Determining Attribute Importance Using an Ensemble of Genetic Programs and Permutation Tests

ANNICA IVERT
Determining Attribute Importance Using an Ensemble of Genetic Programs and Permutation Tests

Relevansbestämning av attribut med hjälp av genetiska program och permutationstester

ANNICA AIVERT
aivert@kth.se

Master’s Thesis in Machine Learning
Degree Program in Information Technology 300 credits
Master Program in Machine Learning 120 credits
Royal Institute of Technology, January 2015

Supervisor at CSC was Örjan Ekeberg
Examiner was Anders Lansner
This thesis work was done at The University of Tokyo

Royal Institute of Technology
School of Computer Science and Communication

KTH CSC
SE-100 44 Stockholm, Sweden

URL: www.kth.se/csc
Acknowledgement

First of all I would like to thank Hitoshi Iba for welcoming me to the Iba laboratory at The University of Tokyo, where this thesis work have been conducted. Also, a big thanks to Claus Aranha, who have provided me guidance and support during this work.

I would also like to thank Stockholms Grosshandels sociedadet (through the Sweden-Japan foundation) and JASSO for their financial support.

Lastly, I would like to thank my supervisor at KTH, Örjan Ekeberg, for his valuable feedback on this report.
Abstract

When classifying high-dimensional data, a lot can be gained, in terms of both computational time and precision, by only considering the most important features. Many feature selection methods are based on the assumption that important features are highly correlated with their corresponding classes, but mainly uncorrelated with each other. Often, this assumption can help eliminate redundancies and produce good predictors using only a small subset of features. However, when the predictability depends on interactions between the features, such methods will fail to produce satisfactory results.

Also, since the suitability of the selected features depends on the learning algorithm in which they will be used, correlation-based filter methods might not be optimal when using genetic programs as the final classifiers, as they fail to capture the possibly complex relationships that are expressible by the genetic programming rules.

In this thesis a method that can find important features, both independently and dependently discriminative, is introduced. This method works by performing two different types of permutation tests that classify each of the features as either irrelevant, independently predictive or dependently predictive. The proposed method directly evaluates the suitability of the features with respect to the learning algorithm in question. Also, in contrast to computationally expensive wrapper methods that require several subsets of features to be evaluated, a feature classification can be obtained after only one single pass, even though the time required does equal the training time of the classifier.

The evaluation shows that the attributes chosen by the permutation tests always yield a classifier at least as good as the one obtained when all attributes are used during training - and often better. The proposed method also fares well when compared to other attribute selection methods such as RELIEFF and CFS.
Referat

Då man handskas med data av hög dimensionalitet kan man uppnå både bättre precision och förkortad exekveringstid genom att enbart fokusera på de viktigaste attributen. Många metoder för att hitta viktiga attribut är baserade på ett grundantagande om en stark korrelation mellan de viktiga attributen och dess tillhörande klass, men ofta även på ett oberoende mellan de individuella attributen. Detta kan å ena sidan leda till att överflödiga attribut lätt kan elimineras och därmed underlätta processen att hitta en bra klassifierare, men å andra sidan också ge missvisande resultat ifall förmågan att separera klasser i hög grad beror på interaktioner mellan olika attribut.

Då lämpligheten av de valda attributen också beror på inlärningsalgoritmen i fråga är det troligtvis inte optimalt att använda sig av metoder som är baserade på korrelationer mellan individuella attribut och dess tillhörande klass, ifall målet är att skapa klassifierare i form av genetiska program, då sådana metoder troligtvis inte har förmågan att fånga de komplexa interaktioner som genetiska program faktiskt möjliggör.

Det här arbetet introducerar en metod för att hitta viktiga attribut - både de som kan klassifiera data relativt oberoende och de som får sina krafter endast genom att utnyttja beroenden av andra attribut. Den föreslagna metoden baserar sig på två olika typer av permutationstester, där attribut permuteras mellan de olika dataexemplaren för att sedan klassifieras som antingen oberoende, beroende eller irrelevanta.

Lämpligheten av ett attribut utvärderas direkt med hänsyn till den valda inlärningsalgoritmen till skillnad från så kallade wrappers, som är tidskrävande då de kräver att flera delmängder av attribut utvärderas.

Resultaten visar att de attribut som ansetts viktiga efter permutationstesten genererar klassifierare som är atminstone lika bra som när alla attribut används, men ofta bättre. Metoden står sig också bra när den jämförs med andra metoder som till exempel RELIEFF och CFS.
5.2 Data ......................................................... 29
   5.2.1 UCI data .............................................. 29
   5.2.2 Cytometry data ...................................... 30
   5.2.3 Constructed data .................................... 31
5.3 Evaluation .................................................. 31

6 Results ....................................................... 33
   6.1 Synthetic data .......................................... 34
      6.1.1 MONK data ......................................... 34
      6.1.2 Constructed data .................................. 37
   6.2 SVM$_{wcbcp}$ ............................................. 38
   6.3 Cytometry data .......................................... 39
      6.3.1 GP .................................................. 40
      6.3.2 SVM ............................................... 43
   6.4 UCI datasets ............................................ 45
      6.4.1 WDBC ............................................... 45
      6.4.2 Wine ............................................... 46
      6.4.3 Vehicle ............................................ 46
      6.4.4 Evaluation ........................................ 47
   6.5 Attribute construction ................................ 48

7 Discussion and Future work ................................ 49
   7.1 Summary of contributions ............................. 49
   7.2 Discussion ............................................. 49

8 Conclusion .................................................. 56

References ..................................................... 58

Appendices .................................................... 62

A Parameters ................................................ 63

B Cytometry data - permutation graphs .................. 65

C UCI data - permutation graphs ........................ 69
Chapter 1

Introduction

With increasingly large amounts of high-dimensional data becoming available, the need to find reliable and cost-effective methods to deal with this data become more and more important. One way to make the data more tangible for the learning algorithms is to reduce the dimensionality and only consider the most important features. Not only can feature extraction result in faster solutions and more reliable predictors, but using less features will also help us better understand the obtained solutions and the underlying process of the phenomena in study. This is especially true if the learning algorithm used have potential to produce easily interpretable rules, like Genetic Programming (GP), where the classifiers can consist of mathematical formulas in a familiar format.

Section 3 will describe the necessary background to this thesis, and in particular Section 3.1 will go into the general problem of feature selection in more detail. The two main categories of feature selection methods, filters and wrappers will be given their own sections, but as for now filter methods can be said to perform a preprocessing of the data by filtering out seemingly irrelevant attributes, often based on correlations between individual attributes and the corresponding target concept. This is done independently of the learning algorithm, whereas wrapper methods work by expanding or reducing subsets of features, whose importance are evaluated by the same learning algorithm that will be used to produce the final classifier.

This thesis will focus on feature selection where the target classifier consists of an ensemble of genetic programs. The formal problem statement can be found in Section 2.

Genetic programs can be envisioned as tree-representations of computer
programs or mathematical formulas, where the operations of choice relate the features to produce the final classification rules. A more in-depth description can be found in Section 3.2. Since genetic programs have the ability to express complex relationships preprocessing methods that do not consider interactions between attributes will not necessarily lend themselves well to GP. On the other hand, wrapper methods are often computationally expensive as they may require the learning algorithm to be run over and over again, and with an increasing amount of features there will also be an exponentially increasing number of possible ways to choose a subset of those. This motivates the use of a feature selection algorithm that is biased with the learning algorithm (GP), but still able to perform feature selection without repetitive re-evaluation for each subset of features.

Kumar Paul and Hitoshi Iba used a voting ensemble of Genetic Programs to predict the cancer class from gene expression data [14]. They observed that many of the attributes frequently used in the genetic programming rules were also known to be associated with different types of cancer. However, very few of the most commonly selected genes were part of the individual rules with the best test accuracies. They hypothesize that this might be due to the genes in the best rules being correlated and only predictive of cancer through their joint interactions, but leave this as an open question to be investigated. If present, such dependent attributes would be more difficult to find as they require a specific configuration to be evolved, and thus are unlikely to appear as often in the rules as attributes with independently predictive powers.

This kind of dependencies seem to be largely overlooked in feature selection literature. Instead, the assumption that the important features are correlated with the class, but mainly uncorrelated with each other is often made [5] [4].

Guyon et al. describe the problems with these approaches in a comprehensive summary over feature selection methods. They also provide a couple of motivating examples, where variables that are useless by themselves become useful when considered together with others [4].

This work aims to not only find features that are important by themselves, but also features that have dependencies, which when exploited provide important predictive powers. In order to do this two types of permutation based evaluation scores are introduced: the within-class permutation and the between-class permutation scores. These will be further described in Section 4, along with some variations.
The basic idea is however that permutations of individual attribute values between data samples will destroy the predictability of the trained classifiers if the attribute has an important role in making the classification. There is however a key difference between obtaining the new attribute value from a sample having the same class-belonging (within-class permutation) or a different one (between-class permutation). If the attribute is independently predictive, swapping its value with that of a sample belonging to the same class might have very little impact on the performance of the classifier, since they are both used to make the same classification. If it, on the other hand, is dependent on other attributes in order to become useful, those dependencies will be broken even when the attribute is permuted with samples within the same class. Making this observation, it is possible to, not only say something about the importance of an attribute, but also in which way it is important.

Since the feature selection works by modifying the data, the classifier only needs to be trained once, avoiding the repetitive re-evaluation of wrappers, while keeping some of their desired properties. There is no need to make any assumptions about the data like in most filters, and since the same learning algorithm is used for both feature selection and classification the desired bias is achieved.

The proposed method will then be evaluated by training classifiers using (1) all the available attributes, (2) the attributes deemed important by the permutation tests and (3) attributes considered important by a selection of standard attribute selection algorithms. Details about the implementation and the data used can be found in Section 5. A comparison of the final classifiers is presented in Section 6. These results will be discussed in Section 7, where possible future work is also outlined. Finally, the findings are summarized in Section 8.
Chapter 2

Problem

2.1 Problem formulation

The motivation behind this thesis is to provide an attribute selection method that selects the best attributes with respect to the learning algorithm used, mainly aiming for a final classifier consisting of an ensemble of genetic programs. The objective is also to be able to detect not only independently predictive attributes, but also attributes that gain most of their predictive powers through joint interactions with other attributes.

The evaluation of the proposed method can thus be summarized with the following questions:

1. Can dependent attributes be detected when tested on artificial data with known dependencies?

2. Does attribute selection using the proposed method result in a better final classifier than attribute selection using other commonly used attribute selection methods?

3. Can meaningful attributes be constructed by adding subtrees created from attributes deemed dependent on each other? That is, can the final classifier be further improved by including such new constructions?
2.2 Limitations

This thesis is concerned with finding a set of features that are relevant for class prediction. However no focus have been put into finding the minimal subset. If two attributes are perfectly correlated, one of them is clearly redundant. Still both will be considered important by the proposed method. Also, the proposed method will not be suitable for very high dimensional data where preprocessing is absolutely necessary in order to find any kind of decent classifier. If the learning algorithm does not build reasonable classifiers when all attributes are given as input, important attributes will not be detectable either. This will be further discussed in Section 7.
Chapter 3

Background

As the main task of this thesis is a feature selection problem, some background to the general problem of feature selection will first be given in Section 3.1. Specific feature selection methods, that later will be used for comparison, will also be briefly explained.

