Automatic feature generation and selection in predictive analytics solutions

Suzanne van den Bosch

Suzanne van den Bosch: *Automatic feature generation and selection in predictive analytics solutions*
Master thesis Computing Science
Faculty of Science, Radboud University
Student number: s4021444

SUPERVISORS:
Arjen de Vries, Radboud University
Edwin de Jong, Quintiq
Stieneke de Lange, Quintiq

Nijmegen, June 2017

Abstract

In this thesis we proposed a feature generation and selection method called Feature Extraction and Selection for Predictive Analytics (FESPA). This method aims to improve the current Predictive Analytics solution of Quintiq, which had only options for manual feature selection and no options for feature generation. We have discovered that the proposed method does not decrease performance. In most cases, however, it does also not improve the performance. For the data sets where the performance increased, the improvement is significant.

Acknowledgements

I would like to thank a number of people who have supported me during the process of this master thesis.

First, I would like to thank Arjen de Vries, for giving me advice about the direction of my research, for ensuring me I did not have to be stressed about little things like the final presentation when it is still two months away, and for helping me whenever I got stuck in figuring out the ExploreKit algorithm, the implementation or the writing of this thesis. I would also like to thank Elena Marchiori for taking the time and effort to be the second reviewer for this thesis.

I would like to thank my colleagues and supervisors at Quintiq. Edwin de Jong for giving me the opportunity to do this research at Quintiq, for helping me define the scope of this thesis, and giving me the freedom to define a research scope that appealed to me most. Stieneke de Lange for patiently explaining the perspective of the user to me and for the useful feedback while writing this thesis. That made this thesis a lot more readable. Harm Buisman for all his help with understanding the Predictive Analytics solution and feedback on my implementation of FESPA in the Quintiq software.

Last, but certainly not least, I would like to thank my boyfriend Frank Blom and my parents, Rob and Marieke, for all their loving support and unwavering trust in my capabilities. Without them none of this would have been possible.

Contents

Chapter 1

Introduction

Data processing is the collection and manipulation of data. In this thesis, we focus on data processing involving machine learning.

The focus of machine learning research has mainly been on the learning algorithms. This may be due to the limited amount of available data. Over the years, the available technology became more advanced and that has created the possibility to store significantly more data. With the increase of data, it has become clear that the representation of the data, which is the input of a learning algorithm, can have a significant impact on the performance of that learning algorithm. Currently, there is more available data, but that data contains more noisy and relatively less useful data. Therefore, adequate data pre-processing is becoming more and more important.

Data pre-processing is a collective name for all methods that aim to ensure the quality of the data. We focus on two methods within data pre-processing called feature generation and feature selection, explained in more detail in Section 1.1 and 1.2 respectively.

Besides feature generation and feature selection, there is a process which aims to detect and correct inconsistent and missing values in the data. This process is called data cleaning. Data cleaning is also an important part of the data pre-processing phase, but is out of the scope of this thesis.

In this thesis, we address the problem of efficient and effective feature generation and feature selection for a specific machine learning problem. We propose a method that will improve the current Predictive Analytics solution of Quintiq, described further in Section 1.3. The proposed method is based on an existing implementation of a feature generation and selection method called ExploreKit, further discussed in Section 2.3.

Our research objective of improving the current Predictive Analytics solution for Quintiq has two aspects that need to be taken into account; first the scientific results should be improved, however since the tool is used by companies in predicting planning aspects on the short until long term horizon it also means it should produce results in a timely manner. This means that, as part of our research, we are not going for the optimum scientific solution, but we try to find a balance between improved results and fast results. Since this balance can be different for each business case, the approach we took aims to make the resulting Predictive Analytics solution flexible and configurable by the user. One example is that feature generation and selection methods have been split and can be run separately, as well as the fact that their parameters can be set.

1.1 Feature generation

Feature generation is also known as feature construction, feature extraction or feature engineering. There are different interpretations of the terms feature generation, construction, extraction and engineering. Some nearly equivalent, yet differing definitions for these terms are; construction of features from raw data [11, 6], creating a mapping to convert original features to new features [12], creating new features from one or multiple features [10].

In the context of this thesis, feature generation and all its synonyms will be interpreted as the process of creating new features from one or multiple features, unless otherwise specified.

Two goals of feature generation can be dimensionality reduction and accuracy improvement [13, 4]. When the goal of a feature generation method is dimensionality reduction, then the result will be a feature space which contains less features than the original feature space. However, when the goal is accuracy improvement, the resulting feature space will most likely contain more features than the original feature space.

We are primarily interested in feature generation methods where the goal is to improve the accuracy of the predictor. Dimensionality reduction does not have a high priority, since the results of feature generation are input to a feature selection phase aims to reduce the dimensionality of the feature space. Even though the feature generation phase does not have to reduce the dimensionality, it certainly has to take care to not generate an extreme amount of new features.

To illustrate the importance of feature generation, consider the following example in Table 1.1. Here we can see the original feature `Date` and dependent feature `Visitors`, which shows a date and the corresponding number of visitors for a theme park. When looking at just these features, there does not seem to be an obvious pattern between the predicting and the dependent feature. With feature generation, we can extract what kind of day the date is, shown in the `IsWeekendDay` column. This tells us whether the date is a weekend day or not. Now we can see a clear pattern where the number of visitors is significantly higher on weekend days than on week days.

Date	Visitors	IsWeekendDay
‘20-05-17’	19234	yes
‘11-04-17’	5735	no
‘29-01-17’	17914	yes
‘04-05-17’	5496	no
‘29-03-17’	5913	no

Table 1.1: Example features Day and Visitors and extracted feature IsWeekendDay{Day}

Another situation where feature generation can improve performance is when there is feature interaction. Then two (or more) features are not relevant or correlated to the dependent feature on their own, but together they have a (high) influence on the dependent feature. For example, take as features the price and quality of a product. Separately, they will not give much indication of whether a product is purchased often. Combined they have a high correlation to the purchase of the product. If the price is low and the quality high, then the product will be purchased often. However, a low price or a high quality without knowing the other value cannot guarantee that the product will be purchased often. If both price and quality are low, then the product will not be purchased by many customers. The same can be said when both price and quality are high.

1.2 Feature selection

Feature selection tries to find the optimal subset of a given feature set. The problem of feature selection is essentially equivalent to the problem of finding the optimal subset of a given set, which has been shown to be NP-hard. There is one simple method to find the optimal subset, namely calculate the evaluation score for each possible subset. The advantage of this method is that it will always find the optimal subset. A disadvantage is that the complexity is $O(2^n)$, where n is the number of features. This means that for 10 features, already 1024 sets have to be evaluated. This disadvantage is even more influential, because feature selection will become more useful the more features are involved.

