a recently developed AutoFE method, namely Learning Feature Engineering (LFE) developed by Nargesian et al. [15]. LFE is a computationally efficient data fusion meta-learning approach which uses

several online data sets to learn the feature construction operators that are most suitable for a given type of feature. This model is extended by incorporating more construction operators, a feature selection method, more feature types, and more types of classifiers.

Organization of the paper. The remainder of this paper is structured as follows. In Section 2, we present and discuss the relevant literature regarding AutoFE and AutoML and conclude which directions will be explored in this paper. The chosen approach is explained in detail in Section 3. Next, the results of the proposed methods are evaluated in Section 4 after which conclusions are made and suggestions for future research are stated in Section 5. The code, the data sets, and specific details which can be used to implement our methods can be found at https://github.com/Casper-de-Winter/GELFE.

2. Literature Review

Feature engineering algorithms and their relevant aspects have been studied by many researchers. In this section, we present a literature review on all subjects which are covered in the scope of our research and discuss why we decided to explore the meta-learning approach for AutoFE. In the following sections, we discuss relevant feature construction and selection techniques and the AutoFE and the AutoML methods. As most research related to automatic feature engineering focuses on binary classification problems, the presented techniques will also be reviewed from that point of view [14, 15, 20]. In Section 2.1, more background information is given on feature construction and selection. Relevant AutoFE and AutoML methods are presented in Sections 2.2 and 2.3, respectively.

2.1. Feature Construction and Selection

Feature Construction. Many operators and techniques have been proposed to construct new features from the raw data. Most of them are unary or binary operators which apply a transformation to one or two features, respectively, to generate a new one. Ternary or in general n-ary operators have hardly been used for AutoFE and will therefore also not be considered in this research.

Unary operators can, for example, use scalers and power transformations to alter the distribution of the feature or discretize a numerical feature into an appropriate categorical one. Binary operators try to capture the relevant relation between two features in a brand new one. Examples of binary operators are basic arithmetic operators such as addition and multiplication, and group-by operators which categorize the

numerical data according to another categorical feature. Constructed features can be used again with another operator to construct even more features. It is specific to each data set the operators that produce the most useful features [3, 21]. However, the drawback of using several different operators is that the number of possible features which can be constructed goes to infinity. This is often where feature selection methods come into play within feature engineering.

Feature Selection. Filter and wrapper are both feature selection methods that have been used within feature engineering algorithms. Wrapper methods evaluate subsets of features and constructed features using an ML algorithm [22]. These methods are already computationally expensive for simple ML algorithms, and are thus not general. Using AutoML as a wrapper function is definitely not computationally efficient and will thus not be considered. Filter methods are faster and more general heuristic selection methods, and can be divided in similarity-based, statistical-based, and entropy-based methods [23]. These different methods can also be combined, for example in a dual feature selection and weighted representation framework [24]. In general, the algorithms in the similarity-based and statistical-based groups all select features individually, which means that they only look at relevance and not at redundancy. In feature engineering, where multiple (20) versions of a relevant feature can be generated, addressing redundancy is essential. Therefore, we focus on entropy-based methods.

This family of selection methods chooses, in every step, the extra feature with the highest potential, given the target variable and the already selected features. Entropy-based methods can, in general, only be applied to categorical data. The linear generalization of these methods is called conditional mutual information (CMI) [23] and its formula can be found in Equation 1. In this formula, $I(\cdot)$ is a function that calculates the information shared, f_{eval} is the evaluated feature, y is the target variable, X the current set of features containing features f_i, and w_1 and w_2 are weights for the different terms.

$$CMI_{f_{eval}} = I(f_{eval}, y) - w_1 \sum_{f_i \in X} I(f_{eval}, f_i) + w_2 \sum_{f_i \in X} I(f_{eval}, f_i | y) \tag{1}$$

The first term is equal to Information Gain [25]. The middle term is a penalty for the information shared between the evaluated feature and X, while the final term is a bonus for the information gained when the evaluated feature is incorporated into X, given the target variable. This is done to not add seemingly relevant but redundant features. Examples of different versions of the CMI formula are Conditional Infomax

Feature Extraction (CIFE) [26] and Minimum Redundancy Maximum Relevance (MRMR) [27]. In CIFE, for example, both weights are 1 to make all terms equally important, whereas MRMR uses $w_1 = |X|$ and $w_2 = 0$ and therefore mainly focuses on redundancy. For more background on CMI, we refer to the work of Li et al. [23].

2.2. Automated Feature Engineering

In the previous section, we showed that the selection problem cannot, in general, be solved to optimality. This can certainly not be done when construction is used to create an infinite number of new features. Due to this, different types of heuristic algorithms were designed to automatically apply feature engineering on a new data set. We can distinguish between algorithm-specific, expansion-reduction, metaheuristic, and meta-learning methods for AutoFE.

Algorithm-specific methods are designed to optimize performance for a single machine learning algorithm, such as FICUS in a decision tree learning [21] or a stack of convolution and pooling layers in a convolutional neural network, for automated feature extraction [28]. As AutoML focuses on many different algorithms, these methods are not general and hence, cannot be applied.

Expansion-reduction methods focus on expanding the feature space with many features after which a selection method is used to only keep the most promising features. ExploreKit, created by Katz et al. [14], creates all possible features given a data set and several operators, after which a relatively fast selection method is used to evaluate the features. However, even for moderately sized data sets, this algorithm could be running for a day due to the enormous expansion. In other researches, random subsets of constructed features are added and evaluated [29]. The results using this approach were not always promising since the number of random subsets and evaluation were still considerably large. Also an expansion-reduction method that is considered is a strategy that combines feature selection from machine and human experts [30]. However, for this, professionals are thus needed.

Most work on automated feature engineering has been focused on meta-heuristics, which is a large family of general and powerful heuristic algorithms for optimization. Due to the enormous search space in which new, relevant features can be found, a metaheuristic can be used as an intelligent search algorithm. Examples of metaheuristic algorithms adapted for feature engineering are evolutionary algorithms as genetic programming, harmony search, and particle swarm optimization [31, 3, 32, 33, 34]. In most approaches,

random feature sets are created and evaluated using filter or wrapper methods in order to evolve towards a final feature set. These methods are general in terms of exploring a large feature space, but often take quite long and therefore are not efficient. The most recently developed sub-field of automated feature engineering uses *meta-learning* [15]. This can be seen as a part of machine learning which uses an algorithm that learns from past experiences to improve the performance on current problems. In meta-learning for AutoFE, a large selection of data sets is used to create *meta-data* and train *meta-models* or *meta-classifiers* for each construction operator. The *meta-models* then store the knowledge about which feature-operator combinations improve the classification task the most. This training phase only needs to be done once, and here it is important that the implemented method can generalize different features to a standard set of meta-features that captures their distribution. Afterwards, given a new feature set, the trained meta-models are used to predict the operator which impacts the classification performance the best. The main advantage of this approach is that, after the meta-models have been trained, the computational cost of recommending an operator is very small.

The described meta-learning method Learning Feature Engineering (LFE) by Nargesian et al. [15] is seemingly the most efficient approach, beating expansion-reduction methods on both accuracy and computation time. The main disadvantage of LFE is that it is not yet general: only one ML algorithm is used to determine the meta-target variable in the meta-data sets and the LFE approach can only be used for numerical features. However, it is possible to add multiple ML algorithms to the training phase while only affecting the training time. Due to these reasons, LFE is chosen as the AutoFE method to be extended and implemented in this paper. In our extension we will not limit ourselves to numerical features and will also incorporate categorical features. Furthermore, we will incorporate 20 more operators and three additional training algorithms. Finally, we will apply selection methods to decide between multiple recommended operators. This last idea can be seen as a combination between meta-learning and expansion-reduction. Meta-learning is used to expand the feature space by recommending new features, after which a selection method filters only the features that are suitable for a particular classification problem.

