Université d'Ottawa · University of Ottawa

Université d'Ottawa · University of Ottawa

FACULTÉ DES ÉTUDES SUPÉRIEURES ET POSTDOCTORALES

FACULTY OF GRADUATE AND POSTDOCTORAL STUDIES

Jefferson SOUZA

AUTEUR DE LA THÈSE - AUTHOR OF THESIS

Ph.D (Computer Science)

GRADE - DEGREE

School of Information Technology and Engineering

FACULTÉ, ÉCOLE, DÉPARTEMENT - FACULTY, SCHOOL, DEPARTMENT

TITRE DE LA THÈSE - TITLE OF THE THESIS

Feature Selection with a General Hybrid Algorithm

N. Japkowicz

DIRECTEUR DE LA THÈSE - THESIS SUPERVISOR

S. Matwin

CO-DIRECTEUR DE LA THÈSE - THESIS CO-SUPERVISOR

EXAMINATEURS DE LA THÈSE - THESIS EXAMINERS

M. Kubat

Y. Mao

F. Oppacher

L. Peyton

J.-M. De Koninck, Ph.D.

LE DOYEN DE LA FACULTÉ DES ÉTUDES SUPÉRIEURES ET POSTDOCTORALES

DEAN OF THE FACULTY OF GRADUATE AND POSTDOCTORAL STUDIES

Feature Selection with a General Hybrid Algorithm

Jerffeson Teixeira de Souza

A thesis submitted to the Faculty of Graduate and
Postdoctoral Studies of the University of Ottawa in partial fulfillment
of the requirements for the degree of Doctor of Philosophy

Ottawa-Carleton Institute for Computer Science

School of Information Technology and Engineering (SITE)
University of Ottawa
Ottawa, Ontario, Canada

Your file Votre référence
ISBN: 0-494-01773-2
Our file Notre référence
ISBN: 0-494-01773-2

To my wife Shirliane
and my friends Igor and Elisa

Artificial Intelligence: the art of making computers that behave like the ones in movies.

Bill Bulko

Contents

List of Figures

List of Tables

Abstract

The *Feature Selection* problem involves discovering a subset of features, such that a classifier built only with this subset would have better predictive accuracy than a classifier built from the entire set of features. A large number of algorithms have already been proposed for the feature selection problem. Although significantly different with regards to 1) the search strategy they use to determine the right subset of features and 2) how each subset is evaluated, feature selection algorithms are usually classified in three general groups: *Filters*, *Wrappers* and *Hybrid* solutions.

In this thesis, we propose a new hybrid system for the problem of feature selection in machine learning. The idea behind this new algorithm, FortalFS, is to extract and combine the best characteristics of filters and wrappers in one algorithm. FortalFS uses results from another feature selection system as a starting point in the search through subsets of features that are evaluated by a machine learning algorithm. With an efficient search heuristic, we can decrease the number of subsets of features to be evaluated by the learning algorithm, consequently decreasing computational effort and still be able to select an accurate subset. We have also designed a variant of the original algorithm in the attempt to work with feature weighting algorithm.

In order to evaluate this new algorithm, a number of experiments were run and the results compared to well-known feature selection filter and wrapper algorithms, such as Focus, Relief, LVF, and others. Such experiments were run over a number of datasets from the UCI Repository. Results showed that FortalFS outperforms most of the algorithms significantly. However, it presents time-consuming performance similar to that of wrappers. Additional experiments using specially designed artificial datasets demonstrated that FortalFS is able to identify and remove both irrelevant, redundant and randomly

class-correlated features.

The FortalFS time-consumption issue is addressed through parallelism. A parallel version of FortalFS based on the master/slave design pattern is implemented and evaluated. In several experiments, we were able to achieve near optimal speedups.

Acknowledgements

In order to be fair to all people who helped me in one way or another to complete this great journey, please allow me to reflect on the whole process that brought me here from Brazil.

The whole "Canada affair" started in 1999. In June of 99 (my *sent-mail-jun-1999* folder confirms) I sent an e-mail to my former undergrad classmate and soon-to-be best friend, Igor Sales, at the time a Master student in Computer Science at U of O. In that message I expressed my "somehow deep" desire to do my doctorate abroad and generally inquired about his experience in Canada. And, as you may suspect, it was an upbeat response. After deliberating with my former undergrad research advisor and Ph.D. candidate, Rossana Castro, and just graduated Dr. Riverson Rios, both from Ottawa U., and exchanging half a dozen more e-mails with Igor, there was not a single doubt in my mind: Canada it is. Thanks, Rossana and Riverson, for the support.

Figuring out where to go was the easy part, getting the money to pay for it was the next goal. Fortunately, two Brazilian government agencies give scholarships to Ph.D. hopefuls. So I applied. And got it. Thanks CAPES. After that, getting accepted shouldn't be, as it wasn't, a huge problem since my marks from my undergrad and masters were not bad and my "little" research experience seemed to be good enough. However, I could not forget to thank professors Fernando Carvalho and Tarcisio Pequeno, both from UFC, for the recommendation letters that, I am sure, helped me in the acceptance process.

Riverson suggested his former advisor, Dr. Stan Matwin, as a possible supervisor. So during the scholarship process, in August of 1999, I contacted Stan for the first time. I couldn't be happier with his response.

I arrived in Canada with my beautiful wife, Shirliane, on the 12th of August of 2000 (We got married exactly three weeks earlier). And guess who was at the airport waiting for us?? Mr. Sales, of course. We went immediately to the university residence, where we

stayed for a couple of weeks, and to Denny's to have lunch. Igor later on that month helped us open a bank account, find an apartment, buy stuff for the apartment, get a phone-line, get familiar with the new city, etcetera, etcetera and etcetera. In the four years in Canada, Igor helped us in so many ways that I could never forget. As you can see, my friend Igor truly deserves my absolute gratitude for the help he gave me from day one. Thank you Igor. I am shamefully positive that I couldn't have done it without you.

During my period as a Ph.D. student, I have received the help and support of several new friends. First I would like to thank my course professors: Dwight Deugo, Mike Just, Dwight Makaroff, Stan Matwin and Nathalie Japkowicz.

I have to send an special "thank you" to Nathalie. Besides being a wonderful machine learning professor, her kind words regarding my final course project changed, for better, the direction of my doctorate. At that point, I was visualizing a bunch of software patterns for data mining applications. In that machine learning course project, I had the idea of improving a well known feature selection algorithm, LVF, by making it a wrapper (don't bother trying to understand that, I also could not understand it very well at that time). I got an A+ in the project and a nice incentive to try to publish the idea in a conference. So I did it. At that moment I started realizing that my idea wasn't so bad and I could convince Stan to go for it. He surprisingly agreed. I then asked Nathalie to be my co-supervisor. The idea I had in that machine learning course about three years ago is the core of this thesis. The rest is history.

I obviously owe a great deal of gratitude for all the support Stan gave me. For helping me in picking my disciplines, for being such a great professor, for the great research advices in the several papers we worked together, for the financial support, and for helping me make my thesis a much much better research document. I have indeed learned a lot from your experience. Thank you for that.

I also have to thank all professors that worked with me in my several TA positions, including Lucia Moura, Paola Flocchini and Heather Betel. A special thank you to Lucia

for the four years of learning and working together. I would like to believe that we really made the File Management course better year after year.

I have to dedicate a paragraph to our new Argentinean friend Elisa, Igor's better half. Elisa, and Igor, have welcomed us with such open arms that made it really hard for us not to feel at home. They were so great to share all they had with us. She accomplished the almost impossible task of making a Brazilian soccer fan admire Argentina for producing such a wonderful human being. Elisa, you have no idea how much you will be missed.

Thanks to all supporting people, the network administrators, the secretaries, the graduate student advisors, the graduate student association staff, the international office people, my officemate Vivi, everybody that contributed to my smooth period in this university.

Eu não poderia esquecer de todos aqueles que ficaram no Brasil torcendo e contribuindo para meu sucesso. Meus pais, Cícero e Ivan, que passaram pelo grande esforço de entender meu desejo de estudar no Canadá. Meu irmão Cidcley e minha cunhada Fatinha, pelo apoio e pelas inúmeras ajudas durante a minha ausência no Brasil. Meus amigos pelo suporte e desejo de sucesso. Meus sogros, Vivaldo e Alzélia, pelas palavras de apoio e compreensão. Obrigado.

Finally, and most importantly, I would like to thank my wife, Shirliane. She has been a perfect partner. I am thankful for having my soulmate with me to share each second of my life and, thanks to her, most of these seconds have been truly wonderful. Love you.