Since evaluating every subset separately is practically infeasible, there is need for a different, smarter way to decide the usefulness of each subset.

If the original feature set contains a thousand features, it is highly likely that not all features positively influence the dependent variable. For example, in text mining it is common practice to remove the stop words; the, in, a, of, as, and, with, and many more. Although stop word removal can be seen as a part of the data cleaning step, it can also be seen as a separate feature selection step. Consider every word to be a feature, then the stop words are features that do not contribute to the prediction of the dependent value.

Feature irrelevance is the problem that some features are simply not correlated to the dependent feature. These features can even have a negative effect on the performance of a model.

1.3 Quintiq

Quintiq is one of the top companies in supply chain planning and optimization. It was started in 1997 as a scheduling company by five programmers. It has since grown to a vast company with offices all around the world and over 12 000 users every day. They have been developing a predictive analytics solution as a stand-alone product or to improve other products. The predictive analytics solution as it is at the start of this research had several functionalities. It gives users the option to load in their own data-set, manually select and deselect features, train a decision tree and use that tree to predict new values. The solution helps users select features by giving them the option to calculate the error per feature. With the feature generation and selection method proposed in this thesis, we aim to improve the performance compared to no selection and manual selection.

Since the classifier method implemented in the Quintiq predictive analytics solution is a decision tree, the proposed method in this thesis only has decision trees in its scope. The decision trees implemented in this thesis are based on the definition by Breiman [2].

1.4 Thesis structure

In section 2, we discuss the literature that is relevant to this thesis. In section 3, we discuss the method that is implemented in this thesis. In section 4, the empirical results are discussed. In section 5, we discuss the implications of the results and conclusions.

Chapter 2

Related Work

In this section, we discuss research done that is relevant for the research done in this thesis. Feature generation and selection methods have been shown to improve the performance of machine learning tasks [9, 5, 14].

2.1 Decision tree

The learning algorithm used in this thesis is a decision tree based on the book by Breiman [2]. The decision tree algorithm builds an acyclic connected graph. Each node represents one of the input features.

Take Figure 2.1 as example. This Figure represents a decision tree build on data of passengers of the Titanic. The input features are the sex and age of the passengers. Based on a part of the data, the decision tree makes a decision in each node and determines the class that decision most likely belongs to. For example, this decision tree would predict that a male passenger of age thirty will have died on the Titanic.

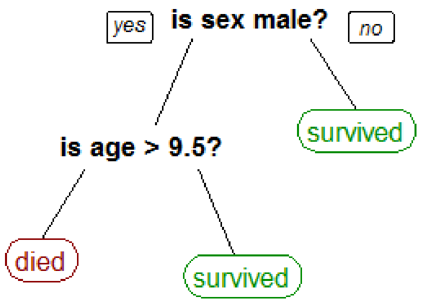


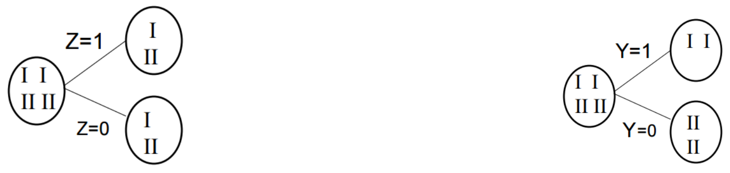
Figure 2.1: Example: Titanic data set

2.2 Information Gain

Information Gain is used as a baseline for the proposed method in this thesis. Information Gain is a method that can be applied to a learning algorithm to find how much information every feature contains.

For a decision tree, it will look at the nodes and for each node calculate how much information is gained by the split. For example, consider the decision tree in Figure 2.2(a). Here the distribution of the classes stays consistent when splitting from the root to the two leaf nodes. This means that doing this split has gained the learner no new information. Feature Z will therefore have a low Information Gain score.

Now consider the decision tree in Figure 2.2(b). Here the distribution of the classes completely changed, with class I being contained in the first leaf and class II being contained in the second leaf. Here the split has gained the learner all the information it needed. Feature Y will therefore have a high Information Gain score.



(a) (b)

Figure 2.2: Examples of (a) no gain and (b) gain, while using Information Gain

The Information Gain method used in this thesis is the variable importance defined by Breiman [2].

2.3 ExploreKit

ExploreKit [7] is the inspiration for the implemented method for feature extraction and feature selection which is analyzed in this thesis. The method proposed in this thesis is implemented for regression tasks whereas ExploreKit is implemented for classification tasks. Since we compare method from this thesis to ExploreKit, we use a subset of the data sets used for ExploreKit. The ExploreKit method provides a combined method for feature extraction and feature selection. The key contribution of this work is to introduce a classifier to predict the usefulness of features extracted.

The ExploreKit algorithm goes as follows. For a fixed number of iterations, they generate candidate features based on the current feature set. F_0 is the original feature set. Then they rank the candidate features by a background classifier. The background classifier is explained in more detail in Section 3.2. Starting with the highest ranking candidate, they evaluate the set containing the current features and the current candidate feature on the classifier, which is either a decision tree, support vector machine or random forest. Based on the outcome of this evaluation, they either (1) add the candidate feature to the current feature set, end the iteration and go to the next iteration, (2) mark this candidate feature as best so far and continue to the next candidate feature in the ranking or (3) discard this candidate feature and continue to the next candidate feature.

2.4 Other methods

Several different implementations of feature extraction and selection methods are FICUS [10], Brainwash [1] and Cognito [8].

FICUS has many similarities to the ExploreKit method, see Section 2.3. One of the differences is that FICUS allows for user-specified feature transformations and constraints. FICUS considers two different categories for transformations, namely domain-independent and domain-dependent.

Brainwash is a method designed to help computer programmers. It considers pieces of code to be features. Based on features that other programmers wrote and used and the features that the current programmer wrote, it recommends new features. After the recommendation, the programmer can use the new feature and give feedback to the system about the usefulness of the new feature.

Cognito implements a greedy incremental search approach. It builds a so-called transformation tree. This transformation tree can be seen as a non-cyclic graph where the nodes represent data sets and the edges represent transformations of those data sets. The root is the original data set. The tree can then be searched using breadth-first or depth-first search to find the best data set.

The method proposed by Shadvar [15], uses the Mutual Information measure to generate new features from the original features. This method reduces the dimensionality of the feature space.

There are several other methods that may be just as good as or better than ExploreKit. Evolutionary algorithms are one of these methods[3]. The idea behind evolutionary algorithms is that it starts with some basic blocks and creates new blocks from combinations of existing blocks. However, evolutionary algorithms are so essentially different that discussing them is outside the scope of this thesis.

