Doctoral Thesis

A New Balance for Efficiency and Accuracy of Feature Selection for High-dimensional Datasets

by

Adrian Pino Angulo

March 2019

Graduate School of Applied Informatics
University of Hyogo

Abstract

Machine Learning has been one of the most hottest trends for the last ten years. Supervised classification as a sub-field of machine learning, is increasingly gaining popularity among researchers due to its versatility and power of application at any field where data is available. Among the most common examples of supervised learning we can find: microarray problem classification, cancer diagnosis and network intruder detection. Supervised classification is a central issue in machine learning and consists on finding a classification function $\ell : \mathbf{D} \rightarrow v(c)$ that is able to classify an arbitrary instance with unknown class from $v(c) \in C$. ℓ is built from analysing the relation between instances in $\mathbf{D}$. The performance of supervised classifiers is often measured in three directions: efficiency, representation complexity and accuracy. The efficiency refers to the time required to learn the classification function ℓ; while the representation complexity often refers to the number of bits used to represent the classification function. All these three factors can be strongly affected when there exist features in $\mathbf{D}$ that do not contain useful information to predict the class variable. Feature selection methods are able to identify and remove unneeded, irrelevant and redundant features from data that do not contribute to the improvement of the accuracy of a predictive model. Feature selection allows us to build models as good or with better accuracy whilst requiring less data. The process of selecting features is composed of two basic components: an evaluation function and a search engine. The evaluation function is a metric that evaluates quantitatively how good are a set of features to discriminate among class labels. On the other hand, the search engine is in charge of generating all the potential sets to be evaluated. Feature selection algorithms can be divided into three broad categories: wrapper, filter and embedded methods. To evaluate a feature set F, wrapper methods use some accuracy score of a classifier after being trained in the dataset projected by F. Wrapper methods are very low in efficiency since training and testing the inferred function is required for each evaluation. Conversely, filters make use of explanatory analysis on data to assign a score to each feature set. Filters are usually less computationally expensive than wrappers, but they output a feature set that is not tuned to a specific type of predictive model. Embedded methods learn which features best contribute to the accuracy of the model while the model is being created. The most common type of embedded feature selection methods are regularization or penalization methods. Filter-based feature selection can be also classified as: *feature ranking*, *pairwise evaluation* and *consistency-based* algorithms. The *feature ranking* methods evaluate relevance of individual features using statistical measures. That is, features are ranked using their individual relevance score and then the top features are selected. Although the ranking feature algorithms are usually simple and fast, they have two serious drawbacks that may affect the performance of supervised classifiers. First, redundant features are likely to be selected. Second, they usually can not detect interacting features. Oppositely to the *feature ranking* algorithms, pairwise

evaluation methods can detect and eliminate relevant features, but also are able to remove redundant features by computing the correlation between features. Consistency-based algorithms can detect interacting features by collectively evaluating relevance (correlation) of a feature set to the class. Although exhaustive search of all possible feature sets is computationally too expensive, the result can be expected to be accurate. In this paper, we propose several feature selection algorithms for high-dimensional data that can efficiently find very accurate solutions when compared with other benchmarking algorithms. Our contribution is as follows.

- We first, propose four new feature selection algorithms based on consistency measures, which are improvements of the current state-of-the-art algorithms: *Steepest-Descent-Consistency-Constrained* (SDCC), the *Linear-Consistency-Constrained* (LCC), *Super Linear-Consistency Constrained*(SLCC), respectively.

- Second, we propose a rule-based feature selection algorithm, namely, *Probabilistic Attribute Value Integration for Class Distinction* (PAVICD), which can detect interacting features and is extremely fast.

- Third, we propose a new version of the pairwise-evaluation-based algorithms, the *Fast Correlation based Filter* (FCBF) and the *Correlation-based Feature Selection* (CFS).

- Lastly, we propose an improvement of the hybrid feature selection algorithm, namely *Genetic Bee Colony for Feature Selection* (GBC).

All the proposed algorithms are tested in terms of accuracy, number of selected features and running time required. Results of the experiments in high-dimensional data exhibits that in most of the datasets our proposed algorithms are faster and more accurate than the original algorithms.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In the field of knowledge discovery, feature selection has been playing a crucial role to detect and remove irrelevant and redundant information from datasets. Feature selection is important not only to find good models that describe specific phenomena with a small number of explanatory variables, but also to improve efficiency and accuracy of machine learning algorithms [21]. In this research we focus on creating feature selection algorithms for high-dimensional data, so that the research community can use them to improve the machine learning process through the efficient and accurate selection of features.

1.1 Background

Machine Learning has been one of the most hottest trends for the last ten years. Supervised classification as a sub-field of machine learning, is increasingly gaining popularity among researchers due to its versatility and power of application at any field where data is available. Among the most common examples of supervised learning we can find: microarray problem classification [6][10], cancer diagnosis [40][34] and network intruder detection[52][2]. Supervised classification is incredibly powerful to make predictions and suggestions by means of inferring a function from labelled training data. The most basic structured data corresponds to a single data matrix

$$\mathbf{D} = \begin{bmatrix} x_1^1 & \cdots & x_1^n & c_1 \\ \vdots & \ddots & \vdots & \vdots \\ x_m^1 & \cdots & x_m^n & c_m \end{bmatrix},$$

where every instance x_j is described by a row vector $[x_j^1, \ldots, x_j^n, c_j]$: x_j^i is a value for the feature f_i; and c_j is a class label, which is a value for the class variable C. The collected data have no utility unless useful information is discovered from them. Supervised classification is a central issue in machine learning and consists on finding a classification

function $\ell : \mathbf{D} \rightarrow v(c)$ that is able to classify an arbitrary instance with unknown class from $v(c) \in C$. ℓ is built from analysing the relation between instances in $\mathbf{D}$ [41]. The performance of supervised classifiers is often measured in three directions: efficiency, representation complexity and accuracy. The efficiency refers to the time required to learn the classification function ℓ; while the representation complexity often refers to the number of bits used to represent the classification function. One of the most common metrics to measure the accuracy of a supervised classifier is the error rate defined as:

$$Err(\ell, \mathbf{D}) = \frac{1}{m} \sum_{j=1}^{m} \overline{\delta}(\ell(x_j), c_j),$$

where m is the number of instances in $\mathbf{D}$ and $\overline{\delta}$ is the complement of the Kronecker's delta function, which returns 0 if both arguments are equal and 1 otherwise. All these three factors can be strongly affected when there exist features in $\mathbf{D}$ that do not contain useful information to predict the class variable. Feature selection plays an essential role in supervised classification since its main goal is to identify and remove irrelevant and redundant features that do not contribute to minimize the error of a given classifier [32]. Basically, the advantages of feature selection include selecting a set of features $\tilde{F} = \{f_{i_1}, \ldots, f_{i_k}\} \subsetneq F$ with:

$$Err(\ell, \mathbf{D}_{\tilde{F}}) \leq Err(\ell, \mathbf{D}),$$

where $\mathbf{D}_{\tilde{F}}$ is the result of projecting $\tilde{F}$ over $\mathbf{D}$. The process of selecting features is composed of two basic components: an evaluation function and a search engine [42]. The evaluation function is a metric that evaluates quantitatively how good are a set of features to discriminate among class labels. On the other hand, the search engine is in charge of generating all the potential sets to be evaluated.

Feature selection algorithms can be divided into three broad categories: wrapper, filter and embedded methods [32]. To evaluate a feature set $\tilde{F}$, wrapper methods use some accuracy score of a classifier after being trained in the dataset projected by $\tilde{F}$. Wrapper methods are very low in efficiency since training and testing the inferred function is required for each evaluation. Conversely, filters make use of explanatory analysis on data to assign a score to each feature set. Filters are usually less computationally expensive than wrappers, but they output a feature set that is not tuned to a specific type of predictive model. Embedded methods learn which features best contribute to the accuracy of the model while the model is being created. The most common type of embedded feature selection methods are regularization or penalization methods [24].

Since we are especially interested in high-dimensional data, in this research we focus only on the filter approach. Although we also propose a new algorithm to speed up the

wrapper approach.

1.2 Motivation

The main motivation of this research lies on the benefits underlying the usage of the feature selection methods. Feature selection brings several benefits to the machine learning process, and in particular, can contribute the following advantages to the supervised learning:

- *Efficiency:* The running time and memory consumption of most of machine learning algorithms depends on the number of features in the datasets. Therefore, applying feature selection reduce the dimensionality of data, and consequently, the running time and memory consumption of machine learning algorithms may drastically decrease.

- *Accuracy:* The more information we have about a problem, not necessarily means the better decisions we can make. When data contains noisy, irrelevant and redundant features, the accuracy of machine learning models can be seriously affected. Since the aim of feature selection is to remove irrelevant and redundant features, the accuracy of machine learning techniques can be improved by only focusing in the relevant information of the data.

- *Resources:* In some real-world problems, the acquisition of data is very expensive. For example, when the measurement of some characteristics of the problem requires chemical analysis. The exclusion of irrelevant features may avoid wasting resources.

- *Scalability:* Some machine learning algorithms, such as those based on diffuse rules, suffer from the curs-of-dimensionality. Their accuracy is given by a mathematical function that grows very fast depending on the number of features (for example, exponential-order functions). The curse-of-dimensionality makes some algorithm no to be scalable to some high-dimensional datasets. Consequently, feature selection makes wider the scope of application of some machine learning algorithms.

- *Comprehensibility of results:* Some machine learning techniques, such as the rule-based and decision-tree-based classifiers returns a model that can be interpretable by researchers. However, when datasets are high-dimensional, this models are rather difficult to interpret due to the huge mount of information. However, feature selection is able to reduce the number of features, and hence, the comprehension of the model is easier.

Feature selection is an endless problem because as time passes the dimensionality of datasets, taken from real-world problems, increases. Therefore, the necessity of removing

features that do not contribute to the solution of our problems, and therefore, my harm the efficiency and accuracy of machine learning algorithms will continue to get higher. Subsequently, the number of feature selection algorithms proposed by researchers also increases in recent years. Some feature selection algorithms can be applied in a wide variety of problems, and the achieved improvement in such algorithms has resulted in valuable contributions in the real word. Other algorithms cannot be used with high-dimensional datasets, but there is a room of improvement so that they became scalable to the size of datasets.

The main motivation of this research is to improve some of the best feature selection algorithms so that they can be applied to high-dimensional datasets without sacrificing efficiency. We are very interested in high-dimensional data analysis, and to enhance our research we propose to make some state-of-the-art algorithms scalable to high-dimensional domains. In our proposal, we not only improve accuracy of such algorithms, but also efficiency.

1.3 Objective

The main goals we aim in this research are the followings.

- *Deep understanding and study of feature selection.* Creating new feature selection algorithms is a difficult task. However, by studying and analysing the benchmarking algorithm we can make easier this task.

- *Improvement of some of the existing methods.* Some of the current feature selection methods are very accurate, but they can not be applied to high-dimensional data. In this research we propose some modifications to these methods, so that they can be scales to high-dimensional data. On the other hand, some methods are very fast, but we also propose to improve the in terms of accuracy.

- *Evaluation of our proposed algorithms.* To validate our proposals we aim to run experiments in high-dimensional datasets. We use datasets that contains data mainly from microarray cancer classification, text mining and artificial data. Basically, we evaluate three parameters when testing an algorithm: quality of solutions, running time and number of features selected.

1.4 Structure of this research

This paper is compose by five chapters, in which we present our whole research, conclusions and future work. In chapter 2, we analyse and describe some of the state-of-the-art

feature selection algorithms. We divide these algorithms in three groups: methods to rank features, methods based on pairwise evaluations and methods based on set-wise evaluations. Since the methods in the last groups outperforms the other algorithms, we focus on improving some of these algorithms in chapter 3.

In chapter 3, we propose four new feature selection algorithms based on set-wise evaluation. Most of these methods are based on state-of-the-art algorithms that have shown good results. However, these algorithms has some gaps and we propose to solve them by proposing a new version of the algorithms that outperform their original versions. We also perform evaluations to validate each proposal.

In chapter 4, we present new proposals, but in the pair-wise-evaluation approach. We are especially interested in this approach, because some of their algorithms can be used in the two-stage search, which is extremely accurate. However, the two-stage algorithms are not scalable to high-dimensional data. Since a pair-wise evaluation algorithm represents the first phase of a two-stage search, improving the efficiency of the pair-wise evaluation algorithm can drastically increase the efficiency of a two-stage search.

1.5 Notation

During the entire document we use the same notation. Therefore, the following is a list of all notations used in this research.

Table 1.1: Notation used in this research

Symbol	Definition
$\mathcal{D}$	Dataset
x_j	j-th instance in the dataset $\mathcal{D}$
f_i	j-th feature in the dataset $\mathcal{D}$
x_j^i	feature value that corresponds to the j-th instance in the i-th feature in $\mathcal{D}$
C	set of all possible class labels in $\mathcal{D}$
$\tilde{F}$	current selected set of features
μ	a feature selection evaluation function
$\mathbb{F}$	entire feature set in $\mathcal{D}$

Chapter 2

An Overview of Feature Selection and Related Work

Today many modern scientific research fields make use of machine learning techniques for speeding up the process of knowledge discovery and making decisions [1]. These problems often involve a large number of variables that incrementally grow as the time passes due to the improvement of measurement techniques and technology. Many factors affect the success of machine learning on a given task. The representation and quality of the example data is first and foremost. Theoretically, having more features should result in more discriminating power. However, practical experience with machine learning algorithms has shown that this is not always the case [54]. Usually, most of these features are either redundant or irrelevant to the predictive model. In typical predictive modelling tasks like supervised classification, extensively large feature sets can lead to poor accuracy, high computational cost and memory usage, and slow speed. If there is too much irrelevant and redundant information present or the data is noisy and unreliable, then learning during the training phase is more difficult. Therefore, selection of the optimal (possibly minimal) feature set giving best possible results is desirable to increase the discriminating power of predictive models [21].

Feature subset selection is the process of identifying and removing as much irrelevant and redundant information as possible [22]. This reduces the dimensionality of the data and may allow learning algorithms to operate faster and more effectively. In some cases, accuracy on future classification can be improved; in others, the result is a more compact, easily interpreted representation of the target concept.

A feature selection algorithm can be seen as the combination of a search method that generates candidate feature subsets, along with an evaluation function, which assigns a score to the candidate feature subset according to its ability to uniquely determine class labels with high likelihood [19].

In the well known survey by Molina et al. [42], the search strategy is further decom-

posed into search organization and generation of successors, and an evaluation function is referred to as an evaluation measure. The search strategy represents sequences of theoretical and/or heuristic decisions on feature sets to investigate. The evaluation function, on the other hand, is an information-theoretic function used to evaluate the feature sets that the search strategy generates, and the evaluation results are input into the search strategy as feedback.

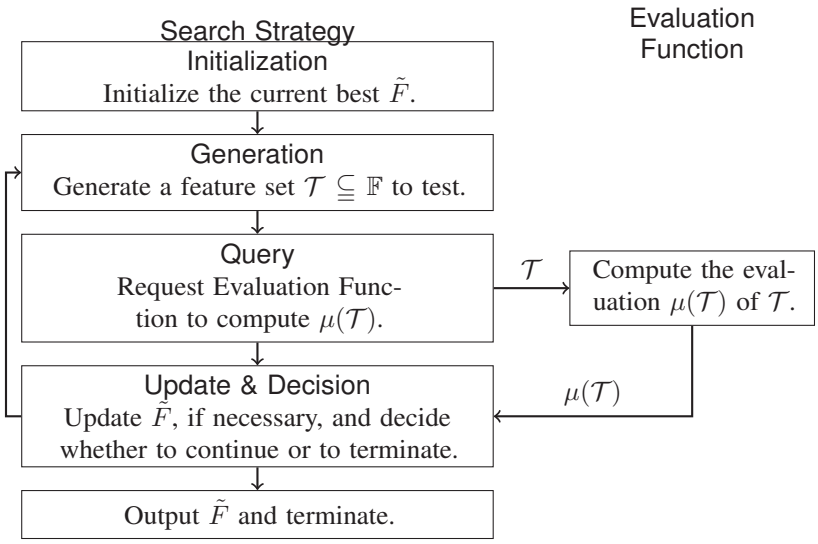


Figure 2.1: The basic framework of feature selection of the filter and wrapper approaches. $\mathbb{F}$ is the entire feature set of a dataset $\mathcal{D}$, and $\tilde{F}$ denotes the current best feature subset.

Figure fig:framework depicts this framework. In Initialization, the current best feature set $\tilde{F}$ is set to an appropriate initial value. For example, we let $\tilde{F} = \emptyset$ for forward selection and $\tilde{F} = \mathbb{F}$ for backward elimination, where $\mathbb{F}$ denotes the entire feature set of the dataset $\mathcal{D}$ input. In Generation, the search strategy generates a feature set $\mathcal{T}$ that is to be investigated and then requests the evaluation function to evaluate $\mathcal{T}$. In Update & Decision, based on $\mu(\mathcal{T})$ returned from the evaluation function, the search strategy updates $\tilde{F}$, if necessary, and decides whether it should continue the search or should terminate it by outputting $\tilde{F}$.

The simplest algorithm, following this general framework, is to test each of the 2^n possible subset of features finding the one which minimizes the error prediction rate. However, this is an exhaustive search of the space, and its computationally cost is prohibitively high. Therefore, alternative search-based techniques have been constantly proposed by the machine learning community.

In large, feature selection includes three approaches, namely, the *embedding approach*, the *wrapper approach* and the *filter approach*. Intuitively speaking, the embedding approach consist of classification algorithms that intrinsically include the feature selection functionality. Decision tree algorithms such as CART [9], ID3 [49] and C4.5 [51] are good examples: Pruning branches corresponds to eliminating irrelevant features.

The wrapper approach relies on a particular classifier algorithm and aims to select feature sets that optimize the performance of the classifier. Wrapper methods generally result in better performance than filter methods because the feature selection process is optimized for the classification algorithm to be used. However, wrapper methods are too expensive for large dimensional database in terms of computational complexity and time since for every feature set to be evaluated, the classifier must be trained and tested on the reduced data [13].

In general, filters are fast due to the fact they do not incorporate learning algorithms and rely on the intrinsic characteristics of the training data to select and discard features. As a consequence, filter methods are generally much faster than wrapper methods, and, as such, are more practical for use on data of high dimensionality. Although in this research we propose two new wrapper algorithms, we are mainly focus on filter algorithms.

2.1 Feature Selection in Supervised Learning

A fundamental issue in supervised classification is to learn the functional relationship $\ell()$ from training instances $X = \{x_1, x_2, \cdots, x_m\}$ with associated correct labels $C = \{c_1, c_2, \ldots, c_t\}$, to correctly determine the class labels for unseen instances [53]. x_j is a vector of real numbers, where $x_j^i \in f_i$ is the value of the i-th feature in the j-th instance. C represents a finite set of possible results associated to a given instance. As an example, x_j might be a vector of values associated to the cells of a tumour biopsy or the cells of the tissue of a healthy patient, whereas C represents whether the patient has cancer or not. Then, the classification algorithm analyses thousands of patients data along with labels containing the correct diagnosis of the patient. The algorithm will then learn a function ℓ that represents the relationship between the patient data and their associated diagnosis. Once ℓ is learnt, new patients without diagnosis can be classified using $\ell(x_p) \rightarrow \mathrm{C}$, where x_p represent the data of the new patient.

In order to solve a given problem of supervised learning, the following steps must be accomplished [53]:

1. *Determine the type of training examples.* Before applying supervised classification algorithms, the researcher should know the type of data disposed. This is especially useful to know whether or not the data need to be preprocessed or even to determine the potential algorithm to use as a learner [26].

2. *Gather a training set.* The input or training set must be representative within the universe of all possible instances of the problem. Hence, a set of input objects is collected along with their corresponding outputs, either from human experts or from measurements.

3. *Determine the input feature representation for the learned function*. The accuracy of the learned function depends strongly on how the input object is represented. Typically, the input object is transformed into a feature vector, which contains a number of features that are descriptive of the object. The number of features should not be too large, because of the curse of dimensionality; but should contain enough information to accurately predict the output.

4. *Determine the structure of the learned function and corresponding learning algorithm*. The learned model can be expressed in several formats such as: decision trees, artificial neuronal networks and list of rules. An approximation of the best model representation can be search through a trial and error process [41].

5. *Complete the design*. Optimizing the parameters of the supervised algorithm via cross validation is essential to reach the highest possible accuracy.

6. *Evaluate the accuracy of the learned function.* After parameter adjustment and learning, the performance of the resulting function should be measured on a test set that is separate from the training set. A wide range of supervised learning algorithms is available, each with its strengths and weaknesses. It is strongly recommended to test several algorithms to choose the one that better fits to the given data. There is no single learning algorithm that works best on all supervised learning problems.

There are plenty of supervised learning algorithms in the literature that mainly differs on the way they represents their learning function. The following algorithms are very representative in the research community and we use many of them to evaluate the proposed feature selection algorithms through cross validation.

- *Naive Bayes:* Bayesian theory is a very simple, but powerful tool in machine learning because hypotheses can be assigned weights based on prior probability. Bayesian methods calculate explicit probabilities for hypotheses. For example, Michie, et al. [36] compared decision tree and neural network methods with a Naive Bayesian classifier found they have some similar features. The Naive Bayes algorithm uses a simplified version of Bayes equation to decide which class a new instance belongs to. The posterior probability of each class is computed, given the feature values present in the instance; the instance is assigned the class with the highest probability. The following equation shows the naive Bayes formula, which makes the assumption that feature values are statistically independent within each class.

$$P(c_t|x_j^1, x_j^2, \ldots, x_j^n) = \frac{P(c_t)\prod_{i=1}^{n} P(x_j^i|c_t)}{P(x_j^1, x_j^2, \ldots, x_j^n)} \tag{2.1}$$

Learning with the Naive Bayes classifier is straightforward and involves simply estimating the probabilities in the right side of the Equation from the training instances. The result is a probabilistic summary for each of the possible classes.

- Support vector machines (SVMs) provide very powerful machine learning algorithms. Unlike regression, an SVM determines a separation hyperplane with a margin so as to maximize the gap between different classes, as illustrated in Fig. 2.2. The SVM divides the dataset into different parts according to data instances called support vectors. The class of a new data instance is determined by the area in which it falls. An SVM in conjunction with a kernel function, which projects data to a higher-dimensional space, can efficiently handle high-dimensional data. An

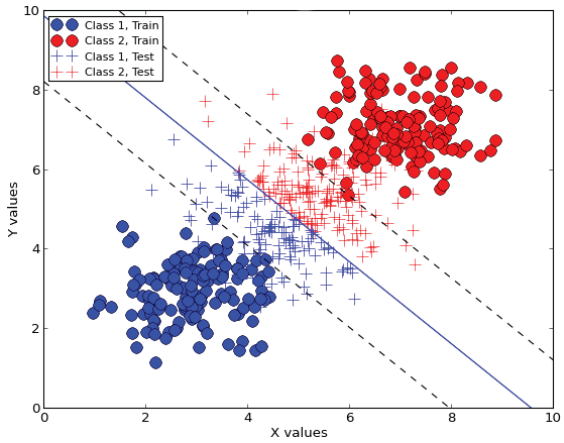


Figure 2.2: Illustration of a support vector machine (SVM)

SVM model is a representation of the examples as points in space, mapped so that the examples of the separate categories are divided by a clear gap that is as wide as possible. New examples are then mapped into that same space and predicted to belong to a category based on which side of the gap they fall.

- *C4.5 Decision Tree* In decision tree learning, data features are compared with decision conditions in order to select a specific category [50]. Along with systems that induce logical rules, decision tree algorithms have proved popular in practice. This is due in part to their robustness and execution speed, and to the fact that explicit concept descriptions are produced, which users can interpret [7]. C4.5 builds decision trees from a set of training data $\{x_1, x_2, \ldots, x_m\}$, using the concept of information entropy. Each instances in the training data has a class value associated from $C = \{c_1, \ldots, c_k\}$. At each node of the tree, C4.5 chooses the feature of the data that most effectively splits its set of samples into subsets enriched in one class

or the other. The splitting criterion is the normalized information gain (IG).

$$IG(f_i; \mathrm{C}) = \sum_{\substack{x_j^i \in f_i, \\ c_t \in C}} P(f_i = x_j^i, \mathrm{C} = c_t) \log \frac{P(f_i = x_j^i, \mathrm{C} = c_t)}{P(f_i = x_j^i) P(\mathrm{C} = c_t)} \tag{2.2}$$

The attribute with the highest normalized information gain is chosen to make the decision.

Recent research has shown that common machine learning algorithms can be adversely affected by irrelevant and redundant features in the training data. As an example, there have been several researches that points out that the simple nearest neighbour algorithm is sensitive to irrelevant features and its accuracy can significantly be improved when noisy feature are removed [31]. The Naive Bayes classifier can be adversely affected by redundant attributes due to its assumption that attributes are independent given the class [35]. Decision tree algorithms such as C4.5 can sometimes overfit training data, resulting in large trees [50]. In many cases, removing irrelevant and redundant information can result in C4.5 producing smaller trees [32].

In the remaining of this chapter, we describe some feature selection algorithms used in this research. Algorithms have been divided in three groups: ranking methods, pair-wise evaluation methods and set-wise evaluation methods.

2.2 Related Work

Researchers have studied various aspects of feature selection. One of the key aspects is to measure the goodness of a feature subset in determining an optimal one [19].

When working with high dimensional data with thousands or hundred thousands features, it is very common that a large number of the features are not informative because they are either irrelevant or redundant with respect to the class variable [66]. However, the search space in the feature selection problem grows exponentially with the increase of dimensionality, as shown in Figure 2.3. In other words, the possible number of solutions is 2^n, where n is the number of features.

Finding the optimal solution is almost impossible in high-dimensional datasets: many problems related to feature selection have been shown to be NP-hard [8]. In the remaining of this section, we describe several feature selection algorithms that fall into one of the categories of: methods to rank features, pairwise evaluation methods or set-wise evaluation methods.

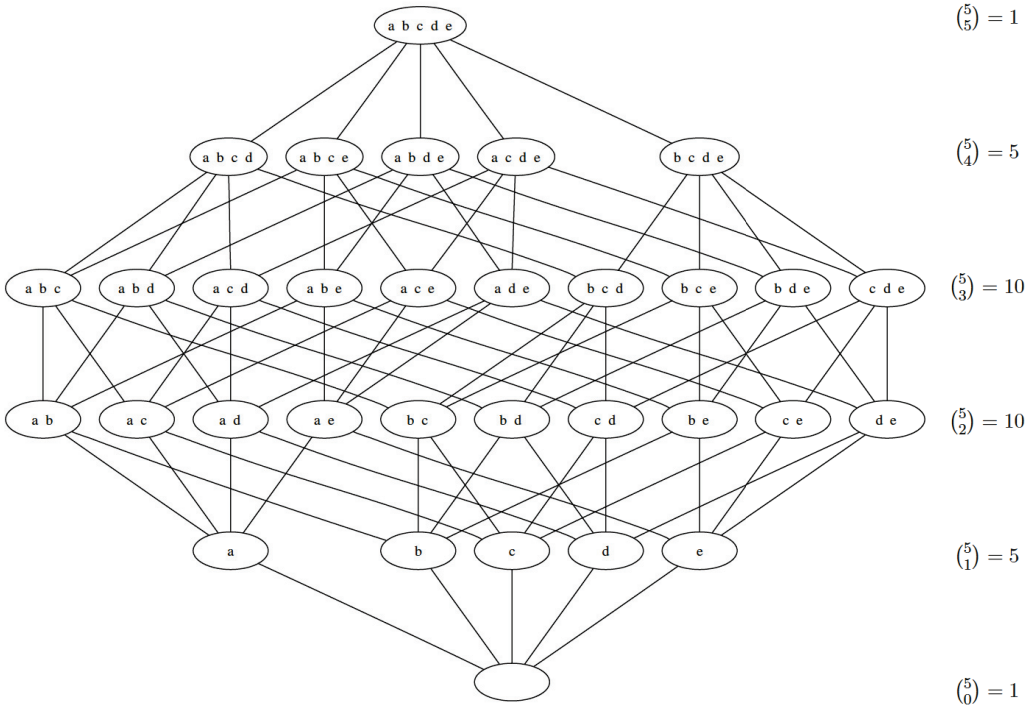


Figure 2.3: Search space for the feature selection problem with five features: a, b, c, d, e.

2.2.1 Methods to Rank Features

The individual relevance score $r(f_i; \mathrm{C})$ of a feature f_i is a common term that refers to the power of a single feature to predict the class feature C. The individual relevance score can be used as a metric to select the features that better predicts the class under certain threshold. That is, features are ranked using their individual relevance score and then the top features are selected. The selection condition can be expressed in number of features to select or in a threshold for r. These algorithms are called feature ranking methods and often use correlation, distance and information measures between a single feature and the class feature to find a set full of high-relevant features.

2.2.1.1 Relief and ReliefF

As an example, RELIEF [29] computes the relevance score of a feature f_i based on the capability of f_i to discriminate among instances of different classes. Assuming instance x_k with class c_+ is randomly sampled from the data, and H_k and M_k are two sets of instances (in the neighborhood of x_k) with class c_+ and c_- respectively, then a feature has high separability power if it has similar values in instances from H_k and different values in instances from M_k. RELIEFF is an extension of RELIEF that handle multiple classes by splitting the data into series of two-class data [33]. The individual relevance of each feature f_i in $\mathbb{F}$ is assessed by computing the average of its separability power in l instances

randomly sampled. That is,

$$RF(f_i;\mathrm{C}) = \frac{1}{|\mathrm{C}|}\sum_{k=1}^{l}(-\frac{1}{|M_k|}\sum_{x_j \in M_k} d(x_k^i, x_j^i) + \sum_{c \neq c(x_k)} \frac{p(c)}{|H_k|(1-P(c))}\sum_{x_j \in H_k} d(x_k^i, x_j^i)),$$

where $P(c)$ is the probability that an instance is labeled with class c and $d(x_k^i, x_j^i) = (x_k^i - x_j^i)/(max(f_i) - min(f_i))$, with $max(f_i)$ and $min(f_i)$ being the maximum and minimum value of feature f_i.

2.2.1.2 Laplacian Score

A similar metric is used by Xiaofei et al. [23] in the Laplacian Score measure. The main difference lying between ReliefF and the Laplacian Score is that the latter does not use information about C, so it is applicable to unsupervised learning as well. Laplacian Score evaluates the quality of a feature f_i according to its agreement with the graph Laplacian matrix. Consider a matrix $\mathbf{W} \in \mathbb{R}^{m \times m}$ that represents the similarity between any pair of instances x_j and x_k such that

$$\mathbf{W}_{jk} = \begin{cases} e^{-\frac{\|x_j - x_k\|}{t}} & \text{if } x_j \text{ and } x_k \text{ are neighbors} \\ 0 & \text{otherwise} \end{cases}, \tag{2.3}$$

where $||x||$ is norm of vector x and the neighbourhood of an instance is defined by a distance function. If we think of matrix $\mathbf{W}$ as a graph of neighbouring instances that are connected by similarity edges, then the Laplacian Score of a feature f_i represents how consistent is f_i with the similarity graph. That is, f_i is consistent with $\mathbf{W}$ if it takes similar values for instances that are near to each other and dissimilar values for instances far from each other. The Laplacian Score metric is computed by

$$LS(f_i; C) = \frac{\sum_{j,k}(x_j^i - x_k^i)^2 \mathbf{W}_{jk}}{\sum_j (x_j^i - \mu_i)^2 \mathbf{A}_{jj}}, \tag{2.4}$$

where $\mu_i = 1/m \sum_j x_j^i$ and $\mathbf{A}$ is a diagonal matrix such that $\mathbf{A}_{ii} = \sum_j \mathbf{W}_{ji}$. This measure seems to be very robust, but according to Zhu et al. in [69] the graph of neighbouring instances is not consistent when the dataset is high-dimensional.

2.2.1.3 Fisher Score

Another popular measure among the ranking feature algorithm is the Fisher Score [18]. Let n_c be the number of instances with class c and let μ_{ic} and σ_{ic}^2 be the mean and variance of the $i-$th value of all instances in the data respectively. The Fisher Score represents the

average of the distances among instances with different classes when the data is projected with feature f_i. Fisher score metric is defined as follows.

$$FS(f_i; C) = \frac{\sum_{c=1}^{|C|} n_c(\mu_{ic} - \mu_i)^2}{\sum_{c=1}^{|C|} n_c \sigma_{ic}^2}. \tag{2.5}$$

2.2.1.4 Information Theory-based Functions

Within the ranking feature selection functions, is the Mutual Information, which can handle categorical features and can be used to measure correlation between a feature and the class:

$$MI(f_i; \mathrm{C}) = \sum_{\substack{x_j^i \in V(f_i), \\ c \in C}} Pr[f_i = x_j^i, \mathrm{C} = c] \log \frac{Pr[f_i = x_j^i, \mathrm{C} = c]}{Pr[f_i = x_j^i] Pr[\mathrm{C} = c]}$$

To compute the mutual information, we use the empirical probability derived from the dataset D: the empirical probability $Pr[f_i = v]$ is the ratio of the number of the instances whose feature value with respect to f_i is identical to v to the total number of instances m and is given by

$$Pr(f_i = v) = \frac{1}{m} \left| \{ j \mid x_j^i = v \} \right|. \tag{2.6}$$

The symmetrical uncertainty $SU(S, C)$ [48], on the other hand, is the harmonic mean between $MI(S; C)/H(S)$ and $MI(S; C)/H(C)$, and a formula to compute it is given as

$$SU(S; C) = \frac{2 \cdot MI(S, C)}{H(S) + H(C)}. \tag{2.7}$$

Mutual Information is biased in favour of features with greater number of values and this is a problem when used for feature selection [67]. The Symmetrical Uncertainty measure deals with this problem by a normalizing function:

$$SU(f_i; C) = 2 \frac{MI(f_i; C)}{H(f_i) + H(C)}$$

The Symmetrical Uncertainty is the harmonic mean between $MI(f_i, C)/H(f_i)$ and $MI(f_i, C)/H(C)$. therefore it is symmetrical and in the range of $[0, 1]$.

2.2.1.5 Support-Vector-Machine-based Recursive Feature Elimination

The classification error-based measures use useful information discovered during the training phase of a classifier to weight the features. As an instance, the Recursive Feature Elimination algorithm evaluates a feature f_i by computing the added error when f_i is

removed from the current set [19].

$$RFE(f_i; C) = \Big(\sum_k \alpha_k \; c(x_k) \; x_k^i \Big)^2 \tag{2.8}$$

where $c(x_k) = \{+1, -1\}$ returns the class corresponding to the instance x_k and α_k is the optimal weight, which can be computed with a linear discriminatory classifier such as SVM by:

$$\min_{\alpha} \frac{1}{2} \sum_{j,k} C(x_j) C(x_k) \alpha_j \alpha_k (x_j x_k + \lambda \delta_{jk}) - \sum_k \alpha_k \tag{2.9}$$

$$\text{s.t } 0 \leq \delta_k \leq \zeta, \sum \delta_k C(x_k) = 0, \tag{2.10}$$

being λ and ζ soft margin parameters (usually fixed to $\lambda = 10^{-4}$ and $\zeta = 10^2$ [19]) and δ_{jk} is the Kronecker function ($\delta_{jk} = 1$ if $j = k$ and $\delta_{jk} = 0$ otherwise). Different from most of feature ranking algorithms, in the Recursive Feature Elimination approach a greedy search is performed to add at the end of the ranking the feature that minimize RFE. Although this atypical way of building a ranking leads to a relatively high computational complexity, the quality of the output is high [19].