Section 3.2 will focus on getting the reader familiar with the target classifier, i.e. (an ensemble of) genetic programs. The general algorithm will be described as well as the problem of bloat and previous attempts to use Genetic Programming for feature selection. The specific classifier used in this thesis, MVGPC, will also be explained. On top of this, a brief introduction to Support Vector Machines is included, as they serve as the secondary learning algorithm in this thesis.

Finally, since the proposed method is based on permutations of the test data similar to an importance measure used in Random Forests, a short description of Random Forests and its usage of permutations has also been included (Section 3.4).

3.1 Feature selection

As the dimensionality of the data increases, the performance of most learning algorithms decreases and there is more to be gained by being able to discard irrelevant features.

There are mainly three categories of feature selection algorithms; Filter, Wrapper and Embedded/Hybrid methods. In the following sections, the first two of these will be described in more detail. Section 3.1.3 and Section 3.1.4
will then focus on two specific feature selection methods, RELIEF and CFS, later used for evaluation purposes.

3.1.1 Filter methods

Filters are preprocessing methods that often produce a ranking of the features according to some ranking criterion. This criterion can be based on correlations (e.g., the Pearson correlation coefficient), mutual information or other information theoretic measurements. A ranking can also be produced by treating each attribute as a single-variable classifier and measuring its discriminative properties in terms of error rate or various other metrics based on false positive/negative rates. Feature selection methods that evaluate each attribute individually are computationally inexpensive, but also have their limitations as they are unable to deal with interacting features [4].

Filters also include methods that use one learning algorithm for preprocessing (feature subset selection) and another for the actual classification task. Decision tree algorithms such as C4.5 have been commonly used for this purpose as they use an information theoretic splitting criteria, and thus reveal more about the importance of the attributes than the more black box type of classifiers. Mark A. Hall mentions a few earlier examples of this usage in his dissertation, for instance two cases where C4.5 is used to select important features for a k-nearest neighbour classifier and a Bayesian network classifier respectively [5]. He also introduces CFS, a correlation-based feature selection method that since then has been widely used. His central hypothesis is however that important features are highly correlated with their class-belonging, but uncorrelated with each other, which leaves room for improvement as this method does not take feature interactions into account.

3.1.2 Wrapper methods

The second broad class of feature selection algorithms is the wrappers. Wrappers evaluate feature subsets using the chosen learning algorithm itself. However, the feature selection is still separated from the task of training the final classifier. During feature selection, the training data with a limited set of features will be fed to the learning algorithm, which will build a classifier and evaluate the suitability of that feature set based on how it fares when faced with unseen test data. This can be done e.g., through leave-one-out cross validation [9].
As the evaluation of features in wrappers is tightly coupled with the learning algorithm in question, they often result in more accurate classifiers than filter approaches. However this does not come without a cost - the computational burden of wrappers is their main drawback. It is generally infeasible to evaluate all possible subsets of features, so instead subsets are often iteratively either expanded or reduced until the presumably optimal set of features has been found. Starting with an empty set of features and iteratively adding features is called forward selection, whereas stepwise exclusion of features from an initial full set of features is called backward elimination. The selection of which features to include/exclude next can be done with heuristics like a hill-climbing or best-first search. A comparison of both these approaches can be found by Ron Kohavi and George H. John [9].

3.1.3 RELIEF

RELIEF is a feature selection algorithm that produces a feature ranking where each feature is considered in the context of all other features, without actually using a wrapper approach. Feature interactions are implicitly dealt with by iteratively updating the feature weights based on the similarities between randomly drawn samples and their neighbouring samples. When a sample, \( X \in C_i \), is drawn, the closest sample from the same class, \( Y \in C_i \), and the closest sample from the other class, \( Z \in C_j \), is found, normally referred to as the near-hit and near-miss, respectively. The intuition is that attributes that are discriminative between the classes should have similar values for samples belonging to the same class, but differ for samples belonging to different classes. Since the samples being compared are selected based on the distance between them, the update of each feature weight will implicitly consider the values of all other features. The formal weight update formula is shown in Equation 3.1.

\[
W_i = W_{i-1} - (x_i - y_i)^2 + (x_i - z_i)^2 \tag{3.1}
\]

RELIEF was introduced in 1992 by Kira and Rendell [8]. In this thesis the extended version RELIEFF [10] will be used during the evaluation of the proposed method.
3.1.4 Correlation based Feature Selection

CFS (Correlation based Feature Selection) is an other well-known feature selection algorithm that will be used for evaluation purposes. It is based on the assumption that good feature sets contain features that are highly correlated with the class, yet uncorrelated with each other [5]. CFS thus attempts to solve the optimization problem of maximizing the chosen features’ correlation with the class, while minimizing their internal feature-feature correlations.

Even though the CFS measure has been shown to work well for a number of real data sets, it still has some obvious shortcomings when the initial independence assumption is invalid. In the MONK data set (Section 5.2.1), for instance, the predictability of the features is strongly dependent on feature interactions and thus this base premise does not hold. It was also shown by Mark A. Hall that CFS failed to select the relevant features for the MONK data [5].

A couple of extensions of CFS that allow it to deal with data containing feature interactions was proposed along with its introduction. The straightforward solution of adding all pairwise combinations of features to the initial feature set proved superior to a version that incorporated the RELIEF feature weights into the CFS measure, even though such an solution can only detect pairwise interactions and further extensions, combining even more features, becomes infeasible for high-dimensional data sets [5].

3.2 Genetic Programming

Genetic Programming was pioneered by John Koza in the early nineties [11], and is an extension of the more general Genetic Algorithms (GA), where the solution to a problem is found by maintaining an evolving population of individuals. The fitness of each individual can be evaluated by a problem specific fitness function and the solution space is traversed following the Darwinian principle of the survival of the fittest, i.e. fit individuals have a higher probability of survival and reproduction of similar offspring. More specific details will be given in the next section.

As this thesis uses an ensemble of genetic programs, Section 3.2.3 will describe the specific learning algorithm used - MVGPC. Section 3.2.2 will then touch upon the subject of bloat, a frequent problem in the evolution of genetic programs. Finally, some previous attempts to use GP for feature
selection will be summarized in Section 3.2.4.

### 3.2.1 Introduction

As previously stated, Genetic Programming is a subset of the broader class Genetic Algorithms, and more specifically evolves computer programs, most commonly represented in a tree structure. Each individual will be built from a set of operator nodes and terminal nodes. Figure 3.1 shows an example of a GP tree, where the operators used are common mathematical operators (multiplication, division etc.). Other sets, such as logical operators or actual program code statements such as if/else can also be used.

![Figure 3.1: Genetic Program. Terminal nodes are shown in red and operator nodes in black.](image)

The suitability of an individual is measured by its fitness. If, for instance, the task is a regression problem, the mean square error might be used as the fitness function, whereas if the problem is a classification problem, the fitness could be measured as a function of the true and false positives/negatives.

The actual search process starts with randomly creating an initial population of individuals. After that, there are a number of slightly different ways to evolve the population, but common for all these is that the individuals will be evaluated in some way, and be more or less eligible to reproduce based on how well they perform.

If tournament selection is used as selection strategy the evaluation will be done by having programs compete against each other, awarding the winner with a higher chance of reproduction. An other variant is roulette wheel
selection, where the probability of selecting an individual for reproduction is proportional to its fitness. If an individual $i$ has fitness $f_i$, this probability becomes $p_i = \frac{f_i}{\sum_{j=1}^{N} f_j}$, i.e. this individual’s fitness, seen in relation to the total fitness of the population. If so called elitism is used, the fittest individual(s) will always be copied to the next generation, before the ordinary reproduction operations take place [7].

The reproduction itself can be affected by two genetic operations - crossover and mutation, both inspired by their biological versions.

The crossover operation is illustrated in Figure 3.2 and 3.3. Two individuals are first selected (Figure 3.2), using the preferred selection strategy.
Then a crossover node is randomly chosen for each of the individuals, and two children created by swapping the subtrees between the individuals (Figure 3.3).

![Figure 3.4: Mutation](image)

The other operation that causes new solution to evolve is the *mutation*. Each node in the offspring will, with a small probability, be swapped by a random node. An example can be seen in Figure 3.4.

In this way, new generations are created, and as they are bred from the fittest individuals from the previous generation, the hope is that fitter and fitter individuals are evolved, until a certain stopping criteria is met. This can be after a certain number of generations, or when a certain fitness has been achieved. A problem that occurs when populations are allowed to evolve for too long will be discussed in the next section.

### 3.2.2 Bloat

Bloat refers to excessive growth of genetic programs, leading to the existence of redundant code. Even though there are still no exact answers to why bloat occurs, there are many theories and they all in some way link bloat to the search for fitter solutions, making it seem almost like a necessary evil [16]. One popular explanation ("fitness causes bloat") is based on the fact that there are more long programs representing a specific solution than shorter ones and as it becomes more difficult to find fitter solutions, children having similar fitness as their parents are given preference. Crossover is usually quite destructive, and thus code segments that do little to change the actual functionality of the program are rewarded.
Bloat makes the programs less interpretable and slows down the search process as the number of possible combinations of subtrees explodes. Also, it will infest the rules with redundant attributes, making it more difficult to determine the importance of attributes merely by inspecting their occurrence in the evolved rules (a common feature selection technique in GP, see Section 3.2.4). Figure 3.5 shows an example of two genetic programs that have the same logical meaning, but where one of them contain completely redundant code. Of course, redundancies are not always this obvious and can also be pierced in between important dependencies, making the truly important relationships difficult to find.

### 3.2.3 Majority Voting Ensembles - MVGPC

MVGPC is an ensemble method that uses an ensemble of genetic programs for classification. Topon Kumar Paul and Hotoshi Iba used MVGPC to classify data samples consisting of a set of gene expression levels as belonging to one of a set of different types of cancer [14]. The fitness function for a GP rule used the Matthews correlation coefficient (MCC), which is a function of the true/false positives and the true/false negatives. The MCC function is shown in Equation 3.2, where $N_{tp}$ refers to the number of true positives, $N_{tn}$ the number of true negatives, and $N_{fp}$ and $N_{fn}$ the number of false positives and
false negatives, respectively.

\[
MCC = \frac{N_{tp}N_{tn} - N_{fp}N_{fn}}{\sqrt{(N_{tn} + N_{fn})(N_{tn} + N_{fp})(N_{fp} + N_{fn})(N_{tp} + N_{fn})}}
\] (3.2)

The classification into multiple classes is done by binary classifications in a "one-versus-rest" approach. In a problem with \(c\) classes \(vc\) GP rules are evolved, where \(v\) is the size of an ensemble. Each set of \(v\) GP rules makes a binary classification for a sample as either belonging to the class \(C_i\) or not, and when presented a new data point each rule votes for a classification. The data point will then be classified as the class that is given the most number of votes. Unlike most other ensemble methods MVGPC evolves \(vc\) rules where each rule aims to correctly classify all the data samples, and all the GP rules are given equal weighting in the final classifier.