2.3. Automated Machine Learning

In this section, we describe how AutoML works, which methods are implemented, and why automated feature engineering can be of any help. The two most used and researched types of AutoML implementations

are genetic programming and Bayesian optimization. Examples of these two types are, respectively, the Tree-based Pipeline Optimization Tool (TPOT), developed by Olson et al. [11], and Auto-SKLearn [10], which is a continuation of Auto-Weka [35]. TPOT uses genetic programming by generating populations of different pipelines and evaluating them using cross-validation until a complete ML pipeline is evolved. Auto-SKLearn, on the other hand, uses Bayesian optimization to find a good pipeline. These pipelines can consist of multiple preprocessing techniques and classifiers. Usually, both AutoML methods take several hours to find good pipelines. The evolved pipelines and performance of both AutoML methods are relatively similar and we have chosen to use TPOT as it has been shown to converge slightly faster [36].

The following classifiers are included in TPOT, each one also contains several hyperparameters which need to be determined: 'Random Forest', 'Decision Tree', 'Extremely Randomized Trees', 'Linear Support Vector Machine', 'Logistic Regression', 'Gradient Boosting', 'Extreme Gradient Boosting', 'Gaussian Naive Bayes', 'Bernoulli Naive Bayes' and 'k-Nearest Neighbors'. These classifiers can also be stacked, meaning that the prediction of one model acts as an extra feature for another algorithm. The AutoML pipeline can also apply preprocessing techniques. While this may seem redundant when also applying AutoFE, the preprocessing techniques are specific to each classifier and significantly more simple. Some examples of these techniques are feature extraction methods and univariate filter selection methods that do not take redundancy into account. The only technique which explores feature interaction is called 'Polynomial Features' which simply adds all products of features to the feature space without doing any selection. The main drawback of these preprocessing techniques is that they are applied to all features and not only the relevant ones, as in AutoFE. These unnecessary computations make the method more inefficient. Also, adding more advanced construction and selection functions to AutoML would further increase the run time, making it even more inefficient. Hence, AutoFE could possibly be an efficient alternative or complement to this stage of AutoML.

It has been shown that AutoML is able to improve the accuracy and execution time of a data science project using these automatic and intelligent methods to find a good ML pipeline [11]. However, drawbacks related to the computation time and the preprocessing techniques exist. Using automated feature engineering, we would like to engineer a data set which results in an improved final accuracy and a lower computation time to find good ML pipelines. We believe that having a general and fast AutoFE block in front of AutoML

is preferred over adding more complexity to the current AutoML implementations. This means that AutoFE is executed first by applying construction and selection techniques, and is then followed by AutoML which applies optional and additional preprocessing techniques and selects the best classifiers. An advantage of this is that the AutoFE method can be made very general. Preprocessing techniques which are specific to certain ML classifiers or certain larger data sets do not need to be taken into account within AutoFE. Furthermore, the engineered data set using AutoFE does not necessarily have to be used in combination with AutoML.

3. Methodology

In this section, we present the automated feature engineering method which we will explore in this research and explain the techniques which are used. As the described LFE method by Nargesian et al. [15] is quite basic, we present the methodology of a more general and extended version of it which will be called Generalized Extended Learning Feature Engineering (GELFE). GELFE can be split into two phases. In Section 3.1, the *training phase* is covered in which several online data sets are used to create meta-data and build meta-models for each operator. This should be done once such that the meta-models can then be used to engineer new features for an unseen data set. This *feature engineering phase*, which exploits the previously built meta-models to construct and select features, is explained in Section 3.2. Finally, we explain how GELFE is evaluated in Section 3.3.

3.1. Training Phase

The training phase is the key part of the meta-learning approach for feature engineering. The meta-models or meta-classifiers which are trained in this phase will be used on any new data set to recommend which operators to consider for each feature. The idea is to use the information about whether it is useful to apply a certain operator on a certain feature to build effective meta-models. The necessary input to create these meta-models are several different data sets, multiple operators, and different classifiers which are used to determine which modifications lead to an improvement. A complete overview of the training phase can be found in Figure 2 and in the next paragraphs each part is explained in more detail.

Data Sets. Many different features from multiple data sets are needed in order to create good meta-models. For this, we use a large number of online available machine learning data sets from the OpenML repository [37]. A main advantage of OpenML is that labels, indicating whether a feature is numerical or

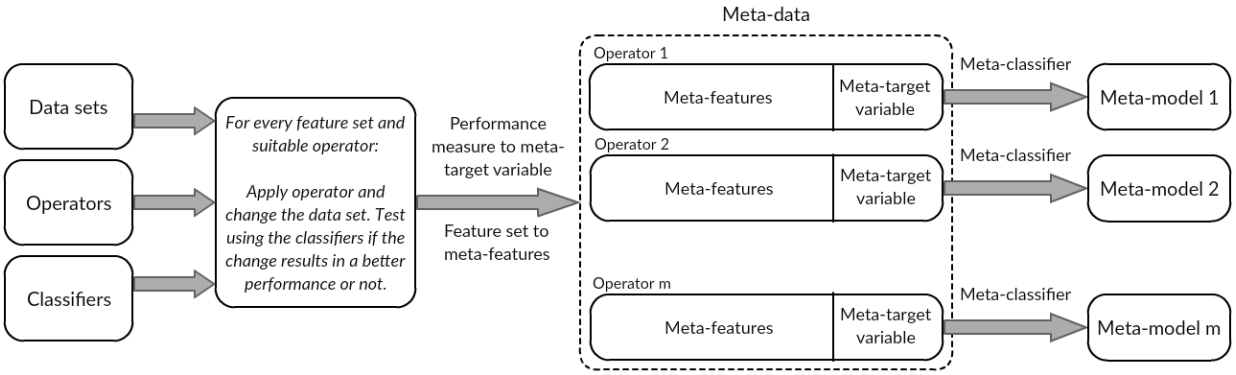


Figure 2: The complete process of the training phase of GELFE.

categorical, are provided for each feature and these are a necessary input for the meta-models. The data sets, which can all be found on GitHub (https://github.com/Casper-de-Winter/GELFE), are downloaded and adapted using several criteria since not all of them are suitable for our research. Namely, the number of instances should be larger than or equal to 500 to ensure that the results found when adding an extra feature are significantly sound and stable. Also, the number of features should be larger than 4 and smaller than 500. The upper limit of 500 is chosen as data sets with thousands of features will, otherwise, be excessively present in the final meta-data. Furthermore, the target variable in all data sets should have at most 10 classes. All multiclass classification problems were transformed to binary problems by converting the largest class to a 1 and the remaining classes to a 0. A small number of missing values is allowed and these are imputed using the Gower distance [38] due to the presence of both numerical and categorical features. Finally, identical data sets or different versions of the same data set were manually deleted. After performing these modifications 165 data sets were left.

Ten of these data sets are later used for the evaluation of our method and the remaining 155 data sets will be used to create meta-models. Together, these data sets contain over 5,000 different features and have a median of 3,000 instances. The number of data sets we use is lower than the number of data sets of Nargesian et al. but we believe that the applied filtering was necessary to standardize the sizes of the data sets and remove identical ones.

Operators. For each of the 34 different operators selected for this research, a meta-model is created. These operators can be split based on the number of input features and the type of each feature.