Although the ranking feature algorithms are usually simple and fast, they have two serious drawbacks that may affect the performance of supervised classifiers. First, redundant features are likely to be selected. Second, they usually can not detect interacting features.

2.2.2 Methods based on Pairwise Evaluation of Relevance

Oppositely to the feature ranking algorithms, pairwise evaluation methods can detect and eliminate relevant features, but also are able to remove redundant features. Most of these algorithms use one of the measures mentioned in the section above. The way most of these algorithms operates is as follows. First, the relevance score $r(f_i, C)$ of each feature in $f_i \in F$ is computed and second, pairwise evaluations $r(f_i, f_j)$ between features are performed to detect features that are highly correlated to others.

2.2.2.1 Fast Correlation based Filter

As an example, the algorithm FCBF (*Fast Corelator based-Filter*) [67] first ranks all features $\{f_1, \ldots, f_n\}$ in the descending order of the Symmetrical Uncertainty scores. Then, starting from the best/first feature in the ranking f_1, it applies a redundancy filter to all of features f_j with $j > i$, and, if $SU(f_i; f_j) > SU(f_j; C)$ holds then it removes f_j. Since the overall complexity of algorithm FCBF is $O(mn \log n)$ where m is the number of instances in the data, this algorithm is scalable to large data.

Algorithm 1 Fcbf [67]

Require: Dataset D described by a feature set $\mathbb{F}$
Ensure: A feature subset $\{\bar{f}_1, \ldots, \bar{f}_q\} \subset \mathbb{F}$.

```
Rank features in F according to SU(f_i, C)
for i = 1, ..., |F| − 1 do
    for j = i + 1, ..., |F| do
        if SU(f_j) < SU(f_i, f_j) then
            F = F \ {f_j}
        end if
    end for
end for
return F
```

2.2.2.2 Correlation-based Feature Selection

CFS is one of the most well-known feature selection algorithms that take advantage of a redundancy filter [22]. The CFS function $Cfs : \mathbb{F}_D \rightarrow \mathbb{R}$ takes an element of $\mathbb{F}_D$, the power set of the entire features of a dataset D, as input. Therefore, an input into the CFS function is a subset of the entire features. On the other hand, the returned real value is the result of evaluation of S from the *class-relevance* and *interior-redundancy* points of view. This design is based on the idea of "*a good feature subset contains features highly correlated with the class variable C, yet uncorrelated to each other*" [22]. Every set $\tilde{F}$ is heuristically evaluated as follows:

$$Cfs(\tilde{F}, C) = \frac{|\tilde{F}|\ \overline{r_{cf}}}{\sqrt{|\tilde{F}| + |\tilde{F}|(|\tilde{F}| - 1)\ \overline{r_{ff}}}} \tag{2.11}$$

, where $\overline{r_{cf}}$ represents the average of the relevance score of each feature in $\tilde{F}$ and $\overline{r_{ff}}$ is the average of the redundancy score of all possible pair of features in $\tilde{F}$. The time complexity of this algorithm is quadratic in terms of number of features. As depicted in Algorithm 2, CFS use the *Greedy Forward* approach as a search engine. Therefore, CFS is not recommended for high-dimensional data.

As Algorithm 2 shows, the greedy forward search initializes S to be with the empty ($S = \emptyset$). In each iteration, every feature f that is not in S is evaluated on the basis of the extent to which the CFS score is improved by adding f to S. The feature that maximizes the CFS score is actually added to S, and the algorithm proceeds to the next iteration. The search stops when no features improve the current CFS score, or the number of iteration exceeds the given threshold n. The feature that is selected in the first iteration has the maximum SU score, because $Cfs(\{f\}) = SU(f; C)$ holds.

Algorithm 2 CFS [22]

Require: Dataset D described by a feature set $\mathbb{F}$
Ensure: A feature subset $S \subset \mathbb{F}$.
1: $S = \emptyset$
2: **for** $k = 1, \ldots, n$ **do**
3: $f_i \in \text{argmax}_{f \in F \setminus S} Cfs(S \cup \{f\})$
4: **if** $Cfs(S \cup \{f_i\}) \leq Cfs(S)$ **then**
5: **break**
6: **end if**
7: $S = S \cup \{f_i\}$
8: **end for**
9: **return** S

2.2.2.3 Maximum-Relevance Minimum-Redundancy

As was stated before, the main goal of feature selection is to identify features 1) that have high correlation with the target class (relevance) but 2) low mutual relevance among them (redundancy). Peng et al. [45] have proposed the algorithm named the *Max-Relevance and Min-Redundancy* algorithm (MRMR), which finds approximate solutions to the aforementioned problem efficiently. MRMR evaluates each subset of genes by the *Mutual Information Difference* measure $\text{MID}_\alpha(\cdot, \cdot)$ defined as shown below:

$$\text{MID}_\alpha(f, \emptyset) = I(f, C); \tag{2.12}$$

$$\text{MID}_\alpha(f, S) = I(f, C) - \frac{2\alpha}{k} \sum_{f' \in S} I(f, f'), \tag{2.13}$$

where $I(f, f')$ represents the Mutual Information between the two genes f and f'. MRMR takes the forward search approach, and hence, the variable S that holds the features selected at each iteration of the for loop (line 2 – 5) is initialized to the empty set (line 1). Then, for each iteration of the for loop, a single feature f that maximizes $\text{MID}_\alpha(f, S)$ is added to S.

Algorithm 3 MRMR [45]

Require: Dataset D described by a feature set $\mathbb{F}$ and a number q of features to select.
Ensure: A feature subset $\{\bar{f}_1, \ldots, \bar{f}_q\} \subset \mathbb{F}$.
1: $S = \emptyset$
2: **for** $k = 1, \ldots, q$ **do**
3: $\bar{f}_k \in \text{argmax}\{\text{MID}_\alpha(f, S) \mid f \in \mathbb{F} \setminus S\}$
4: Add $\bar{f}_k$ to S.
5: **end for**
6: **return** S

The *Minimum Relevance Maximum Relevance* (MRMR) algorithm uses a very similar process to select feature sets [15]. In each iteration the feature $f^* \in F \setminus \tilde{F}$ that optimize

certain evaluation function is selected. Again, the evaluation function corresponds to a balance between the averages of the relevance score and redundancy score of the set of already selected features $\tilde{F}$:

$$f^* = \underset{f_i \in F \setminus \tilde{F}}{\operatorname{argmax}} \left\{ \frac{MI(f_i, c)}{\frac{1}{|\tilde{F}|} \sum_{f_j \in \tilde{F}} MI(f_i, f_j)} \right\} \tag{2.14}$$

2.2.2.4 Sequential Forward Selection-based Validity Index

The SFS-LW *algorithm* evaluates the quality of a feature set by measuring the minimum separation degree between two linearly separable classes in the data. The separation degree between two classes: i and j, is assessed as equation 2.15 shows.

$$FD_{i|j} = d(v_i, v_j) - (r_i + r_j), \tag{2.15}$$

where $d(v_i, v_j)$ represents the distance between the centroids v_i and v_j, and r_i and r_j are radii of the clustered instances in class i and class j respectively. A feature subset S is evaluated by computing the minimum separation degrees between all pair of classes as shown in the following equation.

$$LW_S = \frac{1}{|C|} \sum_{i=1}^{|C|} \min_{i \neq j, j=1,\ldots,|C|} FD_{i|j} \tag{2.16}$$

Figure 2.4 represents the LW_S index graphically.

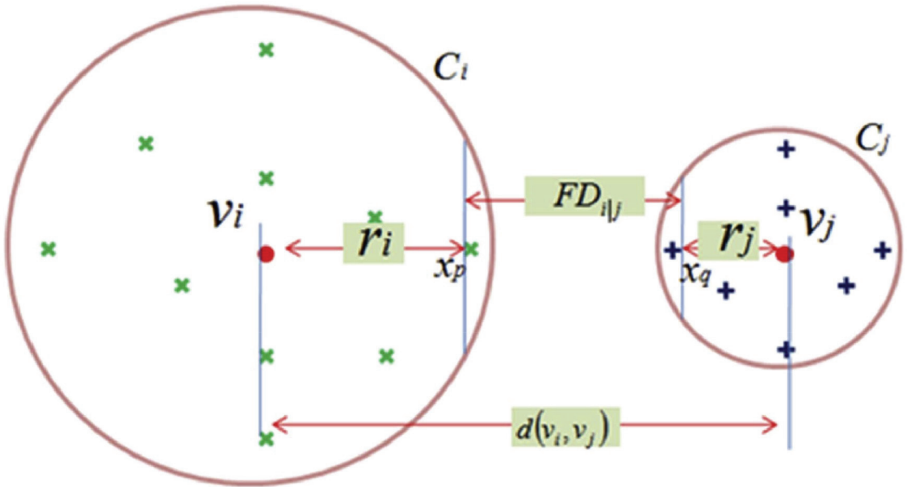


Figure 2.4: Graphical representation of the LW_S index.

The LW-index is used as an evaluation function to evaluate the candidate sets and is combined with the Sequential Forward Search.

2.2.2.5 Supervised Simplified Silhouette Filter

The S^3F algorithm is based on the feature clustering concept [11]. That is, the set of features $\mathbb{F} = \{f_1, f_2, \ldots, f_n\}$ are partitioned into a collection $C^{\mathbb{F}} = \{C_1, C_2, \ldots, C_k\}$ of k mutually disjoint subset of correlated features C_i of $\mathbb{F}$. Features belonging to the same cluster are expected to be highly correlated whereas features in different clusters are expected to have low correlation between them. To build the clusters, S^{3}F uses the k-medoids algorithm [55]. In addition, the distance between features in a cluster is assessed taking into account the correlation between the features and their individual correlation with the class variable, as shown in the following equation.

$$SU_S(f_i, f_j) = \frac{1 - SU(f_i, f_j) + |SU(f_i, C) - SU(f_j, C)|}{2}. \tag{2.17}$$

The medoid (or centroid) of each cluster is the feature more correlated with the class and least correlated with the other features in the cluster. In S^{3}F, the medoid is heuristically determined as follows.

$$\eta_r = \underset{f_i \in Cr}{\operatorname{argmax}} \left\{ \frac{1}{2} \left[\frac{\sum_{f_j \in C_r} SU(f_i, f_j)}{|C_r| - 1} + SU(f_i, C) \right] \right\}. \tag{2.18}$$

Finally, two features are selected from each cluster: the medoid, and the one determined by:

$$\underset{f_i \in Cr}{\operatorname{argmax}} \left\{ \frac{1 - SU(f_i, \eta_r) + SU(f_i, C)}{2} \right\}, \tag{2.19}$$

which is the feature least correlated with the medoid η_r and the most correlated with the class.

Although feature ranking and pair-wise evaluation methods are quite fast and easy to implement, they are not able to detect interacting features. That's why in high-dimensional domains they may output low-quality sets.

To illustrate, consider the class target function $c = f_1 \oplus f_2$ where $\{f_1, f_2, \ldots, f_n\} \in \mathbb{F}$ are binary features and $\oplus$ denotes the *xor* operator. Beforehand, we know $\{f_1, f_2\}$ won't be selected because both features by themselves are uncorrelated with c. If we consider that features in $\mathbb{F} \setminus \{f_1, f_2\}$ can not accurately describe the class then we can not expect a good performance of the classifier after reducing $\mathbb{F}$ by any of the feature ranking or pair-wise evaluation algorithms. Figure 2.5 depicts a numerical version of the aforementioned example. Consistency-based measures are a successful choice to face this problem because they can detect high-order interacting features [68].

According to [27] a feature f_i interacts with a set of features $\tilde{\mathcal{F}}$ when is considered irrelevant based on its individual correlation with the class; but when combined with $\tilde{\mathcal{F}}$, it becomes very relevant. Formally we can say that: f_i interacts with $\tilde{\mathcal{F}}$ if $\mathfrak{Br}(\tilde{\mathcal{F}} \cup \{f_i\}) \leq$

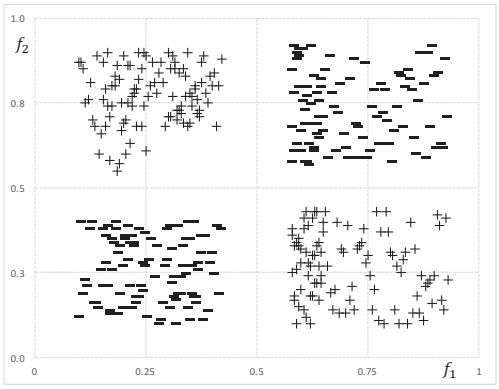


Figure 2.5: Example of how non-relevant features can interact with each other to accurately discriminate between two classes.

$\delta \; and \; \mathfrak{Br}(\tilde{\mathcal{F}}) > \delta$.

2.2.3 Methods based on Set-wise Evaluation of Relevance

Consistency-based algorithms can detect interacting features by collectively evaluating relevance (correlation) of a feature set to the class. Although exhaustive search of all possible feature sets is computationally too expensive, the result can be expected to be accurate. We first introduce the *Bayesian risk* as a consistency measure example and then we define the consistency measure concept. To illustrate, for a dataset , we view a feature of as a random variable and a feature set $\tilde{F}$ as a joint variable. Then, we let $\Omega_{\tilde{F}}$ denote the sample space of $\tilde{F}$, C denotes a variable that describes classes and $\Pr$ denotes the empirical probability distribution of . With these notations, the Bayesian risk is defined by

$$\mathfrak{Br}(\tilde{F}) = 1 - \sum_{x \in \Omega_{\tilde{F}}} \max\{ \Pr_{\mathbf{D}}[\tilde{F} = x, \mathrm{C} = y] \mid y \in \Omega_{\mathrm{C}}\}.$$

This function is also referred to as the *inconsistency rate* in [68]. The Bayesian risk has two important properties, that is, *determinacy* and *monotonicity*, and we first introduce the notion of *consistent feature sets* to explain the properties. For a dataset described by $\mathbb{F}$, a feature set $\tilde{F} \subseteqq F$ is *consistent*, iff, $Pr[\mathrm{C} = y \mid \tilde{F} = x] = 0$ or 1 for all $x \in \Omega_{\tilde{F}}$ and $y \in \Omega_{\mathrm{C}}$.

Then, the determinacy and monotonicity properties are described as follows.

Determinacy. $\mathfrak{Br}(\tilde{F}) = 0$, if, and only if, $\tilde{F}$ is consistent in .

Monotonicity. $\mathfrak{Br}(\tilde{F}) \geq \mathfrak{Br}(G)$, if $\tilde{F} \subseteqq G \subseteqq \mathbb{F}$.

Formally, a consistency measure is defined as a function that returns real numbers on input of feature sets that has the determinacy and monotonicity properties. The *consistency-*

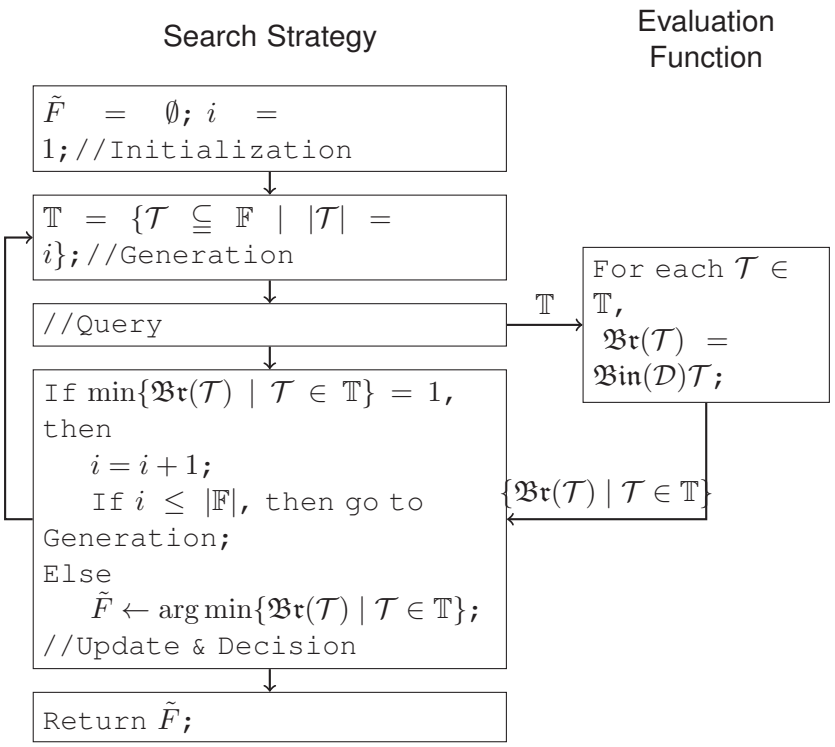


Figure 2.6: FOCUS.

based feature selection, on the other hand, is characterized by use of consistency measures as the evaluation function.

FOCUS [3] is a feature selection algorithm that searches the minimum feature subsets that are consistent in (Fig. 2.6).

FOCUS include two serious problems. One is that the a dataset does not always include consistent feature subsets, and hence, FOCUS cannot find answers in such cases.

The significant difference of the Bayesian risk from the binary consistency function is the property of the Bayesian risk that we can not only determine whether feature sets are consistent but also measure their closeness to the state of being consistent.

The other important problem of FOCUS is its impractically low time-efficiency. Since FOCUS performs exhaustive search, its search space includes $2^{|\mathbb{F}|}$ feature subsets. There have been many attempts to improve the efficiency. *Automatic Branch & Bounds* [43] takes advantage of the monotonicity property of the Bayesian risk to prune unnecessary branches of the search trees. It actually narrows down the search space of FOCUS but is still slow because it performs complete search. Then, heuristic methods play an important role. For example, SETCOVER [12] was the first algorithm the leveraged *sequential forward selection* (*SFS*). The probabilistic and hybrid methods are also useful, and *Las Vegas Filter* [39] and *Quick Branch & Bounds* [30] were good examples.

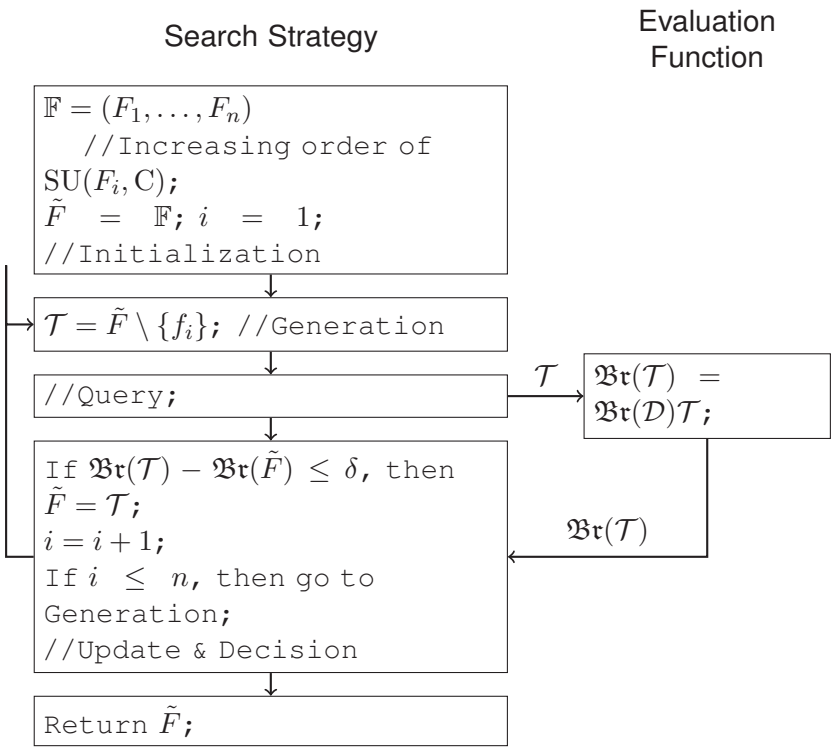


Figure 2.7: Interact. A threshold δ is given as a parameter.

2.2.3.1 Interact

In this regard, INTERACT [68] was an important breakthrough. It investigates only $|\mathbb{F}|$ feature subsets, nevertheless it can exhibit high accuracy by incorporating interaction among features into the evaluation. Fig. 2.7 shows the algorithm of INTERACT: INTERACT receives a dataset that is described by a feature set $\mathbb{F}$ and a threshold δ; In the Initialization step, INTERACT sorts the features in $\mathbb{F}$ into $(f_1, \ldots, f_{|\mathbb{F}|})$ in the increasing order of the symmetric uncertainty $\mathrm{SU}(F, \mathrm{C})$ between each $F \in \mathbb{F}$ and the feature C that represents class labels; Since INTERACT is a *backward elimination* algorithm, the initial value of $\tilde{F}$ is $\mathbb{F}$; Starting from $i = 1$, INTERACT lets $\mathcal{T} = \tilde{F} \setminus \{f_i\}$ and computes $\mathfrak{Br}(\mathcal{T}) - \mathfrak{Br}(\tilde{F})$, which is non-negative by the monotonicity property of the evaluation function; If $\mathfrak{Br}(\mathcal{T}) - \mathfrak{Br}(\tilde{F}) \leq \delta$, INTERACT judges that the feature f_i is not important and eliminates it from $\tilde{F}$; INTERACT repeats the steps of Generation, Query and Update & Decision until no more feature is left untested.

2.2.3.2 Linear-Consistency-Constrained

As was stated in [58], INTERACT usually tends to output $\tilde{\mathcal{F}}$ with high $\mathfrak{Br}(\tilde{\mathcal{F}}); C$ because each set is not evaluated as a whole but each feature is individually evaluated based on its consistency contribution to the current set. Supposing we have a ranked set of features $\mathcal{F} = \{f_1, f_2, \ldots, f_n\}$, such that $SU(f_i) < SU(f_{i+1})$ with $i = \{1, 2, \ldots, n-1\}$, and in

Algorithm:	INTERACT
INPUT:	A consistency measure function $\mathfrak{Br}()$,
	a threshold δ
OUTPUT:	a feature subset $\tilde{\mathcal{F}}$
STEPS:	
	Let $\tilde{\mathcal{F}} = \mathcal{F}$
	Order the features $f \in \tilde{\mathcal{F}}$ in incremental order of $SU\,(f, C)$
	For each $f \in \tilde{\mathcal{F}}$ from the first to the end.
	If $CC\left(f, \tilde{\mathcal{F}}\right) \leq \delta$, let $\tilde{\mathcal{F}} = \tilde{\mathcal{F}} \setminus \{f\}$
	End For.

Figure 2.8: The algorithm of INTERACT

Algorithm:	Linear CC (LCC)
INPUT:	A consistency measure function $\mathfrak{Br}()$, an ordered feature set $\mathcal{F}$
	a threshold δ
OUTPUT:	a minimal subset $\tilde{\mathcal{F}} \subseteq \mathcal{F}$ such that $\mathfrak{Br}(\tilde{\mathcal{F}}; C) \leq \delta$
STEPS:	
	Let $\tilde{\mathcal{F}} = \mathcal{F}$
	If $\mathfrak{Br}(\tilde{\mathcal{F}}; C) > \delta$, abort.
	For each $f \in \tilde{\mathcal{F}}$ from the first to the end.
	If $\mathfrak{Br}(\tilde{\mathcal{F}} \setminus \{f\}; C) \leq \delta$, let $\tilde{\mathcal{F}} = \tilde{\mathcal{F}} \setminus \{f\}$
	End For.

Figure 2.9: The algorithm of LCC

the worst case $CC(f_1; \mathcal{F}) = \delta$, $CC(f_2; \mathcal{F} \setminus \{f_1\}) = \delta$, ..., $CC(f_j; \mathcal{F} \setminus \{f_1, \ldots, f_{j-1}\}) = \delta$, with $j \leq n$; $\mathfrak{Br}(\tilde{\mathcal{F}}; C)$ of the output $\tilde{\mathcal{F}}$ will be equal or higher than $j * \delta$.

As a solution to this drawback, Shin and Xu propose to use $\mathfrak{Br}(\tilde{\mathcal{F}} \setminus \{f\}) \leq \delta$ instead of $CC(f; \tilde{\mathcal{F}})$ to decide whether or not f must be removed [58]. This guaranties that $\mathfrak{Br}(\tilde{\mathcal{F}}; C)$, for the output $\tilde{\mathcal{F}}$, be equal or smaller than δ. In short, the main difference these algorithms have is that LCC evaluates the quality of a set when a feature is removed whereas INTERACT evaluates the quality of a feature based on its consistency contribution to the current set. In other words, the former focuses on the set generated in each iteration while the latter focusses on the quality of each feature individually. In Figure 2.9 the algorithm LCC is shown.

Despite LCC reaches good outputs, is usually trapped by local optima due to the nature of its search. In fact, supposing each feature $f_i \in \mathcal{F}$ is ranked in incremental order, this algorithm outputs from all the sets $\mathcal{H}_i \subseteq \mathcal{F}$, such that $\mathfrak{Br}()(\mathcal{H}_i; C) \leq \delta$, that one which has its least relevant feature, according to *Symmetrical Uncertainty* measure, nearest to the first position in the ranking.

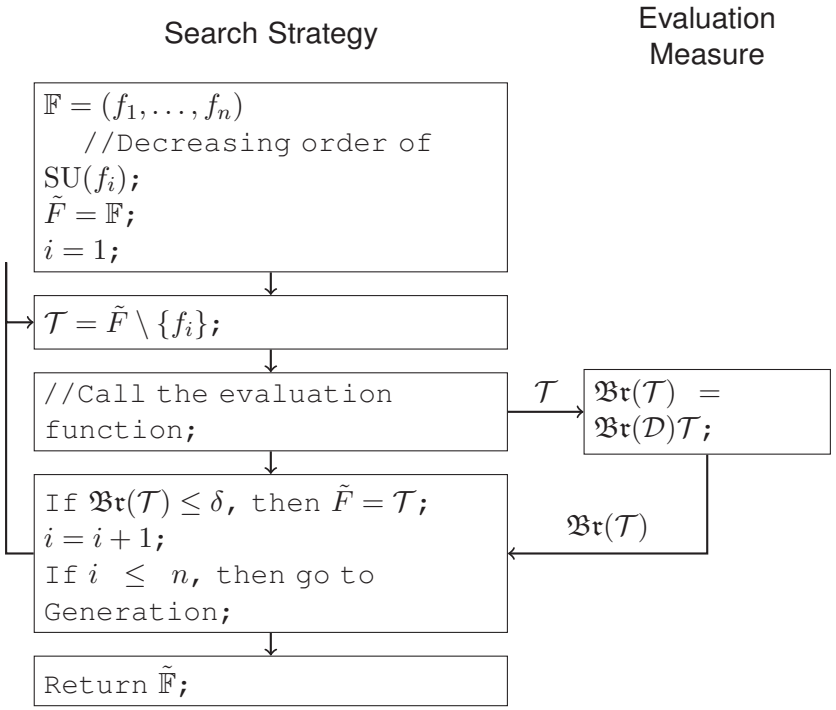


Figure 2.10: Linear Consistency Constrained (LCC) Algorithm

Formally, LCC exactly outputs $\tilde{\mathcal{F}}$ such that:

$$\tilde{\mathcal{F}} = \left\{ \mathcal{H}_k \mid \underset{i=1,..,k,k+1,..}{\mathcal{H}_i \in \mathcal{F}} \;,\; \mu\left(\mathcal{H}_i; C\right) \leq \delta \; and \; \max_{F_j \in \mathcal{H}_i} \left\{ \min \left\{ SU\left(f_j\right) \right\} \right\} \ni \mathcal{H}_k \right\}$$

Unfortunately, most of the time $\tilde{\mathcal{F}}$ is not the global optima.

2.2.3.3 Steepest-Descent-Consistency-Constrained

The *Steepest-Descent Consistency-Constrained* (SDCC) algorithm is less prone to be trapped by local optima [59]. SDCC sets the set $\tilde{\mathcal{F}}$ to the full set of features $\mathcal{F}$ and in each iteration removes a single feature f from $\tilde{\mathcal{F}}$ while $\mathfrak{Br}(\tilde{\mathcal{F}} \setminus \{f\}; C) \leq \delta$. The last $\tilde{\mathcal{F}}$ is the output of SDCC and δ is the minimum allowable inconsistency for this output. In each iteration SDCC selects f to be eliminated by

$$f = \arg\min_{f \in \tilde{\mathcal{F}}} \mathfrak{Br}(\tilde{\mathcal{F}} \setminus \{f\}; C).$$

SDCC needs $\left(|\mathcal{F}| + |\tilde{\mathcal{F}}|\right)\left(|\mathcal{F}| - |\tilde{\mathcal{F}}| + 1\right)/2$ evaluations to remove $|\mathcal{F}| - |\tilde{\mathcal{F}}|$ subsets and output $\tilde{\mathcal{F}}$. In Figure 2.11 the algorithm is presented.

Algorithm:	Steepest-Descent CC (SDCC)
INPUT:	A consistency measure function $\mathfrak{Br}()$, an ordered feature set $\mathcal{F}$ a threshold δ
OUTPUT:	a minimal subset $\tilde{\mathcal{F}} \subseteq \mathcal{F}$ such that $\mathfrak{Br}(\tilde{\mathcal{F}}; C) \leq \delta$
STEPS:	
	Let $\tilde{\mathcal{F}} = \mathcal{F}$
	If $\mathfrak{Br}(\tilde{\mathcal{F}}; C) > \delta$, abort.
	Repeat
	Take $f \in \tilde{\mathcal{F}}$ with the smallest
	$\delta' = \mathfrak{Br}(\tilde{\mathcal{F}} \setminus f; C)$
	If $\delta' \leq \delta$, let $\tilde{\mathcal{F}} = \tilde{\mathcal{F}} \setminus \{f\}$, else break.
	End Repeat.

Figure 2.11: The algorithm of SDCC

Figure 2.12: Example of the search strategy used by SDCC

2.2.3.4 Super-Lcc

Recently, the efficiency of LCC has been improved by conducting binary search instead of linear search [56]. This idea was materialized under the name of SUPER-LCC and works under the assumption that high-dimensional datasets are abundant in irrelevant features that can be removed in mass. By the first to the $(i-1)$-th iterations of the algorithm, the algorithm determines a sequence of indices of features $l_1 < l_2 < \cdots < l_{i-1}$, and defines $\tilde{F} = F \setminus \{f_1, \ldots, f_{l_{i-1}}\} \cup \{f_{l_1}, \ldots, f_{l_{i-1}}\}$. In the i-th iteration, the algorithm finds l_i such that

$$l_i = \underset{j=l_{i-1}+1,\ldots,n}{\operatorname{argmax}} \{\mathfrak{Br}(\tilde{F} \setminus \{f_{l_{i-1}+1}, \ldots, f_j\}) \leq \delta\}.$$

by *Binary Search* due to the monotonicity property of the bayesian risk. SUPER-LCC outputs the same set as LCC but on average has a computational complexity of $O(nm(\log n + \log m))$ where n is the number of features that describes the m instances in D. To the best of our knowledge, SUPER-LCC is the algorithm with better practical performance in both of efficiency and accuracy. According to their authors, for data with more than hundred thousand features, SUPER-LCC needs some seconds to give a response in an ordinary personal computer [56].

2.2.4 Methods with Two-stage Search

A two-stage feature selection algorithm separates its job into two stages that aim at (1) gradual reduction of number of features using a fast algorithm and (2) final and finer

selection of features using a slow but powerful algorithm. The first stage narrows down the search space so that even the slow algorithm at the second stage can find answers within a reasonable time allowance. Usually, a filter-type algorithm is selected for the first stage algorithm, and in particular, MRMR has been extensively used as in the literature because of its efficiency and accuracy [4][25][5][17][16].

2.2.4.1 Genetic Bee Colony for Feature Selection

GBC is a novel hybrid meta-heuristic algorithm that takes advantage of two bio-inspired methods: genetic algorithms (GA) and artificial bee colony (ABC) optimization algorithm. As Figure 2.13 shows, GBC is composed by five phases. In the *Preprocessing* phase, the grand majority of features are removed by the filter-based MRMR algorithm. Afterwards, the first SN candidate solutions are randomly generated in the *Initialization* phase similarily to the initialization phase in the ABC meta-heuristic algorithms [37].

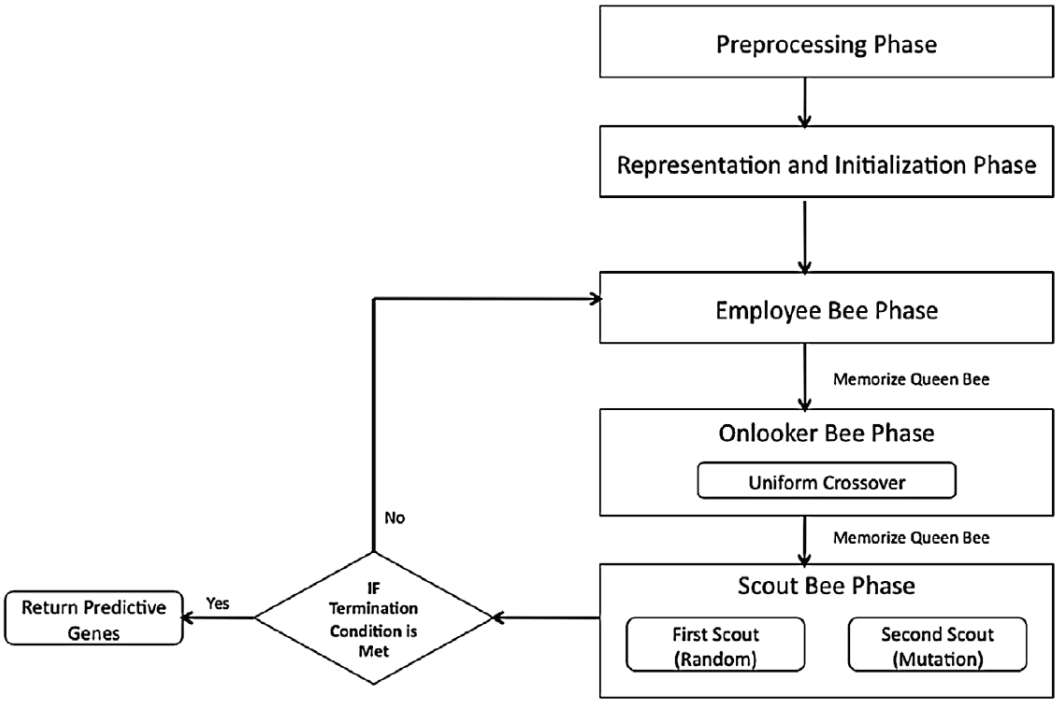


Figure 2.13: The main phases of the GBC algorithm.