### 3.2.4 Feature selection with Genetic Programming

Even though the number of attempts to use Genetic Programs for feature selection seem quite limited, there are a few examples to be found. Durga Prasag Muni et al. has proposed an embedded method that evolves GP classifiers while simultaneously performing feature selection [13]. Each individual in the population is only allowed to use its own subset of features and is evaluated using a multi-objective fitness function that takes both classification accuracy and the number of features used into consideration, rewarding individuals using fewer features. The algorithm was successfully tested on some well-known data sets and showed improved performance when compared to a selection of both filter and wrapper approaches.

Topon Kumar Paul and Hitoshi Iba used MVGPC (Section 3.2.3) to classify different types of cancer based on gene expression data. As a side-effect they noticed that the most commonly selected genes also had well known roles in the development of cancer.

They concluded that valuable information regarding the importance of genes can be gained by looking at the frequencies in which they occur in the evolved rules. However, they also recognize the flaws of this method, as the individual rules with the best test accuracy were mostly made up from non-frequent genes. [14] This simple method may thus only accurately find genes that are independently discriminative as more complex solutions are unlikely to evolve as frequently.
In a similar study, also aimed towards cancer classification, Jianjun Yu et al. found the same genes being repetitively selected during evolution of the GP rules [20]. Similar to the results from the work by Paul and Iba, these genes were previously known to be associated with cancer. To further test the discriminative properties of the most frequently occurring genes, they trained a diagonal linear discriminant analysis (DLDA) and a k-nearest neighbour (KNN) classifier using only this most frequently used subset of genes. The final classifiers were able to show promising results on the validation set and thus the authors concluded that the most common genes in the GP rules were indeed discriminative of the data.

3.3 Support Vector Machines

Support Vector Machines (SVMs) works by mapping the input data to a high-dimensional space, where it is more likely to be linearly separable, and then finding the optimal separating hyperplane by maximizing the margin between the two classes [3]. The intuition is that the larger the margin, the better the chances of good generalization.

![Figure 3.6: Optimal SVM separation.](image)

Figure 3.6 shows an example of such a separation. The blue and green areas have equal width, and thus the optimal separating line is the one separating the two coloured areas. Any line parallel to this one, but shifted in any of the directions, will make it more likely for misclassification of the class which now is closer to the separation line. Also, this line has been chosen such that
any non-parallel separating line also reduces the margin between the closest samples from different classes.

The basic optimization problem solved by a SVM is the maximization of this margin, i.e. the distance between the two closest sample points on either side of the separating hyperplane. Let’s assume that the equation for this hyperplane is:

$$\vec{w}^T \vec{x} + b = 0.$$  \hfill (3.3)

For the plane to actually separate the two classes, the following constraints must be met;

$$\forall i \in C_1 \quad \vec{w}^T \vec{x}_i \geq m,$$  \hfill (3.4)

$$\forall i \in C_2 \quad \vec{w}^T \vec{x}_i < m.$$  \hfill (3.5)

Without loss of generalization, $m$ from Eq. 3.4 and Eq. 3.5 can be set to 1, as $\vec{w}$ always can be scaled to satisfy this condition.

For two points, $p$ and $q$ that lay exactly on the two borders of the margin, the equalities $\vec{w}^T \vec{p} = 1$ and $\vec{w}^T \vec{q} = -1$ hold. Since $\vec{w}$ is the direction orthogonal to the separating line, the distance between $\vec{p}$ and $\vec{q}$ along $\vec{w}$ becomes the width of the margin. The margin width, $2d$ can be expressed as,

$$2d = \frac{\vec{w}^T (\vec{p} - \vec{q})}{||\vec{w}||} = \frac{(1 - b) - (-1 - b)}{||\vec{w}||} = \frac{2}{||\vec{w}||}.$$  \hfill (3.6)

Maximizing the margin, Eq. 3.6, thus equals minimizing $\vec{w}^T \vec{w}$ subject to the constraints 3.4, 3.5 with $m = 1$.

Many problems are however not perfectly linearly separable. In such a case one can introduce slack variables that allows samples points to lie within the margin, or even on the wrong side of the separation line. A sample point which are allowed to break the initial constraints due to a positive slack variable, will however induce a penalty to the objective function that is to be minimized.

The introduction of slack variables may solve a problem that is linearly separable in reality, but seemingly unseparable due to noise. However, for many problems, trying to find a linear separation is not the best alternative. SVMs also provide a solution for more complex separations by the introduction of kernel functions, that allows the sample points to be scattered into an even higher dimensional space, by the mapping provided by the kernel function. The so called kernel trick provides a neat way of implicitly making
this mapping and solving the resulting optimization problem without having to perform the actual computation of the coordinates in the new feature space. The kernel trick will however not be described here, since only the linear version of the SVM has been used in this thesis.

3.4 Measuring importance by permutations in Random Forests

Random Forests is an ensemble method developed by Leo Breiman and Adele Cutler, where the members of the ensemble consist of classification (or regression) trees [2]. Each tree is built by taking a bootstrap sample of the training data and fitting a decision tree to it. Also, for each split only a random subset of the available attributes are considered, which further helps to reduce the correlation between the different trees.

Random Forests have two different ways of assigning importance values to the attributes that it is built from. First of all, the splitting criteria (e.g. Gini impurity) provides information about how much was gained by splitting on a certain variable. For each variable, its importance measure can then be estimated by considering all splits on this variable for all trees in the forest.

The second importance measure uses a feature in Random Forests called out-of-bag (OOB) samples. If the forest is constructed using only a bootstrap of \( \frac{2N}{3} \) samples of the available data, each sample will on average have \( \frac{N}{3} \) trees where it has not been used for training, which means that it can be used as test data for those trees [6].

An OOB error of the forest can thus be calculated by classifying the OOB samples using their corresponding trees. The importance of a feature can be estimated by randomly permuting this feature between the samples and then calculate the new OOB error on the permuted data. A large difference between original and permuted OOB error indicates that the feature plays an important role in the classification, whereas if there is little change in error, the feature is likely to be irrelevant.
Chapter 4

Model

In the following sections the basic idea behind this thesis will be explained, along with pseudo-code for the algorithms introduced.

4.1 Importance measure by frequency count

Earlier approaches using Genetic Programming for feature selection have often measured attribute importance by considering the frequencies in which attributes appear in the genetic programming rules [20][14]. The rational is that an attribute that is chosen often is more likely to be truly discriminative. In this work, the frequency counts will mainly be used for comparison.

4.2 Importance measure by attribute permutation

To measure the importance of an attribute, first ensembles of GP rules are evolved using the training data. The importance of each attribute is then evaluated using Algorithm 3. The overall importance depends on the ratio of the original test accuracy and the between-class permutation accuracy, i.e. the accuracy after a given attribute has been permuted between different classes. The dependency classification further depends on the within-class permutation accuracy, where the attribute instead has been permuted within the same class.
Definitions

**Within-class permutation:** All samples will be given a new value for a specified attribute, taken from an other sample with the *same* class-belonging. See Algorithm 1 and Figure 4.2 (left).

**Between-class permutation:** All samples will be given a new value for a specified attribute, taken from an other sample with a *different* class-belonging. See Algorithm 2 Figure 4.2 (right).

---

**Algorithm 1** Within-class permutation

```
1: procedure WITHINCLASSPERMUTATION(a_i)
2:   for each class k do
3:     for each sample s_i ∈ C_k do
4:       Randomly choose a sample s_j ≠ s_i where s_j ∈ C_k
5:       s_{i,perm}(:) ← s_i(:);
6:       s_{i,perm}(a_i) ← s_j(a_i);
7:   end for
8: end for
9: end procedure
```

**Algorithm 2** Between-class permutation

```
1: procedure BETWEENCLASSPERMUTATION(a_i)
2:   for each class k do
3:     for each sample s_i ∈ C_k do
4:       Randomly choose a sample s_j ≠ s_i where s_j ⊈ C_k
5:       s_{i,perm}(:) ← s_i(:);
6:       s_{i,perm}(a_i) ← s_j(a_i);
7:     end for
8: end for
9: end procedure
```

---

1In order to achieve a real permutation the same sample should never be used twice when assigning new values to the permuted set of samples. Algorithm 1 and 2 has thus been slightly simplified.
Figure 4.1: Left: Three samples from $C_A$. Right: Three samples from $C_B$.

Figure 4.2: Left: Within-class permutation. Attribute $a_2$ is swapped between samples of the same class. Right: Between-class permutation. Attribute $a_2$ is swapped between samples from different classes.
Algorithm 3 Importance measure of an attribute

1: procedure IMPORTANCEMEASURE($a_i$, testData)
2: \hfill \textit{testScore} = average of the scores of all GP trees $\ni a_i$
3: \hfill Initialize all other variables to zero
4: repeat
5: \hfill data $\leftarrow$ withInClassPermutation($a_i$, testData);
6: \hfill for each GP tree $t$ that contains $a_i$ do
7: \hfill \textit{wicScore} $\leftarrow$ wicScore + score(data, $t$);
8: \hspace{2em} end for
9: \hfill data $\leftarrow$ betweenClassPermutation($a_i$, testData);
10: \hfill for each GP tree $t$ that contains $a_i$ do
11: \hfill \textit{becScore} $\leftarrow$ becScore + score(data, $t$);
12: \hspace{2em} end for
13: \hfill \textit{wicScoreTot} $\leftarrow$ \textit{wicScoreTot} + \textit{wicScore}/$N$;
14: \hfill \textit{becScoreTot} $\leftarrow$ \textit{becScoreTot} + \textit{becScore}/$N$;
15: until $R$ rounds have passed
16: \hfill \textit{wicScoreTotal} $\leftarrow$ \textit{wicScoreTotal}/$R$;
17: \hfill \textit{becScoreTotal} $\leftarrow$ \textit{becScoreTotal}/$R$;
18: \hfill \textit{importanceScore} $\leftarrow$ \textit{testScore}/\textit{becScoreTotal}
19: \hfill if \textit{becScore} < \textit{testScore} and \textit{wicScore} < \textit{testScore} then
20: \hfill \textit{dependencyScore} $\leftarrow$ \frac{\textit{testScore} - \textit{becScoreTotal}}{\textit{wicScoreTotal} - \textit{becScoreTotal}}
21: \hfill else
22: \hfill \textit{dependencyScore} $\leftarrow$ 1
23: \hspace{2em} end if
24: \hfill return [$\textit{importanceScore}$, $\textit{dependencyScore}$]
25: end procedure

The intuition is that if an attribute $a_x$ causes a decrease in test accuracy for a sample, $s_i \in C_A$, when its value is swapped with that of $a_x$ in an other sample, belonging to a different class, $s_j \in C_B$, $a_x$ probably provides important information in making the class prediction. The example shown in Figure 4.2 (right) is likely to cause such a decrease, since its value differ noticeably between samples of different classes, but whether or not such a decrease is present in reality also depends on the trained classifiers and whether they use this observation in a sensible way.

Whether or not a seemingly important attribute $a_x$ is dependent on other attributes in order to become predictive can however not be determined based
on the between-class permutation alone. To make a dependency classification the within-class permutation accuracy must also be considered. If \( a_x \) is given a new value from a sample belonging to the same class, \( s_k \in C_A \), and the test accuracy remains fairly unchanged, it is less likely that it is dependent on other attributes in order to become predictive, since such dependencies would have been broken by the permutation and most likely caused a decrease in prediction accuracy. If the within-class permutation on the other hand results in a lower test accuracy, the attribute is likely to be *dependently predictive.*

**Definitions: Predictabilities of an attribute**

*Independent:* The attribute is individually predictive of the class label. Considering more attributes might give better accuracy, but this is mostly due to the effect of the *wisdom of the crowds* and not because of necessary joint interactions.