As you can see, I owe the success of this work to a lot of people who without, I am sure, these four years would not have been so much fun.

Thank you.

Chapter 1

Introduction

Computers are useless. They can only give you answers.

Pablo Picasso

1.1 Motivation

Classification is a key problem in machine learning. Algorithms for classification have the ability to predict the outcome of a new situation after having been trained on data representing past experience. Each example is described by a fixed number of attributes, or *features*, along with a label that denotes its class. More formally, [John et al., 1994] have described classification as follows:

The input to a supervised learning algorithm is a set of n training instances. Each instance X is an element of the set $F_1 \times F_2 \times \cdots \times F_m$, where F_i is the domain of the ith feature. Training instances are tuples (X, Y) where Y is the label, or output. We assume a probability measure p on the space $F_1 \times F_2 \times \cdots \times F_m \times Y$. Given an instance, we denote the value of feature X_i by x_i. The task of the inductive algorithm is to induce a structure (classifier) such that, given a new instance, it is possible to accurately predict the label Y.

A number of factors influence the performance of induction algorithms, including the number and quality of features provided to describe the data, the training and testing data distribution, and others. The factor we focus on in this thesis is the number and quality of features present in the sample data. Intuitively, one could think that the more features there are available to describe some data, the more accurate the classifier created from such data should be. However what is seen in practice is that the more features there are the more training examples are needed to induce an accurate classification model. This result is consistent with the PAC learnability theory [Valiant, 1984] which states that the sample complexity of the hypothesis space considered by a learning algorithm, that is, the number of training instances needed for learning a certain concept, grows exponentially in n, the number of features. Thus, the presence of a large number of features (some of them irrelevant) can hurt the classification capacity of a machine learning algorithm.

This problem is sometimes referred to as "the curse of dimensionality" [Bellman, 1961] and can be explained by the fact that the number of solutions (classifiers) that should be considered from a high dimensional dataset increases exponentially with the number of features, making it more difficult for a learning algorithm to find an accurate one. Also, as a consequence, when a large number of features is present, a greater computational effort is needed in order to compute these features.

In order to reduce dimensionality, two general approaches have been widely used: feature selection and feature discovery. Feature selection attempts to reduce dimensionality by simply eliminating undesirable features while keeping the other features unchanged. Feature discovery, also referred to as constructive induction [Michalski et al., 1983] [Mehra et al., 1989, Pagallo and Haussler, 1990, Matheus, 1991, Craven and Shavlik, 1993], on the other hand refers to the creation of new features that can be performed via feature construction or extraction. Feature construction is the process that discovers missing information about the relationship between features and augments the space of features by inferring or creating additional features while feature extraction uses a mapping function to extract a set of new feature from the original features. Thus, feature selection will not only retain the original physical interpretation of all features but also lead to savings in measurement cost (since some of the features are discarded) [Jain et al., 2000, Guyon and Elisseeff, 2003]. On the other hand, the discovery of new feature may result in improved discriminative ability but such new features may not have a clear physical meaning [Jain et al., 2000].

In this thesis, we deal with the dimensionally reduction problem via feature selection. This choice was motivated by several reasons: first, the selection of features is a simpler solution for the dimensionally reduction problem and yet has been shown to be very effective. Second, researchers have identified some typical limitations of feature extraction algorithms including unrobustness, as defined by [Huber, 1981] as insensitivity to small

deviations from assumptions, and linearity [Laine, 2003]. Finally, feature extraction usually precedes feature selection, since newly created features must be evaluated according to their discriminative ability. This fact makes feature selection a more applicable solution.

1.2 Feature Relevance

In order to better understand the problem we will be dealing with in this thesis, it is essential to be able to differentiate useful (relevant) from useless (irrelevant) features. For this purpose, we will use the concept of *feature relevance* proposed in [John et al., 1994] where two degrees of relevance are defined. *Weak relevance* implies that the feature can sometimes contribute to predictive accuracy. *Strong relevance*, on the other hand, implies that the feature is indispensable is the sense that it cannot be removed without loss of predictive accuracy. Strong and weak relevances are defined as follows:

Definition (Strong relevance)

A feature X_i is strongly relevant iff there exists some feature value x_i, class value y and subset value assignment s_i for which $p(X_i = x_i, S_i = s_i) > 0$ such that

$$p(Y = y|X_i = x_i, S_i = s_i) \neq p(Y = y|S_i = s_i)$$

where S_i is the set of all features except X_i, i.e., $S_i = \{X_1, \ldots, X_{i-1}, X_{i+1}, \ldots, X_m\}$, s_i is a value assignment to all features in S_i and Y is the class label.

Under this definition, a feature X_i is relevant if the probability of that label (given all features) can change when we eliminate knowledge about the value of X_i.

Definition (Weak relevance)

A feature X_i is weakly relevant iff it is not strongly relevant, and there exists a subset of features S'_i of S_i for which there exists some feature value x_i, class value y and subset value assignment s'_i with $p(X_i = x_i, S'_i = s'_i) > 0$ such that

$$p(Y = y|X_i = x_i, S'_i = s'_i) \neq p(Y = y|S'_i = s'_i)$$

where S_i is the set of all features except X_i, i.e., $S_i = \{X_1, \ldots, X_{i-1}, X_{i+1}, \ldots, X_m\}$, s'_i is a value assignment to all features in S'_i and Y is the class label.

Based on these two definitions, features are *relevant* if they are either strongly or weakly relevant, and are *irrelevant* otherwise.

1.3 Feature Redundancy

Another important definition for this thesis is the one of feature redundancy. Lei Yu and Huan Liu [Yu and Liu, 2004] use the definitions of *feature relevance* [John et al., 1994] (above) and *feature's Markov blanket* [Koller and Sahami, 1996] to formally define feature redundancy.

A Markov blanket of a feature X_i is the set of features M_i, such that conditioned on M_i every other feature becomes independent of X_i. Thus, intuitively, knowledge of the values of the Markov Blanket features should make all other features superfluous for classifying X_i. Formally, a Markov blanket is defined in [Koller and Sahami, 1996] as follows:

Definition (Markov blanket)

Given a feature X_i, let M_i be a subset of features such that $X_i \notin M_i$, M_i is said to be a Markov blanket for X_i iff

$$p(F - M_i - \{X_i\}, C \mid X_i, M_i) = p(F - M_i - \{X_i\}, C \mid M_i)$$

where F is the full set of features and C is the target concept.

With this definition of Markov blanket, [Yu and Liu, 2004] defines *redundant cover* and finally *redundant feature*[1] as follows:

Definition (Redundant cover)

A Markov blanket M_i of a weakly relevant feature X_i is called a redundant cover of X_i.

Definition (Redundant feature)

A feature in a set of features G is redundant iff it has a redundant cover within G.

1.4 The Feature Selection Problem

The *Feature Selection* problem involves discovering a subset of features, if one exists, such that a classifier built only with this subset would have better predictive accuracy than a classifier built from the entire set of features. In practice, feature selection algorithms will discover and select features of the data that are *relevant* to the task to be learned.

In addition to irrelevant features, feature selection researchers have identified other

[1]Irrelevant features are not included in this definition.

examples of problematic features which may have a negative impact on the performance of learning systems such as redundant features and randomly class-correlated features. *Irrelevant* features, as defined previously, are those that do not contribute to the predictive accuracy of a particular target concept. *Redundant* features, also as defined above, refer to those that, even when relevant to a target concept, provide mostly information already present in another feature and, in fact, do not contribute to getting better predictors. *Randomly class-correlated* features are correlated to the target class most of the time, and random otherwise.

Thus, both irrelevant, redundant and randomly class-correlated features are worthless and removing them can improve the learning process. In fact, the feature selection process can be seen alternatively as the process of identifying and removing as many irrelevant, redundant and randomly class-correlated features as possible.

1.5 Feature Selection Benefits

Researchers have identified several potential benefits of feature selection. Among them, we emphasize:

- a reduction in the amount of training data needed to achieve learning.
- the generation of learning models with improved predictive accuracy.
- learned knowledge more compact, simpler and easier to understand.
- reduced execution time required for learning.
- reduced storage requirements.

Different research strategies may focus on a single benefit of feature selection or a subset of them over others. In this thesis, we are specially interested in the generation of more accurate learning models via feature selection.

1.6 Feature Selection Approaches

A large number of algorithms have already been proposed for the feature selection problem. Although significantly different with regards to 1) the search strategy they use to determine the right subset of features and 2) how each subset is evaluated, feature selection algorithms are usually classified in three general groups: *Filters*, *Wrappers* and *Hybrid* solutions.