Chapter 3

Methods

In this section, the proposed method will be explained in detail. In this thesis, we develop FESPA, Feature Extraction and Selection for Predictive Analytics, inspired by ExploreKit [7], but aiming to be more user-friendly by separating feature generation and feature selection into two phases. FESPA consists of two separate methods, one for feature generation and one for feature selection. Pseudo-code for all methods is given in Appendix B.

3.1 Feature generation

In order to create new features from old features, we need to define transformations. These transformations define the mapping that will be used to transform one or multiple features into a new feature. The transformations are called operators. We use operators from three different categories; unary, binary and group-by-then. The three categories are discussed in detail below.

Pseudo-code for the feature generation process is shown in Algorithm 1. Users can choose which operators they want to apply in the feature generation process. In the actual implementation the flags `doApplyUnary`, `doApplyBinary` and `doApplyGroupByThen` are changed to a flag per specific operator instead of a flag per category. This gives the user full control over which operators are used.

3.1.1 Unary Operators

Unary operators can be any transformation that transforms a single feature into a new feature. We have defined five different unary operators, motivated by their use in ExploreKit and described below.

Equal range discretizer The 'Equal range discretizer' operator can only be applied to numeric features. As the name implies, this operator discretizes the input feature. The number of groups to which the input feature should be discretized is a user-configurable constant. From the range of the input feature and the number of groups, it calculates the range per group.

For example, if a numeric feature has a minimum value of 0 and a maximum value of 20, then the range of that feature is 20. Take the number of groups to be 10, then the range per group will be $20/10 = 2$. The first group will then be all values in the range [0,2], the second group (2,4], etcetera.

Standard score The 'Standard score' operator can only be applied to numeric features. The resulting feature value is calculated using Equation 3.1.

$$StandardScore(x) = \frac{x - \bar{x}}{\sigma_x} \tag{3.1}$$

where $\bar{x}$ is the average over x and σ_x is the standard deviation of x.

Day of the week The 'day of the week' operator extracts the day of the week from a `Date` feature. The new value will be a number between zero and six, where zero is Sunday and six is Saturday. This way the learner can better find weekly patterns.

Is weekend The 'is weekend' operator converts a `Date` feature to a logical feature. This new feature contains information about whether or not the date is in the weekend.

Hour of the day The 'hour of the day' operator extracts the hour of the day from a time feature.

3.1.2 Binary Operators

The binary operators are defined as follows:

Definition 3.1. Let x and y be numeric features. Let f be a function that performs a mathematical operation. A binary operator is defined as f(x,y).

The currently available mathematical operations for these operators are the standard mathematical operators; addition, division, multiplication.

3.1.3 Group-by-then operators

The group-by-then operators group the data set based on one or multiple features, called the sources, and perform an action on a separate feature, called the target. The action can be one of five mathematical operations; average, maximum, minimum, standard deviation or count. These group-by-then operators can be explained by `SQL` statements. They are a combination of the `GROUP BY` statement and an aggregate function. This function can be either `MIN`, `MAX`, `MEAN`, or `STDEV`. For example, consider the data-set shown in Table 3.1. If 'Age' is the target feature and 'Gender' is the source feature, then the data will be grouped on 'Gender'. This will result in two groups; 'Male' and 'Female'. Assume we use the minimum operation, then the resulting new feature will be as shown in the third column. 'Gender' can not be the target feature, since it is not numeric.

There is also a `COUNT` group-by-then operator, which differs slightly from other group-by-then operators in that it does not use a target feature. The `COUNT` operator groups the data set based on the source features and then counts the number of instances per group.

Gender	Age	GBTM
Male	24	24
Female	22	22
Male	26	24
Male	26	24
Female	24	22

Table 3.1: Example features Gender and Age and extracted feature GroupByThenMinimum{Gender, Age} (GBTM)

3.1.4 Redundant operators

Scaling is a unary operator which was considered, where the resulting feature value is calculated using Equation 3.2.

$$Scaled(x) = cx \tag{3.2}$$

where c is a constant value and x is the original feature value. The scaling operator is not implemented, because scaling of a feature has no impact on the output of the decision tree. In the decision tree every node contains a decision about a feature. Consider the node that decides over feature x_i. Let $x_i > \theta$ be the optimal decision given the current strategy. Then transform x_i with a scaling operator; $x_i' = cx_i$. If $c = 0$, then all feature values will become 0 and that is clearly not desirable. Either, $x_i' > c\theta$ is the optimal decision if $c > 0$, because then $cx_i > c\theta \Leftrightarrow x_i > \theta$. Or, $x_i' < c\theta$ is the optimal decision if $c < 0$, because then $cx_i < c\theta \Leftrightarrow x_i > \theta$. Either way, we receive an equivalent optimal decision.

We have also considered adding subtraction as an additional binary operator. However, the subtraction operator can be defined as the binary addition operator combined with a scaling operator applied to one of the features; $sub(x, y) = add(x, Scaled(y))$. We have proven that scaling does not influence the structure of the decision tree and therefore the subtraction operator will be equivalent to the addition operator.

3.1.5 Linear transformations

We take a special look at the linear transformations among the operators, because the scalar operator was showed to be useless to the decision tree. The linear transformations is similar to the scalar operator.

Definition 3.2. A linear transformation is any formula that falls within the following description.

$$f(x_1, ..., x_i) = c_1x_1 + c_2x_2 + ... + c_ix_i + c_{i+1}$$

We have shown that scaling has no effect on the output of the decision tree and therefore, we can generalize this definition to the following.

$$f(x_1, ..., x_i) = x_1 + x_2 + ... + x_i$$

We show that linear transformations are useful by the following two examples.

Example 1. Let us consider the linear transformation from Equation 3.3.

$$f(x, y) = x + y \tag{3.3}$$

Let the data set be defined as follows.

$$X = \{1, 2, 3, 1, 2, ...\}$$
$$Y = \{0, 1, 2, 0, 1, ...\}$$
$$T = \begin{cases} 1, & \text{if } y = 1 \\ 2, & \text{if } x = 3 \\ 0, & \text{otherwise} \end{cases}$$

The pattern to determine the dependent variable in this data set is dependent on both X and Y. However, the linear transformation $f(x, y)$ will have a consistent value of 2 on every row. If this is used as input, then the learner will no longer be able to determine whether T is 1, 2 or 3. So, considering this data set, both X and Y carry information about the dependent variable, but $f(x, y)$ does not contain any information about the dependent variable.

This example raises a question: "Is a linear transformation of input features ever useful?" The answer to this question is yes, which will be made clear with another example.