In the *Employee Bee* phase the genetic crossover operation is performed between the *Queen Bee*, which is the best solution found so far, and solutions randomly chosen from the population to generate new diverse solutions closer to the optima. Subsequently, the *Scout Bee* phase is accomplished by resetting the solutions trapped by local optima. Also, in this phase, the genetic mutation operation is performed over the *Queen Bee* to intensify the search. In the remainder of this section we briefly explain the different phases aforementioned.

- *Phase 1: The preprocessing phase* In high-dimensional microarray datasets, with hundred of thousands of genes, it is infeasible to apply evolutionary algorithms such as GA and ABC. Therefore, GBC takes advantage of the filter-based algorithm MRMR to remove irrelevant and redundant genes at the very beginning of the search to narrow down the space of solution from 2^n to 2^{qt} subsets, with $qt \ll n$.

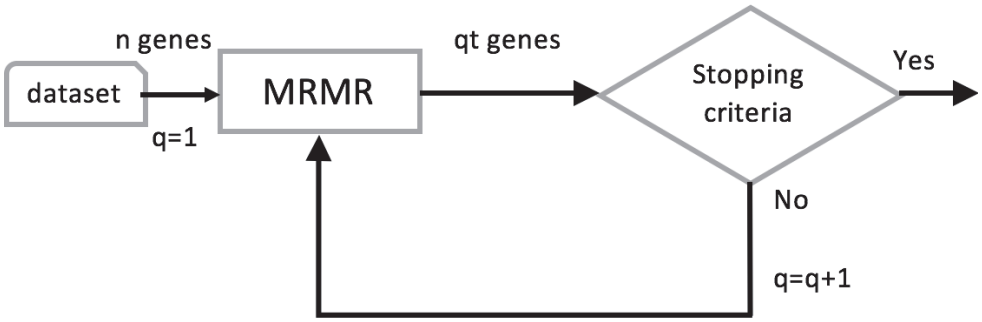


Figure 2.14: The preprocessing phase in the GBC algorithm. t is usually fixed to 50 genes.

As shown in Figure 2.14, MRMR is run several times until the stopping criteria is met. In each run, MRMR returns the subset selected in the previous run plus q additional genes. The preprocessing phase stops when the returned subset can uniquely determine the class variable. That is $SVM(G) = 1.0$, being $SVM(G)$ the accuracy reached by the classifier SVM in the reduced dataset composed by the current qt genes in G.

- *Phase 2: The Initialization phase* In the second phase, GBC generates the initial population composed by SN solutions. Each solution is represented as a group of genes indices that are selected from the subset G, returned by the MRMR algorithm. To build a solution a linear forward selection search is performed. That is, a gene is randomly selected from G, then is tested in the current solution. If the current solution improves by adding the gene, then we continue adding genes while the current solution improves. The solution is built when a gene does not improve it. The i-th gene in a solution is randomly selected according to the following equation:

$$x_j^i = rand(0,1) \times qt, \tag{2.20}$$

where $rand(0,1)$ represents a random number generator in the range of [0,1) with a normal distribution.

- *Phase 3: The employee bee phase* In the *Artificial Bee Colony* (ABC) population-based algorithm, the colony consists of three group of bees: employee bees, onlooker bees and scout bees [28]. The position of a food source represents a potential solution while the amount of nectar in a food source corresponds to the accuracy of

the associated solution. The ABC optimization problem consists on finding the food source with higher amount of nectar through the social cooperation of bees.

GBC algorithm uses this analogy to improve the current population of solutions. GBC sends the employee bees to search in the neighbour of the current SN solutions (food sources) to find solutions that may be closer to the global optima. A neighbour solution is determined by changing the index genes of a current solution by the following equation:

$$v_j^i = \begin{cases} x_j^i + K & \text{if } |S| - K > x_j^i \\ x_j^i - K & \text{otherwise,} \end{cases} \tag{2.21}$$

with $K = rand(-1, 1) \times (x_j^i - x_j^k)$, where $rand(-1, 1)$ denotes a random real number in the range of $[-1, 1]$ and k is a random integer number in $[0, SN - 1]$.

- *Phase 4: The onlooker bee phase* In GBC, the crossover operation is used to share information between employee and onlooker bees in the optimization search space (hive). The employee bees indicates their location of the food sources to the onlooker bees, by a waggling movement.

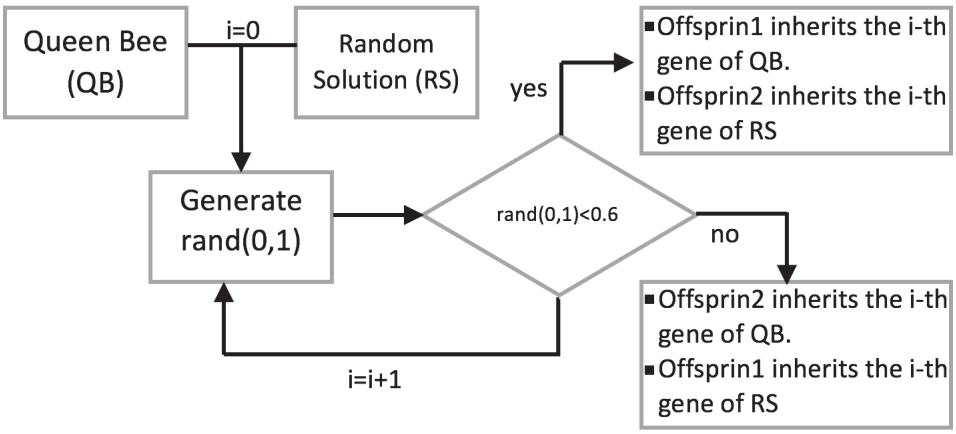


Figure 2.15: Crossover operation between the *Queen Bee* and a solution randomly selected from the population.

As Figure 2.15 depicts, the crossover operation is accomplished by the *Queen Bee*, which is the best solution found so far, and a solution randomly selected from the current population of bees. The probability a solution has to be selected depends only on its accuracy, and can be computed as follows:

$$P(x_j) = \frac{SVM(x_j)}{\sum_{k=1}^{SN} SVM(x_k)}. \tag{2.22}$$

Uniform crossover works by treating each gene independently and making a random choice as to which parent it should be inherit, as shown in Figure 2.15.

- *Phase 5: The scout bee phase* In the GBC algorithm the scout bee phase is two-fold. First, we check all the employee bees in the population and reset all of them that have been trapped by local optima. This is achieved by counting the number of times c that we perform an operation in a solution with no improvements. At this point, if $c > \delta$ then we replace the solution with a new subset randomly generated.

  Second, a mutation operator is applied to the *Queen Bee* to intensify the search in the neighbourhood of the best solution found so far. Consequently, for each gene in the *Queen Bee* we generate a random number r in $[0, 1]$ and if $r < \alpha$ then the i-th gene mutates according to the following equation.

$$v_j^i = \begin{cases} QueenB^i + K & \text{if } |S| - K > QueenB^i \\ x_j^i - K & \text{otherwise,} \end{cases} \tag{2.23}$$

  , with $K = rand(-1, 1) \times (QueenB^i - x_j^k)$, where $rand(-1, 1)$ denotes a random number generator in the range of $[-1, 1]$, k is an integer random number in $[0, SN - 1]$ and $QueenB$ is the best solution found so far.

Chapter 3

First Contribution: Improvement of Accuracy of Set-wise Evaluation Methods

3.1 Introduction

In this chapter, we propose new feature selection algorithms based on consistency measures. First, we propose a new algorithm based on the steepest descent method. Second, we propose a combination of the SUPER-LCC and the SDCC algorithms to improve the accuracy and speed up the SDCC method. The novel approach makes use of a sliding window technique to accomplish better accuracy and efficiency. Lastly, we propose a new hybrid method that is based on the super-lcc and the simulated annealing technique. This method, namely the Hybrid-Consistency-based Simulate Annealing, uses a given classifier to estimate the best threshold δ for the super-lcc algorithm.

3.2 Fast SDCC

INTERACT and LCC algorithms have a linear computational complexity in terms of features. Nevertheless they are usually trapped by local optima. As a solution to this problem, the SDCC algorithm, proposed in [59], presents a good balance to the fundamental tradeoff between the quality of outputs of feature selection algorithms and their efficiency. It has been verified that SDCC outputs better subsets than INTERACT and LCC in terms of consistency. Nevertheless, due to SDCC must evaluate $\left(|\mathcal{F}| + |\tilde{\mathcal{F}}|\right)\left(|\mathcal{F}| - |\tilde{\mathcal{F}}| + 1\right)/2$ subsets to output $\tilde{\mathcal{F}}$, it is not applicable to high-dimensional datasets. It would be very profitable if outputs with similar quality of those returned by SDCC were obtained in a fast way in high-dimensional domains. To make this yearning real, one of the the main

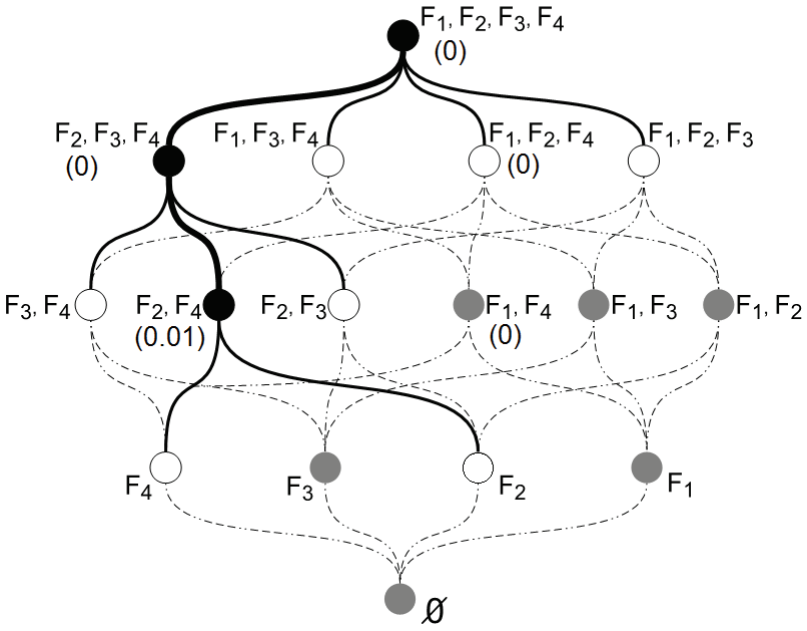


Figure 3.1: Example of the search strategy used by SDCC

contributions of this research is the proposal of two new algorithms, which take advantage of the properties of consistency measures to avoid unnecessary evaluations.

Next, we analyse some weak points of SDCC using the example of Figure 3.1 . Let assume that the entire feature set $\mathbb{F}$ consists of four features: f_1, f_2, f_3, f_4. In Figure 3.1, a node represents a feature subset of $\mathbb{F}$ (we have $2^4 = 16$ nodes in total), while a link between two feature subsets means that SDCC can move from the feature subset placed above to the other placed below by removing a single feature. Thus, starting from $\tilde{\mathcal{F}} = \tilde{F}$, SDCC keeps travelling downward along links, while the visited feature subset meets the condition of $\mathfrak{Br}(\tilde{\tilde{F}}; \mathrm{C}) \leq \delta = 0.01$. In the chart, the inconsistency measurement is represented enclosed in parentheses below the subsets which meet this condition.

In this example, a deficiency relating to the effectiveness of SDCC is revealed. Supposing $\{f_1, f_4\}$ is the smallest of the consistent sets (i.e. the optimal set). SDCC is trapped by a local optima because in the first iteration the effect of removing f_1 and f_3 is similar. Thus, due to a lack of a mechanism to decide which feature removes when there is a tie, SDCC randomly selects a feature. Now, we propose this weakness formally.

> **Weakness 1**. When $\mathfrak{Br}(\mathbb{F} \setminus f_i; \mathrm{C}) = \delta'$ for $i = \{1, 2, \ldots, l\}$ and $\delta' = \min_{F \in \tilde{\mathbb{F}}} \mathfrak{Br}(\tilde{\mathbb{F}} \setminus \{F\}; C)$, the SDCC algorithm arbitrarily selects any feature f_i. When this happens, the probability of SDCC be trapped by a local optima depends on the probability of the removed feature belongs to the optimal set.

Regarding the number of evaluation we can ask ourself the follwoing question. Why SDCC

evaluates, in the first iteration, to $\{f_1, f_3, f_4\}$; $\{f_1, f_2, f_4\}$ and $\{f_1, f_2, f_3\}$ if the set with minimum inconsistency has been already discovered (i.e. $br\tilde{\mathbb{F}} \setminus \{f_1\} = \mathfrak{Br}(\{f_2, f_3, f_4\}\,; \mathrm{C}) = \mathfrak{Br}(\tilde{\mathbb{F}}; \mathrm{C})$). The monotonicity property of consistency measures allows avoiding unnecessary evaluations. Although irrelevant and redundant features are abundant in high-dimensional datasets, we could expect that the number of evaluations executed by SDCC to output $\tilde{\mathcal{F}}$ could be significantly smaller than $\left(|\mathbb{F}| + |\tilde{\mathcal{F}}|\right)\left(|\mathbb{F}| - |\tilde{\mathcal{F}}| + 1\right)/2$. Furthermore, supposing SDCC is in the second iteration where the effect of individually removing f_2, f_3 or f_4 from $\tilde{\mathcal{F}} = \{f_2, f_3, f_4\}$ is evaluated; $\mathfrak{Br}(f_2, f_3; \mathrm{C}) > \delta$ means that f_4 may not be removed from $\tilde{\mathcal{F}}$, and even more, based on the monotonicity property we can say that $\mathfrak{Br}(f_2) > \delta$ and $\mathfrak{Br}(f_3) > \delta$. We generalize this idea below.

> **Lemma 1**. If we remove feature f from $\tilde{\mathcal{F}}$ and $\tilde{\mathcal{F}} \setminus f$ becomes inconsistent, then any subset $\tilde{\mathcal{G}}$ from the power set of $\tilde{\mathcal{F}}$ such that $\tilde{\mathcal{G}} \notin f$ is also inconsistent. Consequently, f may not be removed from $\tilde{\mathcal{F}}$ and must be part of the output.

Consequently, the evaluation $\mathfrak{Br}(f_2; \mathrm{C})$ is unnecesary in the third iteration.

> **Weakness 2**. SDCC execute several unnecesary evaluations.

In order to polish up the SDCC algorithm, we address these issues as follows:

1. If two or more features fulfill the removing condition, then that one less correlated with the class is eliminated.

2.1 If, in an iteration, $\tilde{\mathcal{F}} \setminus \{f\}$ turns inconsistent (i.e. $\mathfrak{Br}(\tilde{\mathcal{F}} \setminus \{f\}\,; C) > \delta$), $\tilde{\mathcal{G}} \setminus \{f\}$, with $\tilde{\mathcal{G}} \subseteq \mathcal{F}$, won't be evaluated in next iterations. Thus, f won't be removed and will be in the final output.

2.2 Furthermore, if the minimum inconsistent measurement is reached in the *i-th* evaluation of an iteration, the *i-th* feature evaluated is immediatly removed and the current iteration is finished. In this way, $\mid \tilde{\mathcal{F}} \mid -i$ evaluations are saved in each iteration.

Note that proposition 2.2 is contradictory to first issue since we suggest to arbitrarily remove the first feature evaluated which, when is removed, do not increase the inconsistency measurement of the current $\tilde{\mathcal{F}}$. Because of that, we also propose to ranking the features before starting the search, in incremental order according to the correlation of each feature $f_i \in \mathcal{F}$ with the class, as in INTERACT and LCC. In this manner, of those all features which do not increase the inconsistency rate of the current $\tilde{\mathcal{F}}$ when are removed, the first evaluated will be the least correlated with the class.

An important observation is that when $\delta = 0$, FSDCC selects the same output of LCC executing $\mid \mathcal{F} \mid$ evaluations. This happens because in FSDCC when $\mathfrak{Br}(\tilde{\mathcal{F}} \setminus \{F\}\,; C) > 0$,

ALGORITHM Fast Steepest-Descent CC (FSDCC)
INPUT A consistency measure function
An ordered feature set $\mathcal{F}$
A threshold δ
OUTPUT A minimal subset $\tilde{\mathcal{F}} \subseteq \mathcal{F}$ such that $\mathfrak{Br}(\tilde{\mathcal{F}}; C) \leq \delta$
STEPS
Let $\tilde{\mathcal{F}} = \mathcal{F}$ and $\xi = \mathfrak{Br}(\mathcal{F}; C)$.
If $\xi > \delta$, abort.
For each $F \in \mathcal{F}$, let $\delta(F) = 0$.
Repeat
Let $\xi' = \delta$ and $F' = \mathsf{Nil}$.
For each $F \in \tilde{\mathcal{F}}$ from the first to the end
If $\delta(F) \leq \xi'$
Let $\delta(F) = \mathfrak{Br}(\tilde{\mathcal{F}} \setminus \{F\}; C)$.
If $\delta(F) = \xi$
Let $\tilde{\mathcal{F}} = \tilde{\mathcal{F}} \setminus \{F\}$ and $F' = F$.
Break.
Else If $\delta(F) < \xi'$
Let $\xi' = \delta(F)$ and $F' = F$.
End If.
End If.
End For.
If $F' = \mathsf{Nil}$, break.
Let $\tilde{\mathcal{F}} = \tilde{\mathcal{F}} \setminus \{F\}$ and $\xi = \xi'$.
End Repeat.
Return $\tilde{\mathcal{F}}$.

Figure 3.2: The algorithm of FSDCC

F will be in the final output, and hence, is useless to evaluate it over again. On the other hand, when $\mathfrak{Br}(\tilde{\mathcal{F}} \setminus \{f\}; C) = 0$, *F* is immediately removed.

Furthermore, when $\delta > 0$, the features *F* such that $\delta \geq \mathfrak{Br}(\tilde{\mathcal{F}} \setminus \{F\}; C) > \mathfrak{Br}(\tilde{\mathcal{F}}; C)$ are evaluated more than once.

The algorithm *Fast Steepest-Descent Consistency-Constrained* (FSDCC) algorithm which take into consideration all these solutions to the weakness of SDCC is depicted in Figure 3.2 .

3.3 Accurate Sdcc

Although we expect that FSDCC considerably improve the performance of SDCC in terms of number of evaluations and consistency rate of outputs, we think it posses some weakness relating to the accuracy obtained by the machine learning algorithms applied to the reduced

data when compared with LCC. This fact could happen because, due to the nature of the search of SDCC and FSDCC, not always the features more correlated with the class tend to remain in the final output.

In order to better understanding this idea, let analyse the following example. Supposing we have, in each itertion, a set $\tilde{\mathcal{F}}$ and for some of its features $f_i \in \tilde{\mathcal{F}}$, $\delta \geq \mathfrak{Br}(\tilde{\mathcal{F}} \setminus \{f_i\}; C) > \mathfrak{Br}(\tilde{\mathcal{F}}; C)$ holds. On the one hand, LCC removes the feature f_i less correlated with the class because is the first feature evaluated that fulfils the removing condition $\delta \geq \mathfrak{Br}(\tilde{\mathcal{F}} \setminus \{f_i\}; C)$. On the other hand, SDCC and FSDCC remove the feature $F = \arg\min_{f_i} \mathfrak{Br}(\tilde{\mathcal{F}} \setminus \{f_i\})$. Therefore, we can say that the final output of LCC will tend to contain the features more correlated with the class whereas SDCC will tend to select a very consistent set which not necessarily will be composed by features highly correlated with the class. Once presented this apparent contradiction, the following step is to discover which of these approaches is the most suitable for the knowledge discovery process.

One way to do this, is by establishing a balance between the consistency contribution of each feature and its respective correlation level with the class. With this propose, next equation is presented:

$$\vartheta(f, C) = \alpha SU(f, C) + (1 - \alpha) \frac{\mathfrak{Br}(\tilde{\mathcal{F}} \setminus \{f\}; C) - \mathfrak{Br}(\mathcal{F}; C)}{\delta - \mathfrak{Br}(\mathcal{F}; C)}. \tag{3.1}$$

Here, $0 \leq \alpha \leq 1$ is a balance parameter which allows to specify, in certain level, the preferable type of feature to seize: those highly correlated with the class, those which are indispensable to compose a very consistent set or a combination of both. When anlyzing all features $f \in \tilde{\mathcal{F}}$, since it is suitable to remove f with small correlation with the class and poor consistency contribution, the main goal will be to remove feature $f = \arg\min_{f \in \tilde{\mathcal{F}}} \vartheta(f, C)$. Furthermore, $\vartheta(f, C)$ has the following properties:

1. When $\delta \geq \mathfrak{Br}(\tilde{\mathcal{F}} \setminus \{f\}; C)$, that is: f is consistent, $0 \leq \vartheta(f, C) \leq 1$ holds.

2. If $SU(g, C) > SU(f, C)$ and $\mathfrak{Br}(\tilde{\mathcal{F}} \setminus \{f\}; C) = \mathfrak{Br}(\tilde{\mathcal{F}}; C)$, then $\vartheta(g, C) \lneq \vartheta(f, C)$. Thus, ϑ is monotonic when is applied in conjunction of a backward search over a ranked set.

3. If $\vartheta(f, C)$ with $\alpha = 0$ is used as evaluation function in FSDCC the output will be the same as if $\mathfrak{Br}(\tilde{\mathcal{F}} \setminus \{f\}; C)$ is used instead.

Taking into consideration these basic properties we have modified (see Figure 3.3) the FSDCC algorithm giving place to a new algorithm named *Accurate Steepest-Descent Consistency-Constrained* (ASDCC).

ALGORITHM Accurate Steepest-Descent CC (ASDCC)
INPUT A consistency measure function
An ordered feature set $\mathcal{F}$
A threshold δ
OUTPUT A minimal subset $\tilde{\mathcal{F}} \subseteq \mathcal{F}$ such that $\mathfrak{Br}(\tilde{\mathcal{F}}; C) \leq \delta$
STEPS
Let $\tilde{\mathcal{F}} = \mathcal{F}$ and $\xi = \mathfrak{Br}(\mathcal{F}; C)$.
If $\xi > \delta$, abort.
For each $f \in \mathcal{F}$, let $\delta(f) = 0$.
Repeat
Let $\xi' = \delta$, $\vartheta = \infty$ and $f' = \mathsf{Nil}$.
For each $f \in \tilde{\mathcal{F}}$ from the first to the end
If $\delta(f) \leq \xi'$
Let $\delta(f) = \mathfrak{Br}(\tilde{\mathcal{F}} \setminus \{f\}; C)$ and $\vartheta' = \vartheta(f, C)$
If $\delta(f) = \xi$ and $\vartheta' < \vartheta$
Let $\tilde{\mathcal{F}} = \tilde{\mathcal{F}} \setminus \{f\}$ and $f' = f$.
Break.
Else If $\delta(f) < \xi'$
Let $\xi' = \delta(f)$.
If $\vartheta' < \vartheta$
$\vartheta = \vartheta'$and $f' = f$
End If.
End If.
End If.
End For.
If $f' = \mathsf{Nil}$, break.
Let $\tilde{\mathcal{F}} = \tilde{\mathcal{F}} \setminus \{f\}$ and $\xi = \xi'$.
End Repeat.
Return $\tilde{\mathcal{F}}$.

Figure 3.3: The algorithm of ASDCC

Table 3.1: Datasets used in the experiment

Dataset	#EXAM.	#FEAT.	#CLASSES
DERMATOLOGY	366	34	6
ARRHYTHMIA	452	279	16
Kr-vs-Kp	3196	36	2
PENDIGITS	10992	16	10
MUSHROOM	8124	22	2
OPTIDIGITS	5620	64	10
NURSERY	12960	8	5
WAVEFORM	5000	40	3
SPECTROMETER	531	100	48
MFEAT-FACTOR	2000	216	10
MFEAT-FOURIER	2000	76	10
MFEAT-KARHUNEN	2000	64	10
MFEAT-PIXEL	2000	240	10
MFEAT-ZERNIKE	2000	47	10
SEGMENT	2310	19	7
SEMEOIN	1593	256	10

3.4 Experimental evaluation

We prove effectiveness of the FSDCC and ASDCC through experiments using the datasets described in Table 3.1 . We verify the advantages of these algorithms over SDCC and LCC in terms of number of features selected, $\mathfrak{Bt}(\tilde{\mathcal{F}}; \mathrm{C})$ and accuracy obtained by three machine learning algorithms applied in the reduced data. *Naive Bayes* (*NB*), *C4.5* and *Support Vector Machine* (*SVM*) were the machine learning algorithms selected because they are very representatives. The results of INTERACT are not shown because they were very low when compared with LCC.

We evaluated all the evaluation variables changing the threshold δ from 0 to 0.1 with 0.01 increments in between for each dataset, and plotted the average over the 16 datasets. Becasue of the SDCC algorithm is sensitive to the order of features, for each δ it was computed the mean (straight line) and the standard deviation (dashed lines) over 10 runs. In each run the order of features was randomly changed.

After evaluating the ASDCC with different values of α, we state $\alpha = 0.75$ is a good value. In Figure 3.4 only some of these evaluations are presented.

According to the experiments represented in Figure 3.4 , FSDCC outputs the most reduced and less inconsistent solutions for almost all δ. The third chart in the first row of Figure 3.4

represents the proportion of the number of evaluations executed between SDCC and LCC, FSDCC and ASDCC. In other words, this chart shows how many times the number of evaluation of SDCC exceeds the number of evaluation of the other algorithms. Although

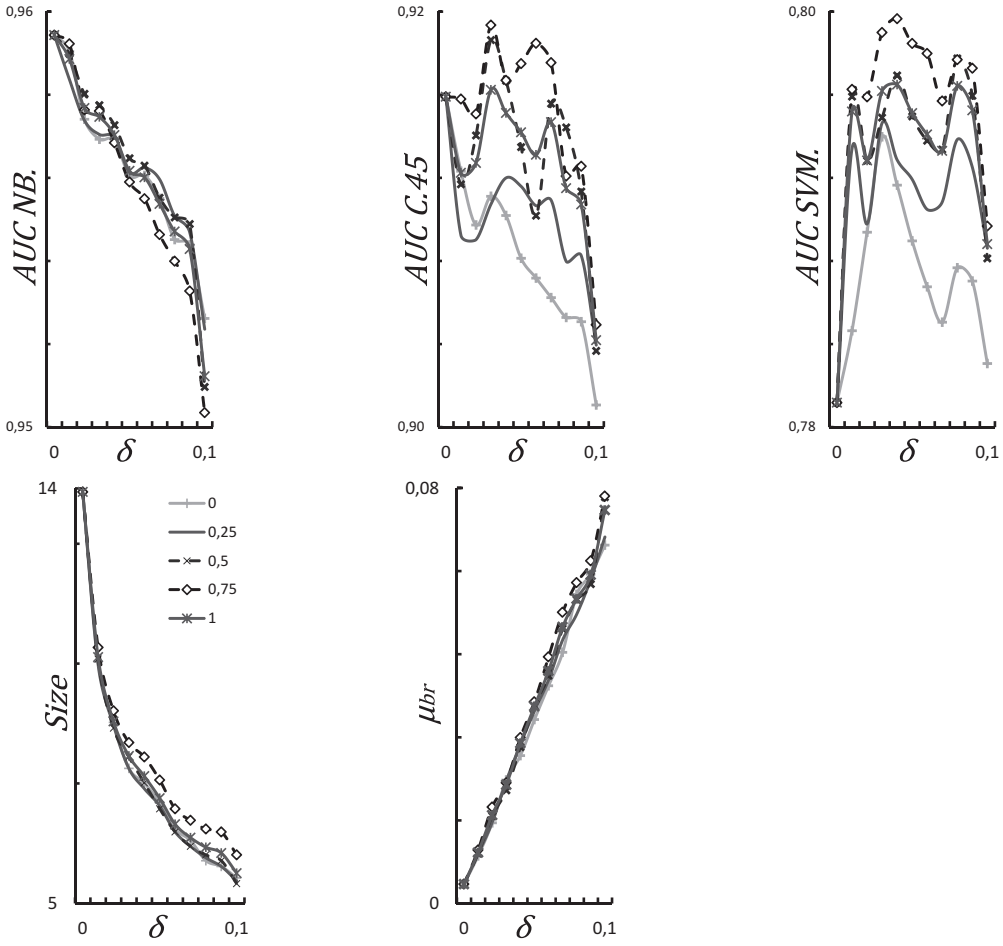


Figure 3.4: Performance of ASDCC algorithm with different values of α

FSDCC and ASDCC have a remarkable difference when compared with LCC when $\delta > 0$, they shown a vaste improvement vis-a-vis SDCC algorithm.

On the other hand, the outputs of ASDCC are quite beneficial for the accuracy reached by the three machine learning algorithms applied after the reduction. Surprisingly, although SDCC and FSDCC output very consistent sets, the accuracy reached by the machine learning algorithms over their results are not so good. A response to this phenomenun could be that each classifier partialy assumes feature dependencies.

Speaking with generality, ASDCC obtain more consistent sets than LCC and more accurate sets than all the algorithms.

Bonferroni Dump non parametric test was used to detect significant differences among the performance of feature selection algorithms. In Figure 3.6 and 3.7 the critical distance which defines the existence of significant differences between the best ranked algorithm and the others with a confidence level of 90% is depicted using a dark shadow. Note that in these charts the ranking value is defined as the average over the rank positions obtained by an algorithm in each dataset.

According to the results shown in Figure 3.5 FSDCC outputs sets significantly smaller than those of LCC and SDCC when $\delta \geq 0.01$ and $\delta > 0.01$ respectively. Relating to the $\mathfrak{Br}()$ measurements the best ranked algorithm for small δ is SDCC.

On the other hand, it was verified that for all δ, ASDCC and FSDCC significantly improved the AUC values obtained by the three machine learning algorithms executed over the outputs of SDCC. Although for C.45 and SVM algorithms significant difference are not so evident when comparing AUC values of ASDCC and FSDCC against LCC, the ranking value tends to be better for the formers when compared with the latter. This means that LCC dropped more relevant features as δ increases and consequently is more sensible to be trapped by local optima.

We measure the time-efficiency of the algorithms by the number of times in which the algorithms compute Bayesian risks since the time to compute Bayesian risks is dominating in the entire execution time of the algorithms. Figure 3.5 shows plots of the averaged ratios of SDCC to the other algorithms.

From Figure 3.5 , we see that FAST SDCC and ACCURATE SDCC compute as many Bayesian risks as LCC for $\delta = 0$, that is, they are as fast as LCC. This is because FAST SDCC and ACCURATE SDCC give up further search in each iteration of Repeat when they detect the first occurence of $\mathfrak{Br}(\tilde{F} \setminus \{f\}) = \mathfrak{Br}(\mathbb{F})$ (Lines 12 to 14 in Figure 3.3), and hence, they behave exactly the same as LCC. For $\delta > 0$, the chart indicates that SDCC computes Bayesian risks 30 times more than FAST SDCC, while it does about 20 to 30 times more than ACCURATE SDCC.

Also, Table 3.2 shows the actual run-time of the algorithms for each dataset measured in seconds for $\delta = 0.01$. The averaged run-time ratios of SDCC and LCC, FAST SDCC

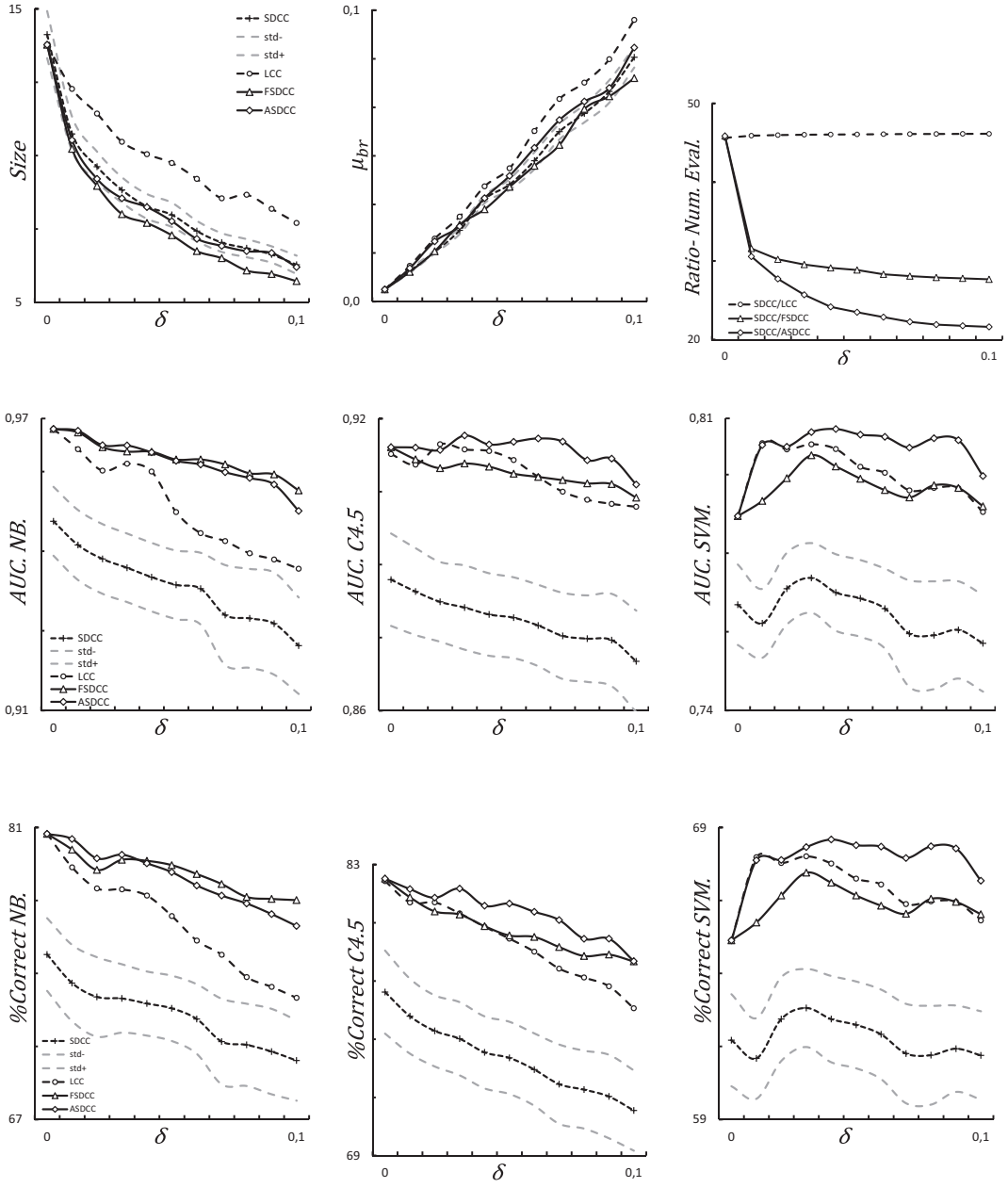


Figure 3.5: Number of features selected, $\mathfrak{Br}()$, proportion of the number of evaluation, and accuracy

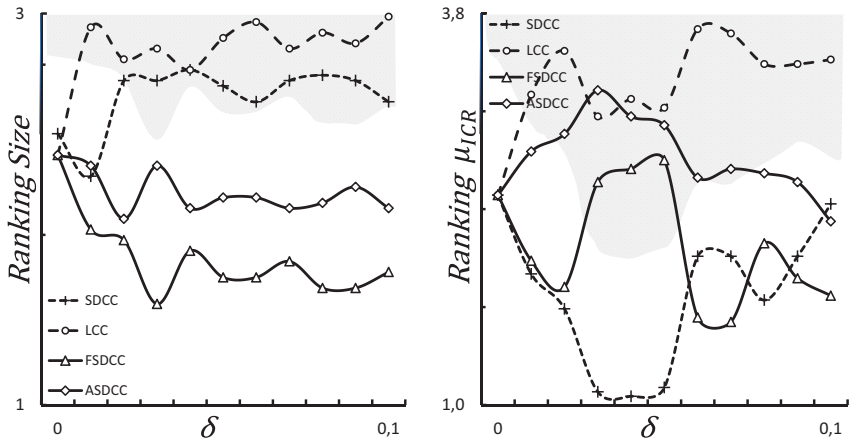


Figure 3.6: Graphical results of Bonferroni-Dunn non parametric test for ranking size and $\mathfrak{Br}()$

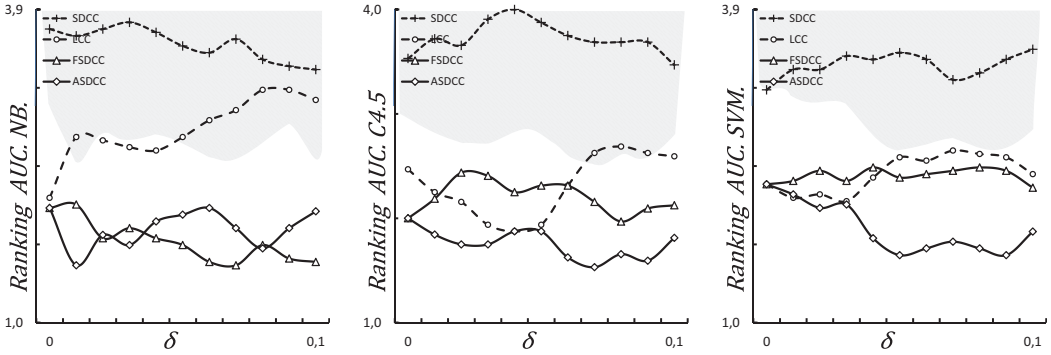


Figure 3.7: Graphical results of Bonferroni-Dunn non parametric test for ranking AUC values

and ACCURATE SDCC are 45.7, 34.6 and 33.4, respectively, and are very closed to the ratios in the numbers of computation of Bayesian risks.