1.6.1 Filters

In a filter model, the feature selection is performed as a pre-processing step to classification. Thus, the selection process is performed independently of the machine learning algorithm to be used to induce the classifier.

In order to evaluate a feature, or a subset of features, filters apply an evaluation function that measures the discriminating ability of the feature or the subset to differentiate class labels. In practice, different evaluation functions are used by different algorithms.

Filters are generally much less computationally expensive than wrapper and hybrid algorithms. However, they may suffer from low performance if the evaluation criterion does not match the classifier well.

1.6.2 Wrappers

As opposed to filters, wrappers [John et al., 1994] do use the learning algorithm as an integral part of the selection process. The idea behind wrappers comes from the fact that the optimal subset of features depends on the specific bias of the learning system, as observed in [John et al., 1994]. Therefore, the selection of features should consider the characteristics of the classifier. Then, in order to evaluate subsets, wrappers use the classifier error rate induced by the learning algorithms as its evaluation function. This aspect of wrappers results in higher accuracy performance for subset selection than simple filters. However, since wrappers have to train a classifier for each subset evaluation, they are often much more time consuming.

1.6.3 Hybrid Systems

The main goal of hybrid systems for feature selection is to extract the good characteristics of filters and wrappers and combine them in one single solution. These systems are designed to select features accurately the way wrappers do, while being time-efficient the way filters are. Hybrid algorithms achieve this behavior usually by pre-evaluating the features with a filter in a way to reduce the search space to be considered by the subsequent wrapper. The term "hybrid" refers to the fact that two different evaluation methods are used, a filter-type of evaluation and the classifier accuracy (like wrappers).

1.7 Thesis Expected Contributions

The expected contributions of the research presented in this thesis are:

A novel and general hybrid solution for feature selection.

This solution combines characteristics of both filters and wrappers in order to achieve precise yet time-efficient feature selection. FortalFS is a general hybrid solution. It allows for the use of different combinations of evaluation criteria. This flexibility permits one to shift its behaviour when wanted. Another novel characteristic of our hybrid algorithm is its search strategy. FortalFS uses a stochastic-seeded generation approach that takes advantage of the knowledge about the features obtained with the filter yet without relying completely on it.

An extensive experimental evaluation of several feature selection algorithms.

We present a comprehensive empirical evaluation of some of the most important feature selection algorithms to date as they get pitted against FortalFS. This evaluation underlies the various strengths and weaknesses of each algorithm.

The extension of a framework for studying and classifying filter and wrapper feature selection algorithms.

We expand Dash and Liu's framework [Dash and Liu, 1997] by covering the most recent developments in feature selection research in terms of filter and wrapper solutions.

A framework for studying and classifying hybrid solutions in feature selection.

By taking into consideration both the type of filter evaluation measure and the classifier used by hybrid feature selection algorithms, we are able to generate a framework

to help us organize and study hybrid methods. This is the first framework ever proposed to organize Hybrid systems.

The study on how parallelism can be used to speedup the performance of feature selection algorithms in general and our novel solution in particular.

We propose and evaluate a parallel version of our algorithm and study how parallelism can aid the generation of more time-efficient feature selection solutions.

Each one of these contributions will be better examined in the final chapter of this thesis.

1.8 Thesis Organization

The remainder of this thesis is composed of the following chapters: In chapter 2 we present two feature selection frameworks that will help us study several selection algorithms that have been previously proposed. Our novel hybrid solution for feature selection, the FortalFS algorithm, is presented in details in Chapter 3. We also present in this chapter a variation of FortalFS named FortalFS-W aimed at enabling our algorithm to work with feature weighting algorithms. Chapter 4 presents a description of the FortalFS evaluation with real-world datasets, initially describing the experimental settings and next the results and analysis. In Chapter 5, FortalFS in evaluated with artificial datasets. In Chapter 6, we describe and evaluate a parallel implementation of FortalFS. Finally, in Chapter 7, we draw some final conclusions and discuss future directions for research.

Chapter 2

Feature Selection: Literature Review

Truth is what stands the test of experience.

Albert Einstein

2.1 Introduction

Some attempts have been made to study feature selection methods based on some framework or structure. Among them we can emphasize the works of Dash and Liu [Dash and Liu, 1997], Blum and Langley [Blum and Langley, 1997] and Liu and Yu [Liu and Yu, 2002].

In [Dash and Liu, 1997], the authors recognize four steps present in any typical feature selection algorithm, namely,

- generation procedure
- evaluation function
- stopping criterion
- validation procedure

The steps are categorized and used to form a framework allowing the grouping of 32 selection algorithms. Finally, the authors give a guideline for applying feature selection methods based on data set characteristics such as data type, data size and noise. Similarly to Dash and Liu's work [Dash and Liu, 1997], Blum and Langley [Blum and Langley, 1997] studied the feature selection process in terms of heuristic search through the space of feature subsets. With that view, they identified four dimensions that determine the nature of the heuristic process and consequently the selection process itself. They divided the methods into filters and wrappers. The dimensions for filters are: starting point, search control, evaluation scheme and halting criterion. Wrappers are defined in terms of starting point, search control, halting criterion and induction algorithm. The authors then characterize a

number of feature selection methods in terms of these four issues. In [Liu and Yu, 2002], Dash and Liu's framework is expanded in several directions, including the discussion of unsupervised feature selection algorithms, thus extending the framework to cover 45 algorithms, and the discussion of newly proposed hybrid algorithms.

In this chapter, we will first describe the feature selection process in its basic steps (Generation of Subsets, Evaluation of Subsets and Stopping Criterion)[1]. Next, we will categorize each step and use these categories to develop a framework that will expand Dash and Liu's framework [Dash and Liu, 1997] by including more recent filter and wrapper feature selection algorithms. There is no conceptual difference between ours and Dash and Liu's framework, since the same dimensions are used in both. In addition, by taking into consideration both the type of filter evaluation measure and the classifier used by hybrid feature selection algorithms, we will introduce another framework for hybrid algorithms to help us organize and study hybrid methods. Finally, we will study feature selection methods used in Support Vector Machines. Due to the recent interest in SVM methods we have decided to describe SVM-based feature selection algorithm, but because of the very unique characteristics of this learning approach, we describe them in a separate section, even though such methods should be considered wrappers and could be included in our first framework.

[1]Thus, we are basically simplifying Dash and Liu's choice [Dash and Liu, 1997] by not including a validation step, since even though it is an important part of any automated learning system, it is not a component of the selection itself.

2.2 The Selection Process

The feature selection process can be seen as a search through the space of subsets of features when trying to find the best subset according to some evaluation criterion. As such, any typical feature selection algorithm can be described in terms of three basic steps:

1. Generation of Subsets = generates the next candidate subset.
2. Evaluation of Subsets = evaluates the current subset.
3. The Stopping Criterion = decides when to stop.

Next, we discuss with more details and categorize each of the three steps described above.

2.2.1 Generation of Subsets

Consider the feature selection process as a search through the space of all subsets of features. For n features in a dataset, there are 2^n possible subsets. Clearly, an exhaustive search of this space may be impractical. More realistic approaches involve some search strategy to reduce the search space.

We present next different approaches that have been applied to generate subsets in feature selection algorithms.

Complete Search

In this approach, all 2^n subsets of features are considered. This way, the optimal subset is guaranteed to be found. The clear disadvantage of this method is in the computational complexity of the search, $O(2^n)$. However, under certain circumstances (discussed later), the search can be complete but not exhaustive, which means that not all 2^n have to be evaluated in order to guarantee an optimal subset.

Heuristic Search

The search through the space of subsets is guided by an heuristic algorithm in a way to avoid a complete search. However, since only a fraction of the search space is considered in the search, there are no guarantees that the optimal subset will be found. Several heuristics have been used for this purpose. We will examine some of them later on this chapter.

Random Search

Algorithms that apply this approach generate a new subset randomly at each iteration. Even though the search space remains 2^n, the exact number of subsets that are considered by the algorithm is controlled by the number of iterations. Consequently, the performance of the search process will depend on the resources available. The two major motivations for developing this sort of algorithm are to avoid getting stuck in local minima as in heuristic search and to capture the interdependence of features which heuristic search is also incapable of doing.

2.2.2 Evaluation of Subsets

Selecting a final subset of features involves picking the best subset according to some evaluation measure. The evaluation method will set a value to each subset based on its ability to distinguish the different target classes. Consequently, the subset with best evaluation measure should be able to generate highly accurate classification models. Different evaluation methods have been used for feature selection. These methods can be broadly grouped into five categories:

Based on Distance

These measures are based on the assumption that instances of a different class are distant in the instance space. The Euclidian Distance is used by a number of algorithms to compute the distance between instances.