Of the 19 unary operators, 15 of them are for numerical features. The first eight numerical unary operators all change the distribution of the feature. These operators are the natural logarithm, the inverse, the square root, the inverse square root, the square, the cube, the robust standardizer, and the arctangent operator. The next unary numerical operator is the quantile operator which transforms the feature values according to their order. More specifically, the smallest value becomes a zero, the largest a one, and all the others are linearly scaled based on their order. The remaining six numerical operators transform the feature into a categorical one. This group includes the equal frequency discretizer with 5 and 10 bins, the equal range discretizer with 5 and 10 bins, the bigger-than-zero operator, and the median-split operator.

Of the 19 unary operators, the remaining four operators are for categorical features. In TPOT and its Python implementation, categorical features should be one-hot encoded before using them as input. This means that ordinal categorical features can lose some crucial information. Therefore, the first unary categorical operator is the Cat-to-Num operator which alters the type of the newly constructed feature. The second operator is the frequency encoder which determines the new values based on the frequency of each category. The final two operators, the binary and the backward difference encoder, are encoders which can work better than the standard one-hot encoding. We have also explored automated grouping operators, but these turned out to be not effective.

The fifteen binary operators consist of nine numerical-numerical operators and six numerical-categorical operators. Categorical-categorical operators have not been explored as they do not seem useful and have not been implemented in AutoFE algorithms before. The list of numerical-numerical operators consists of basic mathematical operations such as addition, subtraction, absolute difference, multiplication, and protected division. Protected division is division in which the value zero is returned when dividing by zero [31, 20]. More complex numerical-numerical operators are the ln-times operator, the hypotenuse operator, the larger-than operator, and the quadrant operator. The last two operators return a categorical feature with two and four levels, respectively.

The set of numerical-categorical operators mostly consists of group-by-then operators. These operators can be used to group numerical values of a feature based on the value of a categorical feature. An example is the average price in the product category to which that product belongs to. The mean, median, and the standard deviation can be saved using three of these operators. Two other operators use the feature value

and these group statistics to create new features, and are called the minus-group-median operator and the group-standardize operator. The final operator is the one-hot-numerical operator which splits the numerical feature into multiple columns (one per category) with numerical information if category is present and zeroes, otherwise.

Classifiers and Determining the Meta-Target Variable.

Every feature set will be tested with multiple classifiers to create the meta-target variable. Four different default ML classifiers have been chosen for this testing procedure: 'Logistic Regression', 'Random Forest', 'Linear Support Vector Machine', and, 'k-Nearest Neighbors'. To limit the computation time, more classifiers were not implemented. We believe that these four classifiers, from different model families, give a good indication of the performance of AutoML when changing a feature.

For each data set, benchmark scores for all classifiers are found using 5-fold cross-validation with the balanced accuracy score (BACC). Then, the data set is slightly changed for one feature set and one operator. The constructed feature is added to the data set or, when the unary operator constructs a new feature that is of the same type as the original feature, it replaces it. In order to avoid data leakage, the operators are applied to each test set using the information from the train set. Afterwards, the same four classifiers are applied to the modified data set to find four new BACC scores, and if at least one of these scores is significantly larger than the benchmark, the meta-target variable becomes one ($\tilde{y} = 1$).

One significant improvement is enough as it is not known beforehand the type of model that would work well on any new data set. If a feature-operator combination shows an improvement on one classifier, the information can be valuable for AutoML when applying a similar classifier. A significant improvement is defined as the minimum of one percentage point of the BACC score and a five percent increase of the BACC score compared to the gap between the benchmark score and 1. This last term means that when the benchmark score is 0.9, the new score should be at least $0.9 + 0.05 \times (1 - 0.9) = 0.905$ to be significant. This necessary improvement is chosen to prevent very small improvements caused by randomness to be significant.

Meta-features. At this moment, we can create for each operator, a list of meta-target variables for different feature sets. In order to create useful meta-models, information about the feature sets should be saved as meta-features. An advantage of using meta-features is that it allows us to relate the usefulness

of an operator to the distribution of the feature by, for example, saying that a certain operator works best for features with a high skewness. Many different meta-features have been implemented in this research. Examples of meta-features for numerical features are the mean, variance, median, range size, mode, mode frequency, and bin frequencies after applying an equal range discretizer. This last meta-feature was the only one originally used in LFE, all the remaining ones are extensions of this research. Furthermore, information about the relation between the feature and the original target variable is saved using their correlation and other measures that describe possible different feature value distributions for instances belonging to $y = 1$ and to $y = 0$.

For categorical features, the meta-features include the number of categories as well as the frequencies of the different category levels. Again, the original target variable is used to find differences in the feature distribution for values belonging to $y = 1$ and $y = 0$ and to find their correlation. A high correlation can, for example, suggest that the feature is ordinal. For binary operators, the used meta-features capture information about both input features and their relation. One can think of the correlation, the rate of both means, or both ranges. For some numerical-numerical operators, the order can be important. Therefore, two meta-observations are saved for each feature set with possibly different meta-target variables. One observation contains first the information of the first feature and then of the second feature, and the other one in the reverse order. During the feature engineering phase, both orders should be evaluated. Details and formulas of all the implemented meta-features can be found in an addendum on GitHub (https://github.com/Casper-de-Winter/GELFE).

Meta-models. The meta-features and meta-target variables compose the meta-data that is used to train the meta-models. One meta-model is trained for each operator and, in the feature engineering phase, is used to decide the operator that will most likely improve the prediction accuracy of a new feature set. In other words, the objective of the meta-models is to accurately predict whether it is useful to apply that operator to a feature set. Furthermore, the prediction probability scores of the different meta-models should be comparable.

Due to the second reason, it is chosen to use hyperparameter tuning on one ML classifier to build the meta-models instead of, for example, using AutoML. The random forest classifier is chosen as it is known to perform well on a wide range of predictive problems [39] and has one of the best performances over

many data sets and all classifiers in AutoML [40]. Based on the hyperparameter options used in AutoML for a random forest classifier and the ranges and options found in online case studies, a large parameter grid was formed to tune the classifier. Specific details about the tuned parameters and their ranges can be found in GitHub (https://github.com/Casper-de-Winter/GELFE). Then, a random grid search with 5-fold cross-validation and the F1-score is used to determine the best model among the set of different models which can be formed using the grid. As the objective is to find a model which can correctly predict if an operator is useful on a certain feature set and the focus is not on correctly predicting the zeroes, the F1-score has been selected as the performance measure.

3.2. Feature Engineering Phase

These trained meta-models can be used to automatically engineer new feature sets for any new input data set. For a new data set, the feature types, labeling each as numerical or categorical, should be added as input in order to know which operators can be applied to which features. In Figure 3, an outline of the feature engineering phase is shown. In this section, we explain both parts in which feature engineering takes place, for unary and binary operators, respectively, and explain how the additional feature selection method can be used.