*Dependent:* The attribute obtains most of its predictive powers by exploiting interactions with other attributes. It might in itself be totally uncorrelated to the class.

*Irrelevant:* The attribute is not helpful in making the class prediction.

If neither of the permutations produces noticeably lower test accuracies the attribute is classified as irrelevant. Figure 4.3 shows expected permutation responses for attributes with different predictive powers.
Figure 4.3: Dependency classification. If none of the permutations decrease the test accuracy noticeably the attribute is classified as irrelevant. If mainly the between-class permutation causes a decrease it is classified as independently predictive, but if both permutations result in decreased predictability the attribute is classified as dependently predictive.

Using the within-class and between-class permutations, the importance of an attribute can thus be classified into one of three different classes; Independently predictive, dependently predictive or irrelevant.

This classification can be done by either looking at the ensemble of trees and how its trees, on average, respond to the permutations, or by examining the responses for each of the trees separately. The problem with looking at the average response from all trees is that genetic programs are very vulnerable to bloat (Section 3.2.2), and thus many attributes may exist as part of trees even if they do not play an active role in how that tree makes the classification. Dependent attributes might thus exist in an "irrelevant way" in a large number of trees, while their true dependencies might only be exploited in a small fraction of the trees. Even though an importance dependency has been found, it might thus be averaged out, unless it has been found by a large enough number of trees. A solution to this problem is to consider each tree individually, and to have each tree make an individual classification for each of the attributes contained in it.

Both approaches will be considered in this thesis. The average approach is described in Section 4.2.1 and the voting approach in Section 4.2.2.
4.2.1 Average classification

Algorithm 3 describes the first approach that uses the average response from all trees containing a specific attribute in order to determine the importance of this attribute. Two importance scores are returned, the first one reflecting the general importance and the second one the level of dependency. A high importance score means that the attribute is important and a high dependency score translates to a high probability of the attribute being dependent on other attributes in order to become predictive.

The tricky part, especially when evaluating dependencies, is to determine good thresholds that separates the important/dependent attributes from the irrelevant/independent ones. In this thesis the ”averaging approach” has only been used to determine the set of important attributes. All attempts to detect dependencies have so far used the voting approach described in the next section.

4.2.2 Voting classification

In the second approach, each individual tree makes a dependency classification for each attribute that is contained within it. The options are, as before, either irrelevant, independently predictive or dependently predictive. Algorithm 4 describes the voting procedure. Each attribute will obtain votes from all trees in which it is contained and the final classification will be decided by the weighted majority class. As dependencies might be difficult to find, a dependent vote is weighted higher than an irrelevant vote. Disregarding the number of irrelevant votes, the attribute is classified as dependent if there are dependent votes above a certain threshold, $\Theta_{DMIN}$, and more dependent votes than independent.
Algorithm 4 Individual tree voting

1: procedure VotingDependencyClassification($a_i$)
2:     irrelevantVotes = 0; indepVotes = 0; depVotes = 0;
3:     for each GP tree $t$ that contains $a_i$ do
4:         wicScore ← 0;
5:         becScore ← 0;
6:         testScore ← computeTestScore($t$);
7:             repeat
8:                 testData ← withInClassPermutation($a_i$);
9:                 wicScore ← wicScore + computeTestScore($t$)
10:                testData ← betweenInClassPermutation($a_i$);
11:                becScore ← becScore + computeTestScore($t$)
12:             until $R$ rounds have passed
13:             if $\frac{becScore}{testAccuracy} > \Theta_{IRRELEVANT}$ then
14:                 irrelevantVotes ← irrelevantVotes + 1
15:             else if $\frac{testScore - becScoreTotal}{wicScoreTotal - becScoreTotal} > \Theta_{DEPENDENT}$ then
16:                 depVotes ← depVotes + 1
17:             else indepVotes ← indepVotes + 1
18:         end if
19:     end for
20:     if depVotes $\geq \Theta_{DMIN}$ and depVotes $\geq$ indepVotes then
21:         return "DEPENDENT";
22:     else if irrelevantVotes $\geq \Lambda$(depVotes + indepVotes) then
23:         return "IRRELEVANT";
24:     else
25:         return "INDEPENDENT";
26:     end if
27: end procedure

4.3 Attribute construction

When an attribute has been chosen as a candidate for being dependently predictive, the possible dependencies need to be determined. The intuition is that when all dependencies have been found, the dependent group of attributes should behave in the same way as an independent attribute when permuted together, that is their within-class permutation accuracy should
be close to the original test accuracy, while the between-class permutation accuracy should be considerably lower.

Algorithm 1 and 2 can easily be extended to allow for multiple attributes to be permuted together, by exchanging the input parameter $a_i$ to a vector $\vec{a}$ with attribute indices and letting the assignment $s_i(\vec{a}) = s_j(\vec{a})$ mean that all the attributes with indices in $\vec{a}$ are assigned values from an other sample, $s_j$.

Figure 4.4 shows a simplified scheme for classifying attributes and finding dependencies. When all attributes have been classified as one of either "irrelevant", "independent" or "dependent", those who are suspected to be dependent will be used to create a set of combined attributes. All pairwise combinations of dependent attributes will be considered for the attribute construction. If, for instance, attribute $a_1$, $a_5$ and $a_7$ are classified as dependent, the combinations $(a_1, a_5)$, $(a_1, a_7)$, $(a_5, a_7)$ will be further investigated by pairwise permutations. The shown scheme is a simplified version since, even though the criteria for classifying pairs of two attributes is somewhat different from classifying single attributes, the same decision box is used in the figure. The latter classification must consider the improvement in accuracy compared to the single-permutation scores. If this improvement is large enough, the attribute combination will be classified as independent, else as irrelevant.

When pairs/groups of attributes that are likely to depend on each other have been found, this information should be exploited somehow when training the final classifier. One advantage of knowing which attributes are likely to be dependent is that one can avoid to break dependency groups when choosing the most important attributes to include, i.e. if one attribute in a dependent group is included, then the rest should be as well.

A more direct exploitation is to construct new attributes that express the possible dependencies. If two attributes, $a_1$ and $a_2$, are likely to be dependent (since the combination $(a_1,a_2)$ was independently classified), the new attributes $a_1 * a_2$, $a_1 + a_2$, $a_1 - a_2$ and $a_1/a_2$ may be added to the set of attributes as a way of providing the learning algorithm with some heuristics.

In the end, the attributes used as input to the learning algorithm will be those classified as singly independent, singly dependent and the pairwise attributes classified as independent, recombined with different operators.
Figure 4.4: Scheme for classifying attributes and finding dependent pairs.
Chapter 5

Method

5.1 Implementation

The following list is a clarification of the different programming languages, packages and parameters used for this thesis.

- MVGPC was implemented in Java using the JGAP package [12]. JGAP (Java Genetic Algorithms Package) is an open source library for Genetic Algorithms and Genetic Programming. The specific parameters used during evolution can be found in Appendix A.

- The SVM was trained using the \texttt{svmtrain()} function from the Statistics Toolbox in MATLAB [17].

- The RELIEFF implementation used is also part of the Statistics Toolbox in MATLAB [17]. The parameter $k$ (number of neighbours) was set to 5 for all cases.

- The relevant features according to CFS was found using the \texttt{fsCFS()} method in the MATLAB version of Weka [19].

- The within-class and between-class permutation tests were implemented in both Java and MATLAB. When calculating the permutation scores, twenty different within-class and between-class permutations were made in order to reduce the variance. For the same reason, the original data was re-sampled five times into different training and test sets, after which a classifier was trained and the features evaluated.
5.2 Data

In evaluating the proposed method a mixture of artificial and real data sets were used. The artificial sets were used to confirm the expected behaviours, while the real sets were used as benchmarks.

5.2.1 UCI data

The UCI Machine Learning Repository is a large and constantly expanding collection of data sets widely used by machine learning researchers [1]. Six UCI data sets will be used in this thesis; the three artificial MONK sets along with the Wisconsin Diagnostic Breast Cancer (WDBC), Wine and Vehicle data sets that were also used in an other research combining Genetic Programming and feature selection [13].

MONK

The MONK problems are an artificial set of problems, containing feature interactions, often used as a benchmark for comparing different machine learning algorithms [18]. The problems describe robots with six attributes;

- Head-shape ($A_1$) $\in \{\text{round (1), square (2), octagon (3)}\}$
- Body-shape ($A_2$) $\in \{\text{round (1), square (2), octagon (3)}\}$
- Is-smiling ($A_3$) $\in \{\text{yes (1), no (2)}\}$
- Holding ($A_4$) $\in \{\text{sword (1), balloon (2), flag (3)}\}$
- Jacket-color ($A_5$) $\in \{\text{red (1), yellow (2), green (3), blue (4)}\}$
- Has-tie ($A_6$) $\in \{\text{yes (1), no (2)}\}$

There are three target concepts, each with 432 training/test instances;

- MONK1: ($A_1 = A_2$ or $A_5 = 1$)
- MONK2: (EXACTLY TWO of $A_1=1$, $A_2=1$, $A_3=1$, $A_4=1$, $A_5=1$, $A_6=1$)
- MONK3: ($A_5 = 3$ and $A_4 = 1$) or ($A_5 \neq 4$ and $A_2 \neq 3$) (5% added noise)

The high degree of feature interactions make many feature selection algorithms fail to find the relevant attributes for the MONK data. CFS was tested on the MONK problems in Mark A. Hall’s dissertation [5], but only
one attribute (out of three) in the MONK1 problem was selected as important, only an average of half of the attributes in the MONK2 problem and just two (of three) important attributes from the MONK3 problem were correctly identified. This is however not surprising, since CFS is based on an assumption about feature independency.

**Wisconsin Diagnostic Breast Cancer (WDBC)**

The WDBC data set contains 569 instances, where patients have been diagnosed with either malign or benign breast cancer. Each sample is comprised of 30 features, derived from images of cell nuclei.

**Wine**

The Wine data set contains 178 instances and 13 attributes derived from chemical analysis of wines from three different cultivars. The task is to classify the cultivar based on the chemical information.

**Vehicle**

The Vehicle data set is listed as an UCI data set eg. by Muni et. al [13], but the 18 attribute real valued set used in this thesis (and by Muni et. al) does not seem to be presently available at the UCI web page. Instead the data was obtained from https://www.sgi.com/tech/mlc/db/.

The Vehicle set consists of four classes (OPEL, SAAB, BUS, VAN), which should be distinguished based on the features extracted from silhouette images of the vehicles. The different vehicles were chosen such that two of them were more similar (OPEL and SAAB) and should thus be more difficult to separate. There are a total of 846 instances contained in the set.

5.2.2 Cytometry data

Cytometry is a group of methods that allow efficient measurement of a large number of cell parameters. The cytometry data used in this thesis has 29 attributes. Similar data have been used by Rodrigo T. Peres, Claus Aranha and Carlos E. Pedreira in a recent paper [15], but the exact data used in this thesis, provided by Claus Aranha, has still not been officially published. It has however been investigated by the same methods used in the paper cited above and a tentative variable ranking has been obtained using the
attributes with the highest weights according to the proposed Differential Evolution-based method. The results have been used for comparison, but it must be noted that the method proposed by Aranha et al. was not aimed at feature selection, and hence might not be very suitable for that purpose. A total of nine different target classes were provided and the aim has been to separate them two and two, where the pairs have been chosen based on the results from the previous study, such that both some easy and some seemingly difficult separations have been included.