Example 2. Consider again the linear transformation from Equation 3.3. Let the data set be defined as follows.

$$X = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 1, 2, ...\}$$
$$Y = \{1, 2, 3, 4, 1, 2, ...\}$$
$$T = \begin{cases} 1, & \text{if } x + y = 5 \\ 0, & \text{otherwise} \end{cases}$$

Now both X and Y are uniform random and contain (almost) no information about the dependent variable. X does contain some information about the dependent variable, because if X is five of higher, then the dependent variable will be 0. Now, $f(x, y)$ contains all information about the dependent variable.

Granted, both these examples will most likely never occur in real-life situations, since real data is noisier than this and the pattern for determining the dependent variable is never this obvious. However, these examples show that it is impossible to determine if linear transformations will be useful for the learner.

3.2 Feature selection

Instead of using a simple criterion like Information Gain, FESPA uses a classifier to predict the usefulness of features. This classifier is called the background classifier. This classifier bases its decision on so-called meta-features. Both the classifier and the meta-features will be explained in more detail below. The pseudo-code for FESPA is shown in Algorithm 2.

In ExploreKit the feature selection method is intertwined with the feature generation. In every iteration the candidate features are calculated. FESPA is different in that feature selection is separated from feature generation which automatically made it impossible to have feature generation in the feature selection. Feature selection can also be applied to just the original features, whereas ExploreKit always kept the original features in the selected set. The reasoning behind separating the feature generation and selection processes is two-fold. First, it is based on the fact that not all original features are per definition relevant to the learning task. Second, it is to increase the user-friendliness of the application. We are allowing the user to get a better result on short notice, for example by just using the improved feature selection process.

Take for example the `DayOfWeekUnaryOperator`. This extracts the day of the week from a `Date` feature. Let's say the dependent variable contains a weekly pattern from this feature. Then the original feature will not give the weekly pattern information to the learner, since the learner can not see that the 24th of May 2017 and the 31th of May 2017 are both a Wednesday. However, the generated attribute can. In this case it makes no sense to keep the original feature in the selected set, since the generated feature provides different information that is not captured by the representation of the original feature.

3.2.1 Meta-features

Meta-features belong to one of two categories; data-set based and feature based. The data-set based meta-features contain information about the data-set. These meta-features are the same for all features of a certain data-set. Among others, these meta-features contain information about the type of the target feature, the number of instances and features in the data-set, the Root Mean Squared Error (`RMSE`) score for the original features and statistics on the Information Gain scores of the original features.

The feature based meta-features are focused on the corresponding feature. These meta-features are calculated for each feature and are therefore not necessarily the same for all features. These meta-features contain information about the parents of the feature, which are non-existent for original features. They also contain information about the Information Gain score for the new feature combined with the original features.

3.2.2 Background learner

For the background learner, we use a collection of data sets. These data sets can be any data sets. We use public data sets[1]. The process for building the background learner is shown in Algorithm 3. In the experiments we have used a leave-one-out method. We select one data set as focus and the rest of the data sets will serve as background data sets.

For each background data set, the data-set based meta-features are calculated. Then all possible new features are generated, given the defined operators see Section 3.1. Next, we generate the meta-features for each feature, both the newly generated features and the original features. Then all features are evaluated separately on a decision tree. If the `RMSE` score is lower than the `RMSE` score of the original features, then the candidate feature is marked 1 and 0 otherwise. So, every feature gets a 1 or a 0. These values are stored in a meta-feature

[1]All sets used in this research come from `https://www.openml.org/`

called `indicator`. This meta-feature is the target variable for the background learner. All meta-features are then combined into one vast data-set.

When determining the `indicator` value, the `RMSE` score of a single original feature is compared to the `RMSE` score of the original data set, which contains all original features. The majority of the original features will not improve the original error score and will therefore get a negative `indicator`. This means that an original feature is less likely to be get a high score by the background learner and less likely to be in the resulting selected feature set.

A decision tree is then trained on all meta-features and stored in a file. This background learner can then be re-used, without needing to train it every time the selection process is executed.

We added a feature to allow users to add their own data sets to the background classifier. Users are more likely to have data sets that have a strong correlation between them, because the data comes from the same branch or belongs to a similar prediction task. Background data sets with a strong correlation with the focus data set will most likely improve the performance of FESPA. Each time the user adds a new data set to the collection of background sets, the trained background classifiers will be removed. This ensures that the background classifiers used are always up-to-date.

Chapter 4

Results

In this section, we will discuss the results of the experiments, with regards to performance, influence background collections, operators, growth of number of features, and run-time approximation.. We have evaluated FESPA on twelve data sets. These data sets are a subset of the data sets used for ExploreKit. This to better compare the performance of ExploreKit to FESPA. For the evaluation of FESPA the parameter settings were as follows.

- The maximum number of features to be selected is set to 20
- The $threshold_{fs}$ parameter in Algorithm 2 is set to 0.01
- The $threshold_{bg}$ parameter in Algorithm 3 is set to 0
- The data sets are split into three equal partitions and used for three-fold cross validation. For each fold of the validation, a different partition is taken as validation set, in such a way that every partition is used as train set, test set and validation set once. After evaluating a method using the three folds, the average of the resulting error scores is taken as final error score.
- The maximum source features for the group-by-then operators is set to two.

4.1 Performance

For the evaluation of the methods the Root Mean Squared Error (`RMSE`) is used.

We define F-SPA to mean applying only the selection phase of the FESPA method and FE-PA to mean applying only the generation phase of the FESPA method.

Table 4.1 gives the `RMSE` scores of no selection, manual selection, information gain, F-SPA, FE-PA and FESPA. Here we can see how FESPA performs compared to the original methods and Information Gain. Colorized arrows indicate whether the performance has gone up or down compared to the performance of the original data set with no selection.

Figure 4.1 shows the improvement of FESPA over the original performance. We can see that for only one data set the performance is decreased. In many cases the performance has remained constant.