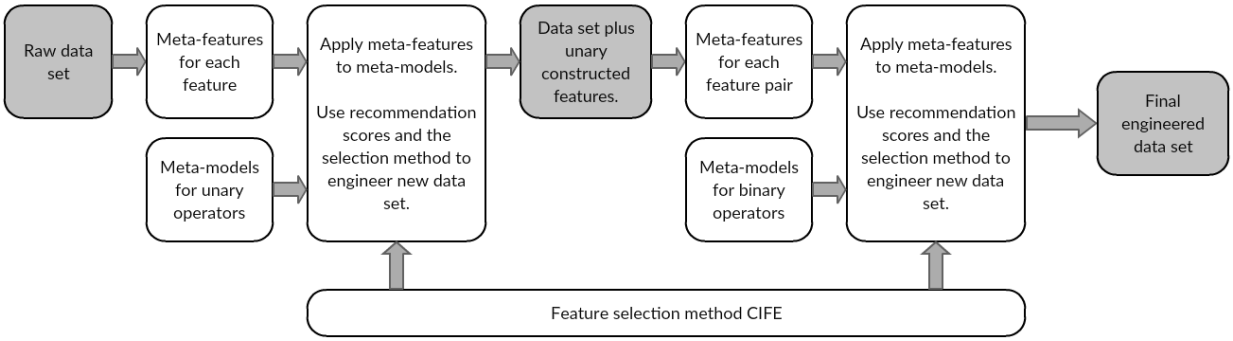


Figure 3: The complete process of the feature engineering phase of GELFE.

First, the meta-models for the unary operators are used to improve the feature space. This is done such that the features which are created can, thereafter, be used to create more new features using the binary operators. A feature set is initially transformed to its corresponding meta-features and used as input for all suitable meta-models. Instead of using the meta-model to predict a one or a zero, the random forests are used

to predict the probability of a one given the meta-feature. This probability can be seen as a recommendation score between 0 and 1 for the expected effectiveness of the operator on that feature set.

In LFE, Nargesian et al. only selected the best of all these recommendation scores for every original feature, given that the score was higher than 0.5 [15]. Disadvantages of this approach are that the current features are not taken into account, that adding multiple new features based on the same feature can be convenient, and that when the number of features is large, say one hundred, one hundred extra features are added. A large amount of extra features does not help the AutoML procedure as it will become much slower and probably include irrelevant transformations.

Therefore, given all the recommendation scores of all feature-operator combinations, the highest `max_unary` or `max_binary` (two constants) are selected in the two phases, respectively. These limitations are based on the number of original features and can vary between 7 and 20. Other imposed limits are that at most two features are allowed to be constructed using the same feature set (one or two features depending on operator arity), that of these two operators, at most one is allowed to be a discretizer, and that at most half of the constructed features are allowed to be created using the same operator. Since evaluating every pair of features with every binary operator meta-model can be significantly time consuming, random feature pairs are evaluated in the binary phase for at most 15 minutes. The AutoFE method would lose efficiency if it would take hours to process all feature pairs in order to construct only a small number of features. All constructed features are always added to the feature set, and we leave the possible removal of an original feature to the selection method.

A multivariate filter selection method can be used in combination with the recommendation scores to engineer the final feature set. The original feature set is expanded and then the selection method chooses the most promising `max_unary` or `max_binary` ones. For this, we use CIFE, which is a special case of CMI, the entropy-based selection method which was explained in Section 2.1. This algorithm is chosen due to its simplicity and proven performance [41].

As CIFE can only be applied to categorical data, some adaptions have been made. A copy is made of the feature set in consideration and all numerical features are discretized using an equal range discretizer with eight bins. Eight bins are chosen as the average between five and ten, which are the bin settings of the discretizer operators. More advanced discretization algorithms (quantile-based discretization) exist for

this purpose but have not been implemented to limit the computation time. However, the discretization of numerical features can lead to a loss of information for certain ML algorithms [42]. Two discretized features with the same entropy can have very different absolute correlation coefficients. Therefore, we combine CIFE with the absolute Pearson correlation coefficient when the evaluated feature is numerical. In that case, the first term in CIFE, see Equation 1, becomes the average between that term and the Pearson correlation. This heuristic adaptation of CIFE ensures that CIFE is less biased towards categorical features or uncorrelated numerical features.

CIFE is implemented as follows. First, a Boolean denotes if the original features should be kept or not. If this is the case, the first selected features are the original features. Then, in each iteration, CIFE adds the feature with the highest CIFE equation score. Features are added until some limit is reached, which can be based on `max_unary` and `max_binary`. Only in the binary phase, a complete feature selection is allowed as seemingly irrelevant original features should not be deleted before exploring the use of binary operators.

In total, six different data sets are engineered in this research. The first is the original data which can be used as benchmark (A). One uses only the unary operators (B) and two use both unary and binary operators (C and D), all three without using the selection method. They only select features based on the recommendation scores. The difference between strategies C and D is that it is allowed in strategy C to construct multiple features using the same feature set, whereas only the one with the highest recommendation score can be used in D. The final two engineering strategies (E and F) use the selection method in combination with the settings of strategy C. In both the unary and the binary phase, the best 50 features are constructed, after which CIFE is used to select between them. In strategy F, the original features can be replaced, which means that the final selection procedure starts with an empty feature set instead of the original feature set.

For clarification, we present an overview of the six data sets and all different settings in Table 1. The number of features in the resulting feature sets of strategies B to F is always higher due to the addition of constructed features. It should be recalled that `max_unary` and `max_unary` can take the values 7, 10, 15, or 20 depending on the number of features p in the original data set X_0. `max_unary` and `max_unary` are 7 if p is below 10, the parameters are 20 if p is larger than 80, and they are 15 if p is between 40 and 80. These values are derived from the logic that more informative features can be derived from a larger feature space and should be added to the data set.

Table 1: A schematic overview of all engineered data sets used for evaluation.

	`max_same_x`	Unary Phase	Unary Selection	Binary Phase	Final Selection
X_{FE}^{A}	-	-	-	-	-
X_{FE}^{B}	2	Add `max_unary`	-	-	-
X_{FE}^{C}	2	Add `max_unary`	-	Add `max_binary` `time_limit` = 15	-
X_{FE}^{D}	1	Add `max_unary`	-	Add `max_binary` `time_limit` = 15	-
X_{FE}^{E}	2	Add at most 50	Reduce added features to `max_unary`	Add at most 50 `time_limit` = 15	Select p + `max_unary` + `max_binary`. X_0 is fixed
X_{FE}^{F}	2	Add at most 50	Reduce added features to `max_unary`	Add at most 50 `time_limit` = 15	Select p + `max_unary` + `max_binary`. X_0 is not fixed

3.3. Automated Machine Learning

Each evaluation data set, of which ten are from OpenML and two are internal ones, is split in a training and testing set using a 60/40 split. All training sets go through the feature engineering phase described in Section 3.2 while the test sets are stored and used in the evaluation phase described in Section 3.4. The training sets are altered during the feature engineering phase according to each of the six strategies mentioned in Section 3.2 and the same modifications are applied to the test set. Only the training sets are used as input for TPOT in order to evolve ML pipelines.

TPOT uses the inputed training sets to generate ML pipelines by repeating an evaluate-select-crossover-mutate process for at most three hours. It is important to note that in total, AutoFE and AutoML has three hours to run, meaning that a higher time spent in feature enginerring implies less time to evolve ML pipelines. To initialize the algorithm, TPOT uses the training sets to generate 30 random tree-based pipelines. Then, the evaluation phase begins and TPOT computes the BACC score of each pipeline and ranks them accordingly. The evaluation stage then selects the top 20 pipelines which simultaneously maximize the classification accuracy and minimize the amount of operators used. Five copies are made for each of the 20 selected pipelines and in the crossover stage, 5% of these offsprings have a randomly chosen subtree removed and replaced with a subtree of another offspring. Finally, 90% of the remaining offsprings randomly suffer a point, insert, or shrink mutation. This cycle is repeated several times and in each run, the best pipeline is stored [43, 11].