Based on Information Gain

First introduced by Quinlan in his classification algorithm ID3 [Quinlan, 1986], information gain refers to the measure of how well a given feature separates instances according to their target classification. Such a statistical measure can be used to compare and consequently select features. Entropy, which measures the (im)purity of a collection of instances, is often used to characterize information gain.

Based on Dependency

These methods are based on the rationale that good subsets contain features highly correlated with (predictive of) the class, yet uncorrelated with (not predictive of) each other. The Pearson's correlation coefficient is an example of a measure used to determine the degree of correlation between a subset and the target class, while the uncertainty coefficient and symmetrical uncertainty coefficient [Press et al., 1988] and the information gain ratio [Quinlan, 1986] can be used to determine the feature-feature and feature-class dependencies.

Based on Consistency

In order to evaluate a given subset of features, its inconsistency rate is calculated by considering only the features of this subset. This rate refers to the number of instance pairs with same feature values but belonging to different classes. This evaluation scheme uses the Min-Features bias when selecting the best subset of features. This bias prefers consistent hypotheses definable over as few features as possible. Consequently, these measures find out the minimally sized subset that satisfies the acceptable inconsistency rate.

Based on Classifier Accuracy

The classifier created from a given subset of features is used as an evaluation function. As described in the previous chapter, algorithms that apply this method are referred to as "wrappers".

2.2.3 The Stopping Criterion

The stopping criterion will determine when the algorithm stops generating new subsets of features and returns the best subset found thus far. A feature selection process may stop under one of the following criteria:

a) a predefined number of features has been selected.

b) a predefined number of iterations has been reached.

c) whether addition (or removal) of any feature does not produce a better subset.

d) an optimal subset according to the evaluation criterion has been obtained.

2.3 A Framework for Filter and Wrapper Feature Selection

2.3.1 Introduction

By taking into consideration the two main characteristics of a feature selection algorithm, namely a generation procedure and an evaluation function, Dash and Liu [Dash and Liu, 1997] proposed a framework for feature selection algorithms that was later improved in [Liu and Yu, 2002].

In this section we will be expanding Dash and Liu's framework by covering the most recent developments in feature selection research in terms of filter and wrapper solutions.

We decided to expand this framework and not the most recent one from Liu and Yu [Liu and Yu, 2002] because the latter also considers unsupervised feature selection algorithms, which are not part of this research.

A total of 52 filter and wrapper algorithms will be described and classified according to this framework.

2.3.2 The Framework

Category *I* - (Generation - Complete, Evaluation - Distance)

Feature selection algorithms in this category are among the first ones to be proposed.

The Branch and Bound algorithm [Narendra and Fukunaga, 1977] finds an optimal subset of features under the monotonicity assumption. This assumption states that a subset of features should not be better, according to some evaluation function, than any larger set that contains the subset. The selection algorithm starts with a full set of features and removes features from it using a depth-first strategy creating a search tree. Nodes whose evaluation value are lower than the current best are not explored since the monotonicity assumption[2] ensures that their children will not contain a better solution, avoiding exhaustive enumeration.

[2]It is important to make clear that assumptions such as monotonicity are actually often invalid for machine learning. In feature selection, for instance, the addition of completely irrelevant features (e.g., social insurance numbers in medical records in a diagnosis task) can significantly worsen the generalization accuracy of a decision tree classifier. In practice, situations like this tend to hurt the performance of algorithms that work under this assumption.

Evaluation Measure	Generation		
	Complete	**Random**	**Heuristic**
Distance	*I* - Branch and Bound Best-First Bobr88		*II* - Relief Relief-F ReliefD EUBAFES Sege84
Information Gain	*III* - MDL		*IV* - SFG DTM SAPPP Koll-Sara96 BDSFS
Dependency			*V* - POE+ACC PRESET CFS FCBF
Consistency	*VI* - Focus MIFES ABB	*VII* - LVF LVI QBB	
Classifier Accuracy	*VIII* - Ichi-Skla84a Ichi-Skla84b AMB&B BS	*IX* - LVW GA Rozsypal03 SA RGSS RMHC-PF1 RVE	*X* - SBS SFS SFFS SBFS FSS BSE BSE-SLASH PQSS BDS For-BRace Back-BRace Schemata RC Queiros-Gels84 ORDERED-FS OBLIVION Feature-Boost

Table 2.1: A Framework for Filter and Wrapper Feature Selection. Each roman number (from I to X) indicates a category.

The Best-First feature selection algorithm [Xu et al., 1989] starts with an empty set of features and generates all possible single feature expansions, see Figure 2.1. The subset with the highest evaluation is then chosen and is expanded in the same way by adding single features. If expanding a subset results in no improvement, the search drops back to the next best unexpanded subset and continues from there. Given enough time a Best-First search will explore the entire search space, so it is common to limit the number of subsets expanded that result in no improvement. The best subset found is returned when the search terminates. This best subset is guaranteed to be the globally best for any criterion satisfying monotonicity principle.

Bobrowski [Bobrowski, 1988] shows that the homogeneity coefficient, due to its monotonicity principal, can be used as an evaluation function for Branch and Bound [Narendra and Fukunaga, 1977] with backtracking and Best-First [Xu et al., 1989].

Category *II* - (Generation - Heuristic, Evaluation - Distance)

The Relief algorithm [Kira and Rendell, 1992b, Kira and Rendell, 1992a] weights each feature according to its relevance to the classification task. Initially all weights are set to zero and then updated iteratively, see Figure 2.2. In each iteration, this non-deterministic algorithm chooses a random instance i in the dataset and estimates how well each feature value of this instance distinguishes between instances close to i. In this process two groups of instances are selected: some closest instances belonging to the same class and some belonging to a different class. With these instances, Relief will iteratively update the weight of each feature according to how well this feature differentiates between data points from different classes while, simultaneously, recognizing data points from the same class. At the end, a certain number of features with the highest weights is selected. In an alternative version, a threshold may be used in such a way that only the features with weights above

Best-First$(D, NumNoImpExp)$

$S_0 = \phi$
Include S_0 in the *Open* list
Closed list $= \phi$
$S_{best} = S_0$
repeat
  $v =$ subset in *Open* with highest $Eval(v, D)$ value
  Remove v from *Open* and include it in *Closed*
  if $Eval(v, D) > Eval(S_{best}, D)$ **then**
    $S_{best} = v$
  Expand v by adding single features
  for each expansion, e, not in *Open* or *Closed*
    Evaluate e and include it in *Open*

---

where:

D - dataset.
$NumNoImpExp$ - maximum number of expansions with no improvements allowed.

Figure 2.1: The Best-First Algorithm.

this value are selected.

Relief$(D, NumIter, NumNeig, NumFea)$

$S = \phi$
Initialize all weights, W_i, to 0
for $i = 1$ **to** $NumIter$
  Randomly choose an instance x in D
  Find its $NumNeig$ $NearHits$ and $NearMisses$
  for $j = 1$ **to** $NumNeig$
    $W_j = W_j - diff(x_j, NearHits_j)^2 + diff(x_j, NearMisses_j)^2$
$S = NumFea$ features with highest W_i
return S

---

where:
  D - dataset.
  $NumIter$ - number of iterations.
  $NumNeig$ - number of neighbors to consider.
  $Threshold$ - selection threshold.

Figure 2.2: The Relief Algorithm.

Relief-F [Kononenko, 1994, Kononenko et al., 1996] extends Relief enabling this algorithm to work with noisy and incomplete datasets and to deal with multi-class problems. RelieveD [John et al., 1994] is a deterministic version of Relief. It uses all instances and all near-hits and near-misses of each instance. This results in the equivalent of running Relief for an infinite amount of time. The EUBAFES (EUclidean BAsed FEature Selection) [Scherf and Brauer, 1997] algorithm weights and selects features similarly to the Relief algorithm. It is also a distance-based approach that reinforces the similarities between

instances that belong to the same class while deteriorating similarities between instances in different classes. A gradient descent approach is employed to optimize feature weights with respect to this goal.

By using an evaluation function that minimizes the sum of a statistical discrepancy measure and the feature complexity measure, Jakub Segen [Segen, 1984] proposes a feature selection algorithm that starts finding the first feature that best separates the classes. Then iteratively adds feature that improve class information until the minimum representation criterion is achieved.