### 5.2.3 Constructed data

In order to test the proposed method on a dataset designed to show its strengths, while at the same time posing a problem for traditional feature selection algorithms, the artificial data set described in Table 5.1 was constructed. No noise was added leaving the classes perfectly separable by taking advantage of either of the relationship \( a_2 = c_i * a_1 \) or \( a_{12} = d_i * a_4 * a_{11} \).

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Class A</th>
<th>Class B</th>
</tr>
</thead>
<tbody>
<tr>
<td>( a_1 )</td>
<td>( \mathcal{N}(2, 2) )</td>
<td>( \mathcal{N}(2, 2) )</td>
</tr>
<tr>
<td>( a_2 )</td>
<td>1.3 * ( a_4 )</td>
<td>0.8 * ( a_1 )</td>
</tr>
<tr>
<td>( a_3..a_5 )</td>
<td>( \mathcal{N}(2, 2) )</td>
<td>( \mathcal{N}(2, 2) )</td>
</tr>
<tr>
<td>( a_6 )</td>
<td>( \mathcal{N}(3, 0.5) )</td>
<td>( \mathcal{N}(4.5, 0.5) )</td>
</tr>
<tr>
<td>( a_7..a_{11} )</td>
<td>( \mathcal{N}(1, 1) )</td>
<td>( \mathcal{N}(1, 1) )</td>
</tr>
<tr>
<td>( a_{12} )</td>
<td>0.8 * ( a_4 * a_{11} )</td>
<td>( a_4 * a_{11} )</td>
</tr>
<tr>
<td>( a_{13..a_{19}} )</td>
<td>( \mathcal{U}(0, 1) )</td>
<td>( \mathcal{U}(0, 1) )</td>
</tr>
</tbody>
</table>

Table 5.1: Constructed data

### 5.3 Evaluation

The proposed method was evaluated on a number of different data sets by first training an ensemble of genetic programs using all attributes, then selecting the most important attributes based on the permutation scores and finally re-training the classifier using only the most informative set of attributes.
The performance of the final classifier was compared to that of the initial classifier and other classifiers trained using the top attributes as selected by a number of competing attribute selection methods. The permutation tests were also applied to a linear SVM classifier, using the same approach as described above.

The main part of the evaluation was done by only separating important attributes from irrelevant ones. As the proposed method also have the ability to separate independently predictive attributes from dependently predictive ones, an attempt to take advantage of this by adding new attributes, consisting of promising combinations of dependent attributes, was also made.
Chapter 6

Results

In the following sections the performance of the final classifiers, trained with and without attribute selection, will be presented. All results shown refer to the test accuracies.

When the within-class and between-class permutations have been used to select the top attributes, the notation \textit{wcbcp} (within-class-between-class-permutation) will be used as a subscript. That is, $GP_{\text{wcbcp}}$ means that permutation tests have been applied to GPs and $ SVM_{\text{wcbcp}}$ that they have been applied to SVM’s.

A summary of the abbreviations used can be found in Table 6.

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\textit{wcbcp}</td>
<td>Within-class-between-class-permutation</td>
</tr>
<tr>
<td>$GP_{\text{wcbcp}}$</td>
<td>Permutation tests applied to MVGPC</td>
</tr>
<tr>
<td>$SVM_{\text{wcbcp}}$</td>
<td>Permutation tests applied to SVMs</td>
</tr>
<tr>
<td>$GP_{\text{mtfs}}$</td>
<td>Multitree genetic programming based FS, Muni et al. [13]</td>
</tr>
<tr>
<td>$c_i$</td>
<td>Class/cluster i</td>
</tr>
<tr>
<td>DE</td>
<td>Differential Evolution</td>
</tr>
<tr>
<td>IRR</td>
<td>Irrelevant</td>
</tr>
<tr>
<td>DP</td>
<td>Dependent</td>
</tr>
<tr>
<td>IDP</td>
<td>Independent</td>
</tr>
<tr>
<td>(v)</td>
<td>Classification by the voting ensemble</td>
</tr>
</tbody>
</table>

Table 6.1: Abbreviations

When an ensemble of genetic programs (MVGPC) is used as the classifier, 42 GP trees for each class is constructed. The operator set \{-, +, /, *, \text{SQRT}\} is always used.
In all permutation graphs shown, the same color coding as in Figure 4.3 (Section 4) is used, i.e. blue refers to the original test accuracy, green to the within-class permutation and red to the between-class permutation.

6.1 Synthetic data

When applying the permutation tests to synthetic data with known dependencies the correct dependencies were most of the time correctly identified. In both cases where some dependent attributes were left unidentified, the other dependencies, which were found, had an overall stronger predictability. It is therefore likely that the weaker predictors had no significant roles in the evolved rules and could therefore not be detected. This will be further discussed in Section 7, but first the actual results will be presented. Section 6.1.1 reports the results for the MONK data and Section 6.1.2 for the constructed data described in Section 5.2.3.

6.1.1 MONK data

It should be noted that the operators \{\,+,-,/,*,SQRT\} are still used for the MONK data set, even though the set \{AND,OR,=\} would have made it easier to find accurate rules.

MONK1

In the MONK1 dataset (\(A_1 = A_2 \text{ or } A_5 = 1\)) attribute \(A_1\) and \(A_2\) are strongly dependent on each other and attribute \(A_5\) is more independently predictive. As seen in Figure 6.1 attribute \(A_1\) and \(A_2\) show a clearly dependent response, as expected. In this case it is easy to assume that they are dependent on each other, since they are the only dependently classified attributes, even though \(A_5\) also shows some signs of having dependencies (the within-class permutation causes a slight decrease in test accuracy). According to the scheme presented in Section 4.3, pairwise permutations should only be done with attributes that both have shown an original dependent response, in this case only \(A_1\) and \(A_2\). However, here all attributes have been pairwise permuted together with \(A_1\) in order to confirm the expected behaviours. When \(A_1\) is permuted together with \(A_2\), the response changes from dependent to a more independent behaviour. This can be seen in Figure 6.4.
Figure 6.1: MONK1 data. $A_1 = A_2$ or $A_5 = 1$

Figure 6.2: MONK1 data. Attribute $A_1$ permuted together with secondary attributes.
**MONK2**

In the MONK2 dataset (EXACTLY TWO of $A_1 = 1$, $A_2 = 1$, $A_3 = 1$, $A_4 = 1$, $A_5 = 1$, $A_6 = 1$) all attributes are pair-wise dependent. $GP_{wcbcp}$ correctly classifies all attributes (the within-class score equals the between-class score in all cases, suggesting a dependency among the variables).

![Figure 6.3: MONK2 data](image)

**MONK3**

In the MONK3 dataset ($A_5 = 3$ and $A_4 = 1$) or ($A_5 \neq 4$ and $A_2 \neq 3$), there are three important attributes. $GP_{wcbcp}$ does however only find two of these ($A_4$ is not found). Still, these results are equivalent to the ones in a previous study [9], where the authors noted that by using only $A_2$ and $A_5$ 97% test accuracy can be achieved.
Figure 6.4: MONK3 data. \((A_5 = 3 \text{ and } A_4 = 1) \text{ or } (A_5 \neq 4 \text{ and } A_2 \neq 3)\). 5 \% added noise

6.1.2 Constructed data

Finally, \(GP_{wcbp}\) is applied to the constructed data set described in Section 5.2.3. Neither RELIEFF nor CFS identifies any of the dependent attributes, even though it is possible to achieve a perfect separation using any of the dependent groups of attributes. \(GP_{wcbp}\) does find one of these groups, but not the other, more complex, one. Figure 6.5 shows the corresponding permutation graph.

\(^1\)Attributes classified as dependent are marked with *.

\(^2\)The complete RELIEFF ranking is 6-10-3-13-7-16-11-4-8-18-9-15-17-5-19-2-12-1-14.
### 6.2 SVM\textsubscript{wc}bcp

Before applying the permutation tests to SVM’s in order to classify real data, a couple of motivating examples will be given. Figure 6.6 (upper row) shows two data sets, one where the two attributes are independent (left) and one where the attributes depend on each other (right). The separating lines obtained by the SVM and the support vectors are also marked in the figure.

Below the data sets the corresponding permutation graphs can be found. As seen in the rightmost example in Figure 6.6, \( A_2 \) is completely dependent on...
$A_1$ in order to be predictive whereas $A_1$ has a certain degree of predictability even without using $A_2$.

This example serves as a intuitive visualization of how the permutation tests works when applied to SVM’s, and why they are unlikely to outperform correlation based methods (a permutation of a variable directly reflects the separability in its dimension), which will be further discussed in Section 7.

![Graphs and bar charts showing independent and dependent examples](image)

Figure 6.6: Upper left: Independent example. Lower left: Corresponding permutation graph (independent). Upper right: Dependent example. Lower right: Corresponding permutation graph (dependent).

### 6.3 Cytometry data

In order to evaluate the suitability of the chosen attributes, classifiers are trained to separate the classes, two and two. The pairings was chosen based
on the results from a previous study of the data described in Section 5.2.2, using a classification algorithm based on Differential Evolution with a Cauchy Schwarz fitness function [15]. Some easily separated examples and some more difficult separations have been included. In Section 6.3.1 the permutation tests are applied to MVGPC and in Section 6.3.2 to a SVM classifier. The top attributes as chosen by the $GP_{wcbcp}/SVM_{wcbcp}$ are then used to train a new classifier and compared to classifiers trained using the top attributes given by other standard feature selection algorithms.

6.3.1 GP

Table 6.3 - 6.9 show a comparison of the top attributes found by DE (the attributes with the highest weights using Differential Evolution and a Cauchy Schwarz based divergence measure to find the most well separating linear projections, as described by Aranha et. al [15]), $GP_{wcbcp}$, GP frequencies (the attributes used most often in the rules) and RELIEFF. The ”GP frequencies” attributes are not used to train new classifiers, but mainly included for comparison - if they always equalled the ones selected by $GP_{wcbpc}$, $GP_{wcbpc}$ would be superfluous.