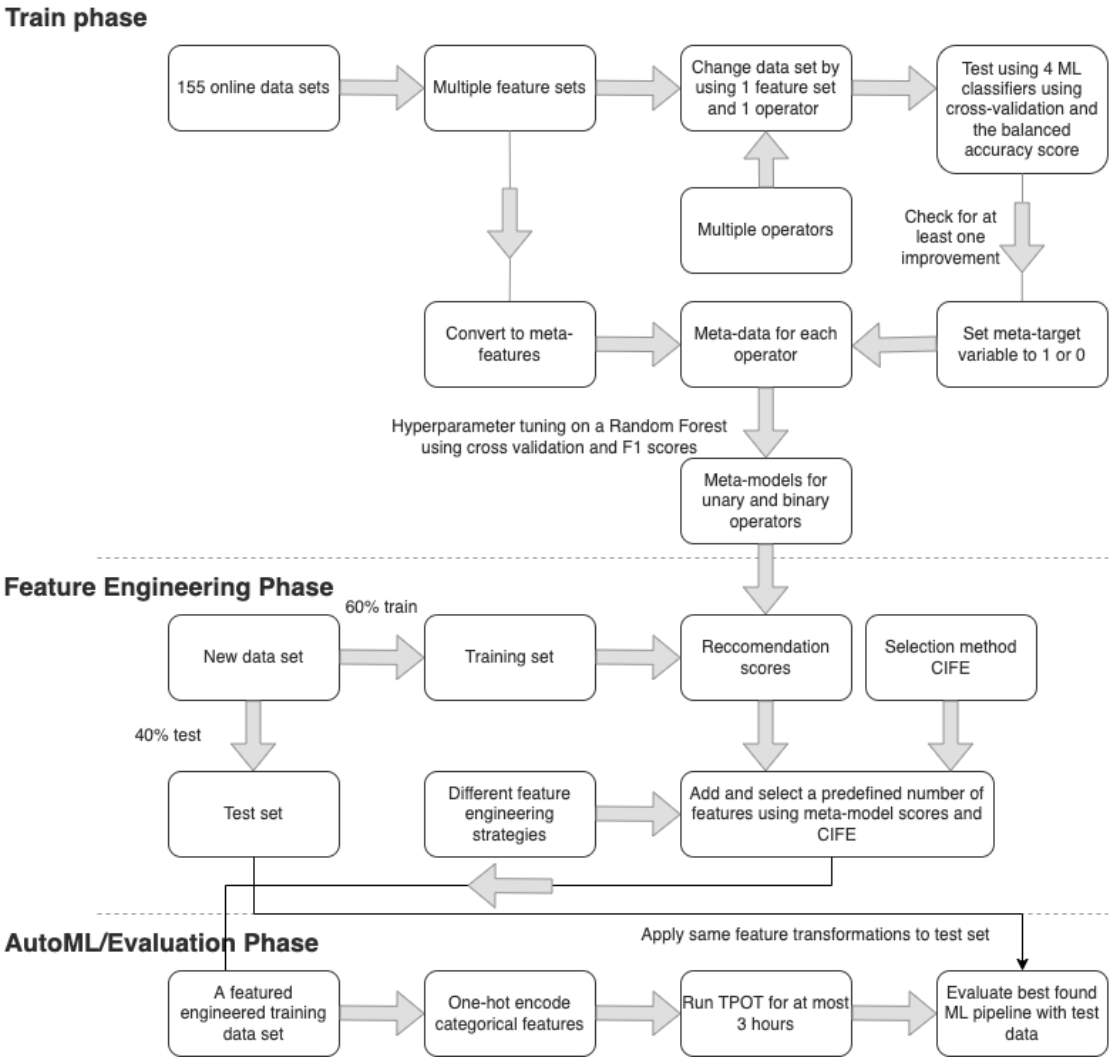


Figure 4: Complete overview of GELFE.

3.4. Evaluation Methodology

The test sets which were previously stored are used, after every minute, to evaluate the accuracy of the best evolved pipeline and determine if there were any improvements. If evidence shows that AutoFE can improve the accuracy of the AutoML pipeline within the same total time limit, it can be concluded that it is indeed beneficial to implement GELFE. Due to the importance of running TPOT for a considerable amount of time, different train/test splits are not considered. To account for this, an additional bootstrap method developed by Davison et al. is used to form proper conclusions regarding the best performing GELFE strategies [44]. Given that the test set has n observations, n observations are sampled a thousand times with

replacement and the BACC score is calculated for each sample. This can be used to form a 95% confidence interval of the test accuracy. Using these confidence intervals, it is possible to state whether or not the model found using one strategy significantly outperforms a model found with another strategy. If the lower limit of one confidence interval is larger than the upper limit of another confidence interval, the first strategy is significantly better than the other one.

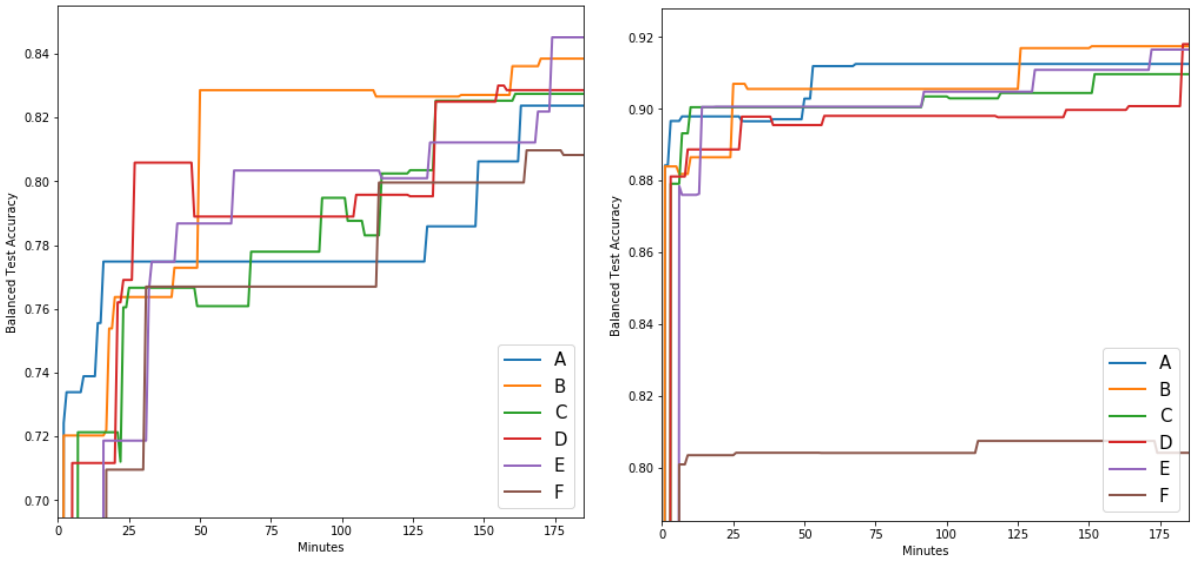
A complete overview of the GELFE algorithm can be seen in Figure 4. The top part of the graph depicts the process of converting 155 data sets to meta-features and meta-target variables to train meta-models for each operator. The middle section then applies the meta-models of each operator on unseen training data sets in order to generate new features and, subsequently, a new data set. This adapted data set is then inputed into TPOT to generate ML pipelines. The best pipelines are evaluated using the test set as can be seen in the last part of the figure.

4. Results

After transforming all feature sets into several meta-observations, which on average took 14 hours per data set, and after training all meta-models, the GELFE procedure can be evaluated. The relevant results are presented in this section. All evaluations have been run on a Microsoft Azure Virtual Machine with 4 2.3 GHz Intel Xeon E5-2673 v4 processors and 128 Gbs of RAM.