Consequently, we can conclude that FAST SDCC and ACCURATE SDCC are 20 to 30 times faster than SDCC and a few times slower than LCC.

Table 3.2: Run-time (sec.) with δ=0.01 (Intel Core i3 2.6GHz and 8GB memory)

Dataset	SDCC	ASDCC	FSDCC	LCC
DER	0.175	0.091	0.092	0.153
ARR	43.83	0.532	0.504	0.398
KRK	2.246	0.747	0.721	0.247
PEN	3.044	1.928	1.708	0.700
MUS	1.066	0.228	0.225	0.276
OPT	14.04	1.268	1.239	0.921
NUR	0.536	0.613	0.534	1.057
WAV	6.557	0.930	0.971	0.632
SPE	3.495	0.877	0.674	0.596
MFA	107.2	1.448	1.586	1.388
MFO	6.638	0.929	0.927	0.729
MKA	5.054	0.691	0.694	0.665
MPI	140.3	1.468	1.399	0.976
MZE	2.119	0.500	0.506	0.521
SEG	0.444	0.206	0.215	0.130
SEM	127.6	1.452	1.420	0.787

3.5 Sdcc with the sliding window method

Steepest-descent is a first-order optimization algorithm that finds a local minimum of a given function by stepping the solution in the direction where the function decreases most quickly [59]. The main advantage of SDCC over LCC can be justified as follows. LCC eliminates the first feature f_i that satisfies $\mathfrak{Br}(\tilde{F} \setminus \{f_i\}; C) \leq \delta$ from $\tilde{F}$, while SDCC tests all $f_i \in \tilde{F}$ and eliminates f_i that minimizes $\mathfrak{Br}(\tilde{F} \setminus \{f_i\}; C)$ such that $\mathfrak{Br}(\tilde{F} \setminus \{f_i\}; C) \leq \delta$. We consider $\tilde{F}$ as a point in the space of subsets of the entire features of . The neighbours of $\tilde{F}$ are determined by $\tilde{F} \setminus \{f_i\}$ for $f_i \in \tilde{F}$; and the distance between $\tilde{F}$ and $\tilde{F} \setminus \{f_i\}$ is given by $\mathfrak{Br}(\tilde{F}\{f_i\}; C) - \mathfrak{Br}(\tilde{F}; \mathrm{C})$. When a function f over feature subsets is $f(\tilde{F}) = |Ft|$, the gradient from $\tilde{F}$ to $\tilde{F} \setminus \{f_i\}$ is $1/(\mathfrak{Br}(\tilde{F} \setminus \{f_i\}; C) - \mathfrak{Br}(\tilde{F}; \mathrm{C}))$. Therefore, an increase of the inconsistency score by eliminating a single feature for SDCC is at least equal than by eliminating a single feature for LCC. This also means that SDCC can eliminate more features than LCC.

Although it is known that SDCC significantly beats LCC in terms of the inconsistency score, SDCC performs $(|F| + |\tilde{F}|)(|F| - |\tilde{F}| + 1)/2$ evaluations to output $\tilde{F}$. Furthermore, we have detected that SDCC removes a lot of features highly-correlated with the class

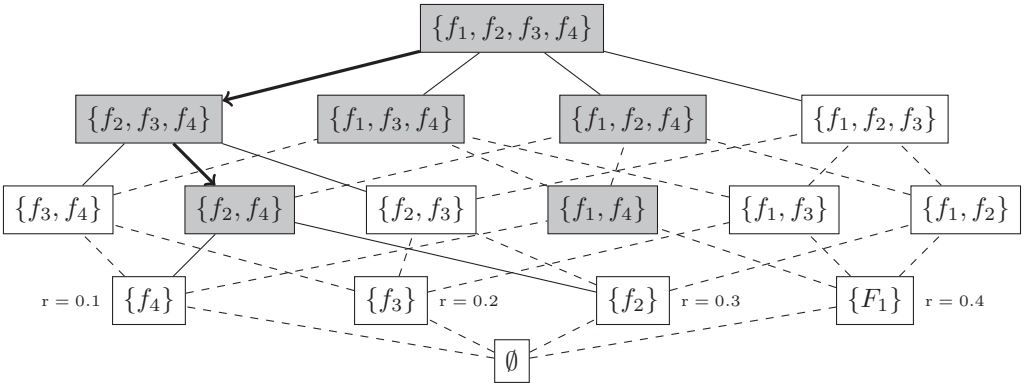


Figure 3.8: An example of search paths by *steepest-descent*. r stands for the individual relevance of a feature.

variable, which may affect the performance of classifiers. In the remainder of this section we discuss some efficiency and effectiveness issues of SDCC. Moreover, we propose a new algorithm to solve these issues.

Figure 3.8 is the Hasse diagram of $F = \{f_1, f_2, f_3, f_4\}$, and the gray nodes represent the feature subsets whose *inconsistency* is zero. With $\delta = 0$, the solid lines represent an example of the paths that SDCC can track. In the first iteration, SDCC investigates the four feature subsets of $\{f_2, f_3, f_4\}$, $\{f_1, f_3, f_4\}$, $\{f_1, f_2, f_4\}$ and $\{f_1, f_2, f_3\}$. The *inconsistency* of three of them are zero, and SDCC chooses $\{f_2, f_3, f_4\}$. In the same way, in the second iteration, SDCC investigates $\{f_3, f_4\}$, $\{f_2, f_4\}$ and $\{f_2, f_3\}$ and chooses $\{f_2, f_4\}$. In the last iteration, SDCC investigates $\{f_4\}$ and $\{f_2\}$ and then terminates.

Problem 1: Small Total Relevance Score: In Figure 3.8, $\{f_2, f_4\}$ and $\{f_1, f_4\}$ are the two candidates that SDCC can select, because they are minimal in the inclusion relation among the feature subsets in F with $\mathfrak{Br}(F; C) \leq 0$. Although the Sdcc selects one of $\{f_2, f_4\}$ and $\{f_1, f_4\}$ arbitrarily, $\{f_1, f_4\}$ is likely to be a better answer than $\{f_2, f_4\}$, because $\mathrm{r}(f_1, \mathrm{C}) + \mathrm{r}(f_4, \mathrm{C}) = 0.5 > \mathrm{r}(f_2, \mathrm{C}) + \mathrm{r}(f_4, \mathrm{C}) = 0.4$. In general, provided all the minimal sets G in F with $\mathfrak{Br}(G; C) \leq \delta$, SDCC arbitrary selects any set G regardless any other information.

Solution to Problem 1 The Individual relevance insensitivity problem occurs because the individual relevance of features has no meaning in the *steepest-descent* algorithm. That is, the *steepest-descent* arbitrarily removes any feature f^- such that $f^- \in \mathrm{argmin}_{f_i \in \tilde{F}}\{\mathfrak{Br}(\tilde{F} \setminus \{f_i\}; C) \mid \mathfrak{Br}(\tilde{F} \setminus \{f_i\}; C) \leq \delta\}$.

A straightforward way to deal with this problem is by removing the feature f^- with the smallest individual relevance. That is, in each iteration remove feature f^-, such that

$$f^- \in \mathrm{argmin}\{r(f; C) \mid f \in \mathrm{argmin}\{\mathfrak{Br}(\tilde{F} \setminus \{f_i\}) | f_i \in \tilde{F}, \mathfrak{Br}(\tilde{F} \setminus \{f_i\}) \leq \delta\}\}. \quad (3.2)$$

To validate the effect of this solution, we have compared SDCC and the corrected that

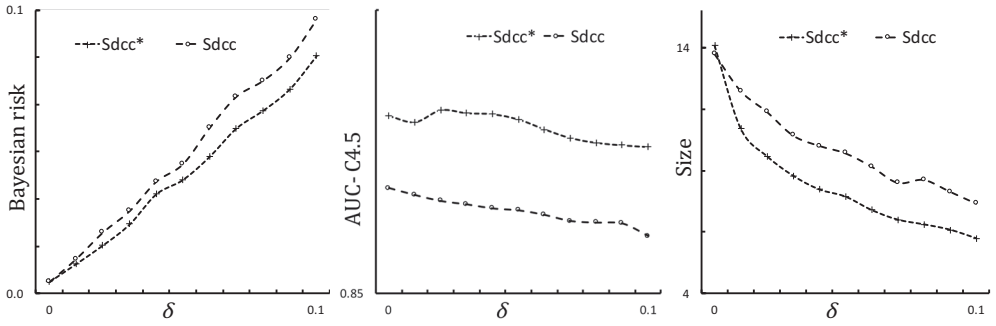


Figure 3.9: Comparison between the original SDCC [59] and its corrected version that searches features based on Eq.(1) in terms of the bayesian risk, the AUC-ROC by *C4.5* classifier and, the number of features selected.

searches features by Eq.1 version using 50 datasets chosen from the UCI machine learning repository [14]. As we expected the corrected version significantly outperforms the original version in terms of the AUC-ROC, the bayesian risk and the number of features selected. Figure 3.9 depicts the averages of the bayesian risk, AUC-ROC when *C4.5* is used as a classifier and the number of features selected across the 50 datasets. The threshold parameter δ varies in the interval $[0, 0.1]$ with an increment of 0.01.

Although these results are quite good, maximization of the average of the individual features (collective relevance) can not be guaranteed because the individual relevance of features is measured back stage. This means that until now the process of removing a feature is composed by two sequential steps and the individual relevance score is only used in the second one. In many cases, this unbalanced trade-off between the bayesian risk and the collective relevance of a set, may lead to undesirable results as stated in Section 2.2.3. We now consider the individual relevance of features as a crucial factor to judge the quality of a feature set, by proposing the *interelevance score* measure defined as follows.

$$IR(\tilde{F}; f_i; C) = (1-\alpha)A(\tilde{F}; f_i; C) + \alpha B(f_i; C)$$

$$\text{with } A(\tilde{F}; f_i; C) = \begin{cases} \frac{\mathfrak{Br}(\tilde{F}\setminus\{f_i\};C)-\mathfrak{Br}(F;C)}{\delta-\mathfrak{Br}(F;C)}, & \text{if } \mathfrak{Br}(F;C) \leq \delta \\ \mathfrak{Br}(\tilde{F}\setminus\{f_i\};C) - \mathfrak{Br}(F;C), & \text{if } \mathfrak{Br}(F;C) = \delta \end{cases}$$

$$B = \begin{cases} \frac{r(f_i;C)-r^-}{r^+-r^-} & \text{if } r^+ > r^- \\ 0 & \text{if } r^+ = r^- \end{cases}$$

where $r^+ = \max_{f_i \in F} r(f_i; C)$, $r^- = \min_{f_i \in F} r(f_i; C)$ and α satisfies $0 \leq \alpha \leq 1$. The *interelevance score* IR is normalization function that evaluates how good is a given feature f_i for the current feature set $\tilde{F}$. IR measures: i) how relevant is f_i and ii) the effect of removing f_i from $\tilde{F}$ from the consistency point of view. Function *A* normalize the *bayesian risk* obtained by removing feature f_i from $\tilde{F}$. $\mathfrak{Br}(F; C)$ and δ are taken

as the minimum and maximum value respectively in the normalization function. We expect that IR metric allows to select interacting feature sets composed by features with high relevance score. Thus, to select f^-, we do not use Eq.1 as a criterion, but use $f^- \in \text{argmin}\{\text{IR}(\tilde{F}; f_i; C) \mid f_i \in \tilde{F}, \mathfrak{Br}(\tilde{F} \setminus \{f_i\}; C) \leq \delta\}$.

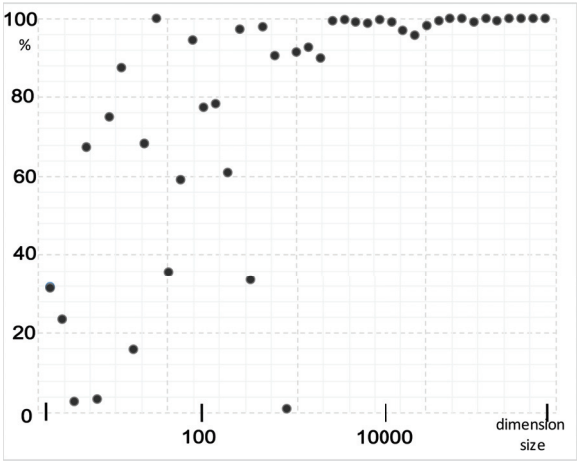


Figure 3.10: Percentage of the first consecutive features $\{f_1, \ldots, f_l\}$ such that $\mathfrak{Br}(F; C) = \mathfrak{Br}(F \setminus \{f_1, \ldots, f_l\}; C)$ to the entire feature set F.

Problem 2: Low scalability to high-dimensional data: Our new version of SDCC may be still slow in datasets with large number of interacting features. Although it is well known high-dimensional datasets are rich in non-interacting features, we do not assume their class variable can be described by a small number of features. Therefore, we now describe two mechanism to reduce even more the number of evaluation of our proposal.

Solution 1 to Problem 2: Eliminating the big mass of irrelevant features by SUPER-LCC. High-dimensional datasets are likely to be abundant in irrelevant and non-interacting features. Assuming $|F|$ is very large, we can expect $\mathfrak{Br}(F \setminus F'; C) = \mathfrak{Br}(F; C)$ with $F' = \{f_1, \ldots, f_l\}$ for a large value of l. To make sure this expectation is true, we randomly picked 44 datasets from the *UCI machine learning repository* and determine l.

The experiments were conducted in small ($|F| \leq 100$), medium ($100 < |F| \leq 10000$) and high-dimensional data ($10000 < |F|$) using $\delta = \mathfrak{Br}(F; C)$. Figure 3.10 depicts the results, and we see that values of l are very close to the numbers of the entire features $|F|$, when the dataset is high-dimensional. This means that for these high-dimensional datasets our *steepest-descent* algorithm will remove a huge number of consecutive features one by one, which is not so efficient. However, recently Shin et al. in [56] have found that l can be determined efficiently by means of *binary search*. In fact, $\{f_1, \ldots, f_l\}$ are removed by the first iteration of SUPER-LCC. This finding broke the premise that consistency-based algorithms were computationally too expensive to apply to high-dimensional data. We use their finding to efficiently remove F' with only a few iterations. We use the first iteration of SUPER-LCC to eliminate the largest $\{f_1, \ldots, f_l\}$ such that $\mathfrak{Br}(F \setminus \{f_1, \ldots, f_l\}; C) \leq$

$\mathfrak{Br}(F;C) + \delta$ and then apply *steepest descent* to the remainder of the features, that is, $\{f_{l+1}, \ldots, f_n\}$.

Solution 2 to Problem 2: Windowing the search. When feature selection is performed using consistency measures, in each iteration of the search we can categorize features as: indispensable, useless and potential features. Being $\tilde{F}$ the current feature set, indispensable features must remain in $\tilde{F}$ in order to keep the *bayesian risk* under the threshold. That is, a feature $f_x \in \tilde{F}$ is indispensable if $\mathfrak{Br}(\tilde{F} \setminus \{f_x\}; C) > \delta$ holds. On the contrary, useless features can be safely removed without degrading the *bayesian score* of $\tilde{F}$. A feature f_y is said to be useless when $\mathfrak{Br}(\tilde{F} \setminus \{f_y\}; C) = \mathfrak{Br}(\tilde{F}; C)$ holds. On the other hand, if a feature is neither of indispensable nor useless then it is a potential feature. That is, for potential feature f, $\mathfrak{Br}(\tilde{F} \setminus \{f\}; C) \leq \mathfrak{Br}(\tilde{F}; C) + \delta$ holds. Potential features are the most interesting type of features: they necessarily become indispensable or useless at any moment of the search and must be evaluated in the next iteration. Speaking about efficiency, the worst case scenario, in a given iteration, is that all features are potential. This means that our version of the *steepest-descent* algorithm needs $|\tilde{F}|$ evaluations to remove the one that minimize IR. To overcome this drawback we propose to limit the search in each iteration to a portion of the features in the current set. This can be done by applying a mobile window search. Let $\overline{d}$ be the average of the differences of the individual relevance of consecutive features in F

$$\overline{d} = \frac{1}{n-1} \sum_{i=1}^{n-1} \Big(r(f_{i+1}; C) - r(f_i; C) \Big) = \frac{1}{n-1} (r^+ - r^-), \tag{3.3}$$

we define the window search w_k in the $k-$th iteration as:

$$w_1 = r^- + \omega(r^+ - r^-) \tag{3.4}$$

$$w_k = w_{k-1} + \lambda \overline{d}, \text{ with } k > 1, \tag{3.5}$$

where $\omega = (0, 1]$ and $\lambda \in \mathbb{R}^+$ are predefined parameters that influence the initial size of the window search w_1 and the acceleration of the expansion of the window search w_k in the $k-$th iteration respectively. If the relevance score of a feature falls into the region of the window $[r^-, w_k)$ then will be evaluated in the $k-$th iteration.

The number of features evaluated in each iteration is not only determined by the position of useless features but also by the size of the window search. This may significantly improve the efficiency of our *steepest-descent* version in datasets abundant in potential features. Let F be the entire feature set and δ be the upper bound of the permissible *bayesian risk* of the output sets. We combine all the solutions given above as follows.

1. The relevance $r(f_i; C)$ of each feature $f_i \in F$ is computed using the *Symmetrical Uncertainty* measure, and F is mapped to $\tilde{F}$ by sorting the features in incremental order of

Algorithm 4 SWCFS Algorithm

Input: D: the dataset
δ: inconsistency score threshold
ω: initial size of the window search
λ: windows size coefficient
Output: $\tilde{F}$ suboptimal set

Rank features in F in incremental order according to SU
Fix $\tilde{F} = F$
Find the maximum l such that $\mathfrak{Br}(\tilde{F} \setminus \{f_1, \ldots, f_l\}; C) = \mathfrak{Br}(\tilde{F}; C)$
Update $\tilde{F} = \tilde{F} \setminus \{f_1, \ldots, f_l\}$
Compute $r^+ = \max_{f_i \in \tilde{F}} r(f_i; C)$ and $w_1 = \omega r^+$
Let $\overline{d}$ be the average of the difference between $SU(f_i; C)$ and $SU(f_{i-1}; C)$ for $f_i \in \tilde{F}$, $k = 1$ and $IR^- = \inf$

repeatTrue $f^- = Null$
$f_i \in \tilde{F}$
if $SU(f_i; C) \leq w_k$ **then** $\delta[f_i] = \mathfrak{Br}(\tilde{F} \setminus \{f_i\}; C)$
$\delta[f_i] > \delta$**continue** $\delta[f_i] = \mathfrak{Br}(\tilde{F}; C) f^- = f_i$, **break** $IR(\tilde{F}; f_i; C) \leq IR^- f^- = f_i, IR^- = IR(\tilde{F}; f_i; C)$ $f^- = Null$**break** $\tilde{F} = \tilde{F} \setminus \{f^-\}$
$k = k + 1$
$w_k = w_{k-1} + \lambda \overline{d}$

Figure 3.11: The algorithm of SWCFS

$SU(f_i; C)$.

2. The maximum set of consecutive useless features $\{f_1, \ldots, f_l\}$ is identified and removed by using the *binary search*.

3. The window size is computed in each iteration.

The *steepest-descent* algorithm is performed using the interelevance score IR by evaluating only the features included in the current window and taking into account the following rules with $f_i \in \tilde{F}$:

Rule 1. If f_i is an useless feature then it is immediately removed from $\tilde{F}$ (line 13).

Rule 2. Else if f_i is indispensable then f_i is not evaluated anymore and never will be removed from $\tilde{F}$ (line 12).

Rule 3. Otherwise the feature f_i that minimize IR is removed from $\tilde{F}$ if $IR(\tilde{F}; f; C) > IR(\tilde{F}; \varnothing; C)$ holds. The algorithm stops when all features have been tested and none of the features can be removed. Figure 3.11 depicts the entire algorithm.

3.6 Experiments of Sddc with the window method

We empirically evaluate the performance of the proposed algorithm and make comparisons with some state-of-the-art feature selection algorithms: RELIEFF (RF)[33], CFS[22], FCBF[67] and SUPER-LCC[56] and ASDCC[47]. We exclude from comparison algorithms SUPER-CWC[56] and FSDCC[47] because we verify they output similar results to SUPER-LCC and ASDCC respectively.

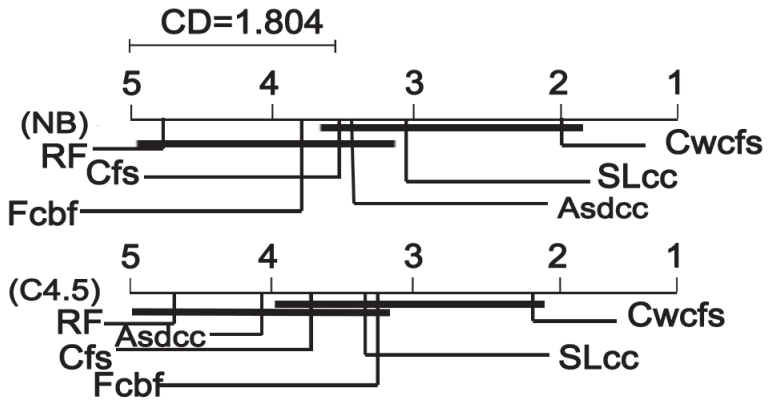


Figure 3.12: Nemenyi test with $\alpha = 0.05$

The configuration of the experiments is as follows. First, we run the feature selection algorithms over the datasets and obtain selected feature subsets for respective algorithms. To evaluate the classification capability of the selected feature sets, we run ten-fold cross validation on the reduced data using two classifiers: NAIVE BAYES and C4.5. The *bayesian risk* parameter δ of SUPER-LCC and SWCFS algorithms was fixed to 0.01. We report results about the AUC-ROC values of both classifiers and the number of features selected by each algorithm. Before running experiments we run SWCFS accross many datasets with different values of α and verified that
$alpha = 0.5$ works well. Table 3.3 shows the AUC-ROC values after running the classifiers on the reduced data and the number of features selected by each algorithm.

Speaking about the size of the output, SWCFS outputs smaller or equal when compared with SLCC. Furthermore, when compared with all the algorithms it turns out that SWCFS is ranked top for a half of the datasets. Speaking about AUC-ROC scores, SWCFS is ranked top for more than the 68% and 62% of the datasets for NAIVE BAYES and C4.5 classifiers respectively. To statistically compare the algorithms, we run Friedman test and statistical differences were found. Figure 3.12 shows the Nemenyi's chart for each classifier. Group of algorithms that are not significantly different are connected with a thick line.

It is apparent that SLCC and SWCFS are compatible in terms of efficiency in high-dimensional data since SWCFS takes advantage of the first iteration of SLCC to remove the less relevant features that are not necessary to create consistency sets. In the case where

Table 3.3: Results of AUC-ROC values for the reduced data and number of features selected by each algorithm

	NB-AUC values						C4.5-AUC values						size					
data	RF	Cfs	Fcbf	SLcc	ASdcc	Swcfs	RF	Cfs	Fcbf	SLcc	Asdcc	Swcfs	RF	Cfs	Fcbf	SLcc	Asdcc	Swcfs
OPT.	.945	.967	.966	.966	.967	.968	.858	.924	.929	.933	.928	.935	30	38	21	9	10	8
ARR.	.468	.850	.854	.848	.850	.848	.464	.738	.737	.733	.733	.733	1	25	12	21	28	21
MAD.	.523	.644	.646	.647	.646	.647	.500	.770	.613	.811	.814	.811	1	6	4	15	12	15
MFE	.966	.973	.985	.986	.970	.991	.972	.968	.961	.952	.954	.964	360	85	136	7	8	6
SEM	.983	.956	.952	.955	.958	.956	.877	.881	.876	.865	.879	.885	175	74	30	31	45	27
AUD	.946	.939	.905	.962	.923	.952	.907	.905	.924	.921	.905	.924	10	6	16	12	9	12
KRV	.969	.930	.968	.972	.970	.983	.972	.930	.959	.997	.995	.997	5	3	7	21	18	15
MF1	.922	.948	.947	.977	.981	.981	.923	.908	.925	.916	.914	.911	90	67	38	8	9	7
MF2	.961	.969	.968	.969	.968	.970	.903	.905	.899	.906	.905	.910	15	12	37	11	13	11
MF3	.979	.986	.986	.981	.984	.984	.907	.915	.907	.920	.907	.924	7	26	57	7	7	7
MF4	.949	.950	.945	.950	.949	.950	.918	.919	.918	.922	.918	.922	3	4	2	5	4	5
MF5	.964	.965	.969	.967	.937	.969	.903	.904	.911	.901	.904	.906	196	103	27	21	17	17
MF6	.925	.955	.955	.957	.955	.957	.859	.880	.884	.871	.871	.871	7	25	14	12	14	12
PEN	.977	.963	.963	.963	.964	.964	.973	.973	.974	.975	.970	.974	16	11	11	7	10	7
SPL	.981	.984	.993	.990	.984	.989	.967	.969	.970	.969	.969	.970	19	6	22	9	8	9
WAV	.510	.945	.932	.938	.945	.946	.500	.858	.882	.877	.876	.884	1	15	6	10	8	9
AVG.	.873	.933	.933	.939	.934	.941	.838	.897	.892	.904	.903	.908	58.5	31.6	.27.5	12.9	13.8	11.8

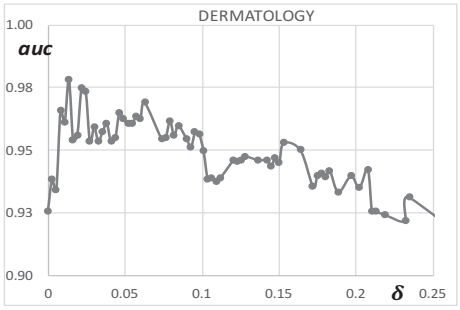
only small number of features are eliminated in the first step of SWCFS, the numbers of evaluations depends on the size of the sliding window. However, if the sliding window is reasonably small then the number of evaluation can be comparable with the number of evaluations of LCC algorithm. Nevertheless, as Figure 3.10 shows, it turns out that high-dimensional data are prone to be rich in irrelevant features that can be removed in the first iteration of SWCFS.

3.7 Simulated-Annealing-based LCC

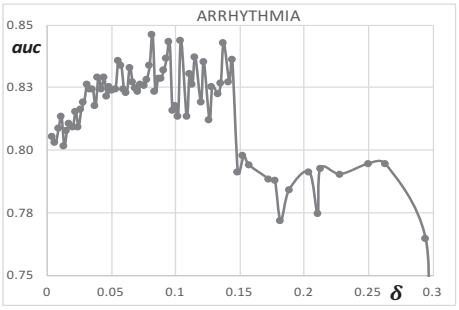
To the best of our knowledge, SUPER-LCC is one of the fastest and accurate feature selection algorithm based on consistency measures. SUPER-LCC works under the assumption that high-dimensional datasets are abundant in irrelevant features that can be removed in mass. By the first to the $(i-1)$-th iterations of the algorithm, SUPER-LCC determines a sequence of indices of features $l_1 < l_2 < \cdots < l_{i-1}$, and defines $\tilde{F} = (F \setminus \{f_1, \ldots, f_{l_{i-1}}\}) \cup \{f_{l_1}, \ldots, f_{l_{i-1}}\}$. In the i-th iteration, the algorithm finds l_i such that

$$l_i = \underset{j=l_{i-1}+1,\ldots,n}{\operatorname{argmin}} \mathfrak{Br}(\tilde{F} \setminus \{f_{l_{i-1}+1}, \ldots, f_j\}) \leq \delta,$$

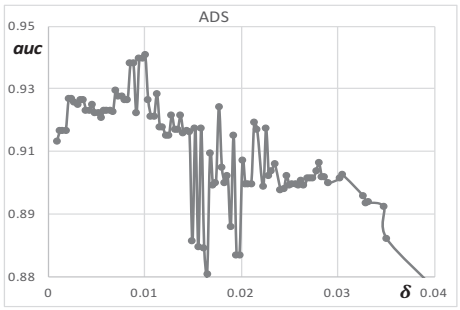
by *binary search* due to the monotonicity property of the bayesian risk. SUPER-LCC outputs the same set as LCC but on average has a computational complexity of $O(nm(\log n + \log m))$, where n is the number of features that describes the m instances in $\mathbf{D}$. To the best of our knowledge, SUPER-LCC is the algorithm with better practical performance in both of efficiency and accuracy in the field of feature selection. According to the authors of [56], for data with more than hundred thousand features, SUPER-LCC needs some seconds



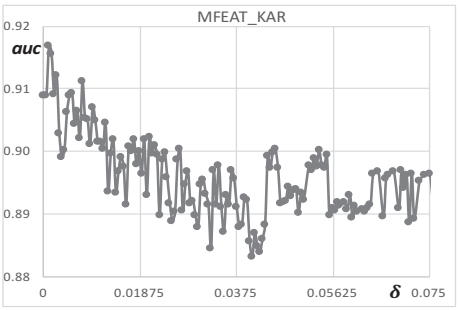
(a) dermatology



(b) arrhythmia



(c) ads



(d) pixel

Figure 3.13: AUC-ROC values of *c4.5* classifier for the outputed value of LCC/SUPER-LCC when varying δ. Five-fold cross validation was used to compute AUC-ROC values.

to give a response in an ordinary personal computer. In the remaining of this paper, we show how to incorporate SUPERLCC and LCC to our approach, but first we present some important properties of LCC/SUPERLCC.

Let $\tilde{F}$ be a subset with $\mathfrak{Br}(\tilde{F}; C) = \beta$, with $\delta = \beta - \epsilon$. For $\epsilon > 0$, LCC outputs a different feature set from $\tilde{F}$. For the output of the i-th iteration of LCC, $F_i = (F \setminus \{f_1, \ldots, f_{l_i}\}) \setminus \{f_{l_1}, \ldots, f_{l_i}\}, \mathfrak{Br}(F_i; C) \leq \delta$ holds.

Figure 3.13 represents a part of the search space of LCC when varying δ.

Theorem 3.7 suggests that, if we run LCC multiple times increasing δ, then the smaller δ is, the smaller the index of the first feature selected by LCC in $F = \{f_n, \ldots, f_1\}$ is. Therefore, if we want to run LCC twice with δ_1 and δ_2 (being $\delta_1 < \delta_2$) over F, and assuming f_{l_1} is the first feature selected by LCC with δ_1, it is reasonable to run LCC with δ_2 not over F, but over $\{f_{l_1}, \ldots, f_1\} \in F$. This is a very important property of LCC to save computational time. In the next section we state how to use it in our new algorithm.

Another important property of LCC/SUPERLCC is as follows. Being br and $\tilde{F}$ the *Bayesian risk* and the outputted set obtained when we run LCC/SUPERLCC with δ, if we let $\delta = br - \alpha$ then LCC/SUPERLCC output a set $\tilde{F}'$ different from $\tilde{F}$. This property suggests that if we perform multiple running of LCC using different and sorted thresholds $\{\delta_m, \delta_{m-1}, \ldots, \delta_0\}$ with $\delta_i > \delta_{i-1}$ and $\mathfrak{Br}(\emptyset; C) \leq \delta_i \leq \mathfrak{Br}(\tilde{F}; C)$ then we obtain $\{F_{\delta_m}, F_{\delta_{m-1}}, \ldots, F_{\delta_0}\}$ different sets. Moreover, if we compute the AUC-ROC values with

a given classifier for all these sets then we can see the AUC-ROC function defined by the interval $[\delta_{min}, \delta_{max}]$.