Data set	O	M	IG	F-SPA	FE-PA	FESPA
Space	0.168	↑ 0.175	↑ 0.180	0.168	↑ 0.179	0.168
Delta elevators	0.00161	↓ 0.00157	↑ 0.00185	0.00161	↑ 0.00176	0.00161
Mammography	0.250	↓ 0.242	↑ 0.294	↑ 0.257	↓ 0.240	↓ 0.245
Diabetes	0.454	↑ 0.460	↓ 0.452	↑ 0.437	↑ 0.505	↓ 0.450
Puma_8	3.719	↓ 3.528	↑ 5.257	3.719	↑ 4.043	3.719
Contraceptive	1.200	↓ 1.178	↑ 1.215	↑ 1.200	↑ 1.128	↑ 1.223
Indian liver	0.485	↓ 0.480	↓ 0.449	↓ 0.471	↑ 0.513	↓ 0.446
CPU	3.861	↓ 7.714	↓ 18.11334	3.861	↑ 3.907	3.861
Heart	0.424	↑ 0.428	↓ 0.411	↑ 0.445	↑ 0.438	↓ 0.405
Wind	4.449	↑ 4.533	↑ 5.927	↑ 4.552	↑ 4.711	↓ 4.443
Credit	0.448	↑ 0.527	↑ 0.603	0.448	↑ 0.588	0.448
German credit	0.490	↓ 0.465	↓ 0.472	↓ 0.432	↑ 0.95	↓ 0.465

Table 4.1: `RMSE` score for no selection (O), manual selection (M), information gain (IG), F-SPA, FE-PA and FESPA

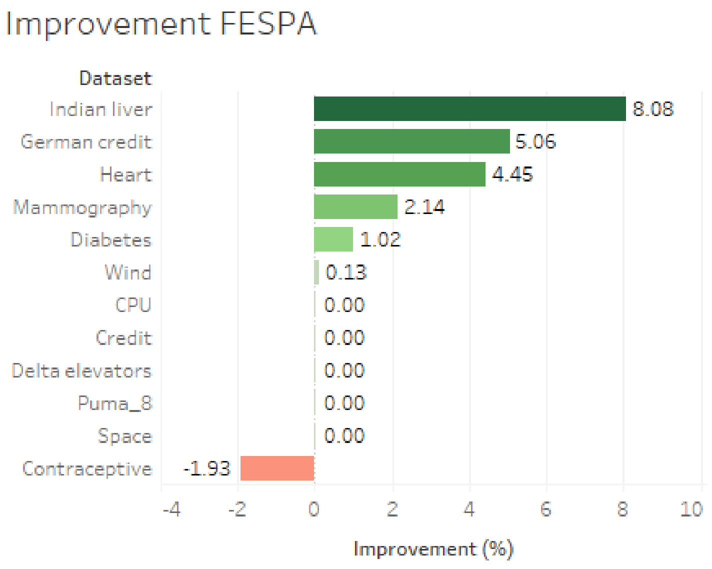
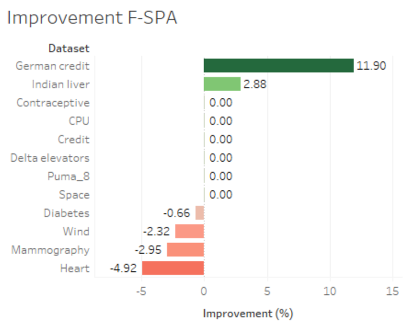
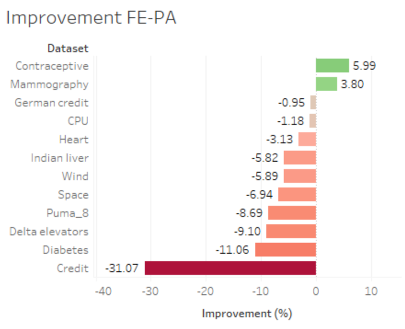


Figure 4.1: Improvement of FESPA over the performance of the original feature set

Figure 4.2 shows the improvement of F-SPA and FE-PA over the original performance. It shows that F-SPA in decreases the performance for a third of the data sets. FE-PA, on the other hand, decreases the performance for almost every data set.



(a)



(b)

Figure 4.2: Improvement of (a) selection phase and (b) generation phase of FESPA, applied seperately

4.2 Influence background collections

An additional experiment on the best performing data set, Diabetes, is conducted. For this experiment, the data sets are grouped as shown in Table 4.2.

Group	Data sets
Medic	Contraceptive, Mammography, Diabetes, Indian liver, Heart
Finance	Credit, German credit
Other	Space, Delta elevators, Puma_8, CPU, Wind

Table 4.2: Grouped data sets

Each group is taken as the collection of background data sets and FESPA is applied on the Diabetes data set. The `RMSE` score for each group is shown in Figure 4.3. Figure 4.3 shows that no group has a significant improvement over any of the other groups.

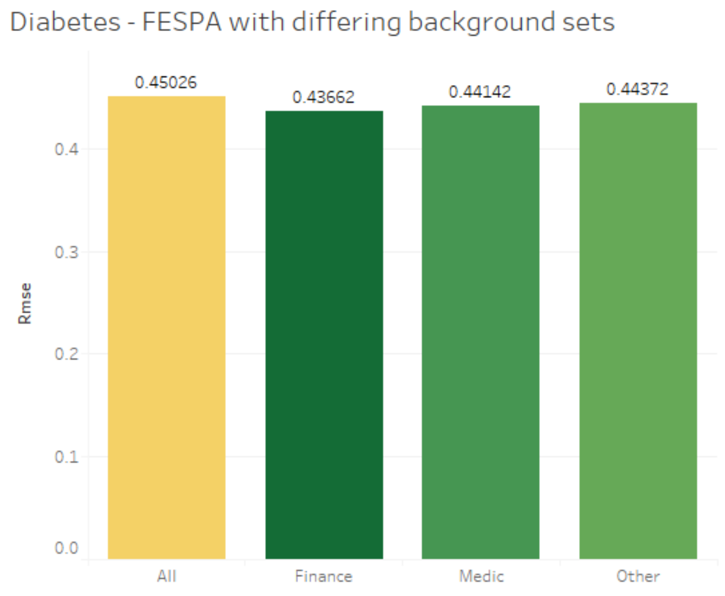


Figure 4.3

Next, we have taken each background data set separately and FESPA is applied on the Diabetes data set. The `RMSE` score for each background data set is shown in Figure 4.4. Figure 4.4 shows that no single data set improves the performance significantly over other data sets.