<table>
<thead>
<tr>
<th>DE</th>
<th>$GP_{wcbcp}$</th>
<th>GP frequencies</th>
<th>RELIEFF</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>12</td>
</tr>
<tr>
<td>12</td>
<td>13</td>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>13</td>
<td>8</td>
<td>13</td>
<td>2</td>
</tr>
<tr>
<td>22</td>
<td>3</td>
<td>23</td>
<td>16</td>
</tr>
<tr>
<td>17</td>
<td>23</td>
<td>2</td>
<td>22</td>
</tr>
</tbody>
</table>

Table 6.3: Top attributes, $c_1/c_3$

<table>
<thead>
<tr>
<th>DE</th>
<th>$GP_{wcbcp}$</th>
<th>GP frequencies</th>
<th>RELIEFF</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>12</td>
<td>12</td>
<td>12</td>
</tr>
<tr>
<td>22</td>
<td>22</td>
<td>15</td>
<td>14</td>
</tr>
<tr>
<td>14</td>
<td>14</td>
<td>7</td>
<td>22</td>
</tr>
<tr>
<td>8</td>
<td>15</td>
<td>14</td>
<td>10</td>
</tr>
</tbody>
</table>

Table 6.4: Top attributes, $c_1/c_5$
### Table 6.5: Top attributes, $c_1/c_8$

<table>
<thead>
<tr>
<th>DE</th>
<th>$GP_{wcbp}$</th>
<th>GP frequencies</th>
<th>RELIEFF</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>12</td>
</tr>
<tr>
<td>22</td>
<td>22</td>
<td>17</td>
<td>10</td>
</tr>
<tr>
<td>12</td>
<td>12</td>
<td>15</td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>17</td>
<td>12</td>
<td>22</td>
</tr>
<tr>
<td>17</td>
<td>13</td>
<td>22</td>
<td>25</td>
</tr>
</tbody>
</table>

### Table 6.6: Top attributes, $c_3/c_6$

<table>
<thead>
<tr>
<th>DE</th>
<th>$GP_{wcbp}$</th>
<th>GP frequencies</th>
<th>RELIEFF</th>
</tr>
</thead>
<tbody>
<tr>
<td>28</td>
<td>28</td>
<td>28</td>
<td>28</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>24</td>
<td>23</td>
</tr>
<tr>
<td>3</td>
<td>24</td>
<td>22</td>
<td>2</td>
</tr>
<tr>
<td>11</td>
<td>3</td>
<td>2</td>
<td>22</td>
</tr>
</tbody>
</table>

### Table 6.7: Top attributes, $c_3/c_7$

<table>
<thead>
<tr>
<th>DE</th>
<th>$GP_{wcbp}$</th>
<th>GP frequencies</th>
<th>RELIEFF</th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
<td>20</td>
<td>20</td>
<td>24</td>
</tr>
<tr>
<td>17</td>
<td>24</td>
<td>24</td>
<td>26</td>
</tr>
<tr>
<td>26</td>
<td>7</td>
<td>7</td>
<td>29</td>
</tr>
<tr>
<td>3</td>
<td>13</td>
<td>1</td>
<td>21</td>
</tr>
<tr>
<td>24</td>
<td>21</td>
<td>21</td>
<td>7</td>
</tr>
</tbody>
</table>

### Table 6.8: Top attributes, $c_6/c_7$

<table>
<thead>
<tr>
<th>DE</th>
<th>$GP_{wcbp}$</th>
<th>GP frequencies</th>
<th>RELIEFF</th>
</tr>
</thead>
<tbody>
<tr>
<td>28</td>
<td>28</td>
<td>28</td>
<td>26</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>27</td>
<td>28</td>
</tr>
<tr>
<td>3</td>
<td>27</td>
<td>2</td>
<td>22</td>
</tr>
<tr>
<td>4</td>
<td>7</td>
<td>20</td>
<td>5</td>
</tr>
</tbody>
</table>
Looking at Table 6.10, first one can note that the classifiers trained using the top attributes from $GP_{wcbcp}$ seem to always improve the accuracy. In some cases, where the test accuracy using the unpruned attribute set already gave relatively good performance, there are however no noticeable difference. In a few cases $GP_{wcbcp}$ also performs better than the other methods, especially in the separation $c_3/c_7$, where the competing methods in fact produce worse classifiers than when all attributes were used, indicating that important attributes have been left out, possibly due to undetected dependencies.

Table 6.10: Performance comparison - cytometry data (MVGPC). The first row for each dataset shows the average performance per tree, and the notation $(v)$ indicates that the performance of the voting ensemble, MVGPC, is shown.
6.3.2 SVM

In a similar way to the previous section, first the important attributes according to the different attribute selection methods will be listed, followed by a comparison of how well they fare when used to train the final SVM classifier.

Attributes chosen by CFS are used as they are expected to perform well in this setting. However, as CFS provides a set of important attributes instead of a ranking, often there are more attributes chosen by CFS than the other two methods (the number of attributes included from RELIEFF is set to equal the number chosen by SVM$_{wcbcp}$, which might give SVM$_{wcbcp}$ a slightly unfair advantage).

<table>
<thead>
<tr>
<th>SVM$_{wcbcp}$</th>
<th>RELIEFF</th>
<th>CFS</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>12</td>
<td>1,6,8,</td>
</tr>
<tr>
<td>22</td>
<td>1</td>
<td>10,12,13,</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>19,22</td>
</tr>
</tbody>
</table>

Table 6.11: Top attributes, 1/3

<table>
<thead>
<tr>
<th>SVM$_{wcbcp}$</th>
<th>RELIEFF</th>
<th>CFS</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>12</td>
<td>8,9,10,</td>
</tr>
<tr>
<td>14</td>
<td>14</td>
<td>12,14,</td>
</tr>
<tr>
<td>22</td>
<td>22</td>
<td>22,29</td>
</tr>
</tbody>
</table>

Table 6.12: Top attributes, 1/5

<table>
<thead>
<tr>
<th>SVM$_{wcbcp}$</th>
<th>RELIEFF</th>
<th>CFS</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>12</td>
<td>1,10,</td>
</tr>
<tr>
<td>12</td>
<td>10</td>
<td>11,12,</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>13,17,</td>
</tr>
<tr>
<td>1</td>
<td>22</td>
<td>22,28</td>
</tr>
</tbody>
</table>

Table 6.13: Top attributes, 1/8

<table>
<thead>
<tr>
<th>SVM$_{wcbcp}$</th>
<th>RELIEFF</th>
<th>CFS</th>
</tr>
</thead>
<tbody>
<tr>
<td>28</td>
<td>28</td>
<td>2,3,4,</td>
</tr>
<tr>
<td>2</td>
<td>23</td>
<td>5,7,</td>
</tr>
<tr>
<td>11</td>
<td>2</td>
<td>11,20</td>
</tr>
<tr>
<td>7</td>
<td>22</td>
<td>26,28</td>
</tr>
<tr>
<td>5</td>
<td>9</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 6.14: Top attributes, 3/6
As can be seen in Table 6.18 the accuracy is nearly always improved when using the top attributes from SVM_{wcbcp} even though the top attributes from CFS perform equally well. RELIEFF does however not seem to choose suitable attributes for SVM classification.

The standard deviation shown is due to the training and test-data being resampled five times.

<table>
<thead>
<tr>
<th>Clusters</th>
<th>All attributes</th>
<th>$SVM_{wcbcp}$</th>
<th>$RELIEFF_{top}$</th>
<th>$CFS_{top}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$c_1/c_3$</td>
<td>92.2 ± 1.5</td>
<td>96.8 ± 0.4</td>
<td>98.2 ± 1.5</td>
<td>99.4 ± 0.5</td>
</tr>
<tr>
<td>$c_1/c_5$</td>
<td>97.4 ± 0.9</td>
<td>99.8 ± 0.4</td>
<td>99.8 ± 0.4</td>
<td>99.2 ± 0.8</td>
</tr>
<tr>
<td>$c_1/c_8$</td>
<td>97.6 ± 1.1</td>
<td>97.4 ± 0.9</td>
<td>97.4 ± 0.9</td>
<td>100 ± 0.0</td>
</tr>
<tr>
<td>$c_3/c_6$</td>
<td>97 ± 1.2</td>
<td>100 ± 0.0</td>
<td>100 ± 0.0</td>
<td>98 ± 1.2</td>
</tr>
<tr>
<td>$c_3/c_7$</td>
<td>71.2 ± 2.9</td>
<td>83.4 ± 2.1</td>
<td>66.6 ± 3.3</td>
<td>80.2 ± 4.3</td>
</tr>
<tr>
<td>$c_6/c_7$</td>
<td>94.4 ± 1.8</td>
<td>98.6 ± 0.9</td>
<td>97.4 ± 0.5</td>
<td>99.2 ± 0.8</td>
</tr>
<tr>
<td>$c_7/c_9$</td>
<td>53.2 ± 1.8</td>
<td>64.8 ± 1.3</td>
<td>53.8 ± 2.1</td>
<td>67.6 ± 2.7</td>
</tr>
</tbody>
</table>

Table 6.18: Performance comparison - cytometry data (SVM)
6.4 UCI datasets

Next, the proposed method is tested on the UCI datasets described in Section 5.2.1. First, in Table 6.19, 6.21 and 6.23, the important attributes as selected by $GP_{wcbcp}$, RELIEFF, CFS and $GP_{mtfs}$ are listed, with an exception for the WDBC data set, where the developers of $GP_{mtfs}$ did not include the actual feature ranking, and hence it has been left out [13].

The voting approach described in Section 4.2.2 has also been tested, and Table 6.20, 6.22 and 6.24 show the resulting dependency classifications. The thresholds chosen, used to determine the dependency class, were very arbitrary and only based on some rough experimental tuning.

6.4.1 WDBC

<table>
<thead>
<tr>
<th>$GP_{wcbcp}$</th>
<th>GP frequencies</th>
<th>RELIEFF</th>
<th>CFS</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>10</td>
<td>25</td>
<td>2,7,8,14,19,21,</td>
</tr>
<tr>
<td>24</td>
<td>8</td>
<td>22</td>
<td>23,24,25,27,28</td>
</tr>
<tr>
<td>28</td>
<td>27</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>21</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>14</td>
<td>29</td>
<td></td>
</tr>
<tr>
<td>27</td>
<td>24</td>
<td>27</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>25</td>
<td>23</td>
<td></td>
</tr>
</tbody>
</table>

Table 6.19: Top attributes, WDBC

| Irrelevant     | 1,2,3,5,6,9,10,11,12,13,15,16,17,18,19,20,22,23,26,29,30 |
| Independent    | (4),7,8,11,14,21,24,(25),27,28                        |
| Dependent      | -                                                        |
| Dependencies   | -                                                        |

Table 6.20: Voting classifications, WDBC
6.4.2 Wine

<table>
<thead>
<tr>
<th>$GP_{wcbcp}$</th>
<th>GP frequencies</th>
<th>RELIEFF</th>
<th>$GP_{mtfs}$</th>
<th>CFS</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>10</td>
<td>1</td>
<td>13</td>
<td>1,2,3,</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>7</td>
<td>10</td>
<td>4,5,6,7,</td>
</tr>
<tr>
<td>13</td>
<td>12</td>
<td>4</td>
<td>1</td>
<td>10,11,</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>9</td>
<td>7</td>
<td>12,13</td>
</tr>
<tr>
<td>12</td>
<td>6</td>
<td>5</td>
<td>6</td>
<td></td>
</tr>
</tbody>
</table>

Table 6.21: Top attributes, Wine

<table>
<thead>
<tr>
<th>Irrelevant</th>
<th>1,2,3,5,6,8,9,11</th>
</tr>
</thead>
<tbody>
<tr>
<td>Independent</td>
<td>4,7,10,12,13</td>
</tr>
<tr>
<td>Dependent</td>
<td>-</td>
</tr>
<tr>
<td>Dependencies</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 6.22: Voting classifications, Wine

6.4.3 Vehicle

<table>
<thead>
<tr>
<th>$GP_{wcbcp}$</th>
<th>GP frequencies</th>
<th>RELIEFF</th>
<th>$GP_{mtfs}$</th>
<th>CFS</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>6</td>
<td>10</td>
<td>7</td>
<td>4,5,6,7,</td>
</tr>
<tr>
<td>10</td>
<td>11</td>
<td>3</td>
<td>13</td>
<td>8,9,11,12,</td>
</tr>
<tr>
<td>6</td>
<td>17</td>
<td>18</td>
<td>10</td>
<td>14,15,16</td>
</tr>
<tr>
<td>12</td>
<td>4</td>
<td>17</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>10</td>
<td>8</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>8</td>
<td>1</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>7</td>
<td>15</td>
<td>16</td>
<td></td>
</tr>
</tbody>
</table>

Table 6.23: Top attributes, Vehicle

46
### 6.4.4 Evaluation

Table 6.26 shows the performance of the final classifiers for the UCI datasets. As before, rows marked \((v)\) show the performance of MVGPC and the others \textit{average per tree} performance.