Furthermore, given the entire feature set F of a dataset, if we fix $\delta^- = \mathfrak{Br}(F; C)$ and $\delta^+ = \mathfrak{Br}(F; C) + \epsilon$ and let be $\Delta = \{\delta^+, \delta^+ - \alpha, \delta^+ - 2\alpha, \ldots, \delta^-\}$, then for a relatively small value of ϵ, if we run LCC with the ordered δ values in Δ, then we may expect to obtain m subsets, which *bayesian risk* are close to $\mathfrak{Br}(F; C)$. Under the assumption these sets may have very low *bayesian risk*, they can be used as the search space for the wrapper search. In the next section we propose a new algorithm that take advantage of Theorem 3.7 to efficiently generate a space of high-quality features given δ^- and δ^+.

3.7.1 Simulated annealing

Simulated Annealing is a stochastic technique used in optimization problems to efficiently scape from local optima. Given a current state of the problem cs, the algorithm generates a candidate (or next) state ns and stochastically decides whether or not the current state is updated to the candidate state. The probability of passing from one state to another is called the transition probability p and often is defined as $p = exp(nf - cf)/T$ where nf and cf are the values of the target function for ns and cs respectively, and T is the temperature variable. In simulated annealing we keep the temperature variable T to simulate the heating process on metal.

We initially fix a high temperature, for example: $T = 0.1/(\log(t + 1))$ with $t = 1$, and then allow it to slowly decrease as the algorithm runs. That is, we can increase t in each iteration. The higher the temperature the more likely to accept solutions that are worse than the current solution. This gives the algorithm the ability to jump out of any local optimums found in early iterations. As the temperature is reduced the algorithm gradually focus on the area of the search space in which hopefully, a close to optimum solution can be found. This gradual cooling process is what makes the simulated annealing algorithm remarkably effective when dealing with large problems which contain numerous local optimums such as the depicted in Figure 1. Figure 2 depicts the general scheme of *Simulated Annealing* we will use for our feature selection algorithm where $rnd(0, 1)$ returns a random number between zero and one according to a uniform distribution and k is an arbitrary constant. Moreover, $continue$ defines the stopping criterion and can be: a) $T > T_{min}$ or/and b) $t < t_{max}$, where T_{min} and t_{max} are predefined constants.

3.7.2 Target function

Now we only need to define the function $next(\delta_c)$ to generate a neighbour state δ_n of the current state δ_c and the target function $f(\delta_c)$. In our problem we state the parameter δ of LCC algorithm as the space of all possible states. In particular, we generate some of the

Algorithm 5 *Simulated Annealing*

Terminology: cs: current state
ns: next state
$next(cs)$: a function that returns a neighbour state of cs
$f(cs)$: the target function

$cs = cs_0$
$cf = f(cs_o)$
$t = 1$

while continue **do** $ns = next(cs)$
$nf = f(ns)$
$T = k/(\log(t+1))$
$p = exp(nf - cf)/T$

if $rnd(0,1) < p$ **then** $cs = ns$
$cf = nf$
$t = t + 1$
return cs

Figure 3.14: The algorithm of *Simulated Annealing*

values of δ in the interval of $\delta^+ \leq \delta \leq \delta^-$ to run LCC and obtain a set of feature subsets $\tilde{F}_\delta$ that can be evaluated by the AUC-ROC function across a fold-cross validation process. We let $\text{AUC}(\mathbf{D}_{\tilde{F}_\delta}, \ell)$ be the target function for the current state δ. $\tilde{F}$ represents the subset of features selected by LCC with δ and $\mathbf{D}_{\tilde{F}}$ represents the the dataset resulted from projecting the set $\tilde{F}$ over $\mathbf{D}$. ℓ is the classifier used in the training and testing process of the cross validation.

3.7.3 Neighbour generator function

The neighbour generation function is crucial in the *Simulated Annealing* algorithm to scape from local optima. Given a current value δ_c of δ, we intend generate a neighbour b of δ_c such that $\text{AUC}(\mathbf{D}_{\tilde{F}_b}, \ell) > \text{AUC}(\mathbf{D}_{\tilde{F}_{\delta_c}}, \ell)$. However, since the target function AUC is unknown this is difficult to achieve.

Another issue in the neighbour generation function is that two different values of δ can lead to the same output of LCC. In this case we may have duplication of candidate sets in the search space of the feature selection problem, which lead to compute the same operations more than once. One way to avoid this is by performing a downward search over the space of δ-values. That is, we generate the search space of feature subsets for the

wrapper evaluator by running LCC(δ_t) with different values of δ in decreasing order. As an instance, given δ^- and a predefined constant ϵ we run LCC with $\delta^- \leq \delta_t \leq (\delta^- + \epsilon) = \delta^+$ such that:

$$\delta_t = \begin{cases} \delta^+ - \beta_t & \text{if } br_{t-1} \geq (\delta^+ - \beta_t) \\ br_{t-1} - \alpha & \text{if } br_{t-1} < (\delta^+ - \beta_t) \end{cases},$$

Algorithm 6 *next(δ_c)*

Terminology:
δ_c: current δ
$step$: a given constant to determine the upper limit of the next state

$mv = step * (rnd(0,1) - 0.5)$
$br_{closest} = closestSBrComputed(\delta_c + mv)$

if $(\delta_c + mv) - br_{closest} > \alpha$ **then** $(F, b) =$ LCC$(\delta_c + mv)$
$list.add((F, b))$
return b
return $br_{closest}$

Figure 3.15: Algorithm to generate the next candidate *Bayesian risk*

where β_t is a propagation function with respect to t such as: $k * t$ or $k/2^t$ (with k as a constant), α is a value as small as required and br_t represents the *bayesian risk* of the set obtained by LCC(δ_t) with δ_t. t is a counter variable that increase in one in each iteration. According to the property of LCC exposed in section 2, when we run LCC with different δ_t always obtain a different set. Therefore, the problem of duplication of sets is solved by this procedure. However, iterating downward may lead to miss the global optima behind. Therefore, we need a mechanism to make a bi-directional search over δ and minimize the number of sets duplicated. Figure 3.15 depicts the proposed function $next(\delta_c)$ to generate a neighbour b given the current state δ_c. In this function, we move δ_c in the space of δ by mv, which is a number generated randomly in the interval $[step + 0.5, step - 0.5]$ (line 1). To minimize the duplicity of feature sets when running LCC with similar values of δ: $br_{closest}$ and $\delta_c + mv$, we fix a threshold α, such that, if $(\delta_c + mv) - br_{closest} > \alpha$ holds, then we consider LCC$(\delta_c + mv) =$ LCC$(br_{closest})$ (line 3), where $br_{closest}$ is the closest and smaller *Bayesian risk* computed so far ($closestSBrComputed(\delta)$) stored in $list$ (line 5).

3.7.4 SALCC: A new algorithm

Now that we have defined some properties of LCC and our scheme for the *Simulated Annealing* algorithm, we propose a new feature selection algorithm namely *Simulated*

Annealing for Linear-Consistency-Constrained-based feature selection(SALCC) as follows.
1. First, we rank the features in F in increasing order of *Symmetrical Uncertainty* (SU) values.
2. Second, we find the *border feature* f_l such that l is maximum and

$$\mathfrak{Br}(\{f_l, \ldots, f_n\}; C) = \mathfrak{Br}(F; C)$$

and fix $\tilde{F} = \{f_{l+1}, \ldots, f_n\}$. This is easily and efficiently achieved by running the first iteration of SUPERLCC described in [56].
At this point, we reduce the search space from $F = \{f_1, \ldots, f_n\}$ to $\tilde{F} = \{f_l, \ldots, f_n\}$. Note that this does not affect the final solution of our algorithm because of Theorem 3.7. This step will make our algorithm scalable for high-dimensional datasets. To make sure this expectation is true, we picked 44 datasets from the *UCI machine learning repository* [14] and determine the percentage of features removed in the first iteration of SUPERLCC. Results are depicted in Figure 3.10 .
3. We run the *Simulated Annealing* algorithm shown in section 3.1 by using the proposed target and neighbour generator functions.

3.7.5 Experiments

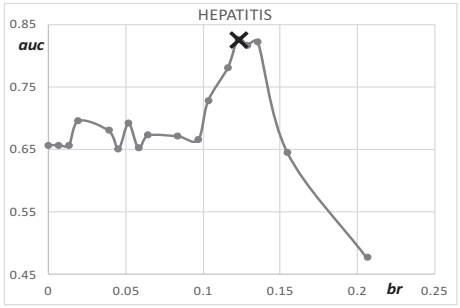
We empirically evaluate the performance of the proposed algorithm in terms of accuracy, and number of times the classifier is used and number of features selected. For the experiments we use the following parameters: $\delta_{max} = 0.4$, $T_{min} = 0.2$, $step = 0.1$, $\alpha = 0.001$ and we use *C4.5* as classifier. Datasets were selected from the *UCI Machine learning repository* and they represent several areas of current researches [14].

Figure 3.17 shows part of the entire search space of sixteen datasets and the suboptimal set reached by our algorithm depicted with a black cross. In datasets a, b, f, h, i, j, k and l, the algorithm found the global optima and in datasets c, d, e, g, m, o and p, the sets found are very close to the optimal solution in terms of AUC-ROC values.

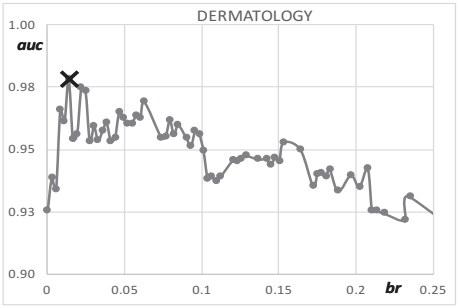
Althoug these datasets are abundant in local optima, we can conclude that SALCC can approximately find the optimal set in many cases. We also conclude that *Simulated Annealing* works very well in these data and can easily scape from local optima.

To evaluate in a more appropiate way SALCC algorithm we perform a ten-fold cross validation comparison with some of the state-of-the-art algorithm in eight text-classification datasets[1]. The algorithms selected for the comparison are: RELIEFF[29], CFS[22], FCBF[67] and SLCC. Table 1 shows that SALCC performs well in most of the datasets when compared with the state-of-the-art algorithms.

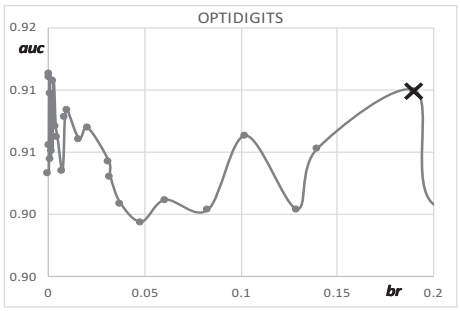
[1]http://tunedit.org/repo/Data/Text-wc



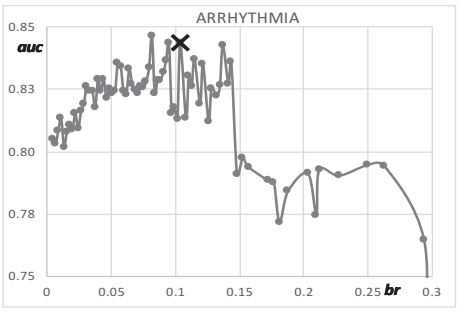
(a) HEP



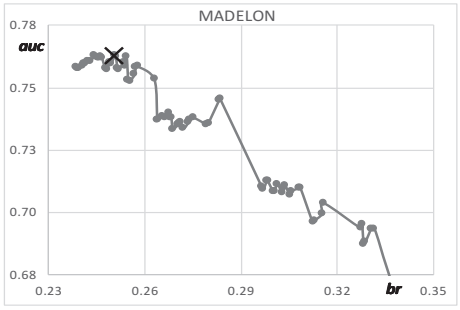
(b) DER



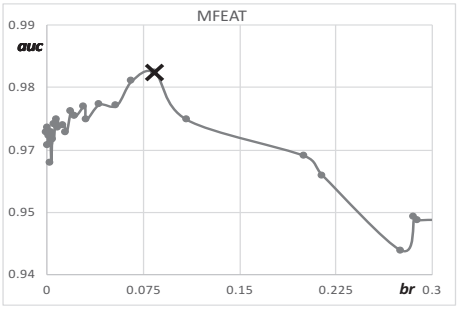
(c) OPT



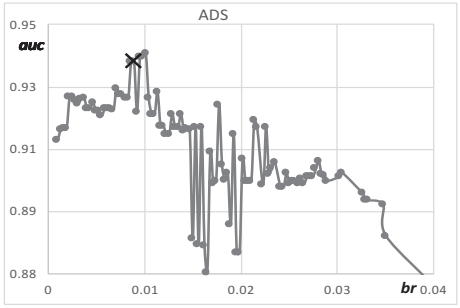
(d) ARR



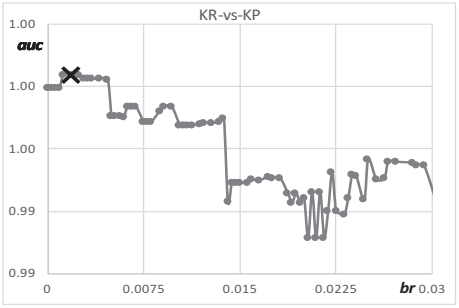
(e) MAD



(f) MFE

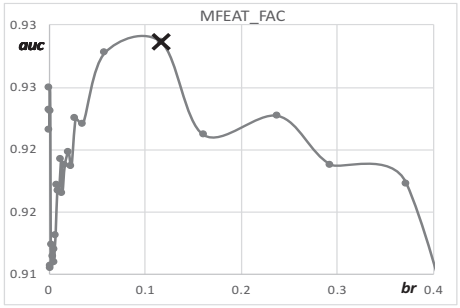


(g) ADS

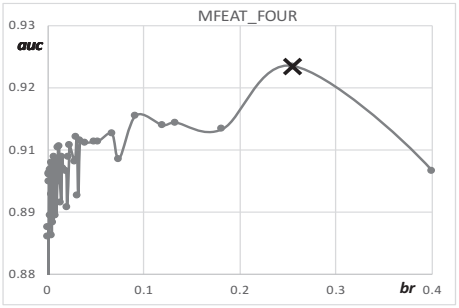


(h) KRV

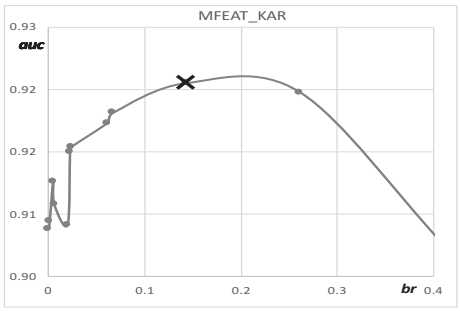
Figure 3.16: search space and set found by the proposed algorithm



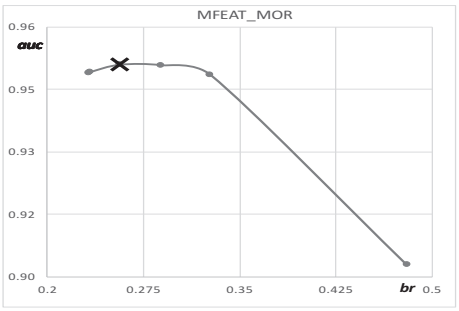
(a) FAC



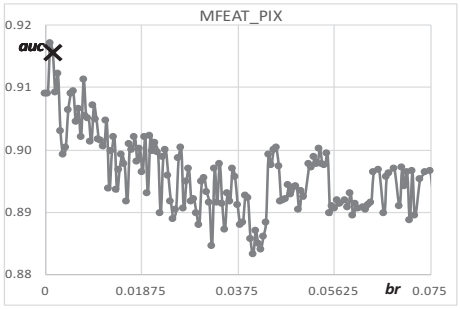
(b) FOUR



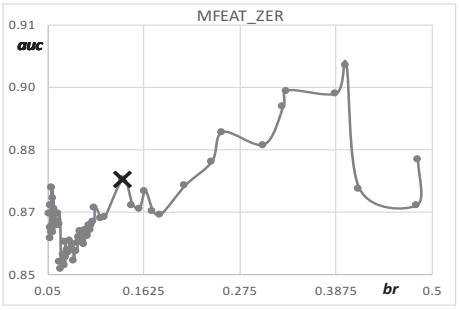
(c) FAR



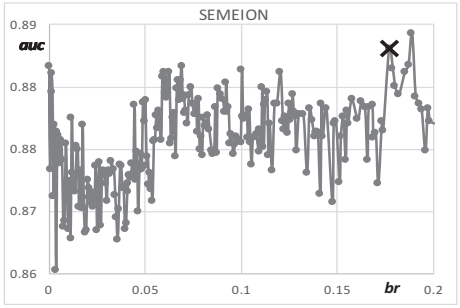
(d) MOR



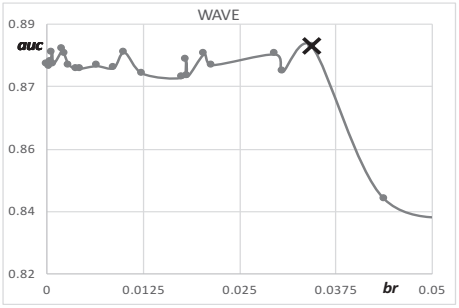
(e) PIX



(f) ZER



(g) SEM



(h) WAV

Figure 3.17: search space and set found by the proposed algorithm

Table 3.4: AUC values comparison among some of the state-of-the-art algorithms and SALCC.

	NB-AUC values					C4.5-AUC values				
dataset	RF	Cfs	Fcbf	SLcc	SaLcc	RF	Cfs	Fcbf	SLcc	SaLcc
tr21	.807	.829	.833	.834	.871	.732	.824	.821	.825	.827
tr41	.612	.721	.794	.748	.748	.587	.741	.788	.714	.714
tr45	.827	.822	.834	.843	.847	.700	.798	.818	.821	.821
wap	.902	.963	.945	.945	.948	.932	.981	.961	.962	.974
tr31	.832	.821	.809	.819	.831	.847	.882	.856	.856	.875
fbis	.709	.722	.734	.729	.741	.776	.742	.739	.747	.747
la2s	.825	.881	.846	.867	.881	.624	.708	.725	.724	.724
la1s	.863	.869	.868	.865	.865	.833	.869	.849	.847	.901
AVG.	.797	.829	.833	.831	.842	.754	.818	.820	.812	.823

Table 3.5 shows the number of times SALCC use the classifier to evaluate a set (ESets), the total number of sets in the range of $[\delta^-, \delta^+]$ (TSets), number of features selected by LCC ($\text{LCC}_{\#f}$) and SALCC ($\text{SALCC}_{\#f}$). As the table depicts, the number of evaluations is relatively small respect to the total number of sets. Also, the number of features selected by SALCC is very small when compared with the number of features selected by LCC. In general we state that SALCC significantly improves the accuracy and size of solutions of LCC.

Table 3.5: Number of times the classifier is used versus the number of all possible sets and the number of features selected by LCC and SALCC.

DATA	ESets	TSets	$\text{LCC}_{\#f}$	$\text{SALCC}_{\#f}$
HEP	10	18	10	3
DER	31	69	10	9
OPT	14	29	13	5
ARR	35	78	23	11
MAD	31	76	15	11
MFE	13	31	8	4
ADS	46	107	39	21
KRV	28	94	29	23
FAC	12	27	12	4
FOU	19	44	14	6
KAR	8	12	9	4
MOR	5	6	5	4
PIX	37	298	16	21
ZER	24	62	12	7
SEM	33	319	29	23
WAV	20	39	12	8

Chapter 4

Second Contribution: Improvement of Efficiency and Accuracy of Pairwise Evaluation Methods

4.1 Introduction

In this chapter we focus on the improvement of the pairwise evaluation methods, such as: CFS and MRMR. We first analyse the algorithm of CFS and exhibit a problem of the implementation of this algorithm in the weka framework. Next, we propose a mechanism to avoid unnecessary evaluations in both of MRMR and CFS algorithms. The proposed algorithms return the same output of their original versions, however, the are much faster.

4.2 Fast CFS

Although CFS was created almost two decades ago, it is still widely used by researchers as a comparison prototype due to its impressive results in terms of accuracy [57][61][46] and has been one of the most popular feature selection functions at all times. In addition, it has been another motivation for many researchers that they can use CFS from the weka interface [62], a well-known useful toolbox for researchers and practionists of machine learning.

Nevertheless, it has been also known that the CFS algorithm used from the weka interface is extremely slow, and this issue has been imposing an unnecessary restriction to application of CFS to high dimensional datasets for long. The aim of this section is to clarify the cause of the issue of CFS and to give a concrete solution. In this section, we reveal the reason why CFS is so slow in the weka framework. In addition, we also propose a reimplementation of CFS to make it faster and scalable to high-dimensional data.

To select k features, in other words, to perform the iteration k times, the greedy forward search requires evaluation of the CFS fucntion $(2n - k + 1)k/2$ times, and the number itself is feasibly small even for high-dimensional datasets: If we assume $k \in O(n)$, this number is a quadratic function of n; if we can assume that k is fixed, it is merely a linear function.

However, in the reality, if we chose the greedy forward search and the CFS function in the weka framework, the resulting run-time will become extremely large frequently. The reason for this problem is because naïve evaluation of the CFS function is time-consuming. In fact, when the size of S is l, evaluation of the CFS function requires computation of l SU values for $Cs(S)$ and $l(l-1)/2$ SU values for $Rs(S)$. Hence, to select k features by the greedy forward search strategy, the number of times to compute SU scores turns out to be

$$\sum_{l=1}^{k}(n-l+1)\left(l+\frac{l(l-1)}{2}\right)=\frac{1}{24}k(k+1)(4nk-3k^2+8n-3k+6),$$

and is $O(n^4)$ if $k \in O(n)$. In the weka framework, the CFS function is naïvely computed, and as a result, the resulting run-time can easily become infeasibly large.

This design of weka is mainly because the weka framework equally deals with arbitrary evaluation function in a uniform manner to use them as a black-box function. To be specific, when the weka framework calls a feature set evaluation function, it inputs a feature set S and a dataset D into the function, and no context information can be taken into account when the function performs evaluation.

Assume that $Cs(S)$ and $Rs(S)$ have been computed. For $R \subset \mathbb{F}_D$ with $R \cap S = \emptyset$, if the values of $Cs(S)$ and $Rs(S)$ could be additionally input to the function, the number of times in which the function has to compute SU values to compute $Cs(S \cup R)$ and $Rs(S \cup R)$ would be reduced. In fact, the formula

$$Cfs(S \cup R) = \frac{Cs(S) + Cs(R)}{\sqrt{k + 2(Rs(S) + Rs(R) + \sum_{f \in R}\sum_{f' \in S} SU(f; f')}} \tag{4.1}$$

indicates that only $(l_R + \frac{l_R(l_R-1)}{2} + l_S l_R$ SU values have to be computed newly, when we let $l_S = |S|$ and $l_R = |R|$. We naturally expect that this new interface to the CFS function would improve the run-time performance of the CFS algorithm drastically.

Note that from now on, for simplicity we use $SU_{i,j}$ to denote $SU(f_i, f_j)$. To better understand our proposal, let's redefine Cfs function for the case where a new candidate

feature f_p is requested to be evaluated by the greedy forward search, as follows:

$$Cfs(S, f_p) = \frac{\frac{1}{k}\sum_{i'=1}^{k-1} SU_{i',c} + \frac{1}{k} SU_{p,c}}{\sqrt{k + k(k-1)\left[\sum_{i'=1}^{k-2}\sum_{j'=i'+1}^{k-1} SU_{i',j'} + \sum_{i'=1}^{k-2} SU_{i',p} + SU_{k-1,p}\right]}}. \quad (4.2)$$

Equation 4.2 is equivalent to $Cfs(S)$. Also equation 4.2 suggests that $Cfs(S, f_p)$ performs i) $k-1$ sums in the summation of the numerator to compute $Cs(S \cup f_p)$, ii) $(k-1)(k-2)/2$ sums in the double summation in the denominator, and iii) $k-2$ sums in the last summation of the denominator, when evaluating a single feature f_p with $|S| = k-1$. We are especially interested in knowing the effect on the efficiency of the greedy forward search when eliminating all these unnecessary summations. Avoiding such sums may drastically increase the efficiency of the feature selection search, specially in high-dimensional datasets. Next, we provide a simple mechanism to store the sums across all the iterations, so that the computation of the sums is not duplicated.

Let's define π_k as the accumulative sum of all $SU_{i',c}$ values for all $f_{i'} \in S$, with $k = \{1, 2, \ldots, |S|\}$. That is:

$$\pi_1 = SU_{1',c} \text{ , with } f_{1'} = \underset{f_i \in \mathbb{F}}{\operatorname{argmax}}\, SU_{i,c} \quad (4.3)$$

$$\pi_k = \pi_{k-1} + SU_{k,c} \text{ , with } f_k = \underset{f_i \in \mathbb{F} \setminus S}{\operatorname{argmax}}\{Cfs(S, f_i)\} \quad (4.4)$$

Now we transform the numerator of $Cfs(S, f_p)$ in Equation 4.2, based on π_k as follows.

$$\frac{1}{k}\sum_{i'=1}^{k-1} SU_{i',c} + \frac{1}{k} SU_{p,c} = \frac{\pi_{k-1}(k-1) + SU_{p,c}}{k} \quad (4.5)$$

Consequently, the numerator of $Cfs(S, f_p)$ now can be computed through an accumulative sum π_{k-1} across iterations of a greedy forward search. This means that with Equation 4.5, CFS only performs $(n-k)$ sums out of $k(n-k)$ sums performed by the original CFS in the k-th iteration.

Analogously, let's define λ_k as the accumulative sum of the SU values across all the pair of features in S for the k-th iteration with $k = \{1, 2, \ldots, |S|\}$. That is:

$$\lambda_1 = 0 \tag{4.6}$$

$$\lambda_2 = SU_{1',2'} \text{ , with } f_{2'} = \underset{f_i \in \mathbb{F} \setminus \{f_{1'}\}}{\operatorname{argmax}} Cfs(\{f_{1'}\} \cup \{f_i\}) \tag{4.7}$$

$$\lambda_k = \lambda_{k-1} + \sum_{i'=1}^{k-1} SU_{i',k} = \lambda_{k-1} + \sum_{i'=1}^{k-2} SU_{i',k} + SU_{k-1,k} \tag{4.8}$$

Also, let's denote λ_k^p as the sum of the SU values of feature f_p with every feature in S when we want to evaluate $S \cup \{f_p\}$, as:.

$$\lambda_k^p = \sum_{i'=1}^{k-1} SU_{i',p} \tag{4.9}$$

Therefore, the squared denominator of $Cfs(S \cup \{f_p\})$ is equivalent to:

$$k + k(k-1)(\lambda_{k-1} + \lambda_{k-1}^p + SU_{k-1,p}) \tag{4.10}$$

Since λ_{k-1} and λ_{k-1}^p are accumulative variables across the iterations, by using this new denominator CFS function only performs $(n-k)$ sums of SU values. Finally, we can rewrite CFS function as follows.

$$Cfs(f_p, k, \pi_{k-1}, \lambda_{k-1}, \lambda_{k-1}^p) = \frac{\frac{1}{k}\left(\pi_{k-1}(k-1) + SU_{p,c}\right)}{\sqrt{k + k(k-1)(\lambda_{k-1} + \lambda_{k-1}^p + SU_{k-1,p})}} \tag{4.11}$$

By replacing the original function $Cfs(S)$ by $Cfs(f_p, k, \pi_{k-1}, \lambda_{k-1}, \lambda_{k-1}^p)$, we can avoid recomputing $Cs(S)$ and $Rs(S)$ every time the evaluation of a feature f_p is required. The greedy forward search is in charge of storing and passing all the required parameters to the new CFS function. For a better understanding of our proposal Algorithm **??** depicts the new algorithm we call *Fast Correlation-based Feature Selection* (FCFS).

The first step in FCFS is to compute the SU values for each par $\langle f_i, C \rangle$ (lines 2-4). Then we add the feature f_i with higher $SU_{i,c}$ to the current solution S (line 5) and update the accumulative sum of SU values in Cs (line 7) as in Equation 4.4.

Algorithm 7 Fcfs

Require: Dataset D described by a feature set $\mathbb{F}$
Ensure: A feature subset S

1: $k = 1, m^* = -\infty$
2: **for all** $f_i \in \mathbb{F}$ **do**
3: $\quad SU_{i,c} = SU(f_i, C)$
4: **end for**
5: $S = \{f_{1'}\}$ where $f_{1'} = \underset{f_i \in \mathbb{F}}{\operatorname{argmax}} SU_{i,c}$
6: $\mathbb{F} = \mathbb{F} \setminus \{f_{1'}\}$
7: Set $\pi_1 = SU_{1',c}$
8: Set $\lambda_1^i = 0$ for all $f_i \in \mathbb{F}$
9: Set $\lambda_1 = 0$
10: **while** $k < n$ **do**
11: $\quad k = k + 1$
12: $\quad$ **for all** $f_i \in \mathbb{F}$ **do**
13: $\quad\quad \lambda_k^i = \lambda_{k-1}^i + SU_{i,k'-1}$
14: $\quad\quad m^i = Cfs(f_i, k, \pi_{k-1}, \lambda_{k-1}, \lambda_{k-1}^i)$
15: $\quad$ **end for**
16: $\quad f_{k'} = \underset{f_i \in \mathbb{F}}{\operatorname{argmax}} m^i$
17: $\quad$ **if** $m^{k'} > m^*$ **then**
18: $\quad\quad m^* = m^{k'}$
19: $\quad\quad S = S \cup \{f_{k'}\},\ \mathbb{F} = \mathbb{F} \setminus \{f_{k'}\}$
20: $\quad\quad \pi_k = \pi_{k-1} + SU_{k',c}$
21: $\quad\quad \lambda_k = \lambda_{k-1} + \lambda_{k-1}^{k'}$
22: $\quad$ **else break**
23: $\quad$ **end if**
24: **end whilereturn** S **=0**

In the second step (lines 10-24), in each iteration we look for the candidate feature f_i that maximizes Cfs. Note that in each iteration all the accumulative sums: λ_k^i (line13), π_k (line 20) and λ_k (line 21) are updated as suggested in Equations 4.9, 4.4 and 4.8 respectively.

4.3 Experimental evaluations of the Fast CFS

The aim of this section is to evaluate the new proposed algorithm in terms of efficiency with respect to the original in weka. Therefore, we focus on the comparison of CFS and FCFS in terms of running time. Furthermore, we use a Mac Book Pro (2012, Apple Inc.) with Intel Core i7 2.9 GHz processor and 8 GB memory. The experiments were conducted as follows.

First, we run both algorithms in fifteen medium-dimensional datasets and measure the running time for each. Table 4.1 shows the attributes of the datasets used in the experiments.

Table 4.1: Characteristics of the data used in the experiments

Dataset	Acronym	#Features	#Instances	#Classes	Source
MultiFeature	MFE	650	2000	10	[44]
Leukemia	LEU	7130	72	2	[60]
CentralNervous	CNS	7130	60	2	[60]
Tumors	TUM	7130	60	2	[44]
MLL	MLL	12583	72	3	[60]
Arcene	ARC	10001	39	2	[20]
TR21	T21	7903	336	6	[65]
TR45	T45	8262	690	10	[65]
BreastCancer	BRE	24482	97	2	[60]
Dexter	DEX	20001	300	2	[20]
StjudeLeukemia	STJ	12559	327	7	[60]
Ecml	ECM	27680	90	43	[44]
Gcm	GCM	16064	144	14	[60]
BurkittLymphoma	BUR	22284	220	3	[60]
Data3	DA3	22278	95	5	[63]
Data1	DA1	54676	123	2	[63]
Data4	DA4	54676	113	5	[63]
Data5	DA5	54614	89	4	[63]
Anthracycline	ANT	61360	159	2	[60]
Data6	DA6	59005	92	5	[63]
Mouse type	MOU	45102	214	7	[60]
Pems	PEM	138673	267	7	[44]
Dorothea	DOR	100001	800	2	[20]

We consider the first fifteen datasets as medium-dimensional, while the last eight datasets as high-dimensional. All the datasets where collected from three machine learning data repositories: Open Machine Learning [60], Machine Learning Data [44] and Tunedit [65]; and two feature selection challenges: NIPS'2003 [20] and RSCTC'2010 [63].

Second, we run FCFS in high-dimensional datasets, where CFS takes more than three days running without ending. With this experiment we will check whether or not FCFS is scalable to high-dimensional data. Third, we run both algorithms in each dataset and measure the running time taken in each iteration. Since the number of sums of SU values in CFS is related to the number of the current selected features k, we expect that the running time of CFS increases as k increases. While we expect that the running time of FCFS remains constant in each k.

Table 4.2 shows the running time in seconds for FCFS and CFS in each dataset. The difference between both algorithms is more remarkable as the data has larger number of features. This is because CFS performs, in each iteration, $k(n-k)((k-1)/2+1)$ sums of SU values, while FCFS only performs $2(n-k)$ sums.

Furthermore, we attempt to run CFS in high-dimensional datasets. However, after

Table 4.2: Running Time (in seconds) of FCFS and CFS in each dataset. AVE. stands for the average of the running time in the first fifteen datasets.

	MFE	LEU	CNS	TUM	MLL	ARC	T21	T45
FCFS	13.76	2.213	1.686	1.917	3.506	5.559	15.09	33.16
CFS	20.44	195.3	243.1	194.2	367.7	547.6	461.5	217.4
	BRE	DEX	STJ	ECM	GCM	BUR	DA3	AVE.
FCFS	31.226	43.607	65.492	31.226	19.894	40.506	20.684	20.908
CFS	19309	2913.1	2855.5	19309	6894.8	7050.8	12271	3884.9
	DA1	DA4	DA5	ANT	DA6	MOU	PEM	DOR
FCFS	75.305	110.76	153.46	149.03	144.47	162.39	177.03	3521.2
CFS	–	–	–	–	–	–	–	–

seventy-two hours running in each data we stop it with no results. Nonetheless, FCFS only took less than three minutes in each dataset, which definitely is a surprising result. Speaking about Dorothea dataset, besides is very high-dimensional, FCFS took around one hour in the selection process because 212 features were selected. This means that CFS may not be applicable to high-dimensional data when there are a large number of relevant features.

Figure 4.1 shows the ratio between the running time of CFS and FCFS for each medium-dimensional dataset. For the datasets Breast Cancer and GCM, FCFS is more than three hundred times faster than CFS, while in the datasets Data3 and ECM, it is more than six hundred times faster on average. In general, for medium-dimensional datasets, FCFS is around one hundred eighty times faster than CFS on average.

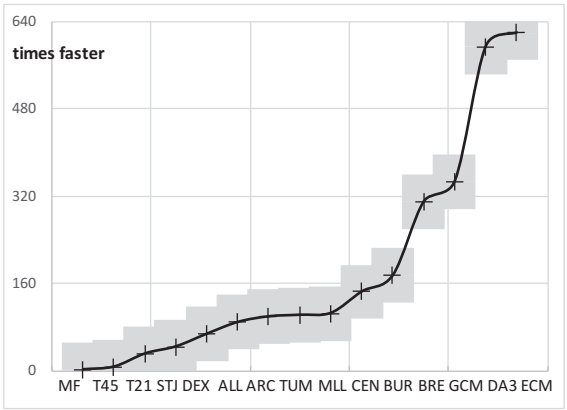


Figure 4.1: Ratio between running time of CFS and FCFS in different datasets. Gray shadow represents the standard deviation across six runs.