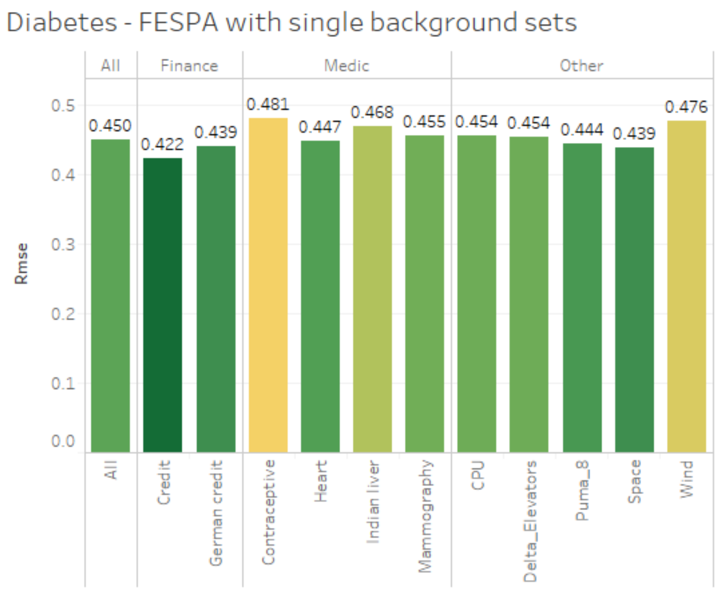


Figure 4.4

4.3 Operators

In Table 4.3, we can see which features were selected by FESPA for the three folds of the cross validation, where it is first fold/second fold/third fold. We can see that features made by unary operators are the least present in the selected set. However, the unary operators are present within the binary and group-by-then operators. For five out of the seven features made by binary features chosen, one or both of the parent features are made by unary operators. Also, the set of features from which the source features are taken consists of all categorical features and for all numeric features the result of the EqualRangeDiscretizer operator applied to them.

Data set	Original	Unary	Binary	GBT
Space	6/6/6	-/-/-	-/-/-	-/-/-
Delta elevators	6/6/6	-/-/-	-/-/-	-/-/-
Mammography	6/-/-	-/-/-	-/2/3	-/-/-
Diabetes	-/8/-	-/-/-	4/-/2	-/-/1
Puma_8	8/8/8	-/-/-	-/-/-	-/-/-
Indian liver	-/-/-	-/-/-	4/3/1	-/1/1
Heart	13/-/-	-/-/-	-/3/6	-/-/-
Credit	15/15/15	-/-/-	-/-/-	-/-/-
German credit	1/1/-	-/-/-	-/1/1	1/0/2
Contraceptive	9/-/-	-/-/-	-/-/-	-/3/5
CPU	12/12/12	-/-/-	-/-/-	-/-/-
Wind	-/14/14	2/-/-	1/-/-	2/-/-

Table 4.3: Operators used for FESPA selected features, for each data set

4.4 Growth of number of features

As mentioned in section 1.1, the feature generator should not produce an extreme amount of new features. We can approximate the number of features generated using the following Equations.

$$maxCombinations(n) = \frac{n * (n-1)}{2} \tag{4.1}$$

$$nUnary(n) = 2n \tag{4.2}$$

$$nBinary(n) = maxCombinations(n) * 4 \tag{4.3}$$

$$nGroupByThen(n) = n * maxCombinations(n-1) * 5 \tag{4.4}$$

To find the complexity of the feature generator, we have to look at the complexity of the aforementioned Equations.

Equation 4.1: $O(maxCombinations(n)) = O(n^2)$

Equation 4.2: $O(nUnary(n)) = O(n)$

Equation 4.3: $O(nBinary(n)) = O(n^2)$

Equation 4.4: $O(nGroupByThen(n)) = O(n^3)$

The complexity of all this combined is equal to the highest complexity among these equations which is $O(n^3)$. Take a look at Figure 4.5, where we have calculated the number of features that would be created and have plotted it against the estimated complexity. Here we can see that the number of features that will be created exceeds the number of features estimated by the complexity function.

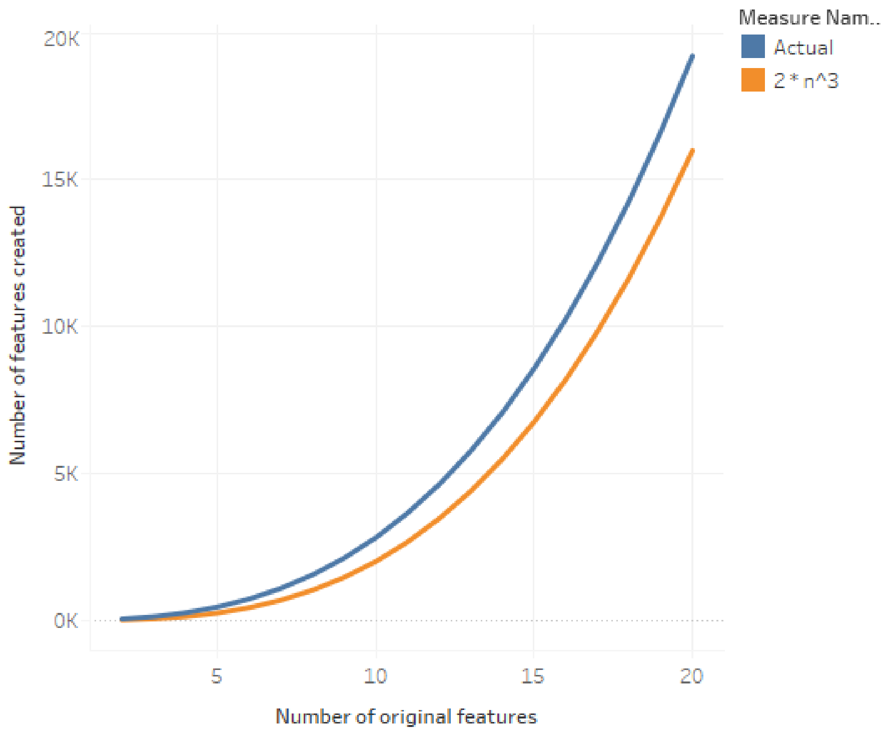


Figure 4.5: Actual generated features vs complexity

4.5 Run-time approximation

What we can see in Figure 4.6 and Figure 4.7 is that the complexity of the run-time of both feature generation and feature selection is non-linear. The run-time is most likely second or third degree polynomial.

For example, you can see that if the original set has 14 features and 30 000 instances, the feature generation phase would take 1500 seconds which is 25 minutes.

The feature selection phase is also clearly the most expensive phase of FESPA. For the same original data set, the run-time can easily be five times as long as the feature generation phase. Take again an original set of 14 features, we were then to apply the feature selection phase to the result of the feature generation, the feature selection phase would take 7000 seconds which is almost two hours.