It should be noted that the decreased voting-accuracy for the Wine and Vehicle set (compared to the average per tree) is due to the fact that they contain more than two classes. Average per tree accuracy always refers to a two-class separation.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>All attributes</th>
<th>$GP_{wcbcp-top}$</th>
<th>$RELIEFF_{top}$</th>
<th>CFS</th>
<th>$GP_{mtfs-top}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>WDBC</td>
<td>93.4 ± 1.6</td>
<td>93.4 ± 3.5</td>
<td>92.2 ± 2.4</td>
<td>96.1 ± 1.4</td>
<td>-</td>
</tr>
<tr>
<td>WDBC (v)</td>
<td>95.7</td>
<td>94.1</td>
<td>93.0</td>
<td>97.3</td>
<td>-</td>
</tr>
<tr>
<td>Wine</td>
<td>93.3 ± 4.7</td>
<td>94.6 ± 4.1</td>
<td>89.7 ± 5.6</td>
<td>94.3 ± 3.4</td>
<td>91.6 ± 6.2</td>
</tr>
<tr>
<td>Wine (v)</td>
<td>94.8</td>
<td>98.3</td>
<td>87.9</td>
<td>96.5</td>
<td>89.6</td>
</tr>
<tr>
<td>Vehicle</td>
<td>62.7 ± 26</td>
<td>78.4 ± 10</td>
<td>74.2 ± 6.9</td>
<td>76.8 ± 11</td>
<td>71.9 ± 10</td>
</tr>
<tr>
<td>Vehicle (v)</td>
<td>47.0</td>
<td>68.1</td>
<td>57.0</td>
<td>67.4</td>
<td>42.3</td>
</tr>
</tbody>
</table>

Table 6.25: Performance comparison - UCI data

<table>
<thead>
<tr>
<th>Dataset</th>
<th>$GP_{all}$</th>
<th>$GP_{wcbcp-top}$</th>
<th>$GP_{mtfs-all}$</th>
<th>$GP_{mtfs-top}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>WDBC</td>
<td>95.7</td>
<td>94.1</td>
<td>97.26</td>
<td>96.31</td>
</tr>
<tr>
<td>Wine</td>
<td>94.8</td>
<td>98.3</td>
<td>95.49</td>
<td>94.82</td>
</tr>
<tr>
<td>Vehicle</td>
<td>47.0</td>
<td>68.1</td>
<td>78.37</td>
<td>78.45</td>
</tr>
</tbody>
</table>

Table 6.26: Performance comparison - learning algorithms
6.5 Attribute construction

The vehicle dataset seem to contain dependencies and was therefore investigated further by adding new attributes, constructed from pairs of seemingly dependent attributes. The hope was that this would improve the accuracy of the resulting classifier.

Looking at Table 6.24, one can note that attributes 2-8, 2-10 and 8-10 are likely to depend on each other. New attributes were therefore added combining these attributes with different operators (*,/,+,-). Even thought the performance was slightly improved compared to the results in the previous section, it cannot be said to be any significant change.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>$GP_{wcecp-voting}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vehicle</td>
<td>79.9 ± 10</td>
</tr>
<tr>
<td>Vehicle (v)</td>
<td>71.0</td>
</tr>
</tbody>
</table>

Table 6.27: Performance after added attribute constructions
Chapter 7
Discussion and Future work

7.1 Summary of contributions

The results from the previous section showed that the proposed method was able to select good features that, when fed to the learning algorithm, resulted in better classifiers than when all attributes were used, in almost all experiments. This improvement was also shown to be in strong contention with other feature selection methods. On top of this, known dependencies in artificial data like the MONK data could be detected and correctly labelled as independently or dependently predictive.

7.2 Discussion

The results presented in Section 6 are thus overall promising, but there is still room for improvement. Even though the predictability is often improved and never worsened when using the top attributes from $GP_{wcbp}/SVM_{wcbp}$ to build the classifier, the improvement can not be said to be significant compared to when using the top attributes selected by eg. CFS.

However, the fact that there are knowingly important attributes (Section 6.1.2) that $GP_{wcbp}$ can find that CFS cannot, combined with the additional knowledge of feature dependencies given by $GP_{wcbp}$ indicates that $GP_{wcbp}$ may be preferable when using a certain type of data sets.

Future work: Apply $GP_{wcbp}$ to other sets of data, preferably with (possibly known) attribute dependencies.

49
Looking at eg. Table 6.10 (Section 6.3.1, Cytometry data) or especially the Table 6.25 (Section 6.4, UCI data) a few instances can be found where \( GP_{wcbcp} \) achieves the same test accuracy per tree as some other feature selection techniques, but still manages to perform better with regard to the voting ensemble. Especially the Vehicle set in Table 6.25 is noteworthy, where CFS has results comparable with \( GP_{wcbcp} \), but even though both RELIEFF and \( GP_{nt.fs} \) also have similar performance per tree, their voting ensembles perform much worse.

This is somewhat surprising, since the permutation tests are applied to the trees themselves and the change in performance is only measured with respect to change in per tree performance. Good performance of ensembles have additional, seemingly unrelated criteria, such as a large diversity among the classifiers. That is, there is no apparent reason for \( GP_{wcbcp} \) to perform better than other methods with respect to the voting ensemble, when they have the same, or even worse, average-per-tree performance.

In Section 6.3.1 the permutation tests were applied to SVM’s, mainly as a demonstrating example. It did not come as a big surprise that \( SVM_{wcbcp} \) could not outperform methods such as CFS. Applying the permutation tests to a linear SVM merely means permuting a variable in one dimension, where the two classes are separated by a line. Similar results should thus easily be obtainable using correlation based methods. Even though the proposed method was constructed having GP or less mathematically evaluable learning algorithms in mind, \( SVM_{wcbcp} \) still showed results very similar to CFS, with the added benefit of offering a feature ranking and dependency detection.

The results from Section 6.1 show that most attribute dependencies are found when \( GP_{wcbcp} \) is applied to known data, however not all. The data set constructed specially for this thesis contain two groups of dependent attributes that, each on their own, can completely separate the data. Even though only the less complex dependency group is found, this does not mean that \( GP_{wcbcp} \) cannot correctly classify the other dependent variables had they been used in a somewhat reasonable way. One flaw with the proposed method is that the classifier trained using all attributes must somewhat properly take advantage of the important attributes in order for them to be classified as important. If there exist dependencies, but these are never exploited, i.e. they are never actually used to obtain a better test accuracy, there cannot be any detectable decrease in accuracy when the attributes are permuted. This makes the proposed method unsuitable for very high-dimensional data where
preprocessing is necessary in order to find any sort of acceptable classifier. However, for genetic programs that can contain a lot of excessive variables due to bloat, and where the relationships responsible for the predictability might be partly occluded and not in their optimal form, \( GP_{wcbep} \), should still be a suitable choice. If the data is too high dimensional it might have to be preprocessed or if one suspects that more complex relationships, that may still be able to contribute to a higher predictability, may not occur in the evolved rules due to premature detection of simpler rules, a backward elimination strategy (eliminating the found important variables) might be adopted.

\[ \textbf{Future work: Trying a backward elimination strategy to find more hidden relationships, by removing easily found important variables one by one.} \]

It is interesting to note that the top attributes according to \( GP_{mtfs} \) show relatively poor performance when used to train a MVGPC classifier. Since both classifiers use GP one would expect them to have similar sets of optimal attributes.

Even though the attributes from \( GP_{mtfs} \) produce worse MVGPC classifiers, \( GP_{mtfs} \) itself performs better on the Vehicle data set than any of the MVGPC classifiers, as can be seen in Table 6.26. Muni et al. report performances of 94.82% and 78.45% for the Wine and Vehicle data sets, respectively. This can be compared to the 98.3% and 68.1% of \( GP_{wcbep} \). Still, using the top attributes from \( GP_{mtfs} \) to train a MVGPC classifier gave as poor performance as 89.6% for the Wine set and merely 42.3% for the Vehicle set, which means that the better performance reported in Muni's paper does not indicate that the the feature selection method presented here is bad (on the contrary it confirms the desired bias towards choosing good attributes for the learning algorithm in question), only that the parameters used during evolution of MVGPC may be non-optimal or that MVGPC itself has room for improvement. The goal of selecting good attributes with respect to the learning algorithm of choice is still achieved, even though there might exist better learning algorithms.

\[ \textbf{Future work:} \text{ Tune the parameters of MVGPC and/or apply the permutation test to another learning algorithm.} \]
The Vehicle set is interesting due to the relatively poor performance and
the seemingly high degree of feature interactions (See Figure C.3). MVGPC
creates four ensembles of genetic programs for this data set - one for each
class. One ensemble will learn how to distinguish between OPEL and \{SAAB,
BUS, VAN\}, while another will concentrate on SAAB versus \{OPEL, BUS,
VAN\}, and so on. This takes focus away from the biggest challenge in this
data set, which is the separation between the two similar classes, SAAB and
OPEL. Discarding the SAAB class completely when distinguishing between
OPEL and \{SAAB, BUS, VAN\} can still achieve 75% accuracy, and such a
solution is likely to be evolved before solutions that concentrate on classifying
SAAB samples correctly. The attributes used to recognize the BUS or VAN
classes may also be irrelevant or even disruptive when discriminating between
the SAAB and OPEL classes, making it difficult to further steer evolution
towards a perfect classification of all samples.

The option to have rules that make pairwise separation between classes
might seem tempting at first, but since SAAB samples will also be shown to
classifiers trained to separate between eg. OPEL and VAN classes exclusively,
this naive solution will produce many false positives.

One alternative is to create three classifiers for each member of the ensem-
bles and have an internal decision process based on their classifications. In the
element SAAB versus \{OPEL, BUS, VAN\}, three classifiers, \(C_1, C_2\) and \(C_3\),
that can discriminate between SAAB/OPEL, SAAB/BUS and SAAB/VAN,
respectively, would be created. Such a classifier is illustrated in Figure 7.1. If
\(all\) these subclassifiers classify a sample as SAAB, the final classification for
this member of the ensemble will be SAAB. If \(at least one\) of the classifiers
makes the classification \(\neg SAAB\) the final classification will also be \(\neg SAAB\).
This avoids the problem of false positives, since each individual classifier still
only separates between eg. SAAB and \{OPEL, BUS, VAN\} and will never
be forced to place invalid votes.