To see the performance of both algorithms in more detail, Figure 4.3 shows the running time per iteration in each medium-dimensional dataset. As depicted, running time of FCFS is constant across iterations. However, running time of CFS increases as k is larger. This phenomena is more visible in datasets with large number of features (last charts in the

figure).

4.4 MRMR+ and CFS+

One of the first feature selection algorithm that attempts to find a solution close to the ideal set is CFS. However, CFS is quadratic in terms of number of features evaluated. On the other hand, MRMR is able to select an approximate set of the ideal set by leveraging the *Mutual Information* metric and a greedy search that leads to the evaluation of only $(|\mathbb{F}| - \frac{q}{2})(q+1)$ pairs of features to select q features. This makes MRMR one of the most powerful algorithms in the history of feature selection. Although MRMR and CFS are one of the most popular algorithms in the field of feature selection, we have discovered that both are still subject to improvements in terms of efficiency.

The main goal of feature selection is to identify features 1) that have high correlation to the target class (relevance) but 2) low mutual relevance among them (redundancy). Although there are more than one methods to evaluate relevance and redundancy, the following shows a way that uses mutual information $I(X, Y)$ between two random variables X and Y. Indeed, we view features of a dataset as random variables and assume that features are associated with the empirical probability distributions derived from the dataset. For a set S with k features and the class variable C, the class relevance of S can be evaluated by

$$S = \frac{1}{k} \sum_{f \in S} I(f, C), \tag{4.12}$$

while the redundancy of S can be evaluated by

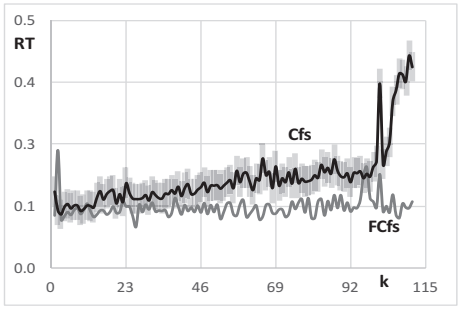
$$S = \frac{1}{k(k-1)} \sum_{(f,f') \in S^{(2)}} I(f, f'). \tag{4.13}$$

We let $S^{(2)} = S^2 \setminus \Delta$ for $\Delta = \{(f, f) \mid f \in S\}$. With relevance and redundancy, Peng et al. [45] have redefined the concept of feature selection as a process to find feature sets that maximize relevance and minimize redundancy. Because relevance and redundancy are in a trade off relation in general, this idea can be formulated as the optimization problem to find

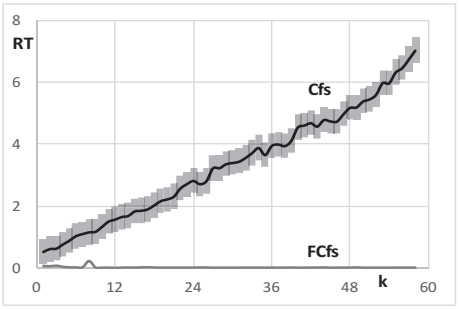
$$S^* \in \operatorname{argmax} \{S - \alpha S \mid S \subseteq \mathbb{F}\}. \tag{4.14}$$

The coefficient α is a parameter to adjust the balance between relevance and redundancy, and $\mathbb{F}$ represents the entire set of features that describes a dataset D but does not include the class variable C.

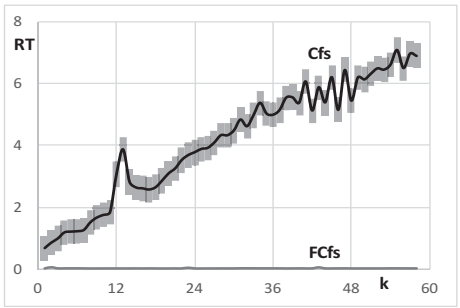
Finding an exact solution to this problem requires investigating the entire $2^{|\mathbb{F}|}$ subsets of $\mathbb{F}$, and the required computation is an exponential function of the size of $\mathbb{F}$. For this



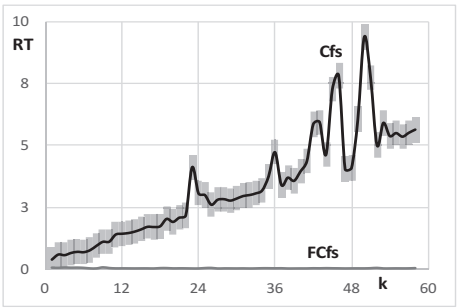
(a) FAC



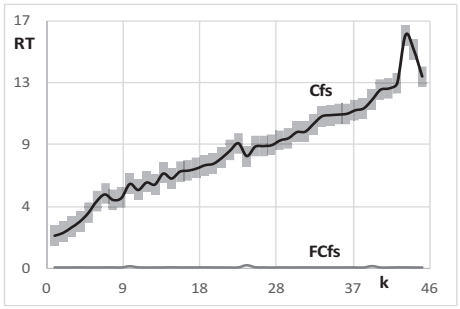
(b) FOUR



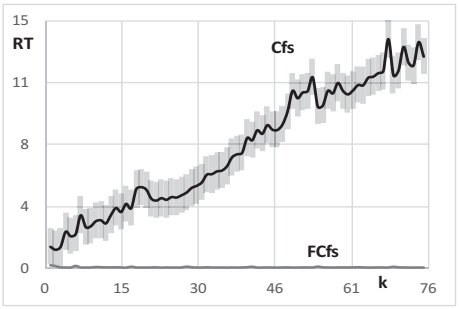
(c) FAR



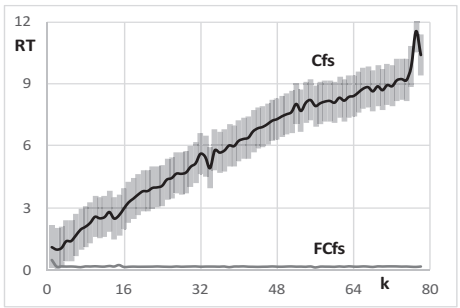
(d) MOR



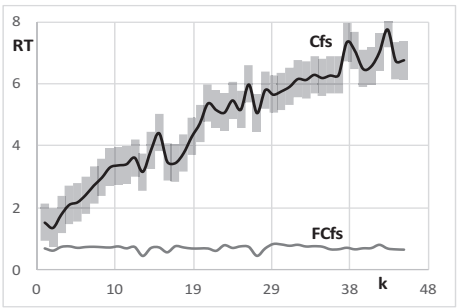
(e) PIX



(f) ZER

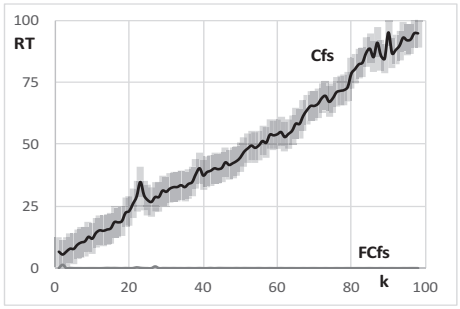


(g) SEM

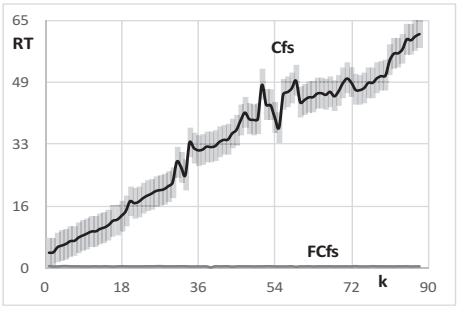


(h) WAV

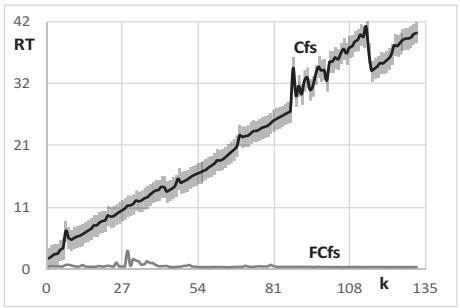
Figure 4.2: Running Time required by CFS (bold curve) and FCFS (gray curve) in each iteration of the greedy forward search. Thick gray curve represents the standard deviation of the results of CFS across six runs.



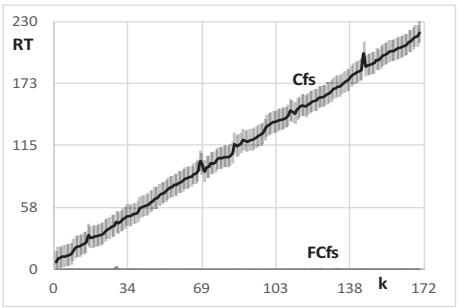
(a) FAC



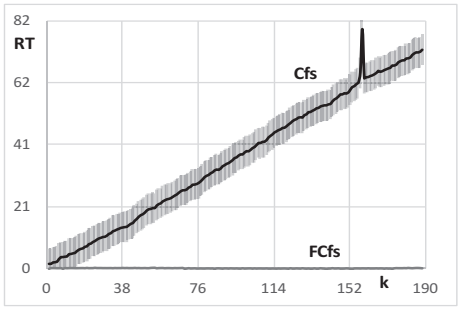
(b) FOUR



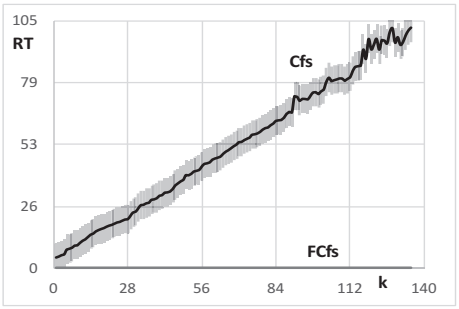
(c) FAR



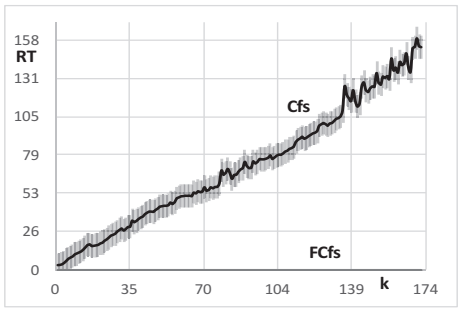
(d) MOR



(e) PIX



(f) ZER



(g) SEM

Figure 4.3: Running Time required by CFS (bold curve) and FCFS (gray curve) in each iteration of the greedy forward search. Thick gray curve represents the standard deviation of the results of CFS across six runs.

reason, it is common in practice to specify the number of features to select, say q. This reduces the computational complexity of the problem to $O(n^q)$, and if q is not too large, solving the problem is practical.

In the reality, however, we often need to cope with large datasets that include many features, and such large datasets tend not to be effectively described by a small number of features. Hence, to select features that sufficiently describe such datasets, we cannot help specifying a large q. Furthermore, for large datasets that includes many instances, computing $I(f, C)$ and $I(f, f')$ in $\cdot$ and $\cdot$ is time consuming.

In this regard, Peng et al. [45] have proposed the algorithm named the *Max-Relevance and Min-Redundancy* algorithm (MRMR), which finds approximate solutions to the aforementioned problem efficiently. Instead of evaluating $\cdot$ and $\cdot$, MRMR evaluates the *mutual information difference* measure $\mathrm{MID}_\alpha(\cdot, \cdot)$ defined as shown below:

$$\mathrm{MID}_\alpha(f, \emptyset) = I(f, C); \tag{4.15}$$

$$\mathrm{MID}_\alpha(f, S) = I(f, C) - \frac{2\alpha}{k} \sum_{f' \in S} I(f, f'). \tag{4.16}$$

The ground of investigating $\mathrm{MID}_\alpha(\cdot, \cdot)$ can be explained as follows. We assume that S has been determined and look for $f \in \mathbb{F} \setminus S$ that minimizes $\delta(f)$ defined by $\delta(f) = S \cup \{f\} - \alpha S \cup \{f\} - S + \alpha S$. Since we have

$$\begin{aligned}
\delta(f) = &-\frac{1}{k(k+1)} \sum_{f' \in S} I(f, C) + \frac{1}{k+1} I(f, C) \\
&+ \frac{2\alpha}{(k-1)k(k+1)} \sum_{(f', f'') \in S^{(2)}} I(f', f'') - \frac{2\alpha}{k(k+1)} \sum_{f' \in S} I(f, f') \\
= &\frac{1}{k+1} \cdot \mathrm{MID}_\alpha(f, S) + O(1/k^2),
\end{aligned}$$

maximizing $\delta(f)$ is approximately equivalent to maximizing $\mathrm{MID}_\alpha(f, S)$.

Algorithm 11 describes algorithm of MRMR. To be precise, the original algorithm of MRMR uses $\alpha = \frac{1}{2}$.

Algorithm 8 MRMR [45]

Require: Dataset D described by a feature set $\mathbb{F}$ and a number q of features to select.
Ensure: A feature subset $\{\bar{f}_1, \ldots, \bar{f}_q\} \subset \mathbb{F}$.

```
S = ∅
for k = 1, ..., q do
    f̄_k ∈ argmax{MID_α(f, S) | f ∈ F \ S}
    Add f̄_k to S.
end for
return S
```

MRMR takes the forward search approach, and hence, the variable S that holds the features selected at each iteration of the for loop (line 2 – 5) is initialized to the empty set (line 1). Then, for each iteration of the for loop, a single feature f that maximizes $\mathrm{MID}_\alpha(f, S)$ is added to S.

MRMR is also a greedy algorithm, and in fact, a feature that has been selected at an iteration will never be investigated again in the consequent iterations.

Since computation of mutual information is dominant in the computational complexity of the algorithm, we count the number of computation of mutual information to evaluate the computational complexity. At each iteration, the algorithm computes $\mathrm{MID}_\alpha(f, S)$ for $(n - k + 1)$ features, and $\mathrm{MID}_\alpha(f, S)$ includes k values of mutual information. Hence, the algorithm computes $(n - k + 1)k$ mutual information values at each iteration, and the total number of computing mutual information is

$$\sum_{k=1}^{q}(n - k + 1)k = \frac{(3n - 2q + 2)q(q + 1)}{6}, \tag{4.17}$$

where $n = |\mathbb{F}|$. This number is not small enough to perform feature selection on large datasts. For example, our experiment has shown that the widely used implementation of MRMR in Weka [62] requires 3,531 seconds, that is, almost an hour, to process a dataset that includes 100,000 features and 800 instances, which is relatively large but not very large.

The inefficiency of the current implementation of MRMR is due to duplication and redundancy when computing mutual information: the algorithm computes the same mutual information values more than one times; Also, it executes unnecessary computation of mutual information. In the next section, we will propose a new algorithm, named MRMR+, that improves the efficiency of the original MRMR significantly by solving the problems of duplication and redundancy.

4.4.1 The proposed algorithm: MRMR+

4.4.1.1 The ideas to solve the problems

We start with describing the two problems of MRMR of duplication and redundancy. For a better understanding of the explanation, we introduce the *partial mutual information information* functions $\mathrm{pMID}_{\alpha,f,S}(i)$ defined as

$$\mathrm{pMID}_{\alpha,f,S}(i) = I(f, C) - \frac{1}{k}\sum_{j=1}^{i} I(f, \bar{f}_j), \tag{4.18}$$

where $S = \{\bar{f}_1, \ldots, \bar{f}_k\}$. It is evident that $\mathrm{pMID}_{\alpha,f,S}(k) = \mathrm{MID}_\alpha(f, S)$ holds.

Duplication. When the same feature f is evaluated at the iterations to determine $\bar{f}_k$ and $\bar{f}_{k+1}$, Algorithm 11 computes the values of $I(f, \bar{f}_1), \ldots, I(f, \bar{f}_{k-1})$ duplicatedly.

Redundancy. Assume that the algorithm has determined $S = \{\bar{f}_1, \ldots, \bar{f}_k\}$ and has also finished the investigation of the features in $T \subset \mathbb{F} \setminus S$ at the iteration to determine $\bar{f}_{k+1}$. For $m^* = \max\{\mathrm{MID}_\alpha(f', S) \mid f' \in T\}$ and $f \in \mathbb{F} \setminus S \setminus T$, if $\mathrm{pMID}_{\alpha,f,S}(i) < m^*$ holds for $i < k$, the feature f cannot update m^*, and hence, it is redundant to compute $I(f, \bar{f}_j)$ for $j > i$, because $\mathrm{pMID}_{\alpha,f,S}(j)$ is a decreasing function with respect to j. Algorithm 11, nevertheless, computes the entire $I(f, \bar{f}_1), \ldots, I(f, \bar{f}_k)$.

To solve these problems, we introduce a two-dimensional array $A[\,][\,]$ to store pairs of a feature f and a sum $\sum_{i=1}^{j} I(f, \bar{f}_i)$. For a feature f, if j is the maximum integer such that $s = \sum_{i=1}^{j} I(f, \bar{f}_i)$ has been computed, the pair of (f, s) is an element of the array $A[j][\,]$. When the value of $s' = \sum_{i=1}^{j'} I(f, \bar{f}_i)$ is necessary for $j' > j$, we have only to compute $\sum_{i=j+1}^{j'} I(f, \bar{f}_i)$ by leveraging the value (f, s) stored in $A[j][\,]$. Thus, we can avoid the duplicated computation of mutual information. To solve the problem of redundancy, we skip computing $I(f, \bar{f}_{j+1}), \ldots, I(f, \bar{f}_k)$, whenever $I(f, C) - \frac{2\alpha}{k} \sum_{i=1}^{j} I(f, \bar{f}_i)$ becomes no greater than the current maximum m^* of $\mathrm{MID}_\alpha(f, S)$. The element $(f, \sum_{i=1}^{j} I(f, \bar{f}_i)$ is stored in $A[j][\,]$.

4.4.1.2 A description of the algorithm

Algorithm 9 shows an algorithm modified by adding the aforementioned mechanism to Algorithm 11 to avoid duplication and redundancy.

The following is a description of Algorithm 9.

- In the lines 1 to 3, the values of $I(f, C)$ for $f \in \mathbb{F}$ are computed and stored in the variables r_f.
- Each iteration of the outer **for** loop (lines 6 to 24) is to find $\bar{f}_{k+1}$ to add to $S = \{\bar{f}_1, \ldots, \bar{f}_k\}$. The variables f^* and m^* are to the feature that maximizes the MID value among the features investigated so far and the corresponding maximum MID value.
- Each iteration of the inner **for** loop (lines 8 to 22), the features in $A[j][\,]$ are investigated.
- The **while** loop (lines 11 to 13) updates $s' = \sum_{i=1}^{j'} I(f, \bar{f}_i)$ until $r_f - 2\alpha s'/k \leq m^*$ holds.
- If $\mathrm{MID}_\alpha(f, S) = r_f - 2\alpha s'/k > m^*$ holds (lines 14 to 16), the values of the variables f^* and m^* are updated.

Algorithm 9 MRMR+

Require: Dataset D described by a feature set $\mathbb{F}$ and a number of features q to select.
Ensure: A feature subset $\{\bar{f}_1, \ldots, \bar{f}_q\} \subset \mathbb{F}$.

for $f \in \mathbb{F}$ **do**
    $r_f = I(f, C)$ ▷ Compute mutual information
end for
$\bar{f}_1 \in \operatorname{argmax}\{r_f \mid f \in \mathbb{F}\}$
$A[0] = \{(f, 0) \mid f \in \mathbb{F} \setminus \{\bar{f}_1\}\}$
for $k = 1, \ldots, q-1$ **do**
    $(f^*, m^*) = (null, -\infty)$
    for $j = A.size() - 1, \ldots, 0$ **do**
        for $(f, s) \in A[j]$ **do**
            $(j', s') = (j, s)$
            while $j' < k$ **and** $r_f - 2\alpha s'/k > m^*$ **do**
                $(j', s') = (j' + 1, s' + I(f, \bar{f}_{j'+1}))$ ▷ Compute mutual information
            end while
            if $r_f - 2\alpha s'/k > m^*$ **then**
                $(f^*, m^*) = (f, r_f - 2\alpha s'/k)$
            end if
            if $j' > j$ **then**
                Remove (f, s) from $A[j]$.
                Add (f, s') to $A[j']$.
            end if
        end for
    end for
    $\bar{f}_{k+1} = f^*$
end for
return $\{\bar{f}_1, \ldots, \bar{f}_q\}$

- In addition, if $j' > j$ holds, that is, if at least one new value of $I(f, f.)$ has been computed (lines 17 to 20), the feature f is moved from $A[j][\,]$ to $A[j'][\,]$.

4.4.1.3 A thought experiment

In our experiments using real datasets (Section4.5), we see that our new algorithm MRMR+ can reduce the number of computing mutual information significantly, and as a result, can improve the run-time performance of MRMR. In this section, on the other hand, we conduct a simple thought experiment to verify the effectiveness of MRMR+. In the experiment, for simplicity, we assume the following properties of a dataset.

- All of the features except the class variable are mutually independent, that is, $I(f, f') = 0$ holds for any distinct features f and f'.
- No $I(f, C)$ is identical, that is, $I(f, C) \neq I(f', C)$ holds for any distinct features f and f'.

Under these assumptions, $\mathrm{pMID}_{\alpha,f,S}(i) = I(f, C)$ always holds.

When MRMR+ has selected $S = \{\bar{f}_1, \ldots, \bar{f}_k\}$, the number of features that the algorithm has to investigate in the iteration to determine $\bar{f}_{k+1}$ is $\ell = |\mathbb{F}| - k = n - k$. For some of them but not for all, the algorithm newly computes mutual information scores $I(f, \bar{f}_j)$. The number of such features is probabilistic depending on the order of investigating features.

To evaluate the expectation of this number, we let $p_{\ell,m}$ denote the probability that our algorithm newly computes mutual information scores for m of the entire ℓ features. To evaluate $p_{\ell,m}$, we consider a random permutation π of ℓ integers $\{1, \ldots, \ell\}$ and denote $\pi = (\pi(1), \ldots, \pi(\ell))$. We convert π into $(\mu(1), \ldots, \mu(2))$ by letting $\mu(i) = \min\{\pi(j) \mid j \leq i\}$ and determine $m(\pi) = |\{\mu(1), \ldots, \mu(\ell)\}|$. Then, we have $p_{\ell,m} = \Pr[m(\pi) = m]$ assuming a uniform random distribution for π.

The following evidently hold for all $\ell \geq 1$: $p_{\ell,m} = p_{\ell-1,m-1} + m \cdot p_{\ell-1,m}$; $p_{\ell,0} = 0$; and $p_{\ell,m} = 0$ for any $m > \ell$. Using this notation, the expected number N_ℓ of mutual information scores computed during the current iteration is evaluated by

$$N_\ell \leq \sum_{m=1}^{\ell} p_{\ell,m} \cdot m \cdot k,$$

because the algorithm computes new mutual information scores for m features with the probability $p_{\ell,m}$ and the number of the mutual information scores computed is bounded above by k. Hence, the expected number of the mutual information scores that the

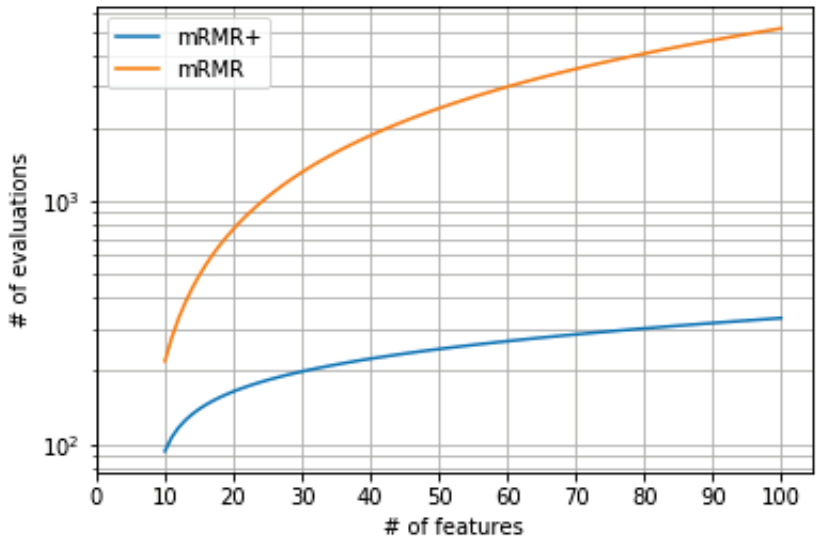


Figure 4.4: A thought experiment: comparison of MRMR+ and MRMR

algorithm computes entirely is bounded above by

$$\bar{N}_{n,q} = n + \sum_{k=1}^{q-1} N_{n-k} = n + \sum_{k=1}^{q-1} \sum_{m=1}^{n-k} p_{n-k,m} \cdot m \cdot k.$$

Figure 4.4 plots $\bar{N}_{n,q}$ as the number of evaluations for MRMR+ for $q = 10$ and $n = 10, 11, \ldots, 99, 100$. At the same time, the numbers given by Eq. (5.3) are plotted as the number of evaluations for MRMR in orange. Note that the y-axis is in a log scale. The values plotted for MRMR+ are just upper bounds, and the actual values should be smaller. Nevertheless, the values for MRMR+ is significantly smaller than those for MRMR. For example, when $n = 100$, the value for MRMR+ is 330.5, whereas that for MRMR is 5,170.

4.4.2 Extension of our proposal

We realized that the problem of duplication and redundancy are not only proper of MRMR algorithm. Must of the algorithms, which use greedy searches suffer from these problems. As an instance, in this section we will analyse the algorithm of *Correlation-based Feature Selection* (CFS).

The CFS evaluation function measures a set of features on the basis of: "A good feature subset contains features highly correlated with the class variable, yet uncorrelated to each other" [22]. The following equation determines the Cfs score when S contains k features.

$$Cfs(S) = \frac{k * Cs(S)}{\sqrt{k + k(k-1)Rs(S)}}, \tag{4.19}$$

where $Cs(S)$ and $Rs(S)$ denotes the average of the correlation between the features in S and the class variable, and the average of the correlation between each possible pair of features in S respectively. $Cs(S)$ and $Rs(S)$ in $Cfs(S)$ are computed by using the *Symmetrical Uncertainty* correlation function. CFS can be used with a wide variety of search strategies. However, according to the reviewed literature the basic greedy forward search is preferred to relieve the computational cost. The main difference between CFS and MRMR is that the stopping criteria of MRMR is fixed by the number of features to select while in CFS when there is no improvement of the current CFS score, the search is stopped.

4.4.2.1 Proposed algorithm: CFS+

With the top priority of developing a new strategy that yields to the reduction of the number of evaluations of CFS in a greedy forward search, we first define the relevance contribution of feature f to the current set S as:

$$Cfs(f,S) = \frac{\sum\limits_{f' \in S} SU(f',C) + SU(f,C)}{\sqrt{(k+1) + 2\Big(SR(S) + \sum\limits_{f' \in S} SU(f,f')\Big)}}, \tag{4.20}$$

where $SR(S)$ is the sum of the redundancy between every pair of features in S and is defined as:

$$SR(S) = \sum_{j=1}^{k-1} \sum_{l=j+1}^{k} SU(f'_j, f'_l) \tag{4.21}$$

Note that $Cfs(f,S)$ is equivalent to $Cfs(S \cup \{f\})$. Therefore, $Cfs(S) - Cfs(f,S)$ represents the effect of adding f to S. Now, we deeper derive the last term in the denominator of $Cfs(f,S)$ to realize that $\sum_{f' \in S} SU(f,f')$ represents the sum of the SU values between f and the features selected in the $p+1, p+2, \ldots, k$-th iterations.

$$Cfs(f,S) = \frac{\sum\limits_{j=1}^{k} SU(f'_j,C) + SU(f,C)}{\sqrt{(k+1) + 2\Big(SR(S) + \sum\limits_{j=1}^{p} SU(f,f'_j) + \sum\limits_{j=p+1}^{k} SU(f,f'_j)\Big)}} \tag{4.22}$$

Now, if we assume that feature f_i is not correlated to any feature in S (i.e. $\sum_{j=p+1}^{k} SU(f_i, \bar{f}_j) = 0$) then we obtain the upper bound of the Cfs score, in the k-th iteration, when only $SU(f_i, \bar{f}_0)$, $SU(f_i, \bar{f}_1)$, $\ldots$, $SU(f_i, \bar{f}_p)$ is known.

We define this upper bound as $Cfs_i^{p,k}$. Taking advantage of the knowledge of $Cfs_i^{p,k}$ we can safely avoid evaluating feature f_i, in the k-th iteration, when $Cfs_i^{p,k} \leq Cfs^*$ holds,

being Cfs^* the best Cfs score found so far in the current iteration. Figure 10 depicts the algorithm of CFS+.

Algorithm 10 Algorithm of CFS+

Require: Dataset D described by a feature set $\mathbb{F}$
Ensure: A feature subset $\{\bar{f}_1, \ldots, \bar{f}_t\} \subset \mathbb{F}$..

for $f \in F$ **do**
  $r_f = SU(f, C)$ ▷ Compute symmetrical uncert.
end for
$\bar{f}_1 = \text{argmax}_{f \in F}\, r_f$
$A[0] = \{(f, 0) \mid f \in F\} \setminus \{\bar{f}_1\}$
$sRel = r_{\bar{f}_1}\ ,\ sRed = 0$
for $k = 1 \ldots, n-1$ **do**
  $(f^*, m^*) = (null, -\infty)$
  for $j = A.size() - 1, \ldots, 0$ **do**
    for $(f, s) \in A[j]$ **do**
      $(j', s') = (j, s)$
      while $j' < k$ **and** $(sRel + r_f)/\sqrt{(k+1) + 2(sRed + s')} > m^*$ **do**
        $(j', s') = (j' + 1, s' + SU(\bar{f}_{j'}))$ ▷ Compute symmetrical uncert.
      end while
      $temp = (sRel + r_f)/\sqrt{(k+1) + 2(sRed + s')}$
      if $temp > m^*$ **then**
        $(f^*, m^*) = (f, temp)$
      end if
      if $j' > j$ **then**
        Remove (f, s) from $A[j]$
        Add (f, s') to $A[j']$
      end if
    end for
  end for
  $\bar{f}_{k+1} = f^*$
end for
return S

4.5 Results and Discussion

The aim of this section is to evaluate the new proposed algorithms in terms of efficiency and accuracy. Although we are very interested in the results of our proposal in microarray and cancer data anlaysis, we also use some other high-dimensional datasets within the field of text mining (datasets: Tr21, Tr41, Tr45, wap, Fbis, Tr31, New3s and Ohscal).

We split this section to report the experimental results into three parts: in the first part, we compare MRMR with MRMR+ in terms of running time and number of evaluations; In the second part, we compare MRMR+ with some benchmark feature selection algorithms in terms of running time and accuracy; Lastly, we test the effect of replacing the MRMR

Table 4.3: Characteristics of the datasets used in the experiments.

Data	Acronym	#Features	#Instances	#Classes	Source
Leukemia	LEU	7130	72	2	[60]
Central Nervous	CNS	7130	60	2	[60]
Tumors	TUM	7130	60	2	[44]
Dexter	DEX	20001	300	2	[20]
Arcene	ARC	10001	100	2	[20]
Tr21	T21	7903	336	6	[65]
Tr41	T41	7455	878	10	[65]
Tr45	T45	8262	690	10	[65]
Wap	WAP	8461	1560	20	[65]
Fbis	FBI	2001	2463	17	[65]
Stjude Leukemia	STJ	12559	327	7	[60]
Breast Cancer	BRE	24482	97	2	[60]
ECML	ECM	27680	90	43	[44]
Hepatitis C	HEP	22278	123	4	[44]
Burkitt Lymphoma	BUR	22284	220	3	[60]
La2s	LA2	12433	3075	6	[65]
La1s	LA1	13196	3204	6	[65]
Ohscal	OHS	11466	11162	10	[65]
New3s	NEW	26833	9558	44	[65]
Tr31	T31	10129	927	7	[65]
Data1	DA1	54676	123	2	[63]
Data4	DA4	54676	113	5	[63]
Data5	DA5	54614	89	4	[63]
Data6	DA6	59005	92	5	[63]
Anthracycline	ANT	61360	159	2	[60]
Mouse type	MOU	45102	214	7	[60]
Ovarian tumor	OVA	54622	283	3	[60]
Various Cancer	VAR	54676	383	10	[60]
Dorothea	DOR	100001	800	2	[20]
Pems	PEM	138673	267	7	[44]

filter with MRMR+ in the two-stage search algorithms.

Table 4.3 shows the attributes of the datasets used in the experiments that where collected from three machine learning data repositories: Open Machine Learning [60], Machine Learning Data [44] and Tunedit [65]; and two feature selection challenges: NIPS'2003 [20] and RSCTC'2010 [63]. Datasets in Table 4.3 are grouped in three groups according to their number of features and are classified in: low-dimensional, medium-dimensional and high-dimensional.

4.5.1 Comparing MRMR with MRMR+

We compare the running time and the number of evaluations of MRMR+ and CFS+ with their respective original versions. For the experiments, we fix the number of features to select by MRMR to fifty. Table 4.4 depicts the results for the running time and number of evaluations. Speaking of the running time, MRMR+ shows remarkable improvement compared with MRMR. In particular, in datasets like LA2, LA1, OHS and NEW, MRMR needs more than two minutes while MRMR+ only needs a few seconds. Results regarding CFS+ are not as good as MRMR+. However, it is remarkable that CFS+ outperforms CFS. Speaking of the number of evaluations, we can state that our method to avoid unnecessary sums of SU values works very good. However, for most of the datasets, MRMR+ outperforms CFS+ again.

Figure 4.7 describes (a) how many times the proposed algorithms are faster than their originals in run-time, and (b) how many percentages of the evaluations of the originals can be avoided when using the proposed algorithms. We see that the extent of improvement by MRMR+ to MRMR is significant. For most of the datasets, MRMR+ is more than ten times faster than MRMR, and MRMR+ executes only a few percentages of the computation that MRMR has to execute. Although the extent of improvement by CFS+ is less noticeable compared with the results for MRMR+, CFS+ is still around two to five time faster than CFS. Also, CFS+ is able to avoid more than forty percentages of the evaluations performed by CFS.

To go deeper in the results of our proposal, Figure 4.8 and 4.9 depicts the cumulative number of evaluations performed by the original CFS/MRMR (blue curve), MRMR+(orange curve) and CFS+(green curve) in each iteration for some of the datasets in Table 4.3. Note that vertical axes are represented in $\log_{10}$ scale.

In Figure 4.8 and 4.9, the curve CFS-MRMR represents the number of evaluation of both of MRMR and CFS. It is remarkable a drastic improvement in terms of number of evaluations when the original MRMR and CFS algorithms are compared with their improved methods. Another observation is that MRMR+ evaluates less feature sets than CFS+.