The twelve data sets used for evaluation vary significantly in both the number of numerical and categorical features (8 to 167), the number of observations (6,500 to 98,000), and their imbalance levels. First, all training sets were used to feature engineer the different strategies A to F. It was found that the most applied operators were discretizers, group-by-then operators, and arithmetic operators such as multiplication. The AutoFE procedure finished within minutes, except for the strategies which implemented CIFE for larger data sets. The maximum computation time was around an hour such that AutoML still had enough time to evolve good pipelines.

Since, in every minute, we save the best-found pipeline for each training set, we can graph the test score over time. The graphs for the bank-marketing and electricity data sets from OpenML can be found in Figure 5.



(a) bank-marketing test score over time

(b) electricity test score over time

Figure 5: The test accuracy over time for the six AutoFE strategies with bank-marketing and electricity.

Large increases can be found when comparing the scores at the start and at the end of the evolution process of TPOT. Also, not every evolved pipeline which performs better on the train part results in an increase of the test accuracy. The time it takes to complete AutoFE can be seen in the bottom left corner as the strategies with more engineering take more time to start TPOT. It should be noted that another advantage of using strategy A is that it contains fewer features and thus more AutoML generations can be evaluated within the three hours. However, at some point in time, almost all feature engineering strategies surpass the test score found with the benchmark strategy A since useful information is added by GELFE.

In the data set bank-marketing, several strategies significantly outperformed the benchmark strategy. The best BACC test score of 0.845 is found with strategy E. Using the graph with the test accuracy of electricity, it can be seen that strategy F, which uses CIFE and allows original features to be deleted, is not robust. This is the case for multiple data sets. It can be concluded that adding several features on top of the original ones is more reliable. In Table 2, the results of all twelve data sets are shown. This includes the confidence interval after three hours, information regarding the best strategy, and the strategies which are significantly outperformed by the best strategy for each data set.

It can be seen that for most of the data sets, not a single strategy was significantly better than another and the differences between the six strategies are not large. On two data sets, adult and bank-marketing,

Table 2: The confidence intervals of all strategies and all data sets after three hours of AutoFE and AutoML. The best score is underlined and the strategies which are not significantly outperformed by the best strategy are marked in italics.

data set	*A*	*B*	*C*	*D*	*E*	*F*
adult	[0.797,0.809]	*[0.816,0.829]*	*[0.812,0.826]*	*[0.815,0.827]*	[0.800,0.813]	[0.775,0.789]
bank-marketing	[0.815,0.832]	*[0.830,0.847]*	[0.818,0.836]	*[0.819,0.838]*	*[0.837,0.853]*	[0.798,0.818]
electricity	*[0.908,0.916]*	*[0.914,0.921]*	*[0.905,0.914]*	*[0.914,0.922]*	*[0.912,0.921]*	[0.798,0.810]
GesturePhase	*[0.747,0.774]*	*[0.744,0.771]*	*[0.736,0.763]*	*[0.721,0.750]*	*[0.736,0.763]*	*[0.719,0.748]*
higgs	*[0.714,0.723]*	*[0.714,0.723]*	*[0.717,0.726]*	*[0.719,0.728]*	*[0.712,0.720]*	*[0.715,0.724]*
house_16H	*[0.866,0.882]*	*[0.872,0.887]*	*[0.869,0.885]*	*[0.866,0.882]*	*[0.868,0.884]*	*[0.862,0.877]*
jannis	*[0.807,0.815]*	*[0.805,0.814]*	*[0.804,0.813]*	*[0.806,0.814]*	*[0.805,0.814]*	[0.796,0.805]
musk	*[1.000,1.000]*	*[1.000,1.000]*	*[1.000,1.000]*	*[1.000,1.000]*	*[1.000,1.000]*	*[0.999,1.000]*
mv	*[0.999,1.000]*	*[0.998,0.999]*	*[0.998,0.999]*	*[0.999,1.000]*	*[0.999,1.000]*	*[0.999,1.000]*
nomao	*[0.958,0.966]*	*[0.957,0.965]*	*[0.959,0.967]*	*[0.960,0.968]*	*[0.960,0.967]*	*[0.955,0.963]*
Internal_Mil	*[0.933,0.944]*	*[0.934,0.944]*	*[0.933,0.944]*	*[0.931,0.942]*	*[0.931,0.942]*	*[0.933,0.944]*
Internal_Tor	*[0.856,0.867]*	*[0.857,0.868]*	*[0.855,0.866]*	*[0.853,0.864]*	*[0.854,0.866]*	*[0.855,0.867]*

Table 3: The average BACC test scores of the benchmark strategy and all AutoFE strategies, including the average of the best AutoFE strategy for each data set.

Strategy	*A*	*B*	*C*	*D*	*E*	*F*	Best AutoFE strategy
Average test score	0.8721	0.8755	0.8727	0.8726	0.8742	0.8565	0.8768

significant performance increases were found by applying feature engineering. Besides the equal accuracy found for the data set musk, doing no feature engineering was slightly better than all feature engineering strategies, which is partially due to the more generations which can be processed for the smaller data sets. In Table 3, the average results for all strategies can be found.

On average, the TPOT BACC test score was 0.872 for strategy *A* whereas this value was higher for all strategies *B* to *E*. The best AutoFE strategy for each data set scored 0.877 on average. This means that AutoFE can be used on these twelve data sets to find a 0.54% increase. Only smaller differences were found between the strategies *B* to *E*. The best two strategies on average are *B* and *E*. The higher average for strategy *B* does not mean that binary operators should not be explored, only that this data set contains fewer features and thus more pipelines could be evaluated. The average score of strategy *E* indicates that it is beneficial to use the selection method to decide which of the many features should be added to the feature space.

We proceed with showing the overall results for all twelve evaluation data sets regarding the best and top strategy after three hours of feature engineering and pipeline optimization, where a strategy that is not significantly outperformed by the best strategy is called a top strategy. In Table 4, the best strategy is given,

along with the information which other strategies are belonging to the top strategies. All other strategies for a certain data set are thus the ones which are significantly outperformed by the best strategy for a certain data set.

Table 4: An overview which strategies are the best and the top strategies for all evaluation data sets.

Name	X_{FE}^{A}	X_{FE}^{B}	X_{FE}^{C}	X_{FE}^{D}	X_{FE}^{E}	X_{FE}^{F}
adult		Best	Top	Top		
bank-marketing		Top		Top	Best	
electricity	Top	Top	Top	Best	Top	
GesturePhase	Best	Top	Top	Top	Top	Top
higgs	Top	Top	Top	Best	Top	Top
house_16H	Top	Best	Top	Top	Top	Top
jannis	Best	Top	Top	Top	Top	
musk	Best	Best	Best	Best	Best	Top
mv	Best	Top	Top	Top	Top	Top
nomao	Top	Top	Top	Best	Top	Top
Internal_Mil	Top	Best	Top	Top	Top	Top
Internal_Tor	Top	Best	Top	Top	Top	Top

For the musk data set, five strategies are denoted as the best strategy as the test score for all these data sets was 1.000 after three hours. It can be seen that for eight out of the twelve evaluation data sets, no significant differences were found. Strategy *B* and *D* always belonged to the set of top strategies. On two data sets, adult and bank-marketing, significant performance increases were found with doing feature engineering.

For all data sets, it is also interesting to analyze the effect of both AutoML and AutoFE on the considered data sets. Therefore, in Table 5, we present the confidence intervals for the balanced test accuracy when only training a standard Random Forest classifier (RF), when only AutoML is used for three hours (strategy *A*), and the best-found scores when the five feature engineering strategies are applied together with the previous two methods.