Category *III* - (Generation - Complete, Evaluation - Information Gain)

MDL [Pfahringer, 1995] applies the minimum description length principle [Rissanen, 1978] as an objective function for evaluating the goodness of feature subsets. The algorithm searches the space of subsets with the Best-First strategy and constructs a simple decision table [Kohavi, 1995] for each subset encountered. The minimum description length method is then used to evaluate both the simplicity and the accuracy of each decision table. The subset that generated the *best* decision table is returned.

Category *IV* - (Generation - Heuristic, Evaluation - Information Gain)

The SFG (Sequential Forward Generation) [Cardie, 1993] algorithm for feature selection starts with an empty set and adds features sequentially according to some information criterion until all or a pre-specified number of features have been added. DTM (Decision Tree Method) [Cardie, 1993] applies the same strategy as SFG, but uses entropy to evaluate the features. A decision tree is generated (with C4.5) from a partition of the training

data. The features that remain on the tree after pruning are selected as best subset. SAPPP (Selection of Attributes with Processing based on Progressive increasing Partitions) [Leite and Brazdil, 2002] uses sampling to make DTM more stable. It uses a sequence of progressively larger samples to grow decision trees until certain stopping criterion is satisfied, e.g. number of iterations. Finally, the algorithm simply combines the features of all trees to obtain the final subset, Figure 2.3.

SAPPP$(D, InitSampleSize, Quotient, NumIter)$

$S = \phi$
$SampleSize = InitSampleSize$
for $i = 1$ **to** $NumIter$
  Use the first $SampleSize$ instances as data sample
  Construct a decision tree on this data sample
  S_{aux} = features used within this decision tree
  $S = S \cup S_{aux}$
  $SampleSize = SampleSize \times Quotient$
return S

---

where:
  D - dataset.
  $InitSampleSize$ - initial sample size.
  $Quotient$ - increase factor of sample size
  $NumIter$ - number of iterations.

Figure 2.3: The SAPPP Algorithm.

Koller and Sahami [Koller and Sahami, 1996] utilize ideas from probabilistic reasoning to characterize irrelevant features as those that give us the least additional information

beyond what we would obtain from the other features. This intuition is captured via a formal notion of *conditional independence* between variables which is realized via *Markov Blankets*. The selection algorithm applies a backward elimination strategy.

The BDSFS (Boosted Decision Stump Feature Selection) algorithm [Das, 2001] applies a forward selection search strategy. The selection of the next feature to be considered is based on the information gain criterion and takes into consideration the weight of each dataset instance. This selected feature is then added to the set that will be returned and used to create a decision stump (used as the weak learner), which updates the weights of the dataset examples by assigning higher weights to examples that have often been misclassified in this round. The process repeats until a pre-specified number of features has been selected, in a process very similar to boosting. A variant of this algorithm, referred to as BDSFS-2, avoids the pre-specification of the number of features to be returned. To archive that, new features should be added to the final subset as long as this addition results in increased training accuracy. Another variant of this algorithm, BBHFS, is described in the next section.

Category *V* - (Generation - Heuristic, Evaluation - Dependency)

Mucciardi and Gose propose seven selection algorithms in [Mucciardi and Gose, 1971]. Among them, POE+ACC combines two evaluation measures: POE (Probability of Error) and ACC(Average Correlation Coefficient). Starting with the feature with smallest POE, the next feature to be selected in the one that produces the minimum weighted sum of POEs and ACC. ACC is the mean of the correlation coefficients of the candidate features with the features previously selected at that point. The process is repeated until a certain number of features have been selected.

The PRESET [Modrzejewski, 1993] ranks the features through the significant value of each feature based on rough set theory [Pawlak, 1991]. This significance measure expresses how important a feature is regarding classification. This measure is based on dependency of features.

The CFS feature selector [Hall, 1998, Hall, 2000] ranks the worth of subsets of features rather than individual features. It is based on the hypothesis that a good feature subset is one that contains features highly correlated with (predictive of) the class, yet uncorrelated with (not predictive of) each other. Since exhaustive enumeration of all possible feature subsets is prohibitive in most cases, CFS uses different strategies (forward selection, backward elimination and best first) to search the space the subsets. These heuristic algorithms take into account the usefulness of individual features for predicting the class along with the level of inter-correlation among them. CFS first calculates a matrix of feature-class and feature-feature correlations from the training data. Then a score of a subset of features assigned by the heuristic is defined as: $M_S = \frac{k\overline{r_{cf}}}{\sqrt{k+k(k-1)\overline{r_{ff}}}}$ where M_S is the heuristic merit of the feature subset S containing k features, $\overline{r_{cf}}$ is the average feature-class correlation, and $\overline{r_{ff}}$ is the average feature-feature inter-correlation. Different approaches have been proposed to estimate correlations in this equation, among them we have symmetrical uncertainty, normalized symmetrical MDL and symmetrical relief.

Yu and Liu introduce and use the concept of *predominant correlation* in their FCBF (Fast Correlation-Based Filter) algorithm [Yu and Liu, 2002]. Initially, features are individually evaluated according to a symmetric uncertainty (SU) measure [Press et al., 1988]. A threshold δ decided by the user is used to select features to S'. The correlation between a feature f_i and a class C, denoted by $SU_{i,c}$, is said to be predominant if $SU_{i,c} > \delta$ and there exists no other feature f_j in S' such that $SU_{j,i} \geq SU_{i,c}$. If there exists such f_j to a feature f_i, f_j is called a redundant peer to f_i. A feature is said to be predominant to a class if its correlation to the class is predominant or can become predominant after remov-

ing its redundant peers. Consequently, the feature selection is a process that identifies all predominant features to a class concept and removes the rest.

Category *VI* - (Generation - Complete, Evaluation - Consistency)

The Focus system [Almuallim and Dietterich, 1991, Almuallim and Dietterich, 1994] starts searching through all individual features looking for one that perfectly represents the original dataset, i.e., such feature will be found when the inconsistency count for this feature is equal to the one of the initial dataset. If no single feature is found to meet this criterion, combinations of features of size two, three and so on are considered until a perfect subset is discovered, see Figure 2.4. Unlike LVF, this inconsistency measure simply counts the number of instances with same feature values but belonging to different classes.

Focus$(D, InitialInc, NumFea)$

for $i = 1$ **to** $NumFea$
  For each subset S of size i
    if $Inconsistency(S, D) = InitialInc$ **then**
      return S

---

where:
D - dataset.
$InitialInc$ - initial inconsistency.
$NumFea$ - number of features in the original dataset.

Figure 2.4: The Focus Algorithm.

The use of the initial inconsistency of the dataset as a threshold allows Focus to work well with noisy data[3]. A drawback of Focus is that it can be computationally prohibitive if the number of features to consider is not small, due to the fact that it applies an exhaustive search to go through the subsets of features. However, this exhaustive search may allow Focus to find the optimal subset of features. Since it stops as soon as it finds a subset with the required inconsistency, Focus is a deterministic solution.

The MIFES algorithm [Oliveira and Vincentelli, 1992] looks for a subset of features of minimal cardinality that is sufficient to classify all the examples. It represents the set of instances in the form of matrix where each column corresponds to a positive example and each row corresponds to a negative example in the training set. A feature f is said to cover an element of the matrix if it is able to distinguish between the positive and the negative instance associated with the element, i.e, the algorithm searches for a subset of minimal cardinality such that each element of the matrix is covered by at least one feature in this subset. Starting with a full set of features, new subsets are generated by removing one feature at a time until no more covers can be found.

ABB [Liu et al., 1998] is a Branch and Bound algorithm [Narendra and Fukunaga, 1977] with its bound set to the inconsistency rate δ of the data set with the full set of features. It starts with the full set of features, removes one feature from this set to generate new subsets until no more feature can be removed while the consistency criterion is still satisfied.

Category *VII* - (Generation - Random, Evaluation - Consistency)

LVF [Liu and Setiono, 1996b] randomly searches the space of subsets using a Las Vegas algorithm. At every iteration, a new subset is randomly generated. If the number of

[3]It is important to emphasize that this characteristic is not present in the original version on Focus.

features in this subset is smaller than or equal to that of the current best subset, this subset is evaluated with the use of an inconsistency count. This count is based on the intuition that the most frequent class label among the instances matching this subset of features is the most probable class label. An inconsistency threshold is used to reject subsets that have an inconsistency rate greater than it, see Figure 2.5. This process is repeated *NumIter* times[4]. LVF is in fact a non-deterministic and *anytime* algorithm, being able to output several partial results during processing.

The LVS algorithm [Liu and Setiono, 1998] was designed to make LVF scalable. It achieves that by decreasing the number of inconsistence checks the algorithms need to perform without sacrificing the quality of the selection.