Table 4.4: Results for the running time and number of evaluations of the original and the proposed algorithms. Number of evaluations is expressed in 10^3 units.

Low-Dimensional Datasets										
	LEU		CNS		TUM		DEX		ARC	
	RTime	NEval	RTime	NEval	RTime	NEval	RTime	NEval	RTime	NEval
Mrmr	1.95	348	1.32	348	1.27	348	27.3	978	2.98	488
Mrmr+	0.66	9.65	0.26	10.3	0.19	10.2	4.21	2.85	0.28	3.68
Cfs	2.38	390	1.79	411	1.77	411	44.9	1756	14.4	747
Cfs+	1.43	106	0.52	88.9	0.69	81.9	5.73	112	4.9	236
	T21		T41		T45		WAP		FBI	
	RTime	NEval	RTime	NEval	RTime	NEval	RTime	NEval	RTime	NEval
Mrmr	8.81	385	25.6	364	21.3	403	51.3	413	12.1	96.7
Mrmr+	0.47	2.99	1.03	1.8	0.85	1.79	1.59	1.57	0.51	1.42
Cfs	26.3	589	4.27	37.2	21.3	370	8.43	16.9	1.71	9.98
Cfs+	6.14	184	0.75	1.48	1.21	13.2	1.08	.038	0.35	0.02
Medium-Dimensional Datasets										
	STJ		BRE		ECM		HEP		BUR	
	RTime	NEval	RTime	NEval	RTime	NEval	RTime	NEval	RTime	NEval
Mrmr	17.9	614	15.9	1198	14.8	1355	10.3	1090	20.6	1090
Mrmr+	1.05	4.47	5.73	14.7	8.95	1.83	6.29	21.3	6.79	30.5
Cfs	45.1	1636	24.3	2394	57.4	4690	87.3	3728	84.2	3220
Cfs+	25.2	392	12.8	1143	19.2	2805	26.9	2120	47.8	1396
	LA2		LA1		OHS		NEW		T31	
	RTime	NEval	RTime	NEval	RTime	NEval	RTime	NEval	RTime	NEval
Mrmr	126	607	139	645	320	560	906	1313	37.3	495
Mrmr+	3.10	1.6	3.29	1.5	9.54	1.22	28.8	1.54	1.43	2.6
Cfs	15.4	49.7	16.1	52.7	8875	16041	30.6	53.6	4.28	20.2
Cfs+	3.41	9.42	3.29	0.01	2023	6074	8.76	0.005	0.79	0.02
High-Dimensional Datasets										
	DA1		DA4		DA5		DA6		ANT	
	RTime	NEval	RTime	NEval	RTime	NEval	RTime	NEval	RTime	NEval
Mrmr	80.8	2677	92.9	2677	89.4	2674	145	2889	119	3005
Mrmr+	41.7	81.3	56.1	29.1	60.1	67.5	61.3	23.6	51.7	11.8
Cfs	85.1	5789	115	9334	126	11065	144	12603	141	7845
Cfs+	24.9	1977	33.2	4709	76.5	6030	73.2	6991	41.6	936
	MOU		OVA		VAR		DOR		PEMS	
	RTime	NEval	RTime	NEval	RTime	NEval	RTime	NEval	RTime	NEval
Mrmr	176	2208	233	2675	262	2677	3531	4898	2724	1724
Mrmr+	24.4	31.4	41.8	41.1	50.5	35.9	211	3.39	62.3	4.27
Cfs	193	8865	459	14384	766	14997	3919	21177	192.2	277.3
Cfs+	126	4911	103	5741	402	8647	752.8	2475	82.34	8.57

Table 4.5: Accuracy and Running time of several benchmark feature selection algorithms

	SVM			Running Time			
	S^3F	SFS-LW	MRMR	S^3F	SFS-LW	MRMR	MRMR+
STJ	.729	.739	.743	12.9	18.3	22.4	1.03
BRE	.845	.889	.889	15.1	20.5	17.8	5.22
ECM	.814	.801	.759	17.3	12.1	14.6	8.46
HEP	.807	.796	.811	21.2	13.8	13.1	6.54
BUR	.902	.934	.910	31.5	17.6	22.5	3.85
LA2	.944	.948	.950	341	89.1	124	3.26
LA1	.863	.857	.861	429	145	131	4.28
OHS	.704	.704	.709	521	287	331	8.46
NEW	.855	.874	.874	1206	789	906	25.6
T31	.741	.727	.751	128	48.9	32.6	1.23
DA1	.712	.712	.707	391	76.4	83.7	54.2
DA4	.892	.920	.934	207	78.1	93.5	62.6
DA5	.689	.708	.671	408	84.8	88.6	61.9
DA6	.722	.741	.731	213	98.6	137	48.7
ANT	.843	.811	.859	789	168	122	51.6
MOU	.732	.770	.775	870	178	173	22.2
OVA	.842	.758	.758	801	201	237	49.2
VAR	.927	.943	.943	798	229	266	50.7
DOR	.730	.708	.783	-	4821	3504	208
PEM	.924	.917	.921	-	2980	2698	58.9

4.5.2 Comparing Mrmr+ with benchmark algorithms

In this section we compare MRMR+ with two benchmark feature selection algorithms: the LW-index (SFS-LW) [38] and the Supervised Simplified Silhouette Filter (S^3F) [11].

To compare MRMR+ with SFS-LW and S^3F algorithms we run experiments and collect the running time and the accuracy of each algorithm. For each dataset, we generate m pair of training and test data subset, where m is the number of instances in the data. Each test data represents an instance in the dataset and the train data is composed by the rest of the instances. First, we run the feature selection algorithms in the training data and apply then we reduce the test data according to the features eliminated by the algorithms. Second, we train the C-SVM-with-RBF-Kernel with the reduced training data and then test it on the reduced test data to determine the Area Under the Receiver Operating Characteristic curve (AUC-ROC) score. In addition, the average of the running time taken by each algorithms when is applied to the train data is collected.

From the results depicted in Table 4.5, we observe that MRMR wins in twelve of the twenty datasets used, in terms of accuracy. However, MRMR is slower than SFS-LW in the majority of the datasets. Nonetheless, exhibiting the same results of MRMR, in terms of accuracy, MRMR+ is on average fourteen times faster than SFS-LW.

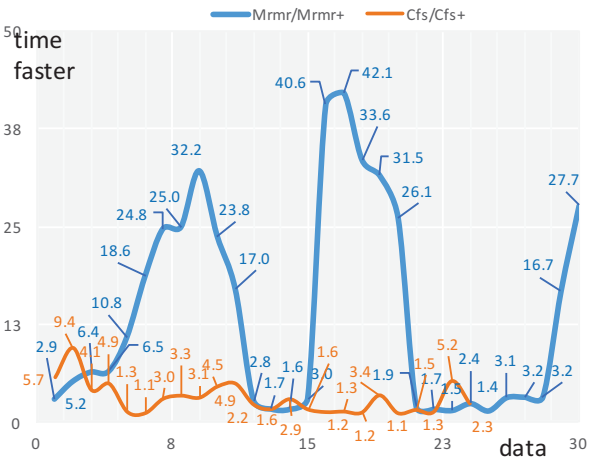
4.5.3 Testing MRMR+ in the two-stage selection algorithms

A two-stage feature selection algorithm separates its job into two stages that aim at (1) gradual reduction of number of features using a fast algorithm and (2) final and finer selection of features using a slow but powerful algorithm. The first stage narrows down the search space so that even the slow algorithm at the second stage can find answers within a reasonable time allowance. Usually, a filter-type algorithm is selected for the first stage algorithm, and in particular, MRMR has been extensively used as in the literature because of its efficiency and accuracy [4][25][5][17][16]. However, as was shown in Section 4.5.1, when the dataset is high-dimensional, MRMR is likely to be too slow. Therefore, in this section, we investigate the effect of replacing MRMR with MRMR+ with two popular instances of the two-stage selection algorithm: Genetic Bee Colony (GBC) [4] and MRMR-GA [16].

To test the effect of replacing MRMR with MRMR+ in the two-stage selection methods, we run experiments using both filters. In table 4.6, the results are shown. The columns labeled with MRMR$_{GB}$ and MRMR+$_{GB}$ specify the running time of MRMR and MRMR+ in the GBC algorithm, while the columns labeled with MRMR$_{GA}$ and MRMR+$_{GA}$ do the running time of MRMR and MRMR+ in the MRMR-GA algorithm. The columns of GBC, GBC+, GA and GA+ specify the total running time of the corresponding entire two-stage algorithms. It is remarkable that in most of the datasets the time required by MRMR represents more than the fifty percentage of the total running time of GBC and MRMR-GA. Therefore, introducing MRMR+ to both algorithms results in substantial improvement of the efficiency in GBC and MRMR-GA. In the context of GBC, for the first eighteen datasets, GBC+ is approximately three times faster than the original GBC on average. For GBC we could not have an answer, for the datasets DOT and PEM, after running the algorithm for more than forty-eight hours. Speaking about MRMR-GA, MRMR-GA+ is around four times faster than the original MRMR-GA on average. Moreover, in the datasets DOR and PEM, which are very high-dimensional, MRMR-GA+ is 11 and 3.5 times faster than the original MRMR-GA, respectively.

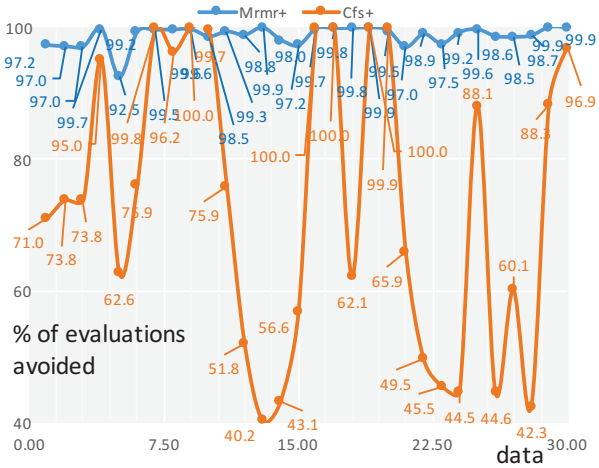
Table 4.6: Running time taken by MRMR and MRMR+ in GBC and MRMR-GA.

	MRMR$_{GB}$	GBC	MRMR+$_{GB}$	GBC+	MRMR$_{GA}$	GA	MRMR+$_{GA}$	GA+
STJ	31.1	38.9	7.45	16.1	84.3	102	13.5	31.4
BRE	16.2	22.9	4.71	9.64	58.7	72.8	14.02	28.5
ECM	31.1	49.3	19.7	35.3	32.9	53.0	17.81	37.6
HEP	8.01	29.4	4.92	21.6	22.4	52.1	9.34	36.8
BUR	23.3	156	4.87	136	21.4	47.6	5.31	30.4
LA2	1290	1341	83.2	148	132	194	31.7	90.4
LA1	2519	2977	91.6	503	139	201	25.4	83.0
OHS	345	432	17.2	103	215	267	29.3	83.3
NEW	3451	4420	82.7	1029	2437	3012	61.5	686
T31	34.2	64.3	2.54	29.7	89.4	145	6.44	63.2
DA1	934	1271	206	541	203	298	69.9	167
DA4	318	571	51.1	296	221	322	86.4	191
DA5	421	929	32.7	541	184	271	107	203
DA6	1473	2927	219	1782	309	438	103	229
ANT	3732	5213	291	1697	378	522	81.6	221
MOU	1349	2168	154	924	382	471	39.3	146
OVA	569	921	31.7	390	469	635	80.2	240
VAR	2981	3172	149	382	671	789	109	227
DOR	-	-	619	1134	8701	8994	505	801
PEM	-	-	514	921	5167	6926	109	1954



0.45

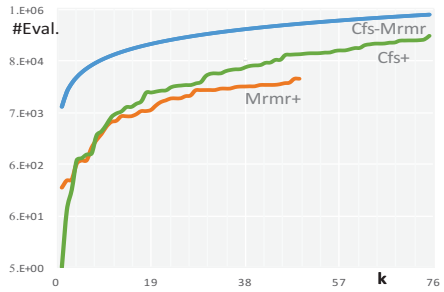
Figure 4.5: The factor of improvement in run-time



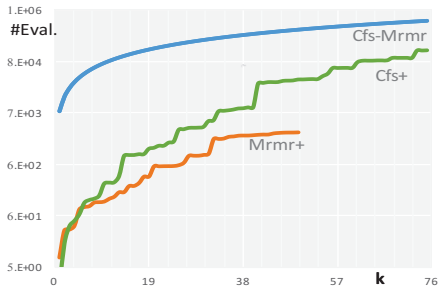
0.45

Figure 4.6: % of evaluations avoided by the improved algorithms

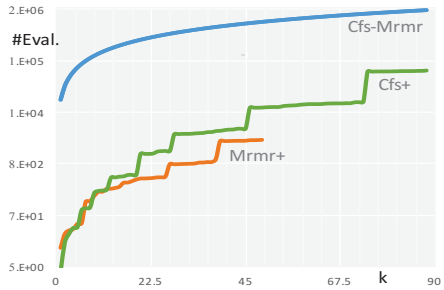
Figure 4.7: Extent of improvement of MRMR+ and CFS+. The blue curve represents the comparison between MRMR+ and MRMR, while the orange curve does the comparison between CFS+ and CFS algorithm. The horizontal axis represents datasets: 1:LEU 2:CNS 3:TUM 4:DEX 5:ARC 6:T21 7:T41 8:T45 9:WAP 10:FBI 11:STJ 12:BRE 13:ECM 14:HEP 15:BUR 16:LA2 17:LA1 18:T31 19:OHS 20:NEW 21:DA1 22:DA4 23:DA5 24:DA6 25:ANT 26:MOU 27:OVA 28:VAR 29:DOR 30:PEMS.



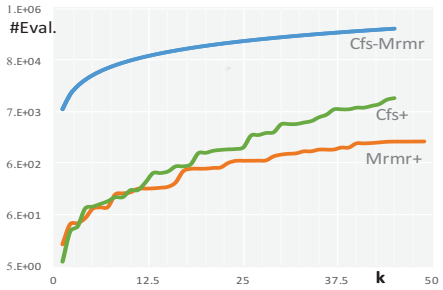
(a) FAC



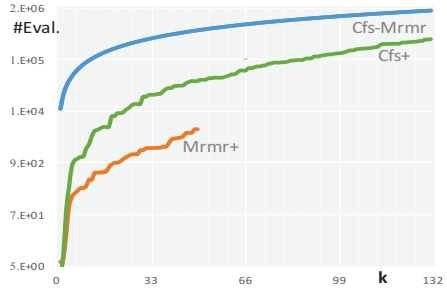
(b) FOUR



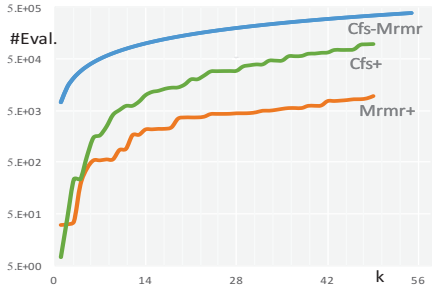
(c) FAR



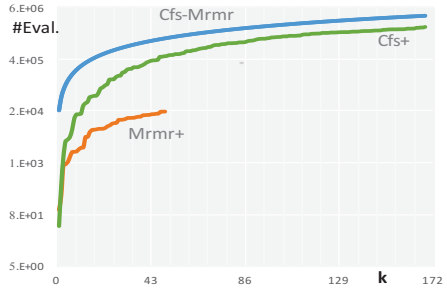
(d) MOR



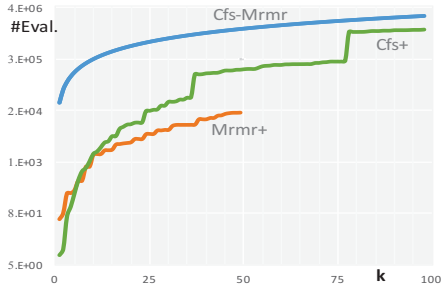
(e) PIX



(f) ZER

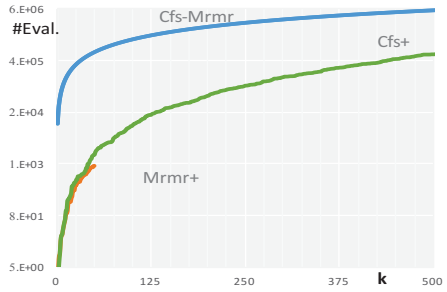


(g) SEM

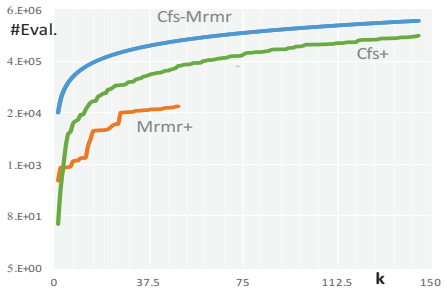


(h) WAV

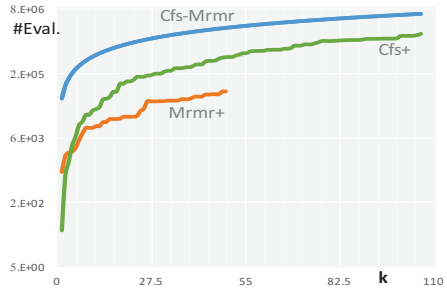
Figure 4.8: Cumulative function of the number of evaluations required in each iteration by CFS/MRMR(blue curve), CFS+(green curve) and MRMR+(orange curve). Vertical axis is expressed in $\log_{10}$ scale.



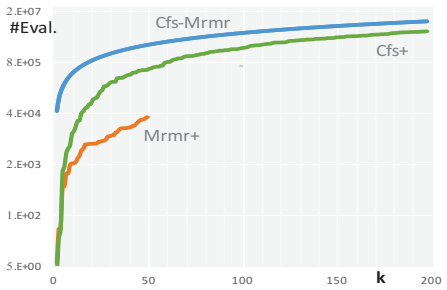
(a) FAC



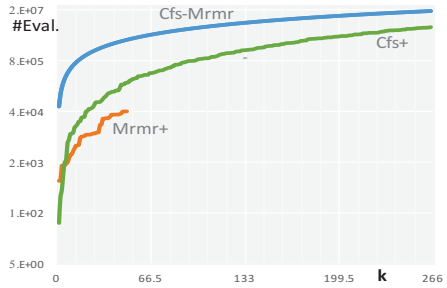
(b) FOUR



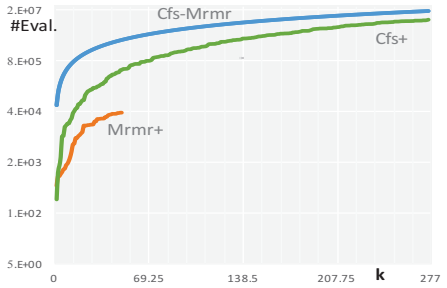
(c) FAR



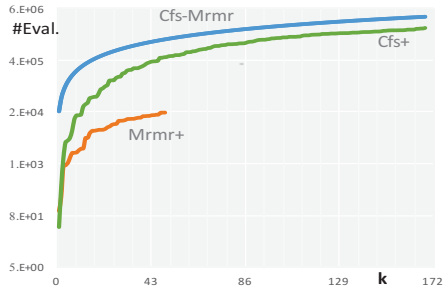
(d) MOR



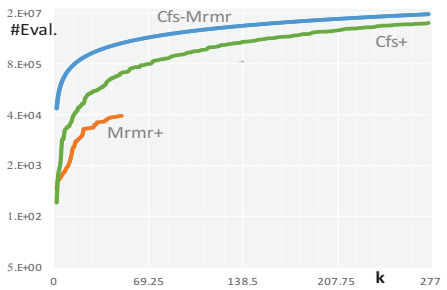
(e) PIX



(f) ZER



(g) SEM



(h) WAV

Figure 4.9: Cumulative function of the number of evaluations required in each iteration by CFS/MRMR(blue curve), CFS+(green curve) and MRMR+(orange curve). Vertical axis is expressed in $\log_{10}$ scale.

Chapter 5

Third Contribution: Improvement of Efficiency and Accuracy of Two-stage Algorithms

Since we are especially interested in high-dimensional data, we made some modifications to the GBC algorithm, so that GBC could be effectively use in high-dimensional domains. In this section, we present some improvements made to the GBC algorithm in terms of efficiency and accuracy.

5.1 The MRMR algorithm

The main goal of feature selection is to identify features 1) that have high correlation with the target class (relevance) but 2) low mutual relevance among them (redundancy). Peng et al. [45] have proposed the algorithm named the *Max-Relevance and Min-Redundancy* algorithm (MRMR), which finds approximate solutions to the aforementioned problem efficiently. MRMR evaluates each subset of genes by the *Mutual Information Difference* measure $\mathrm{MID}_\alpha(\cdot,\cdot)$ defined as shown below:

$$\mathrm{MID}_\alpha(f,\emptyset) = I(f,C); \tag{5.1}$$

$$\mathrm{MID}_\alpha(f,S) = I(f,C) - \frac{2\alpha}{k}\sum_{f'\in S} I(f,f'), \tag{5.2}$$

where $I(f,f')$ represents the Mutual Information between the two genes f and f'. MRMR takes the forward search approach, and hence, the variable S that holds the features selected at each iteration of the for loop (line 2 – 5) is initialized to the empty set (line 1). Then, for each iteration of the for loop, a single feature f that maximizes $\mathrm{MID}_\alpha(f,S)$ is added to S.

MRMR is used in GBC as a filter to remove redundant and irrelevant genes prior to the

Algorithm 11 MRMR [45]

Require: Dataset D described by a feature set $\mathbb{F}$ and a number q of features to select.
Ensure: A feature subset $\{\bar{f}_1, \ldots, \bar{f}_q\} \subset \mathbb{F}$.

```
S = ∅
for k = 1, ..., q do
    f̄_k ∈ argmax{MID_α(f, S) | f ∈ \S}
    Add f̄_k to S.
end for
return S
```

search. Although MRMR drastically reduces the search space, we have found that the time taken by MRMR is extremely large in relation to the total running time of GBC. Figure 5.1 depicts the percentage of the running time of MRMR in GBC (lighter area), and the percentage of the running time of the rest of the algorithm (darker area). The datasets used, are six microarray datasets from the Rough Sets and Current Trends in Computing conference (RSCTC'2010) discovery challenge [64]. The datasets in Figure 2.13 are sorted according to their number of genes. As can be seen, for the first three datasets, which have fewer genes, the running time of MRMR is nearly to the fifty percentage of the entire running time of the GBC algorithm, while for the remain of the datasets, the running time of MRMR represents more than the sixty percentage of the total running time.

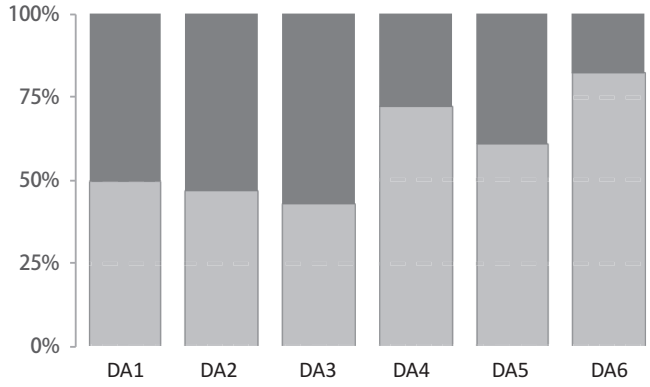


Figure 5.1: Percentage of time required by MRMR (the lighter area) in the GBC algorithm.

In fact, at each iteration, MRMR computes $\mathrm{MID}_\alpha(f, S)$ for $(n - k + 1)$ features, and $\mathrm{MID}_\alpha(f, S)$ includes k values of mutual information. Hence, the algorithm computes $(n-k+1)k$ mutual information values at each iteration, and the total number of computing mutual information is

$$\sum_{k=1}^{q}(n - k + 1)k = \frac{(3n - 2q + 2)q(q + 1)}{6}, \tag{5.3}$$

where $n = |\mathbb{F}|$. This number is not small enough to perform gene selection on large

microarray datasets.

In chapter 4, we present a solution to the inefficiency by proposing the algorithm MRMR+, which find the same set as MRMR+, but in a significantly shorter time. To test the effect of replacing MRMR with MRMR+ over GBC, we run experiments in the RSCTC'2010-challenge-datasets and collect the running time of both versions as shown in Figure 5.2.

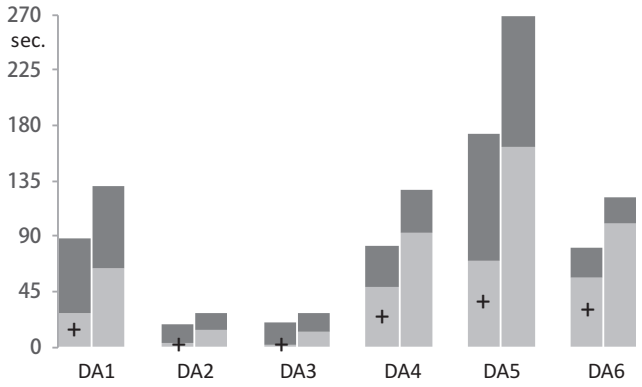


Figure 5.2: Comparison in running time (in seconds) in the GBC algorithm when using the original MRMR and the faster MRMR+ in the filter phase. The area with the + symbol represents the MRMR+ while the the darker area is the running time of the rest of the GBC algorithm. Results of the original GBC is on the right of the MRMR+.

In all cases MRMR+ is more than two times faster than the original MRMR, which significantly improves the running time of GBC.

5.2 Initialization Phase

Most population-based techniques for solving optimization problems in artificial intelligence, generate the first solutions of the population randomly. While this is essential to ensure diversity at the early stage of the search, it also may affect the convergence speed of the algorithm. GBC, in the *Initialization Phase* does not make use of neither the relevance nor the redundancy scores of each gene to build the SN initial solutions. Instead, GBC generates the solutions randomly. Since we work with high-dimensional microarray data, we are very interested in making GBC to converge faster towards the most promising solutions. Therefore, in the *Initialization* phase we propose to make use of the relevance and redundancy score of each gene computed, which are computed by MRMR, to efficiently create an initial population composed by diverse, but accurate solutions.

We define our proposal as follows. Given a set of all genes sorted according to the order they were selected by MRMR, we randomly select a feature and test it in the current solution (initially the empty solution), if the accuracy of the current solution is increased,

then we add the gene, otherwise we stop the search and start creating a new solution by the same procedure. This is in fact, the same procedure GBC uses to create the initial population. However, additionally we assign to each gene g_i a probability $P_{\gamma_j}(i)$ to be selected when creating the j-th solution, as follows:

$$P_{\gamma_j}(i) = \frac{1-\gamma_j}{1-\gamma_j^n} \times \gamma_j^{i-1}, \tag{5.4}$$

where n is the number of features in the data and γ_j is a decreasing function in the range of $(0, 1)$, as follows:

$$\gamma_j = 1 - \frac{1}{j+1} \tag{5.5}$$

Figure 5.3 represents the shape of the probability function $P_{\gamma_j}(i)$. As can be inferred from Figure 5.3, $P_{\gamma_j}(i)$ has several properties:

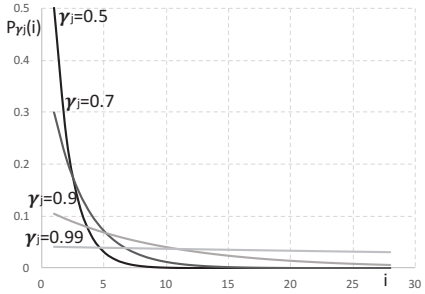


Figure 5.3: Chart representing the probability of choosing feature f_i to be tested in the j-th solution.

- First, as a probability function, $P_{\gamma_j}(i)$ is exhaustive, that is: $\sum_{i=1}^{n} P_{\gamma_j}(i) = 1$.
- Second, $P_{\gamma_j}(i)$ is a decreasing function. Therefore,

  $$P_{\gamma_j}(1) > P_{\gamma_j}(2) > \cdots > P_{\gamma_j}(n), \tag{5.6}$$

  always holds. This means that first genes in the ranking are likely to be selected. Genes will be ranked in the same order they were selected by the MRMR algorithm. Therefore, the first genes in the ranking, are highly correlated with the class variable and are not highly correlated with other features in the ranking.
- Third, the larger the number of solutions already built, the more equal are the probabilities to be chosen for all genes. That is,

  $$P_{\gamma_t}(i) > P_{\gamma_{t+1}}(i) > \cdots > P_{\gamma_{SN}}(i). \tag{5.7}$$

However, for a sufficiently large value of j, for example with $SN/2 \leq k \leq SN$,

$$P_{\gamma_k}(1) \approx P_{\gamma_k}(2) \approx \cdots \approx P_{\gamma_k}(n), \tag{5.8}$$

holds.

$P_{\gamma_j}(i)$ guaranties that the first solutions are likely to contain genes with high correlation with the class and low correlation with the other genes. Therefore, this solutions may have high accuracy. For the rest of the solutions the probability of selecting any gene tends to $1/SN$. Consequently, we can expect that the initial population will be composed by two type of solutions: solutions with high accuracy and random solutions. This creates a good synergy in the search since now we have a diverse population with some accurate solutions that may make the algorithm converge faster. To evaluate our proposed method, we run experiments in several datasets and measure the accuracy of the solutions in the population of the original method (gray curve) and the proposed (black curve). Figure 5.4 depicts the results. To build the chart we sorted the solutions according to their accuracy for both method.

5.3 Intensification

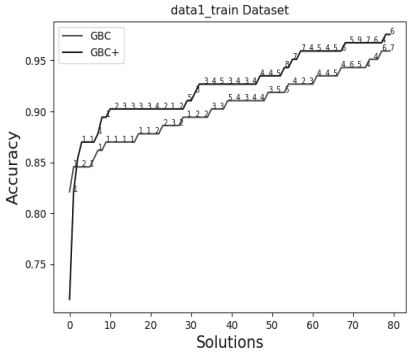
Metaheuristic optimization algorithms often performs well approximating solutions because they first, diversify the search looking for candidate solutions without making any assumption about the underlying fitness landscape. Second, they intensify the search looking for more promising solutions once they explore diverse regions in the solution space. In GBC, the intensification process is accomplished by means of the genetic crossover and mutation operations in the *onlooker bee* and *scout bee* phases, respectively. However, through experiments we realized that in the *scout bee* phase the mutation operation does not make any effect in the intensification process due to the extremely low mutation probability of genes.

To carry out a richer intensification process we adopt the following method.

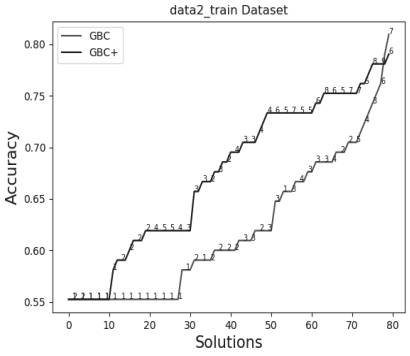
- First, we determine the goodness of a gene according to its occurrence in the solutions of the population. A gene $f_i \in S$ that is in solutions S, with $SVM(S) > \mu$ and is not in solutions R, with $SVM(R) \leq \mu$, must have high goodness. We define the goodness $\ell(f_i)$ of feature f_i as follows.

$$\ell(f_i) = \frac{\sum\limits_{S \in P} \delta^{+}_{f_i,\mu} \times SVM(S)}{\theta^{+}} - \frac{\sum\limits_{S \in P} \delta^{-}_{f_i,\mu} \times SVM(S)}{\theta^{-}}, \tag{5.9}$$

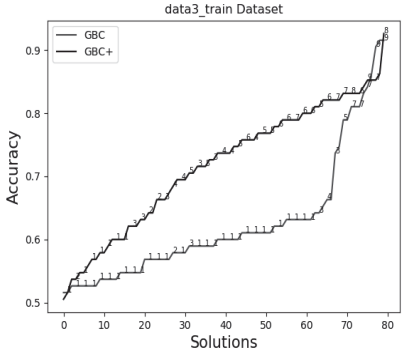
  where $\delta^{+} = 1$ if $\delta^{+} > \mu$ and $\delta^{+} = 0$ if $\delta^{+} \leq \mu$, being μ the average of the fitness



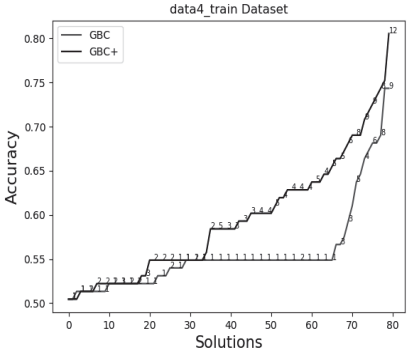
(a) FAC



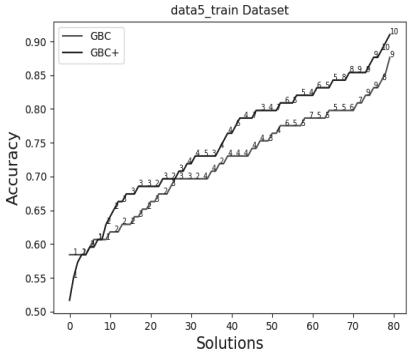
(b) FOUR



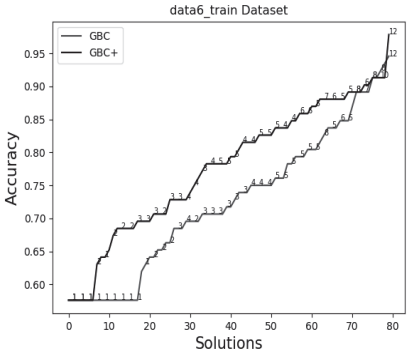
(c) FAR



(d) MOR



(e) PIX



(f) ZER

Figure 5.4: Accuracy of the solutions in the population of the original method (black) and the proposed method (gray curve). Number of selected genes are located over each solution.

of all solutions in the population, δ^- has opposite value to δ^+, and θ^+ and θ^- are the sum of the fitness of all solutions S in the population such that $SVM(S) > \mu$ and $SVM(S) \leq \mu$,respectively. $\ell(f_i)$ is a normalized coefficient in the range of $[-1, 1]$. A value of 1 means gene f_i is in all solutions where $\delta^+_{f_i,\mu} = 1$ and is not in any solution where $\delta^-_{f_i,\mu} = 1$. A value of -1 means f_i is present in all solutions where $\delta^-_{f_i,\mu} = 1$ and not in any solutions with $\delta^+_{f_i,\mu} = 1$.