Given this table and the previously discussed results, the following can be concluded about AutoFE and AutoML in this study. For all data sets, the impact of the AutoML pipeline optimization is larger than the impact of feature engineering. One important remark is that a standard random forest is not a fair comparison against the extensive evolution process of AutoML which also makes this difference extra large. Moreover, the meta-models used for feature engineering are not built specifically for a random forest. For the data set electricity, among others, the performance of AutoFE + RF was lower than the performance of RF, suggesting

Table 5: The confidence intervals for four different strategies. RF denotes a standard Random Forest classifier fitted to the train data and evaluated on the test data set. The AutoML confidence intervals are the ones found after evolving a pipeline for three hours. AutoFE can be used before both of these ML methods and the best-found confidence interval over the different feature engineering strategies is stated.

Data Set	RF	AutoML	AutoFE + RF	AutoFE + AutoML
adult	[0.773, 0.787]	[0.797, 0.809]	[0.778, 0.792]	[0.816, 0.829]
bank-marketing	[0.652, 0.673]	[0.815, 0.832]	[0.655, 0.676]	[0.837, 0.853]
electricity	[0.878, 0.888]	[0.908, 0.916]	[0.849, 0.859]	[0.914, 0.922]
GesturePhase	[0.652, 0.679]	[0.747, 0.774]	[0.662, 0.691]	[0.744, 0.771]
higgs	[0.676. 0.685]	[0.714, 0.723]	[0.676, 0.685]	[0.719, 0.728]
house_16H	[0.830, 0.848]	[0.866, 0.882]	[0.823, 0.842]	[0.872, 0.887]
jannis	[0.776, 0.785]	[0.807, 0.815]	[0.778, 0.787]	[0.806, 0.814]
musk	[0.919, 0.951]	[1.000, 1.000]	[0.986, 0.998]	[1.000, 1.000]
mv	[0.994, 0.996]	[0.999, 1.000]	[0.994, 0.996]	[0.999, 1.000]
nomao	[0.952, 0.960]	[0.958, 0.966]	[0.954, 0.962]	[0.960, 0.968]
Internal_Mil	[0.916, 0.928]	[0.933, 0.944]	[0.916, 0.929]	[0.934, 0.944]
Internal_Tor	[0.847, 0.858]	[0.856, 0.867]	[0.846, 0.858]	[0.857, 0.868]

that in this context the AutoFE method should not be applied. However, for this data set, the performance of AutoFE + AutoML was better than AutoML alone. This insight shows that AutoFE is general as it is not biased towards one specific algorithm.

5. Conclusion

This paper focuses on the combination of two subfields of machine learning: feature engineering and model selection. Both parts of the ML life cycle are important and time-consuming such that it can be conveninent to automate them, especially given the shortage of data scientists. Both AutoFE and AutoML have been able to show their potential in literature, but have never been combined in order to find their joint potential.

In this paper, we have explored the use of meta-learning as a relatively fast and general AutoFE approach called GELFE in combination with the more often applied AutoML method TPOT. A large collection of data sets was gathered in order to create meta-models for multiple operators. These were then used to predict the best operators for each feature set. In the different AutoFE strategies, a number of constructed features with the best recommendation scores were added to the raw data set. Furthermore, a selection method based on conditional mutual information was implemented in order to select between a large number of recommended new features.

The strategies were evaluated using several benchmark data sets and significant increases were found using AutoFE for some data sets, while significant decreases were never found. On average, AutoFE improved the test accuracy after three hours by 0.54%. Even though this difference is not extremely large, it can already be valuable for companies with a large turnover. Moreover, this increase was found even though more pipelines could be evaluated for the benchmark strategy within the time limit of three hours. Therefore, a final conclusion can be formed. It can be stated that automated feature engineering can be used to increase the accuracy of the general automated machine learning process, while not increasing the computation time. Similar to AutoML, this AutoFE method can be used to make the complete ML process more efficient as it can quickly find multiple feature recommendations for further exploration.

In further research, more in-depth research should be done on the different strategies and time limits for each data set. New strategies can also be made using the information presented in this paper. Perhaps, fewer features should be added to improve the trade-off between more information and more AutoML generations. Besides more research on new strategies, more directions are possible now that we have shown the initial usefulness of combining AutoFE and AutoML. Meta-models and strategies can also be created for multi-class classification and regression problems and in-depth analyses can be performed on the importance of the different operators and meta-features. Also, we would like to compare our approach with the use of LLMs for AutoFE in conjunction with AutoML [45, 46]. Last, we would like to compare our approach with state-of-the-art deep learning and ensemble learning solutions in concrete domains [47, 48].

References

[1] Gartner, Top trends from Gartner hype cycle for digital government technology, 2018, Gartner, accessed: 1 March 2024 (September 2018).
URL https://gtnr.it/2MJGtoM

[2] I. H. Witten, E. Frank, M. A. Hall, C. J. Pal, Data Mining: Practical Machine Learning Tools and Techniques, Morgan Kaufmann, 2016.

[3] A. Elola, J. Del Ser, M. N. Bilbao, C. Perfecto, E. Alexandre, S. Salcedo-Sanz, Hybridizing cartesian genetic programming and harmony search for adaptive feature construction in supervised learning problems, Applied Soft Computing 52 (2017) 760–770.

[4] P. Christensson, GIGO definition, accessed: 1 March 2024 (2015).
URL https://techterms.com/definition/gigo

[5] insideAINews, Infographic: The data scientist shortage, insideAINews, accessed: 1 March 2024 (August 2018).
URL https://bit.ly/2w1dYZs

[6] W. Markow, S. Braganza, B. Taska, S. Miller, D. Hughes, The quant crunch: How the demand for data science skills is disrupting the job market, in: Burning Glass Technologies, IBM, Burning Glass, Business-Higher Education Forum, 2017, accessed: 2022-02-22.
URL https://ibm.co/2yiLsWS

[7] P. Dangeti, Statistics for Machine Learning, Packt Publishing Ltd, 2017.

[8] L. A. Kurgan, P. Musilek, A survey of knowledge discovery and data mining process models, The Knowledge Engineering Review 21 (1) (2006) 1–24.

[9] A. Karpatne, I. Ebert-Uphoff, S. Ravela, H. A. Babaie, V. Kumar, Machine learning for the geosciences: Challenges and opportunities, IEEE Transactions on Knowledge and Data Engineering 31 (8) (2019) 1544–1554.

[10] M. Feurer, A. Klein, K. Eggensperger, J. Springenberg, M. Blum, F. Hutter, Efficient and robust automated machine learning, in: Proceedings of the 28th Annual Conference on Neural Information Processing Systems (NIPS 2015), Curran Associates, 2015, pp. 2962–2970.

[11] R. S. Olson, N. Bartley, R. J. Urbanowicz, J. H. Moore, Evaluation of a tree-based pipeline optimization tool for automating data science, in: Proceedings of the 18th Genetic and Evolutionary Computation Conference 2016 (GECCO 2016), ACM, 2016, pp. 485–492.

[12] E. LeDell, S. Poirier, H2O AutoML: scalable automatic machine learning, in: 7th ICML Workshop on Automated Machine Learning (AutoML 2020), 2020.

[13] J. Ma, D. Lei, Z. Ren, C. Tan, D. Xia, H. Guo, Automated machine learning-based landslide susceptibility mapping for the three gorges reservoir area, China, Mathematical Geosciences 56 (5) (2024) 975–1010.