QBB [Dash and Liu, 1998] applies a combination of LVF and ABB (Automatic Branch and Bound) [Liu et al., 1998]. First, it runs LVF and records all returned subsets. Then it runs ABB over each one of these subset by using them as starting sets. The algorithm keeps the minimum sized subsets so that ABB is more focused.

Category *VIII* - (Generation - Complete, Evaluation - Classifier Accuracy)

The AMB&B (Approximate Monotonicity with Branch and Bound)
[Foroutan and Sklansky, 1987] is a variation of the classic Branch and Bound
[Narendra and Fukunaga, 1977] algorithm. This algorithm allows non-monotonic functions to be used, typically classifiers, by relaxing the condition that terminates the search on a specific node. The algorithm considers a threshold error rate r and a certain tolerance Δ is placed on this threshold. Subsets of features with error rate below r are said to be feasible and below $r + \Delta$, conditionally feasible. AMB&B will explore both feasible

[4]In their original paper, the authors chose the number of iterations to be $77 \cdot N^5$.

LVF$(D, \delta, NumIter)$

S_{best} = All features in D
for $i = 1$ **to** $NumIter$
  Randomly choose a subset of features, S_i
  if $Card(S_i) \leq Card(S_{best})$ **then**
    if $Inconsistency(S_i, D) \leq \delta$ **then**
      if $Card(S_i) < Card(S_{best})$ **then**
        $S_{best} = S_i$
        output S_i
      else
        output S_i
return S_{best}

---

where:

D - dataset.

δ - inconsistency threshold.

$NumIter$ - number of iterations.

Figure 2.5: The LVF Algorithm.

and conditionally feasible subsets. However, only feasible subsets can be chooses as final solution.

The Beam Search (BS) feature selection algorithm [Doak, 1992] works similarly to the Best-First algorithm [Xu et al., 1989], except that it limits the scope of the search with a bounded queue. Here, subsets will be placed on the queue from best to worse. At every iteration, the best subsets is extracted from the queue. All possible subsets by adding a feature to it are generated and placed back in the queue in their proper position.

Ichino and Sklanksy have proposed two algorithms for feature selection based on specific classifiers, namely: linear classifier [Ichino and Sklansky, 1984a] and box classifier [Ichino and Sklansky, 1984b].

Category IX - (Generation - Random, Evaluation - Classifier Accuracy)

LVW [Liu and Setiono, 1996a] is a variation of LVF [Liu and Setiono, 1996b] that evaluates its subsets based on an estimated error rate obtained from a learning algorithm instead of an inconsistency measure.

The Genetic Algorithm-based feature selector (GA) [Vafaie and De Jong, 1992] applies a simple genetic algorithm to search through the subsets of features. Each chromosome, containing N genes representing the N features of the dataset, represents a subset to be considered. For each gene in a chromosome, the value 1 means that feature is present in the subset and the value 0 that it is not.

The algorithm starts by randomly generating an initial population of subsets, see Figure 2.6. Each subset is then evaluated and, based on this evaluation, a number of genetic operations (selection, reproduction and mutation) is applied to this population. The fit-

ness value calculated for each subset during evaluation will express how well this subset represents the original dataset. A new population of subsets is generated and the process restarts until a pre-specified number of populations has been considered. At the end, the subset with higher fitness value is selected.

GA$(D, NumberPop, SizePop)$

Generate initial population with size $SizePop$
Evaluate each subset of the initial population
for $i = 1$ **to** $NumberPop$
  Generate a new population P by:
    1) Selecting elements of the previous population
    2) Combining them via reproduction
    3) Mutating a few of them
  Evaluate each subset of P
S_{best} = best subset in P
return S_{best}

---

where:
D - dataset.
$NumberPop$ - number of populations.
$SizePop$ - size of each population.

Figure 2.6: The Genetic Algorithm.

The instance and feature selection algorithm proposed in [Rozsypal and Kubat, 2003] also employs a genetic search strategy similar to the one in [Vafaie and De Jong, 1992]. However, in order to select both instances and features, it uses two chromosomes, one

representing the selected instances and the other the selected features. The instance chromosome contains integers that point to training instances and the feature chromosome contains integers pointing to features. When used to create classifiers, the algorithm selects the instances determined by the first chromosome and describes them with the features determined by the second chromosome. The algorithm uses a fitness function that makes it possible to place emphasis either on maximizing the classification accuracy or minimizing the number of retained instances and features. The k-Nearest Neighbour algorithm is used to create the classifiers.

The Simulated Annealing-based feature selector (SA) [Doak, 1992] mimics annealing for optimization. The search starts with a random subset. In each iteration a new subset is generate by "mutating" the previous one. Like in annealing temperature is set high in the beginning and then is cooled down to an equilibrium. While the temperature is high, the algorithm accepts with high probability new subsets even if they represent a lost in terms of accuracy. As temperature drops, new subsets are accepted only if they reflect improvements.

The RGSS (Random Generation plus Sequential Selection) [Doak, 1992] system will insert randomness into SFS and SBS by generating a random subset of features and performing both SFS and SBS on this subset.

RMHC-PF1 [Skalak, 1994] applies a randomized hill-climbing search that uses the nearest neighbor algorithm as evaluation method. The algorithm starts with a random subset of features. Next, it *mutates* this subset by adding or removing one feature at random. After evaluation, if this new subset outperforms the old one, it is accepted as new best. The process repeats until a maximum number of iterations have been performed.

The RVE [Stracuzzi and Utgoff, 2002] wrapper algorithm starts by computing the num-

ber of features to be removed at each step in such a way that the cost of removing all irrelevant features is minimized (the number of relevant features must be known). Next, RVE iteratively removes the irrelevant features randomly according to the computed schedule. After each step the resulting subset of relevant features in evaluated.

Category X - (Generation - Heuristic, Evaluation - Classifier Accuracy)

SFS (Sequential Forward Selection) [Devijver and Kittler, 1982, John et al., 1994], also known as Forward Wrapper, starts from the empty feature set, and in each iteration generates a new subset by adding a feature selected by some evaluation function, usually classifier error rate, see Figure 2.7. This algorithm is also knows as Forward Wrapper. Similarly, SBS (Sequential Backward Selection) or Backward Wrapper [Devijver and Kittler, 1982, John et al., 1994] starts from the complete set, and in each iteration generates new subsets by discarding a feature.

Methods applying bidirectional hillclimbing have also been proposed. To improve the performance of the SFS and SBS algorithms, Pudil et al. [Pudil et al., 1994] proposed modifications of the SFS and SBS algorithms they called *floating sequential algorithms.* The forward and backwards variants are referred to as SFFS and SBFS, respectively. The SFFS algorithm is similar to the SFS algorithm except that, when a new subset S_k is constructed, the algorithm enters a loop in which it constructs smaller subsets from S_k by sequentially removing one feature at a time until removing a feature results in a subset that is not better than the previous best subset. SBFS modifies SBS in a similar way. FSS [Caruana and Freitag, 1994] start with an empty set of features and greedily add or delete features one at a time. BSE [Caruana and Freitag, 1994] start with a full set of features and greedily add or delete features one at a time. BSE-SLASH [Caruana and Freitag, 1994] is a more greedy variant of BSE that eliminates after every step all features not used in

SFS(D)

$S = \phi$
repeat
  Select the feature A with smallest ErrorRate($S \cup A$)
  $S_{aux} = S \cup A$
  if ErrorRate(S_{aux}, D) < ErrorRate(S, D) **then**
    $S = S_{aux}$
until $S \neq S_{aux}$
return S
until S_{best} has not changed in the last $NumNoImpExp$ iterations

---

where:
  D - dataset.

Figure 2.7: The SFS Algorithm.

what was learned at that step. This algorithm is based on the observation that learning methods (e.g., C4.5/ID3) do not use all features when a large number of them is available. PQSS ((P,Q) Sequential Search) is a generalization of FSS and BSE. P is the number of the features to be added at each step, Q is the number of features to be deleted at each step. If $P < Q$, search starts from full set. If $P > Q$, search starts from empty. In the first case, the first best subset with P features is found. Then the best subset with $(P - Q)$ features is found. If the new subset has better performance than the subset with P features, the subset with $(P - Q)$ features becomes our new subset. The process repeats until no improvements can be done. BDS (Bi-Directional Search) [Doak, 1992] is simply the parallel implementation of SFS and SBS, i.e., SFS is performed from the empty set and

SBS is performed from the full set. In order to guarantee that SFS and SBS converge to the same solution, we must ensure that features already selected by SFS are not removed by SBS and features already removed by SBS are not selected by SFS.