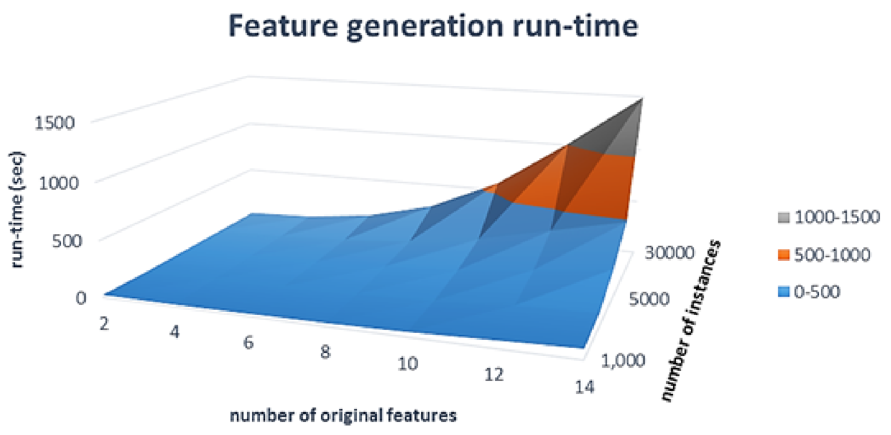


Figure 4.6: Run-time in seconds for feature generation

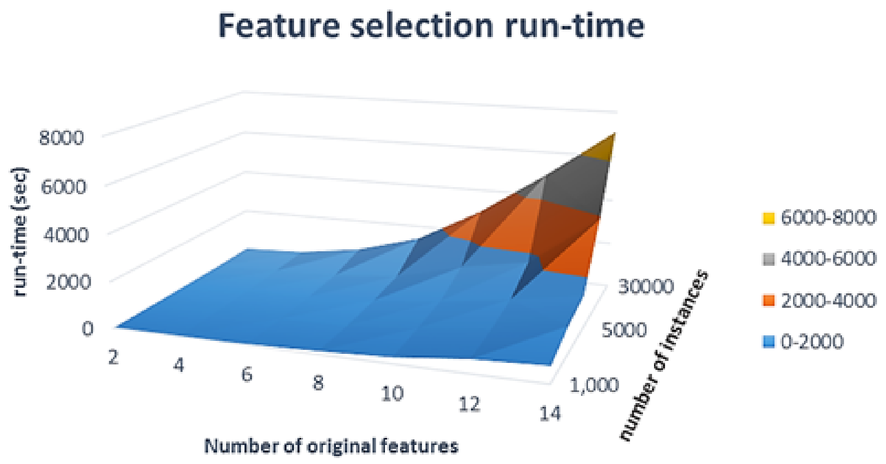


Figure 4.7: Run-time in seconds for feature selection after the feature generation phase, on a table with x original features

Chapter 5

Discussion

In this section, we will discuss implications of the results from Section 4, with regards to performance, operators and time versus accuracy. We will also discuss some ideas for future research.

5.1 Performance

As we can see in Figure 4.1, FESPA has a negative improvement over the original performance for just one data set. We can conclude from this that it is beneficial to apply FESPA, given the user has time. Obviously, applying FESPA, or any other feature generation and/or selection method, is more time-consuming than taking the original feature set as the selected set. More on run-time in Section 5.4. Table 4.1 shows us that generally FESPA outperforms manual selection and Information Gain selection. It may not improve the performance of more data sets than manual selection and Information Gain selection, but FESPA decreases the performance of only one data set whereas the other selection methods decrease the performance of all data sets where they do not improve the performance.

When we compare Figure 4.1 to Figure 4.2(a), we can see that the feature selection phase of FESPA improves the performance if the input contains only original features or also generated features.

When we compare Figure 4.1 to Figure 4.2(b), we see that applying only the feature extraction phase of FESPA decreases the performance in most cases. A user should always apply a selection method after the feature extraction phase of FESPA, since this will improve the performance. Applying the selection method of FESPA gives the greatest improvement and is even faster than the Information Gain method for data sets over 10 original features with all operators applied.

I have found that most original features have as meta-feature indicator a zero. Using a different check on line 8 of Algorithm 3 to give more original features a positive indicator can improve the performance of FESPA on original data sets.

Any small difference between FESPA and ExploreKit may also be due to the difference of implementation, as ExploreKit was implemented in `Java` and FESPA was implemented in `R`. For such a relatively complex method, `Java` is a better choice than R for a programming language, as it is more object-oriented. In Java, it is easier to design custom objects that have their own parameters and methods. However, since R was already integrated in the Quintiq software, it was easier to implement FESPA in R.

5.2 Influence background collections

The hypothesis for the experiment from Section 4.2 was that a data set from a certain group would benefit most from background data sets from that group. However, the Diabetes data set is from the Medic group and in Figure 4.3 we can see that the Medic group is not the best performing group. In Figure 4.4 we can see that the best performing background data sets are in the Finance and Other group. Not only do the best performing background data sets not belong to the Medic group, one of the data sets with the worst error score does belong to the Medic group. This tells us that there is no apparent correlation within a certain group of data sets.

5.3 Operators

In table 4.3, we can see which features are selected by FESPA. For five of the twelve data sets, FESPA could not find one or more features, generated or original, that improve the original performance. As a result, FESPA then simply returns the original feature set, see Algorithm 2. This ensures there is never a decrease in performance.

The binary operators addition and multiplication are most present in the selected sets, whereas division is never selected. The group-by-then operators with aggregate functions `Avg` and `Stdev` are by far the most represented in the selected set.

For nearly half of the runs, FESPA actually selected features. Therefore, even though the binary division operator and the group-by-then `Count`, and `Min` operators are not present in any selected set, we cannot confidently conclude that these operators will never be of importance for a data set.

5.4 Time versus accuracy

During this research, there has been one main theme. This was the problem of finding a balance between improving the accuracy and keeping the run-time within acceptable duration.

From a research point-of-view it is interesting to find the method and parameter settings that optimize the performance. However, from a business point-of-view, the user-friendliness of the application has to be maintained. This means that the run-time of the implemented methods cannot exceed a certain duration or the functionality would not be possible to use by day to day operational tasks. The exact maximum acceptable duration depends on the application, the business operation this is applied to, the method, and the run-time of alternative methods.

The bottleneck for FESPA is the group-by-then operators, since the complexity of these operators is n^3, where n is the number of features, see Section

4.4. We conclude that if results are needed in limited amount of time, it is best to use only a few or none of the group-by-then operators. Since in our results the most chosen group-by-then operators were `Avg` and `Stdev`, we recommend to choose these.

The results from Section 4.5 have lead to the decision to make the solution more adaptable by the user to make it fit their time frame. Therefore, in the implementation, users can choose which operators to apply. If the user chooses less operators, then the run-time of both the feature generation phase and the feature selection phase will decrease. This makes for a much more user-friendly application.

5.5 Future work

The application can be made more user-configurable by enabling the user to not only select which operators to use, but also which operators to use for which features. This will also help keep the resulting feature space of the feature generator small. However, this will require a greater understanding of the method than the current implementation.

In future work, a functionality can be added that indicates how long the feature generator or selector will run given the selected settings. Either by defining a function with the possible settings as arguments, or as a prediction task in itself.