For the sake of argument, lets assume that the probability of misclassifi-
cation is \(p\) for each of the three classifiers, the probability of misclassification
in the original MVGPC implementation is \(p_{mvGPC}\), and that \(p < p_{mvGPC}\) since
it should be easier to distinguish between two classes than four. For a sam-
ple, \(s_i \in \{OPEL, BUS, VAN\}\), say \(s_i \in OPEL\), the probability of correctly
classifying the sample \(s_i\) becomes \((1 - p) + p(1 - 0.5^2) = 1 - 0.25p \geq (1 - p) >
(1 - p_{mvGPC})\). Here it is assumed that the classifiers that have not been trained
on the OPEL class, eg. the one separating between SAAB and BUS will have
50% chance of making the classification \(\neg SAAB\). That is, even if the classi-
A classifier trained to separate between SAAB and OPEL fails to make the correct classification, the final classification can still end up correct due to the random classification of the other two classifiers. The probability of correctly classifying a $\neg SAAB$ sample is thus much higher than that of the MVGPC classifier.

However, for a sample, $s_i \in SAAB$, all three classifiers must classify the sample correctly, i.e. the probability of correct classification is $(1 - p)^3$, which may or may not be higher than $(1 - p_{mvgpc})$ depending on how much the classification accuracy can be improved by considering pairwise match-ups instead of the one-versus-the-rest approach. Let's assume that this improvement is $c$, i.e. $(1 - p) = c(1 - p_{mvgpc})$. To achieve a better true...
positive rate with this new approach \((1-p)^3 = c^3(1-p_{mgpc})^3 > (1-p_{mgpc})\) must hold. This implies that \(c > \left(\frac{1}{(1-p_{mgpc})^2}\right)^{1/3} = \frac{1}{(1-p_{mgpc})^{2/3}}\). If the \(GP_{mgpc-top}\) accuracy from Table 6.25 (0.78) is used, \((1-p) \geq 0.92\) leads to more true positives. However, incorporating the false positive rates, which does account for 75% of the cases in a 4-class problem, gives an average total classification accuracy, \((1-\bar{p}_{avg})\), of \(\frac{3n(1-0.25p)+(1-p)^3}{4}\) or more generally for a \(k\)-class problem:

\[
(1-\bar{p}_{avg}) = \frac{(k-1)(1-0.25p) + (1-p)^3}{k} \tag{7.1}
\]

Setting \((1-p) = c(1-p_{mgpc})\) and \((1-p_{mgpc}) = 0.78\), gives an average performance of 0.78 when \((1-\bar{p}_{avg}) = 0.70\). That is, if the performance of the pairwise separation is 0.70, this new approach is equivalent to the MVGPC approach, and with any higher performance, the new classifier is preferable. Intuitively it should be easier to separate one class from another instead of three others, so this approach seems promising as, in this case, the required accuracy is even lower than that of MVGPC.

It may also be preferable to separate the important features chosen such that each pairwise class separation is given its own feature set. As mentioned earlier, features used to separate between SAAB and BUS might not be important in the separation between SAAB and OPEL.

Even though it might at a first glance seem computationally expensive to train \((k-1)\) times as many classifiers, this is not necessarily the case. Actually, if the learning algorithm has quadratic time complexity this new solution is faster, since each pairwise classifier only need to be trained on \(\frac{2}{k}\) parts of the data, making the total training time \((k-1)(\frac{2}{k}n)^2 < n^2\) when \(k \geq 3\) and \(n\) is the total number of samples. With linear time complexity the method will however be slower, since \((k-1)(\frac{2}{k}n) = 2n - \frac{2}{k}n > n\) when \(k \geq 3\).

\[\text{Future work: Modify MVGPC and make each member of an ensemble separating class } c_i \text{ from the rest consist of } k \text{ internal classifiers as described above. Select different feature sets for different pairwise separations.}\]

This suspicion that a poor SAAB vs OPEL separation is mainly responsible for the poor performance in the Vehicle set, may also have been a reason that the attribute construction in Section 6.1.2 did not show any
significant improvement. The accuracy, when discarding this discrimination, might already have been fairly high and the attributes found mainly used to separate between the easier classes. In any case, more investigation on the subject of attribute construction is needed.

Future work: Investigate attribute construction further or other ways to make use of the dependency classifications.

Finally, there is one problem with the way dependent attributes are paired up. The underlying assumption is that when more attributes in a dependent group is permuted together, the accuracy should be improved. This is clearly true for a group of two attributes. It is however most likely difficult to determine the exact attribute dependencies when the dependent group consists of three or more attributes. Consider a tree $T_i$ that contains three dependent attributes, $a_1$, $a_2$ and $a_3$. When one of these is permuted, say $a_1$, the within-class score is decreased by a factor that should reflect the fact that only $2/3$ of the dependent attributes are intact. However, the proposal to permute attributes pairwise in order to find dependencies might fail in this situation. If $a_1$ and $a_2$ are given the values from another tree $T_j$, the within-class permuted $T_i$ now only has its original attribute $a_3$, with $a_1$ and $a_2$ from $T_j$, but the number of dependent attributes from the same tree is still the same, i.e. $2/3$.

Finding exactly which attributes that depend on each other might thus be difficult for large groups of attributes where the dependencies are fairly equal in strength.

Future work: Empirically and/or theoretically investigate the behaviour of dependencies consisting of more than two attributes, when the attributes are permuted one and one versus two and two.
Chapter 8

Conclusion

This work has outlined and evaluated a feature selection method based on two different permutations tests, referred to as the within-class and between-class permutationsm, \( wcbcp \). The proposed method has been tested on some common benchmark data as well as pairwise cell match-ups from cytometry data.

Using only the top attributes chosen by \( GP_{wcbcp} \) to train new classifiers led to improved prediction accuracies, compared to the classifiers built having access to all attributes, in all but a few cases where there was no significant difference. This indicates that truly important attributes had indeed been selected by \( GP_{wcbcp} \), or at least that the most important attributes had not been discarded.

The classifiers built from the top attributes selected by \( GP_{wcbcp} \) also showed results comparable with those using attributes based on other feature selection methods like RELIEFF and CFS. In some cases the performance was better, in some other fairly equal, but never significantly worse.

Even though primarily targeted towards genetic programs, the permutations test were also coupled with SVM’s as a way to show its applicability to other learning algorithms. The results showed that \( SVM_{wcbcp} \) fared equally well as CFS, and even better than RELIEFF even though this might be due to the relatively few training examples available. \( SVM_{wcbcp} \) was not expected to outperform eg. correlation based attribute selection methods since, intuitively, permutations of one attribute (dimension) in the context of linear SVM’s can be translated to simply evaluating the class separability in this dimension.

Thus, even though the permutation tests can be applied to other learning
algorithms, like linear SVM’s, the initial motivation was to use them together with classifiers that are able to combine attributes in ways that make it difficult for traditional feature selection methods to select appropriate features. This difficulty was demonstrated with a simple example using a perfectly separable artificial data set, where neither CFS or RELIEFF could detect the interacting attributes needed in order to perfectly separate the classes. \( GP_{wcbcp} \) could however easily detect the most prominent dependencies and was also able to successfully classify the attributes contained in the MONK data set as being either irrelevant, independent or dependent, with the exception of one important attribute in the MONK3 data set that was not found. However, this was probably not due to a direct vulnerability of \( GP_{wcbcp} \) but instead representative of the fact that important attributes cannot be found unless they are used to some extent in the first classifiers evolved. This is of course a limitation in itself and might become a problem when the data is very high-dimensional.

Only a minor attempt was made at attribute construction, which did not show any significant improvement, however this topic is left as future work, with some outlined guidance in Section 7.

It is now time to revisit the questions posed in the problem statement:

1. **Can dependent attributes be detected when tested on artificial data with known dependencies?**
   
   Yes, the dependent attributes were detected in the MONK data sets and also the most prominent ones in the constructed data set.

2. **Does attribute selection using the proposed method result in a better final classifier than attribute selection using other commonly used attribute selection methods?**
   
   No, it cannot be confirmed by the experiments in this thesis that the proposed method always is preferable. However, it was always at least as good as the other feature selection methods used, and considering the positive answer on the previous question, has potential to outperform other methods when applied to data containing dependencies that, when exploited, provide important discriminative properties.

3. **Can meaningful attributes be constructed by adding subtrees created from attributes deemed dependent on each other? That is, can the final classifier be further improved by including such new constructions?**

   57
No, this has not been shown in this thesis. There was only one attempt made at attribute construction, which produced a slight improvement, however not significant. Further investigation is needed.

Finally, to summarize, the proposed method has shown to be able to detect independently as well as dependently predictive attributes. It has also shown classification accuracies on a par with some standard feature selection methods and have potential to produce even better results if applied to the right type of data, containing dependencies that are difficult to find for standard attribute selection algorithms.
Bibliography


Appendices
Appendix A

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Population size</td>
<td>1000</td>
</tr>
<tr>
<td>Number of generations</td>
<td>50</td>
</tr>
<tr>
<td>Crossover probability</td>
<td>0.9</td>
</tr>
<tr>
<td>Mutation probability</td>
<td>0.1</td>
</tr>
<tr>
<td>Reproduction probability</td>
<td>0.1</td>
</tr>
<tr>
<td>Max crossover depth</td>
<td>7</td>
</tr>
<tr>
<td>Max initial depth</td>
<td>6</td>
</tr>
<tr>
<td>Select from previous generation</td>
<td>0.10</td>
</tr>
</tbody>
</table>

Table A.1: JGAP parameter
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ensemble size / class</td>
<td>42</td>
</tr>
<tr>
<td>Number of re-samplings (train/test data)</td>
<td>5</td>
</tr>
<tr>
<td>Number of permutations / evaluation</td>
<td>20</td>
</tr>
<tr>
<td>$\Theta_{DMIN}$</td>
<td>10</td>
</tr>
<tr>
<td>$\Lambda$</td>
<td>3</td>
</tr>
<tr>
<td>$\Theta_{IRRELEVANT}$</td>
<td>0.95</td>
</tr>
<tr>
<td>$\Theta_{DEPENDENT}$</td>
<td>0.50</td>
</tr>
</tbody>
</table>

Table A.2: Other parameters

\(^1\)From Algorithm 4.
Appendix B

Cytometry data - permutation graphs

Figure B.1: Test accuracy, within-class accuracy and between-class accuracy for the separation between cluster 1 and 3.
Figure B.2: Test accuracy, within-class accuracy and between-class accuracy for the separation between cluster 1 and 5.

Figure B.3: Test accuracy, within-class accuracy and between-class accuracy for the separation between cluster 1 and 8.
Figure B.4: Test accuracy, within-class accuracy and between-class accuracy for the separation between cluster 3 and 6.

Figure B.5: Test accuracy, within-class accuracy and between-class accuracy for the separation between cluster 3 and 7.
Figure B.6: Test accuracy, within-class accuracy and between-class accuracy for the separation between cluster 6 and 7.

Figure B.7: Test accuracy, within-class accuracy and between-class accuracy for the separation between cluster 7 and 9.
Appendix C

UCI data - permutation graphs

Figure C.1: Test accuracy, within-class accuracy and between-class accuracy, WDBC data.
Figure C.2: Test accuracy, within-class accuracy and between-class accuracy, Wine data

Figure C.3: Test accuracy, within-class accuracy and between-class accuracy, Vehicle data