[14] G. Katz, E. C. R. Shin, D. Song, Explorekit: Automatic feature generation and selection, in: Proceedings of the 16th IEEE International Conference on Data Mining (ICDM 2016), IEEE, 2016, pp. 979–984.

[15] F. Nargesian, H. Samulowitz, U. Khurana, E. B. Khalil, D. Turaga, Learning feature engineering for classification, in: Proceedings of the 26th International Joint Conference on Artificial Intelligence (IJCAI 2017), Vol. 17, 2017, pp. 2529–2535.

[16] M. Gada, Z. Haria, A. Mankad, K. Damania, S. Sankhe, Automated feature engineering and hyperparameter optimization for machine learning, in: Proceedings of the 7th International Conference on Advanced Computing and Communication Systems (ICACCS 2021), IEEE, 2021, pp. 981–986.

[17] R. Kohavi, G. H. John, Wrappers for feature subset selection, Artificial intelligence 97 (1-2) (1997) 273–324.

[18] S. Li, D. Wei, Extremely high-dimensional feature selection via feature generating samplings, IEEE Transactions on Cybernetics 44 (6) (2014) 737–747.

[19] J. M. Kanter, K. Veeramachaneni, Deep feature synthesis: Towards automating data science endeavors, in: Proceedings of the 2nd IEEE International Conference on Data Science and Advanced Analytics (DSAA 2015), IEEE, 2015, pp. 1–10.

[20] B. Tran, B. Xue, M. Zhang, Genetic programming for feature construction and selection in classification on high-dimensional data, Memetic Computing 8 (1) (2015) 3–15.

[21] S. Markovitch, D. Rosenstein, Feature generation using general constructor functions, Machine Learning 49 (1) (2002) 59–98.

[22] H. Liu, L. Yu, Toward integrating feature selection algorithms for classification and clustering, IEEE Transactions on Knowledge and Data Engineering 17 (4) (2005) 491–502. doi:10.1109/TKDE.2005.66.

[23] J. Li, K. Cheng, S. Wang, F. Morstatter, R. P. Trevino, J. Tang, H. Liu, Feature selection: A data perspective, ACM Computing Surveys 50 (6) (2018) 94.

[24] Z. Zhang, F. Zhang, L. Mao, C. Chen, H. Ning, DFS-WR: A novel dual feature selection and weighting representation framework for classification, Information Fusion 104 (2024) 102191.

[25] S. Kullback, R. A. Leibler, On information and sufficiency, The Annals of Mathematical Statistics 22 (1) (1951) 79–86.

[26] D. Lin, X. Tang, Conditional infomax learning: an integrated framework for feature extraction and fusion, in: Proceedings of the 9th European Conference on Computer Vision (ECCV 2006), Springer, 2006, pp. 68–82.

[27] H. Peng, F. Long, C. Ding, Feature selection based on mutual information: criteria of max-dependency, max-relevance, and min-redundancy, IEEE Transactions on Pattern Analysis & Machine Intelligence 8 (2005) 1226–1238.

[28] F. Hassan, S. F. Hussain, S. M. Qaisar, Fusion of multivariate EEG signals for schizophrenia detection using CNN and machine learning techniques, Information Fusion 92 (2023) 466–478.

[29] O. Dor, Y. Reich, Strengthening learning algorithms by feature discovery, Information Sciences 189 (2012) 176–190.

[30] M. Abdar, A. Mehrzadi, M. Goudarzi, F. Masoudkabir, L. Rundo, M. Mamouei, E. Sala, A. Khosravi, V. Makarenkov, U. R. Acharya, et al., Binarized multi-gate mixture of bayesian experts for cardiac syndrome X diagnosis: A clinician-in-the-loop scenario with a belief-uncertainty fusion paradigm, Information Fusion 97 (2023) 101813.

[31] V. V. De Melo, W. Banzhaf, Kaizen programming for feature construction for classification, in: Genetic Programming Theory and Practice XIII, Springer, 2016, pp. 39–57.

[32] K. Krawiec, Genetic programming-based construction of features for machine learning and knowledge discovery tasks, Genetic Programming and Evolvable Machines 3 (4) (2002) 329–343.

[33] M. G. Smith, L. Bull, Genetic programming with a genetic algorithm for feature construction and selection, Genetic Programming and Evolvable Machines 6 (3) (2005) 265–281.

[34] B. Xue, M. Zhang, Y. Dai, W. N. Browne, PSO for feature construction and binary classification, in: Proceedings of the 15th Annual Conference on Genetic and Evolutionary Computation (GECO 2013), Vol. 8602 of Lecture Notes in Computer Science, ACM, 2013, pp. 137–144.

[35] C. Thornton, F. Hutter, H. H. Hoos, K. Leyton-Brown, Auto-weka: Combined selection and hyperparameter optimization of classification algorithms, in: 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD 2013), ACM, 2013, pp. 847–855.

[36] A. Truong, A. Walters, J. Goodsitt, K. Hines, C. B. Bruss, R. Farivar, Towards automated machine learning: Evaluation and comparison of automl approaches and tools, in: 2019 IEEE 31st International Conference on Tools with Artificial Intelligence (ICTAI), IEEE, 2019, pp. 1471–1479. doi:10.1109/ICTAI.2019.00209.

[37] J. Vanschoren, J. N. Van Rijn, B. Bischl, L. Torgo, OpenML: networked science in machine learning, ACM SIGKDD Explorations Newsletter 15 (2) (2014) 49–60.

[38] J. C. Gower, A general coefficient of similarity and some of its properties, Biometrics (1971) 857–871.

[39] J. Brownlee, How to know if your machine learning model has good performance, accessed: 1 March 2024 (April 2018).
URL https://bit.ly/2K0E2ZY

[40] R. S. Olson, W. La Cava, Z. Mustahsan, A. Varik, J. H. Moore, Data-driven advice for applying machine learning to bioinformatics problems, arXiv preprint arXiv:1708.05070 (2018).

[41] X. Tang, Y. Dai, Y. Xiang, L. Luo, An interaction-enhanced feature selection algorithm, in: Proceedings of the 22nd Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD 2018), Springer, 2018, pp. 115–125.

[42] J. M. Sotoca, F. Pla, Supervised feature selection by clustering using conditional mutual information-based distances, Pattern Recognition 43 (6) (2010) 2068–2081.

[43] R. S. Olson, TPOT repository, accessed: 1 March 2024 (2016).
URL https://epistasislab.github.io/tpot/

[44] A. C. Davison, D. V. Hinkley, Bootstrap Methods and their Application, Vol. 1, Cambridge University Press, 1997.

[45] S. Malberg, E. Mosca, G. Groh, FELIX: Automatic and interpretable feature engineering using LLMs, in: Proceedings of the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases 2024 (ECML PKDD 2024), Vol. 14944 of Lecture Notes in Computer Science, Springer, 2024, pp. 230–246.

[46] N. Hollmann, S. Müller, F. Hutter, Large language models for automated data science: Introducing CAAFE for context-aware automated feature engineering, in: Proceedings of the 37th Annual Conference Advances on Neural Information Processing Systems (NIPS 2023), Curran Associates, 2023, pp. 44753–44775.

[47] W. Li, L. Wu, X. Xu, Z. Xie, Q. Qiu, H. Liu, Z. Huang, J. Chen, Deep learning and network analysis: classifying and visualizing geologic hazard reports, Journal of Earth Science 35 (4) (2024) 1289–1303.

[48] D. Lei, J. Ma, G. Zhang, Y. Wang, X. Deng, J. Liu, Bayesian ensemble learning and Shapley additive explanations for fast estimation of slope stability with a physics-informed database, Natural Hazards (2024) 1–30.