- Second, we sort the genes according to their goodness ℓ and store them in two different sets. Genes with $\ell(f_i) > 0$ are sorted in increasing order and stored in S^+ while genes with $\ell(f_i) \leq 0$ are sorted in decreasing order and stored in S^-. Afterwards, we run a greedy forward selection search starting with the *Queen Bee* solution and using genes in S^+. That is, we test adding to *Queen Bee*, all genes in $S+$, one by one, and the gene that maximize $SVM(QueenB \cup \{f_i\})$ is added to $QueenB$. We stop searching when neither of the genes improves the current $QueenB$. Subsequently, a greedy backward search is performed using the genes in S^-. That is, we start with the current *Queen bee*, and test eliminating genes in S^- from $QueenB$, if present. The gene that maximize $SVM(QueenB \setminus \{f_i\})$ is removed from $QueenB$. The search stops when no feature improves $QueenB$.

To test our method, we run GBC twice for each dataset: first, we run the original GBC and second we replace the mutation operation with our proposed intensification method. Figure 5.5 depicts the results.

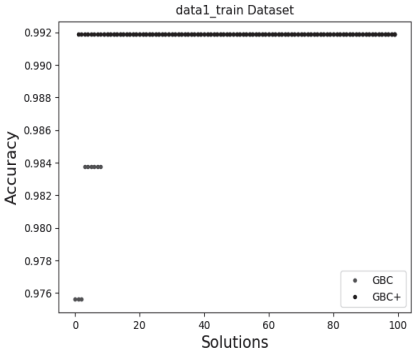
5.4 Minor improvements

In addition to the improvements we have proposed in the previous sections, we have detected some small gaps on the design of the original GBC, that could lead to undesirable results. Our final proposals is as follows.

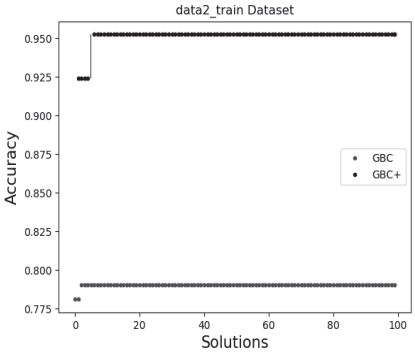
- First, we consider two more stopping criteria in the GBC algorithm: i) stopping the search when $SVM(QueenB) \geq \lambda$, and ii) stopping the search when the current $QueenB$ remains the same after t consecutive cycles.

- Second, we implement, a *lookup table* that stores the solutions already evaluated and their respective accuracy. Every time a candidate solution is going to be evaluated, first we inspect the lookup table to avoid duplicated evaluations.

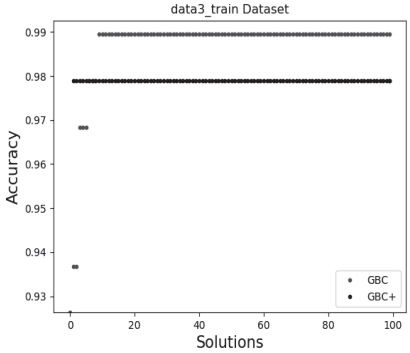
These modifications may look naive. However, we run experiments to determine the evaluations saved by implementing these modifications in the original GBC algorithm and results were very positive. Table 5.1 shows the results. We report that in neither of the datasets the accuracy varies with respect to the original GBC.



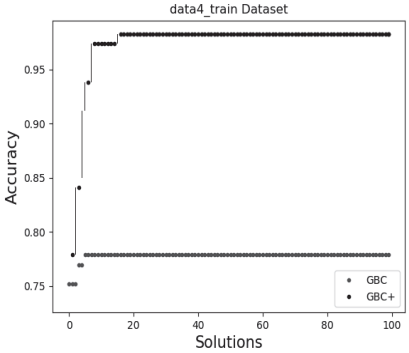
(a) FAC



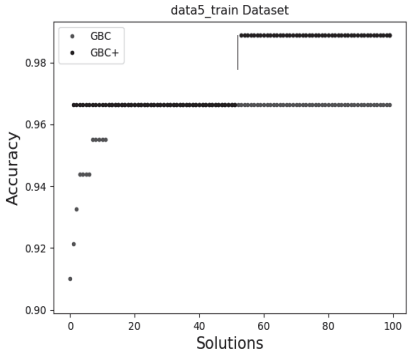
(b) FOUR



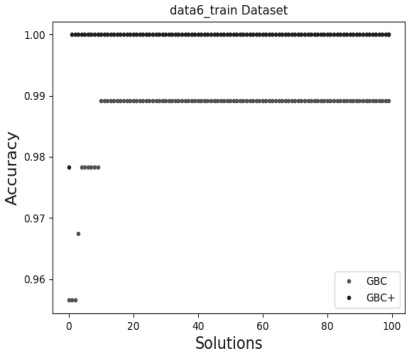
(c) FAR



(d) MOR



(e) PIX



(f) ZER

Figure 5.5: Comparison between the original GBC (Gray points) and GBC with the proposed method of intensification (black points). The line between two black points quantifies an improvement in the *Queen Bee* by the proposed method.

Table 5.1: Percentage of time saved by the minor improvements respecto to the original GBC algorithm. Values are expressed as % of number of evaluation saved / percentage of running time saved. AVE. stands for average.

	stopping criteria		lookup table	all
	i)	ii)		
DA1	79 / 41	-	94 / 55	81 / 44
DA2	75 / 73	-	41 / 40	85 / 85
DA3	64 / 62	-	51 /50	84 / 84
DA4	64 / 52	-	72 / 63	88 / 73
DA5	60 / 27	-	76 / 52	73 / 52
DA6	63 / 26	92 / 68	93 / 61	86 / 53
AVE.	67 / 47	92 / 68	71 / 53	83 / 65

Table 5.2: Accuracy of GBC and GBC+ in several datasets.

Alg\Data	STJ	BRE	ECM	HEP	BUR	DA1	DA2	DA3
Gbc	0.76	0.94	1	1	0.96	0.99	0.93	0.97
Gbc+	0.94	0.96	1	1	0.98	0.99	0.97	1
	DA4	DA5	DA6	ANT	MOU	OVA	VAR	PEM
Gbc	0.85	0.98	1	0.87	0.68	0.96	0.89	-
Gbc+	0.98	0.98	1	0.99	0.99	0.98	0.99	0.87

5.5 Experimental evaluation

To compare GBC with GBC+, we run experiments and collect the running time, the accuracy of each algorithm and the number of genes selected. For each dataset, we generate m pair of training and test data subset, where m is the number of instances in the data. Each test data represents an instance in the dataset and the train data is composed by the rest of the instances. First, we run the feature selection algorithms in the training data and then we reduce the test data according to the features eliminated by the algorithms. Second, we train the C-SVM-with-RBF-Kernel and *C4.5* classifiers with the reduced training data and then test them on the reduced test data to determine the Area Under the Receiver Operating Characteristic curve (AUC-ROC) score. In addition, the average of the running time taken by each algorithms and the number of genes selected, when is applied to the train data, is collected.

From the results depicted in Table 5.2, we observe that MRMR wins in twelve of the twenty datasets used, in terms of accuracy.

Table 5.3: Running time of GBC and GBC+.

Alg\Data	STJ	BRE	ECM	HEP	BUR	DA1	DA2	DA3
Gbc	548	50.3	907	87.6	187	196	83.6	88.3
Gbc+	8.52	16.7	31.1	7.06	51.8	120	35.5	15.7
	DA4	DA5	DA6	ANT	MOU	OVA	VAR	PEM
Gbc	337	155	157	244	346	331	1704	-
Gbc+	101	67.9	106	187	42.6	123	387	421

Table 5.4: Number of genes selected by GBC and GBC+ in the experiments.

Alg\Data	STJ	BRE	ECM	HEP	BUR	DA1	DA2	DA3
Gbc	1	7	1	9	10	6	9	7
Gbc+	22	12	2	11	14	7	13	10
	DA4	DA5	DA6	ANT	MOU	OVA	VAR	PEM
Gbc	9	10	11	7	1	6	13	-
Gbc+	20	11	12	19	27	10	39	27

Chapter 6

Summary of algorithms proposed in this research

In this section we briefly describe the algorithms proposed in this research.

6.1 Fast SDCC

The algorithm FSDCC is an improved version of the algorithm SDCC. It has been verified that SDCC outputs better subsets than INTERACT and LCC in terms of consistency. Nevertheless, because SDCC must evaluate $\left(|\mathcal{F}| + |\tilde{\mathcal{F}}|\right)\left(|\mathcal{F}| - |\tilde{\mathcal{F}}| + 1\right)/2$ subsets to output $\tilde{\mathcal{F}}$, it is not applicable to high-dimensional datasets. FSDCC avoids two main deficiencies of SDCC. First, SDCC removes arbitrarily any feature f with $min\mathfrak{Br}(F \setminus \{f\})$. By contrast, FSDCC evaluates individual correlation of features f and eliminates f with lower relevance in earlier stages. As a result, the final solution tends to be higher in terms of collective relevance. Secondly, in the algorithm of SDCC when $\min \mathfrak{Br}(F \setminus \{f\}) = \mathfrak{Br}(F)$ is found, the algorithm still continues evaluating the rest of the features, which is definitely redundant. To solve this problem, FSDCC sorts the features in increasing order according to the correlation of the features with the class. Then, when FSDCC find a feature f with $\min \mathfrak{Br}(F \setminus \{f\}) = \mathfrak{Br}(F)$, f is immediately removed, and a new iteration is started. In this way, FSDCC guaranties to remove the feature with the lowest loss of consistency $\mathfrak{Br}(F \setminus \{f\})$ and the lowest individual correlation with the class. Furthermore, a lot of unnecessary evaluations is avoided.

6.2 Accurate Sdcc

Although we expect that FSDCC considerably improves the performance of SDCC in terms of the number of evaluations and consistency rate of the outputs, we think it posses some

weakness relating to the accuracy obtained by the machine learning algorithms applied to the reduced data when compared with LCC. The output of LCC will tend to contain features more correlated with the class, whereas SDCC tends to select more consistent sets, but they are not necessarily composed by features highly correlated with the class. ASDCC solves this problem by establishing a balance between the consistency contribution of each feature and its respective correlation with the class. With this purpose, the algorithm uses the combined measure determined by:

$$\vartheta\left(f,C\right)=\alpha SU\left(f,C\right)+\left(1-\alpha\right)\frac{\mathfrak{Br}(\tilde{\mathcal{F}}\setminus\left\{f\right\};C)-\mathfrak{Br}(\mathcal{F};C)}{\delta-\mathfrak{Br}(\mathcal{F};C)}. \tag{6.1}$$

The first SU (symmetrical uncertainty) represents the individual correlation of f to C, while the second term does the normalized consistency loss when f is removed. $0\leq\alpha\leq1$ is a balance parameter which allows to specify, in a certain level, the preferable type of feature to seize: those highly correlated with the class, and simultaneously indispensable to compose a consistent set. When anlyzing all features $f\in\tilde{\mathcal{F}}$, since it is suitable to remove f with small correlation with the class and poor consistency contribution, the main goal will be to remove feature $f=\underset{f\in\tilde{\mathcal{F}}}{\arg\min}\,\vartheta\left(f,C\right)$.

6.3 Sdcc with a sliding window method

The algorithm of LCC evaluates only one feature in each iteration to decide whether or not it will be removed. On the contrary, the algorithm of SDCC evaluates all the features to determine which should be removed. SDCC is more accurate than LCC in terms of Bayesian risk, but LCC is faster than SDCC. The algorithm of the SDCC with a sliding window technique (SwCfs) can be placed at an intermediate point between LCC and SDCC. SwCfs uses a windows to determine the number of features to be evaluated in each iteration. The size of the windows can be one feature, as in LCC, all the features as in SDCC, or any values in between.

Let F be the entire feature set and δ be the upper bound of the permissible *Bayesian risk* of the output sets. Our proposed algorithm follows the following rules:
1.F is converted into $\tilde{F}$ by sorting the features in an incremental order of their symmetrical uncertainty score $SU(f_i;C)$.
2. The maximum set $\{f_1,\ldots,f_l\}$ with $brF\setminus\{f_1,\ldots,_l\}<\delta$ is identified and removed by using the *binary search*.
3. The window size is computed in each iteration.

The *steepest-descent* algorithm is performed using the interelevance score IR by evaluating only the features included in the current window and taking into account the following rules with $f_i\in\tilde{F}$:

Rule 1. If $\mathfrak{Br}(\tilde{F} \setminus \{f_i\}) = \mathfrak{Br}(\tilde{F})$ then f_i it is immediately removed from $\tilde{F}$ (line 13).

Rule 2. If $\mathfrak{Br}(\tilde{F} \setminus \{f_i\}) > \delta$, then f_i will never be evaluated again and never removed from $\tilde{F}$ (line 12).

Rule 3. Otherwise, the feature f_i that minimizes IR is removed from $\tilde{F}$ under the condition of $IR(\tilde{F}; f; C) > IR(\tilde{F}; \varnothing; C)$ holds. The algorithm stops when all features have been tested and none of the features can be removed any more.

6.4 Simulated-Annealing-based LCC

To the best of our knowledge, SUPER-LCC is one of the fastest and the most accurate feature selection algorithm based on consistency measures. However, SUPER-LCC uses a parameter δ that is critical for the accuracy of outputs, and it is a bothersome job to determine appropriate δ values. In fact, researchers often choose $\delta = 0$ when running SUPER-LCC, since the optimal value for δ it is unknown a priori. Simulated-Annealing-based LCC (SALCC) solves this problem by leveraging the simulated annealing search to find a suitable value for δ for SUPER-LCC algorithm.

The SALCC algorithm is as follows.

1. First, we rank the features in F in an increasing order of *Symmetrical Uncertainty* (SU) values.

2. Second, we find the *border feature* f_l such that l is maximum and

$$\mathfrak{Br}(F \setminus \{f_l, \ldots, f_n\}; C) = \mathfrak{Br}(F; C)$$

and fix $\tilde{F} = \{f_{l+1}, \ldots, f_n\}$. This is easily and efficiently achieved by running the first iteration of SUPER LCC described in [56].

At this point, we reduce the search space from $F = \{f_1, \ldots, f_n\}$ to $\tilde{F} = \{f_l, \ldots, f_n\}$.

3. We run the *Simulated Annealing* algorithm shown in Section 3.1 by using the proposed target and neighbour generator functions.

6.5 MRMR+ and CFS+

The inefficiency of the current implementation of MRMR is due to duplication and redundancy when computing mutual information: the algorithm computes the mutual information values for the same features smore than one times. Also, it executes unnecessary computation of mutual information. To overcome this issues, we propose a new algorithm, named MRMR+, that improves the efficiency of the original MRMR significantly by solving the problems of duplication and redundancy.

To solve these problems, we introduce a two-dimensional array $A[\,][\,]$ to store pairs of a feature f and a sum $\sum_{i=1}^{j} I(f, \bar{f}_i)$. For a feature f, if j is the maximum integer such that $s = \sum_{i=1}^{j} I(f, \bar{f}_i)$ has been computed, the pair of (f, s) is an element of the array $A[j][\,]$. When the value of $s' = \sum_{i=1}^{j'} I(f, \bar{f}_i)$ is necessary for $j' > j$, we have only to compute $\sum_{i=j+1}^{j'} I(f, \bar{f}_i)$, because $s' = s + \sum_{i=j+1}^{j'} I(f, \bar{f}_i)$ and (f, s) is stored in $A[j][\,]$. Thus, we can avoid the duplicated computation of mutual information. To solve the problem of redundancy, we skip computing $I(f, \bar{f}_{j+1}), \ldots, I(f, \bar{f}_k)$, whenever $I(f, C) - \frac{2\alpha}{k} \sum_{i=1}^{j} I(f, \bar{f}_i)$ becomes no greater than the current maximum m^* of $\mathrm{MID}_\alpha(f, S)$. The element $(f, \sum_{i=1}^{j} I(f, \bar{f}_i))$ is stored in $A[j][\,]$.

We realized that the problems of duplication and redundancy are not proper of MRMR algorithm. Most of the algorithms, which use greedy searches suffer from the same problems. As an instance, the same technique as stated above can be applied to the algorithm of *Correlation-based Feature Selection* (CFS).

The CFS evaluation function measures a set of features on the basis of: "A good feature subset contains features highly correlated with the class variable, yet uncorrelated to each other" [22]. The following determines the Cfs evaluation function when S contains k features.

$$Cfs(S) = \frac{k * Cs(S)}{\sqrt{k + k(k-1)Rs(S)}}, \tag{6.2}$$

where $Cs(S)$ and $Rs(S)$ denote the average of the correlation between the features in S and the class variable, and the average of the correlation between each possible pair of features in S, respectively. $Cs(S)$ and $Rs(S)$ in $Cfs(S)$ are computed by using the *Symmetrical Uncertainty* correlation function. CFS can be used with a wide variety of search strategies. However, in our review of the literature, the basic greedy forward search is preferred to relieve the computational cost. Another main difference between CFS and MRMR is that the stopping criteria of MRMR is fixed by the number of features to select, while CFS stops when there is no improvement of the current CFS score.

6.6 Improvement of GBG algorithm: GBC+

In Chapter 5, we present some improvements to the GBC algorithm in terms of efficiency and accuracy. First, we replace MRMR used in GBC to filter the data in the first step by MRMR+. In our experiments presented in Chapter 4, it is exhibited that MRMR+ is around 30 to 40 times faster than MRMR. Second, the initialization phase in GBC is made totally at random. In GBC+, we have invented a mechanism to create solutions closer to the optimal by taking into account the MRMR+ score computed in the previous phase of the algorithm. This allows to build an initial population of solutions can lead us to faster convergence. Third, the intensification process in GBC is also made totally at random. In GBC+, we

make use of the SVM scores to intensify solutions with higher accuracy. According to experiments, this step made GBC+ superior to GBC in terms of accuracy.

Chapter 7

Conclusion

The main purpose of this research is to build feature selection algorithms that can be applied to high-dimensional data. Most of the algorithms we provide in this research are fast and very accurate. We first, propose the FASDD and ASDCC algorithms, which are an improved version of the SDCC algorithm. SDCC have two main problems: i) the only information used to select features is the bayesian risk measurement, and ii) even when SDCC reach the lower-bound value for the bayesian risk, it still continuous evaluating hoping a lower value will be found. Both of the FSDCC and ASDCC solve such problems. Afterwards, we propose in this research the SWSDCC, which leverage the binary search to remove a huge mass of irrelevant features in some seconds. In addition, SWSDCC uses a mobile window to intensify the search in the more promision region of the search space. This allows to avoid many evaluations and to maximize the collective relevance in the returned solution. SWSDCC is considered an improved version of LCC and SDCC algorithms. Then, we propose the SALCC, which is an improvement of the SUPERLCC algorithm. The basic idea underlying in SALCC is that the threshold δ, which is the maximum inconsistency rate allowed for the final solution, plays an important role to improve the selected set. We discovered that varying the value of δ the quality of the solution can drastically change. Therefore, we use the simulated annealing algorithm to search across the domain of the value of δ. At the same time we use a wrapper approach to investigate the variation of the quality of the solutions why we vary δ. SALCC is fast, and highly accurate.

We also offer some contributions in the scope of the pairwise-evaluation-based algorithms. We propose the CFS+ and the MRMR+ algorithms, which are three and fourteen times faster that their original version respectively. CFS+ and MRMR+ return the same solution as their original algorithm. However, they are much faster, since they were provided of a mechanism that can avoid many unnecessary evaluations that harms the efficiency of their original versions.

Finally, we propose an improvement of the hybrid GBC algorithm, which is one of the most accurate feature selection algorithms ever created. Although GBC is very accurate,

we detect some weak points on its design that makes this algorithm extremely slow. Our proposal, namely GBC+ is around four time faster than GBC and also is more accurate. We run experiments in fourteen datasets and GBC+ always was better or equal to GBC in terms of accuracy.

Acknowledgements

I would like to express my very great appreciation to my advisor Professor Dr. Yoshihiro Shin for his professional and valuable guidance during the planning and development of this research. At the same time, I also deeply appreciative the Dean Dr. Nishimura for his extremely precious help and comprehension during these years at the University of Hyogo. I would also like to thank several Professors who helped me during my research and living in Japan: Dr. Tetsuji Kuboyama, Dr. Takako Hashimoto and Dr. Basabi Chakraborty. I would also thank the members in my University, specially the faculty members and the staff at the secretarial office for their patience and support.

Bibliography

[1] Abdallah M. and Elkeelany O. A survey on data acquisition systems daq. In *2009 International Conference on Computing, Engineering and Information*, pages 240–243, April 2009.

[2] Almseidin M., Alzubi M., Kovacs S., and Alkasassbeh M. Evaluation of machine learning algorithms for intrusion detection system. In *2017 IEEE 15th International Symposium on Intelligent Systems and Informatics (SISY)*, 277-282, 2017.

[3] Hussein Almuallim and Thomas G. Dietterich. Efficient algorithms for identifying relevant features. In *In Proceedings of the Ninth Canadian Conference on Artificial Intelligence*, pages 38–45. Morgan Kaufmann, 1992.

[4] Hala M. Alshamlan, Ghada H. Badr, and Yousef A. Alohali. Genetic bee colony (gbc) algorithm: A new gene selection method for microarray cancer classification. *Computational Biology and Chemistry*, 56:49-60, 2015.

[5] Hala M. Alshamlan, Ghada Hany Badr, and Yousef AlOhali. mrmr-abc: A hybrid gene selection algorithm for cancer classification using microarray gene expression profiling. In *BioMed research international*, 76-92, 2015.

[6] Angela Angeleska, Nataa Jonoska, and Masahico Saito. Rewriting rule chains modeling dna rearrangement pathways. *Theor. Comput. Sci.*, 454:5–22, October 2012.

[7] Taiwo Oladipupo Ayodele. Types of machine learning algorithms. In *New advances in machine learning*. InTech, 98-112, 2010.

[8] Rivest Ronald Blum Avrib. Training a 3-node neural network is np-complete. *Neural Networks*, 5(1):117 – 127, 1992.

[9] Breiman L., Friedman J., Olshen R., and Stone C. *Classification and Regression Trees*. Wadsworth and Brooks, Monterey, CA, 112-122, 1984.

[10] Leo Wang and Kit Cheung. *Classification Approaches for Microarray Gene Expression Data Analysis*, pages 73–85. Humana Press, Totowa, NJ, 2012.

[11] Thiago F. Coves and Eduardo R. Hruschka. Towards improving cluster-based feature selection with a simplified silhouette filter. *Information Sciences*, 181(18):3766 – 3782, 2011.

[12] Dash M. Feature selection via set cover. In *Proceedings 1997 IEEE Knowledge and Data Engineering Exchange Workshop*, pages 165–171, Nov 1997.

[13] Dash M. and Liu H. Feature selection for classification. *Intelligent Data Analysis*, 1(1):131 – 156, 1997.

[14] Dua Dheeru and Efi Karra Taniskidou. UCI machine learning repository, 2017.

[15] Ding C. and Peng. H. Minimum redundancy feature selection from microarray gene expression data. In *Computational Systems Bioinformatics. CSB2003. Proceedings of the 2003 IEEE Bioinformatics Conference. CSB2003*, pages 523–528, Aug 2003.

[16] Ali El Akadi, Aouatif Amine, Abdeljalil El Ouardighi, and Driss Aboutajdine. A two-stage gene selection scheme utilizing mrmr filter and ga wrapper. *Knowledge and Information Systems*, 26(3):487–500, Mar 2011.

[17] Elyasigomari V., Lee D.A., Screen H.R.C., and Shaheed M.H. Development of a two-stage gene selection method that incorporates a novel hybrid approach using the cuckoo optimization algorithm and harmony search for cancer classification. *Journal of Biomedical Informatics*, 67:11 – 20, 2017.

[18] Quanquan Gu, Zhenhui Li, and Jiawei Han. Generalized fisher score for feature selection. In *Proceedings of the Twenty-Seventh Conference on Uncertainty in Artificial Intelligence*, UAI'11, pages 266–273, Arlington, Virginia, United States, 2011. AUAI Press.

[19] Isabelle Guyon and André Elisseeff. An introduction to variable and feature selection. *J. Mach. Learn. Res.*, 3:1157–1182, March 2003.

[20] Isabelle Guyon, Steve Gunn, Asa Ben Hur, and Gideon Dror. Result analysis of the nips 2003 feature selection challenge. In *Proceedings of the 17th International Conference on Neural Information Processing Systems*, NIPS'04, pages 545–552, Cambridge, MA, USA, 2004. MIT Press.

[21] Mark A Hall. *Correlation-based feature selection for machine learning*. PhD thesis, The University of Waikato, 1999.

[22] Mark A. Hall. Correlation-based feature selection for discrete and numeric class machine learning. In *Proceedings of the Seventeenth International Conference on*

Machine Learning, ICML '00, pages 359–366, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc.

[23] Xiaofei He, Deng Cai, and Partha Niyogi. Laplacian score for feature selection. In Y. Weiss, B. Schölkopf, and J. C. Platt, editors, *Advances in Neural Information Processing Systems 18*, pages 507–514. MIT Press, 2006.

[24] Mdlina Hodorog and Josef Schicho. A regularization approach for estimating the type of a plane curve singularity. *Theoretical Computer Science*, 479:99 – 119, 2013. Symbolic-Numerical Algorithms.

[25] Hui-Huang Hsu, Cheng-Wei Hsieh, and Ming-Da Lu. Hybrid feature selection by combining filters and wrappers. *Expert Systems with Applications*, 38(7):8144 – 8150, 2011.

[26] Jianglin Huang, Yan-Fu Li, and Min Xie. An empirical analysis of data preprocessing for machine learning-based software cost estimation. *Information and Software Technology*, 67:108 – 127, 2015.

[27] Aleks Jakulin, Ivan Bratko, Dragica Smrke, Janez Demšar, and Blaž Zupan. Attribute interactions in medical data analysis. In Michel Dojat, Elpida T. Keravnou, and Pedro Barahona, editors, *Artificial Intelligence in Medicine*, 229–238, 2003.

[28] Dervis Karaboga and Bahriye Akay. A comparative study of artificial bee colony algorithm. *Applied Mathematics and Computation*, 214(1):108 – 132, 2009.

[29] Kenji Kira and Larry A. Rendell. A practical approach to feature selection. In Derek Sleeman and Edwards, editors, *Machine Learning Proceedings 1992*, pages 249 – 256. Morgan Kaufmann, San Francisco (CA), 1992.

[30] J. Kittler, P. Somol, and P. Pudil. Fast branch bound algorithms for optimal feature selection. *IEEE Transactions on Pattern Analysis Machine Intelligence*, 26:900–912, 2004.

[31] Ron Kohavi and Chia hsin Li. Oblivious decision trees, graphs, and top-down pruning. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, pages 1071–1077. Morgan Kaufmann, 1995.

[32] Ron Kohavi and George H. John. Wrappers for feature subset selection. *Artificial Intelligence*, 97(1):273 – 324, 1997. Relevance.

[33] Igor Kononenko. Estimating attributes: Analysis and extensions of relief. In Francesco Bergadano and Luc De Raedt, editors, *Machine Learning: ECML-94*, pages 171–182, 1994.

[34] Konstantina Kourou, Themis P. Exarchos, Konstantinos P. Exarchos, Michalis V. Karamouzis, and Dimitrios I. Fotiadis. Machine learning applications in cancer prognosis and prediction. *Computational and Structural Biotechnology Journal*, 13:8 – 17, 2015.

[35] Pat Langley and Stephanie Sage. Induction of selective bayesian classifiers. In *Proceedings of the Tenth International Conference on Uncertainty in Artificial Intelligence*, UAI'94, pages 399–406, San Francisco, CA, USA, 1994. Morgan Kaufmann Publishers Inc.

[36] Thomas W Lee and Terence R Mitchell. An alternative approach: The unfolding model of voluntary employee turnover. *Academy of Management Review*, 19(1):51–89, 1994.

[37] Wan li Xiang and Mei qing An. An efficient and robust artificial bee colony algorithm for numerical optimization. *Computers and Operations Research*, 40(5):1256 – 1265, 2013.

[38] Chuan Liu, Wenyong Wang, Qiang Zhao, Xiaoming Shen, and Martin Konan. A new feature selection method based on a validity index of feature subset. *Pattern Recognition Letters*, 92:1 – 8, 2017.

[39] Huan Liu and Rudy Setiono. A probabilistic approach to feature selection - a filter solution. In *Proceedings of the Thirteenth International Conference on International Conference on Machine Learning*, ICML'96, pages 319–327, San Francisco, CA, USA, 1996. Morgan Kaufmann Publishers Inc.

[40] Elisabetta De Maria, Franois Fages, Aurlien Rizk, and Sylvain Soliman. Design, optimization and predictions of a coupled model of the cell cycle, circadian clock, dna repair system, irinotecan metabolism and exposure control under temporal logic constraints. *Theoretical Computer Science*, 412(21):2108 – 2127, 2011. Selected Papers from the 7th International Conference on Computational Methods in Systems Biology.

[41] Thomas M. Mitchell. *Machine Learning*. McGraw-Hill, Inc., New York, NY, USA, 1 edition, 1997.

[42] Molina, L. C. Belanche, L. and Nebot A. Feature selection algorithms: a survey and experimental evaluation. In *2002 IEEE International Conference on Data Mining, 2002. Proceedings.*, pages 306–313, 2002.

[43] Narendra P. M. and Fukunaga K. A branch and bound algorithm for feature subset selection. *IEEE Trans. Comput.*, 26(9):917–922, September 1977.

[44] Cheng Soon Ong. Towards open machine learning: Mloss.org and mldata.org. In *2011 IEEE International Workshop on Open-source Software for Scientific Computation*, pages 12–12, October 2011.

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

[46] Adrian Pino Angulo. Gene selection for microarray cancer data classification by a novel rule-based algorithm. *Information*, 9(1):6, Jan 2018.

[47] Adrian Pino Angulo and Kilho Shin. Fast and accurate steepest-descent consistency-constrained algorithms for feature selection. In Panos Pardalos, Mario Pavone, Giovanni Maria Farinella, and Vincenzo Cutello, editors, *Machine Learning, Optimization, and Big Data*, pages 293–305, Cham, 2015. Springer International Publishing.

[48] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes in C (2Nd Ed.): The Art of Scientific Computing*. Cambridge University Press, New York, NY, USA, 1992.

[49] Quinlan., J.R. Induction of decision trees. *Machine Learning*, 1(1):81–106, Mar 1986.

[50] Quinlan, J.R. Induction of decision trees. *Machine Learning*, 1(1):81–106, Mar 1986.

[51] Quinlan J.R. *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993.

[52] Gordon Thomas Rohrmair and Gavin Lowe. Using data-independence in the analysis of intrusion detection systems. *Theor. Comput. Sci.*, 340(1):82–101, June 2005.

[53] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall Press, Upper Saddle River, NJ, USA, 3rd edition, 2009.

[54] Sharma N. and Saroha K. Study of dimension reduction methodologies in data mining. In *International Conference on Computing, Communication Automation*, pages 133–137, May 2015.

[55] Weiguo Sheng and Xiaohui Liu. A hybrid algorithm for k-medoid clustering of large data sets. In *Proceedings of the 2004 Congress on Evolutionary Computation (IEEE Cat. No.04TH8753)*, volume 1, pages 77–82 Vol.1, June 2004.

[56] Shin K., Kuboyama T., Hashimoto T., and Shepard D. Super-cwc and super-lcc: Super fast feature selection algorithms. In *2015 IEEE International Conference on Big Data (Big Data)*, pages 1–7, Oct 2015.

[57] Shin K. and Miyazaki S. A fast and accurate feature selection algorithm based on binary consistency measure. *Comput. Intell.*, 32(4):646–667, November 2016.

[58] Kilho Shin and Xian Ming Xu. Consistency-based feature selection. In Juan D. Velásquez, Sebastián A. Ríos, Robert J. Howlett, and Lakhmi C. Jain, editors, *Knowledge-Based and Intelligent Information and Engineering Systems*, pages 342–350, 2009.

[59] Kilho S. and Xian Ming X. A consistency-constrained feature selection algorithm with the steepest descent method. In Vicenç Torra, Yasuo Narukawa, and Masahiro Inuiguchi, editors, *Modeling Decisions for Artificial Intelligence*, pages 338–350, 2009.

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

[61] Guangtao Wang, Qinbao Song, Baowen Xu, and Yuming Zhou. Selecting feature subset for high dimensional data via the propositional foil rules. *Pattern Recognition*, 46(1):199 – 214, 2013.

[62] Witten I.H., Frank E., Hall , and Pal C.J. *Data Mining: Practical Machine Learning Tools and Techniques*. The Morgan Kaufmann Series in Data Management Systems. Elsevier Science, 2016.

[63] Marcin Wojnarski. Rsctc'2010 discovery challenge: Mining dna microarray data for medical diagnosis and treatment. In *Rough Sets and Current Trends in Computing*, pages 4–19, 2010.

[64] Marcin Wojnarski, Andrzej Janusz, Hung Son Nguyen, Jan Bazan, ChuanJiang Luo, Ze Chen, Feng Hu, Guoyin Wang, Lihe Guan, Huan Luo, Juan Gao, Yuanxia Shen, Vladimir Nikulin, Tian-Hsiang Huang, Geoffrey J. McLachlan, Matko Bošnjak, and Dragan Gamberger. Rsctc'2010 discovery challenge: Mining dna microarray data for medical diagnosis and treatment. In Marcin Szczuka, Marzena Kryszkiewicz, Sheela Ramanna, Richard Jensen, and Qinghua Hu, editors, *Rough Sets and Current Trends in Computing*, pages 4–19, 2010.

[65] Marcin Wojnarski, Sebastian Stawicki, and Piotr Wojnarowski. TunedIT.org: System for automated evaluation of algorithms in repeatable experiments. In *Rough Sets and*

Current Trends in Computing (RSCTC), volume 6086 of *Lecture Notes in Artificial Intelligence (LNAI)*, pages 20–29. Springer, 2010.

[66] Eric P. Xing, Michael I. Jordan, and Richard M. Karp. Feature selection for high-dimensional genomic microarray data. In *Proceedings of the Eighteenth International Conference on Machine Learning*, ICML '01, pages 601–608, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc.

[67] Lei Yu and Huan Liu. Feature selection for high-dimensional data: A fast correlation-based filter solution. In T. Fawcett and N. Mishra, editors, *Proceedings, Twentieth International Conference on Machine Learning*, volume 2, pages 856–863. Morgan Kaufmann Publishers Inc., 2003.

[68] Zheng Zhao and Huan Liu. Searching for interacting features. In *Proceedings of the 20th International Joint Conference on Artifical Intelligence*, IJCAI'07, pages 1156–1161, San Francisco, CA, USA, 2007. Morgan Kaufmann Publishers Inc.

[69] Linling Zhu, Linsong Miao, and Daoqiang Zhang. Iterative laplacian score for feature selection. In Cheng-Lin Liu, Changshui Zhang, and Liang Wang, editors, *Pattern Recognition*, pages 80–87, 2012.