RC [Domingos, 1997] takes into consideration the fact that some features will be relevant only in some parts of the space. In many senses RC is very similar to SBS, but with the distinction that it makes local, instance-specific decisions on feature relevance as opposed to global ones. This is done by looking at the nearest example of some instance that belongs to the same class and considering that the features along which they differ are irrelevant. These features are then deleted if such deletion results in accuracy improvement. The process is repeated for all instances and stops when no improvements are possible.

Queiros and Gelsemas algorithm [Queiros and Gelsema, 1984] works similarly to SFS, but evaluates individual features under various settings by considering different interactions with the set of features previously selected. The authors used Bayesian classifier error rate as the evaluation function.

Moore and Lee [Moore and Lee, 1994] introduce For-BRace and Back-BRace, two selection algorithms that implement the idea of "racing" feature subsets, For-BRace starting with an empty set and Back-BRace with a full set. During this race if leave-one-out cross-validation indicates that a subset is unlikely to have the lowest estimated error, its evaluation is terminated, reducing the computational cost of fully evaluating each subset. A blocking scheme is also applied to avoid running subsets with nearly identical predictions right to the end. In the same paper, the authors proposed Schemata, another selection algorithm that uses a search variant called schemata search. Here, rather than starting with an empty or full set of features, the search begins with all features marked as "unknown". In each iteration, a feature is chosen and raced between being in the subset or excluded from it.

The ORDERED-FS algorithm [Ng, 1998] starts finding the subset of features with minimum error on the training set for each subset size. Next, it evaluates all of these subsets on a hold-out set and returns the one with smallest error. This algorithm has sample complexity (and error) that grows logarithmically in the number of irrelevant features.

OBLIVION [Langley and Sage, 1994] performs pruning to remove features from an initially full oblivious decision tree (all nodes at the same level test the same feature) when that do not hurt classification accuracy. In each step the pruned tree is evaluates with cross-validation.

The FeatureBoost algorithm [O'Sullivan et al., 2000] can be seen as a variant of boosting where features are boosted rather than examples. While standard boosting algorithms alter the distribution by emphasizing particular training examples, FeatureBoost alters the distribution by emphasizing particular features. This distribution is updated at each boosting iteration by conducting a sensitivity analysis on the features used by the model learned in the current iteration. The distribution is used to increase the emphasis on unused features in the next iteration in an attempt to produce different sub-hypotheses. At the end, the sub-hypotheses are combined.

2.4 A Framework for Hybrid Feature Selection

2.4.1 Introduction

Only a few hybrid solutions for feature selection have been proposed thus far. By taking into consideration both the type of filter evaluation measure and the classifier used by hybrid feature selection algorithms, we are able to introduce a framework to help us organize and

study hybrid methods. The filter evaluation methods used in our framework are based on Distance, Information Gain, Dependency and Consistency. These classes of evaluation functions were described previously in Section 2.2.2. The classifiers are Decision Tree, k-Nearest Neighbour, Gaussian classifier and "Others". The "Others" class is used to indicate that the algorithms may employ any other learning algorithm.

2.4.2 The Framework

Table 2.2 presents the framework for hybrid feature selection algorithms. In the table, a plus sign ($^+$) next to a particular method in a certain category indicates that such a method can be adapted to fall under this category, even though no practical attempt has been made to do so.

Filter	**Classifier**			
Evaluation Measure	**Decision Tree**	**k-Nearest Neighbour**	**Gaussian Classifier**	**Others**
Distance				
Information Gain	Bala96 BBHFS Xing01$^+$	Sebban02 Xing01	Xing01	Xing01$^+$
Dependency	ADHOC	ADHOC$^+$	ADHOC$^+$	ADHOC$^+$
Consistency				

Table 2.2: A Framework for Hybrid Feature Selection.

In [Sebban and Nock, 2002], the authors start by proposing a new criterion to estimate the relevance of features called *Relative Certainty Gain* (RCG)[5]. The RCG evaluation

[5]Since this new evaluation method is similar to the Information Gain criterion used by several filter

method assumes that the leaner's ability to correctly classify label instances depends on the existence of wide geometrical structures (characterized using a Minimum Spanning Tree built on the learning data) of identical label points. Next, a new filter for feature selection is proposed, where subsets are generated by a greedy forward selection algorithm and evaluated according to the RCG measure. The authors then address the similarities between the Minimum Spanning Tree (MST) and the 1-Nearest Neighbour (1-NN) graph, which allows for the replacement of the MST by the 1-NN graph. The 1-NN graph besides being less expensive to compute also helps shifting the behaviour of the proposed feature selection algorithm toward wrapper approaches, even though classification accuracy is not used.

The filter component of Xing, Jordan and Karp's hybrid algorithm [Xing et al., 2001] is build in three phases. Initially, *unconditional univariate mixture modeling* is used mainly to discretize the measurements for a given feature. Next, the algorithm ranks all features according to an *information gain* measure which is used as initial filter. The remaining features are then passed to the more computationally expensive third phase. In this final filter step, the authors propose the use of *markov blanket filtering* to select subsets of features for each subset cardinality. Finally, each subset is evaluated via cross validation and the best one is returned.

Bala and others [Bala et al., 1996] propose a feature selection hybrid strategy that integrates genetic algorithms and decision tree learning. In their algorithm, a genetic system drives the search for subsets of features. The fitness value for each subset F to be maximized is expressed as: $Fitness(F) = Inf(F) - Cost(F) + Acc(F)$. $Inf(F)$ is a value based on a technique that estimates the discriminatory power of each feature calculated using an entropy measure. $Cost(F)$ is a simple measure of cost which is directly

feature selection algorithms, we have considered this feature selection method to fall under the Information Gain category.

proportional of the cardinality of subset F. Finally, $Acc(F)$ is a measure of the classification accuracy of feature subset F obtained by inducing a decision tree.

The BBHFS (Boosting Based Hybrid Feature Selection) algorithm [Das, 2001] is an extension of the filter BDSFS-2. The BDSFS (Boosted Decision Stump Feature Selection) algorithm [Das, 2001] applies a forward selection search strategy. The selection of the next feature to be considered is based on the information gain criterion and takes into consideration the weight of each dataset instance. This selected feature is then added to the set that will be returned and used to create a decision stump (used as the weak learner), which updates the weights of the dataset examples by assigning higher weights to examples that have often been misclassified in this round. The process repeats until a pre-specified number of features has been selected, in a process very similar to boosting. A variant of this algorithm, referred to as BDSFS-2, avoids the pre-specification of the number of features to be returned. To achieve that, new features should be added to the final subset as long as this addition results in increased training accuracy. In BBHFS, a learning algorithm is used to drive the search. For that matter, the reweight process is changed so that the weak hypothese used in each round of the boosting process are the concepts the learning algorithm would learn from the unweighted training set when using just the features in the set thus far. However, both the selection of the next feature to be added, performed on the basis of weighted information gain, and stopping criterion remain the same as in BDSFS-2.

The hybrid algorithm ADHOC [Richeldi and Lanzi, 1996] comprises two main steps, namely the *Data Reduction* step and the *Feature Selection* step. In the first step an iterative process is applied to explore dependencies between data and to partition the set of observed features into a small number of clusters (factors). The search for true association between the data is based on the concept of feature *profile*, that denotes which other features one is related to. In the feature selection step, ADHOC selects at most one feature from each

of the factors (data dimension) that has been discovered in the data reduction step by using a wrapper approach. Several heuristics were investigated in this phase, with genetic algorithms (GA) having excellent results. In addition to its selection capabilities, ADHOC is able to rank the features by analyzing the distribution of features in the final population generated by the genetic algorithm.

2.4.3 Discussion

We can make a few remarks about the framework for hybrid feature selection algorithms described above. First, it is clear that the number of hybrid solutions proposed to this date is still very small. Second, one can verify that there is a concentration of information gain-based methods. However, since information gain is the evaluation method used in the highly accepted Decision Tree algorithm, it is only natural that new methods consider this approach. Finally, we can confirm that there is a considerable number of combinations of evaluation methods that have not been tried. For instance, no methods based on distance or consistency have been proposed thus far. In addition, even though Xing01 and ADHOC allow for the use of any classifier, no practical attempt has been made to use the bias of other learning algorithms such as Naive Bayes, Neural Nets, SVM, and others.

As a conclusion, it is reasonable to assume that the feature selection field could benefit from a general algorithm that could assume any position in the framework just by "plugging" different evaluation methods. That flexibility would allow for such an algorithm to have its behaviour shifted when required.