Possibly, the application can be extended with a user-configurable classifier method. Currently, all classifiers implemented are decision trees. But, for example, if users want to use support vector machines instead of decision trees. The predictions of the meta-features will currently be based on the performance of the background set features on a decision tree. However, whether a certain feature is influential can depend on the type of classifier used. Therefore it is recommended to make the classifier method for the entire process user-configurable. Then the predictions of the meta-features will be based on the performance of the background set features on support vector machines.

Bibliography

[1] Michael R Anderson, Dolan Antenucci, Victor Bittorf, Matthew Burgess, Michael J Cafarella, Arun Kumar, Feng Niu, Yongjoo Park, Christopher Ré, and Ce Zhang. Brainwash: A data system for feature engineering. In *CIDR*, 2013. 7

[2] L. Breiman, J. Friedman, R. Olshen, and C. Stone. *Classification and Regression Trees*. Wadsworth and Brooks, Monterey, CA, 1984. 4, 5, 6

[3] Edwin D De Jong and Tim Oates. A coevolutionary approach to representation development. In *Proc. of the ICML-2002 Workshop on Development of Representations*, 2002. 7

[4] Rezarta Islamaj, Lise Getoor, and W John Wilbur. A feature generation algorithm for sequences with application to splice-site prediction. *Feature Selection for Data Mining: Interfacing Machine Learning and Statistics*, page 42, 2006. 2

[5] Uday Kamath, Kenneth De Jong, and Amarda Shehu. Effective automated feature construction and selection for classification of biological sequences. *PloS one*, 9(7):e99982, 2014. 5

[6] James Max Kanter and Kalyan Veeramachaneni. Deep feature synthesis: Towards automating data science endeavors. In *Data Science and Advanced Analytics (DSAA), 2015. 36678 2015. IEEE International Conference on*, pages 1–10. IEEE, 2015. 2

[7] Gilad Katz, Eui Chul Richard Shin, and Dawn Song. Explorekit: Automatic feature generation and selection. In *Data Mining (ICDM), 2016 IEEE 16th International Conference on*, pages 979–984. IEEE, 2016. 6, 9

[8] Udayan Khurana, Deepak Turaga, Horst Samulowitz, and Srinivasan Partharathy. Cognito: Automated feature engineering for supervised learning. In *International Conference on Data Mining*. IEEE, 2016. 7

[9] Hugh Leather, Edwin Bonilla, and Michael O'boyle. Automatic feature generation for machine learning–based optimising compilation. *ACM Transactions on Architecture and Code Optimization (TACO)*, 11(1):14, 2014. 5

[10] Shaul Markovitch and Dan Rosenstein. Feature generation using general constructor functions. *Machine Learning*, 49(1):59–98, 2002. 2, 7

[11] MuhammadArif Mohamad, Haswadi Hassan, Dewi Nasien, and Habibollah Haron. A review on feature extraction and feature selection for handwritten character recognition. *International Journal of Advanced Computer Science & Applications*, 1(6):204–212, 2015. 2

[12] Hiroshi Motoda and Huan Liu. Feature selection, extraction and construction. *Communication of IICM (Institute of Information and Computing Machinery, Taiwan) Vol*, 5:67–72, 2002. 2

[13] Michael L Raymer, William F. Punch, Erik D Goodman, Leslie A Kuhn, and Anil K Jain. Dimensionality reduction using genetic algorithms. *IEEE transactions on evolutionary computation*, 4(2):164–171, 2000. 2

[14] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015. 5

[15] Ali Shadvar. Dimension reduction by mutual information feature extraction. *International Journal of Computer Science & Information Technology*, 4(3):13, 2012. 7

Appendix A

Overview data sets

In the table below, an overview is given of the data sets used in this research.

Data set	#features	#instances	% numeric features
Contraceptive	9	1.473	66.6%
CPU	12	8.192	100%
Credit	15	690	40%
Delta elevators	6	9.517	100%
Diabetes	8	768	100%
German credit	20	1.000	35%
Heart	13	270	46%
Indian liver	10	585	90%
Mammography	6	11.183	100%
Puma_8	8	8.192	100%
Space	6	3.107	100%
Wind	14	6.574	100%

Table A.1: Characteristics of used data sets

Appendix B

Algorithms pseudo-code

Algorithm 1: Feature generation

```
Data: dataset
Result: dataset containing original and generated features
begin
    if doApplyUnary then
        dataset := applyUnaryOperators(dataset)
    end
    if doApplyBinary then
        binaryFeatures := applyBinaryOperators(dataset)
    end
    if doApplyGroupByThen then
        groupedFeatures := applyGBTOperators(dataset)
    end
    return combine(dataset, binaryFeatures, groupedFeatures)
end
```

Algorithm 2: FESPA - feature selection

```
Input: dataset, rankingThreshold
Output: dataset containing selected features
1  begin
2      selectedSet := $\emptyset$
3      features := dataset.GetFeatures()
4      originalError := EvaluateOnLearner(features.GetOriginal())
5      meta_features := generateAllMetaFeatures(features)
6      rankedFeatures := RankFeatures(features, meta_features)
7      PruneRankedFeatures(rankedFeatures, rankingThreshold)
8      foreach feature in rankedFeatures do
9          improvement := EvaluateOnLearner(selectedSet) -
            EvaluateOnLearner(selectedSet $\cup$ feature)
10         if improvement > $threshold_{fs}$ then
11             selectedSet := selectedSet $\cup$ feature
12         end
13     end
14     if selectedSet == $\emptyset$ then
15         selectedSet := features.GetOriginal()
16     end
17     return selectedSet
18 end
```

Algorithm 3: Building the background learner

```
Input: bg_datasets: background datasets
Output: dataset containing all meta-features of the background datasets
begin
    foreach bg_dataset do
        originalFeatures := dataset.GetFeatures()
        originalError := EvaluateOnLearner(originalFeatures)
        foreach feature in originalFeatures do
            feature_meta := generateFeatureMetaFeatures(feature)
            featureError := EvaluateOnLearner(feature)
            if featureError $\leq$ originalError - $threshold_{bg}$ then
                feature_meta.SetIndicator(1)
            end
            else
                feature_meta.SetIndicator(0)
            end
        end
        dataset_meta := generateDatasetMetaFeatures(bg_dataset)
        dataset := generateFeatures(background_dataset)
        foreach feature in dataset do
            feature_meta := generateFeatureMetaFeatures(originalFeatures
             $\cup$ feature)
            featureError := EvaluateOnLearner(feature)
            if featureError $\leq$ originalError - $threshold_{bg}$ then
                feature_meta.SetIndicator(1)
            end
            else
                feature_meta.SetIndicator(0)
            end
        end
    end
    return meta-features
end
```