2.5 Support Vector Machine Methods

2.5.1 Introduction

In [Guyon and Elisseeff, 2003], the authors survey all papers presented in the special issue of JMLR on variable and feature selection. The high number[6] of Kernel and Support Vector Machine (SVM) methods published on this special issue is a clear indication of the great attention researchers are currently giving to these techniques. In this section, we will briefly summarize a few feature selection methods for SVM learning after a short introduction.

2.5.2 Support Vector Machines

Support vector learning is based on the class of hyperplane functions

$$(w \cdot x) + b = 0, w \in R^N, b \in R,$$

considering the decision functions

$$f(x) = sign((w \cdot x) + b).$$

The optimal hyperplane (see Figure 2.8), the one with maximal margin of separation between classes, can be uniquely constructed by solving a constrained quadratic optimization problem whose solution vector w has an expansion $w = \sum_i v_i x_i$ in terms of a subset

[6]More than half of the papers on this JMLR special issue fall under kernel and support vector machine learning.

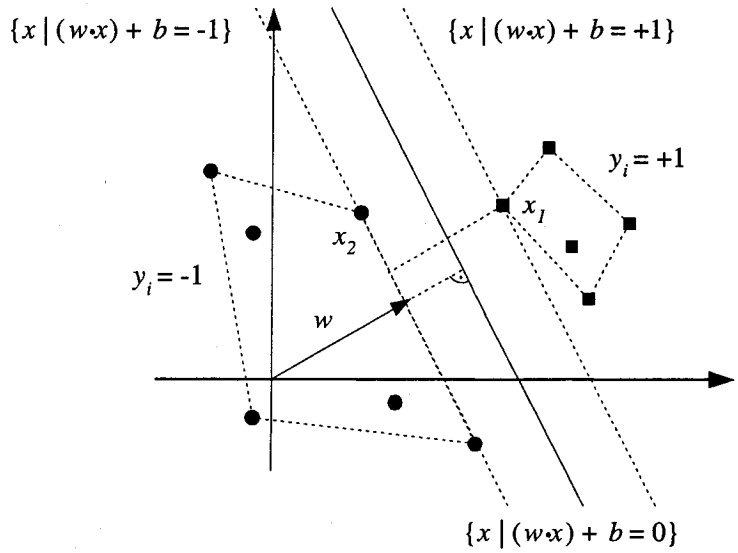

Figure 2.8: A separable classification toy problem: separate balls from squares. The optimal hyperplane is orthogonal to the shortest line connecting the convex hulls of the two classes (dotted), and intersects it half way. There is a weight vector w and a threshold b such that $y_i((w \cdot x_i) + b) > 0$. Rescaling w and b such that the point(s) closest to the hyperplane satisfy $|(w \cdot x_i) + b| = 1$, we obtain a form (w, b) of the hyperplane with $y_i((w \cdot x_i) + b) \geq 1$.

of training instances (called support vectors) that lie on the margin. An interesting characteristic of this learning algorithm is that both the quadratic optimization problem and the final decision function

$$f(x) = sign(\sum_i v_i(x \cdot x_i) + b)$$

depend only on the dot product between instances.

The idea of SVM machines is to map the data into some high-dimensional dot product space F, referred to as *feature space*, via a nonlinear function $\Phi : R^N \rightarrow F$ and perform

the learning algorithm described above in F. As mentioned before, this only requires the evaluation of dot products that can, in some cases, be replaced by a simple *kernel* K that can be evaluated more efficiently.

By substituting $\Phi(x_i)$ for each training instance x_i and "simulating" the computation if F performing the optimal hyperplane algorithm in F, we end up with the following nonlinear decision function

$$f(x) = sign(\sum_{i=1}^{l} v_i \cdot K(x, x_i) + b).$$

where v_i are computed as the solution of the quadratic programming problem.

The norm of the weight vector w, used in several SVM algorithms, is defined as:

$$\| w \| = \sqrt{w_1^2 + w_2^2 + \ldots + w_n^2}$$

This brief overview of SVM (including the graph) was extracted from [Hearst et al., 1998]. For a more thorough and mathematical introduction, see [Müller et al., 2001].

2.5.3 Feature Selection Algorithms for SVMs

[Guyon et al., 2002] proposes an application of the Recursive Feature Elimination (RFE) method (an instance of the backward feature elimination approach) using SVM to tackle the problem of gene selection for cancer classification. The SVM-RFE algorithm starts with all features. At each iteration, it trains a new SVM restricted to the current subset

of features, computes the weight vector w, calculates a ranking criteria $c_i = (w_i)^2$ and removes the feature with smallest c_i.

[Rakotomamonjy, 2003] investigates three criteria for feature selection derived from generalization error bounds in SVM: the weight vector norm $\| w \|^2$, defined as and upper bounds of the *leave-one-out* error. The author then derives zero-order and first-order criteria. The zero-order criteria employ directly the original bounds and the first-order uses a derivative of the zero-order criteria. A feature selection algorithm similar to the SVM-RFE algorithm is finally proposed. This algorithm also applies a backward elimination approach as search strategy and removes features according to the evaluation criterion being used.

In [Weston et al., 2000], the authors propose a feature selection method for SVM based on finding the best variable subset which minimizes the radius/margin bound $R^2 \| w \|^2$ [Vapnik, 1998], where R is the radius of the smallest sphere that contains all the data in the feature space. The search is performed via gradient descent.

A feature selection method is derived in [Weston et al., 2003] that minimizes the zero-norm of w, defined as $\| w \|_0^0 = card\{w_i | w_i \neq 0\}$, where *card* is set cardinality. The authors also propose a new method of minimizing the zero-norm.

2.6 Conclusion

In this chapter, we have first expanded a feature selection framework which helped us classify and describe several filter and wrapper selection algorithms according to their generation procedure and evaluation criterion. The study of feature selection algorithms

under this framework brings some structure to the area in a way that it makes easier to predict/understand the behaviour of the algorithms based on their categories.

In addition, we have developed a framework for classifying hybrid feature selection algorithms by taking into consideration both the type of filter evaluation measure and the classifier used by such methods. From the study of this framework a few facts may come to one's attention in what refers to the hybrid solutions. First, one can conclude that more research in this area could bring benefits. Furthermore, it could be useful to have a general algorithm that could assume any position in the framework by employing different evaluation methods at different times.

Chapter 3

The FortalFS Algorithm

If we knew what it was we were doing, it would not be called research, would it?

Albert Einstein

3.1 Introduction

In this chapter we will propose a new hybrid algorithm, named FortalFS, for the feature selection problem.

The idea behind FortalFS is to extract and combine the best characteristics of filters and wrappers into one algorithm, namely, an efficient heuristic used to search through subsets of features and a precise evaluation criterion, respectively. Thus, the FortalFS algorithm uses results from another feature selection system as a starting point in the search through subsets of features that are evaluated by a machine learning algorithm. Therefore, with an efficient heuristic, we can decrease the number of subsets of features to be evaluated by the learning algorithm, consequently decreasing computational effort (the major advantage of filters) and still be able to select an accurate final subset (the major advantage of wrappers).

3.2 The Algorithm

Initially, the k best subsets returned by a single run of a feature selection system (or the single results of k different runs, if such an algorithm returns only one best subset per execution) are stored into a two-dimensional array, see Figure 3.1. This array will then be condensed into a new array, called *Adam*, that will simply store the number of times each feature appeared in the k best subsets. Next, FortalFS will iteratively generate new subsets of features in a stochastically guided fashion using *Adam* as a seed and evaluate them with a learning system. The generation of a new subset is such that features with high value in *Adam* have a better chance of being selected than those with a low one at each iteration. At the end, the subset with best accuracy will be returned. If subsets tie

it terms of accuracy, the one with the lowest cardinality is returned. This algorithm is described graphically in Figure 3.2.

FortalFS($D, NumIter$)

O = FeatureSelector(D)
$Adam$ = CalculateAdam(O)
for $i = 1$ **to** $NumIter$
  S = GenerateSubset($Adam$)
  if ErrorRate(S, D) < ErrorRate(S_{best}, D) **then**
    $S_{best} = S$
  else
    if ErrorRate(S, D) = ErrorRate(S_{best}, D) **and**
      $Card(S) < Card(S_{best})$ **then**
        $S_{best} = S$
return S_{best}

---

where:
D - dataset.
$NumIter$ - number of iterations.

Figure 3.1: The FortalFS Algorithm.

We describe next, in detail, each of the methods used by FortalFS.

FeatureSelector(D) runs a feature selection system getting the k best subsets generated and storing them into the two-dimensional vector O.

There are a few characteristics that make a feature selection algorithm suitable to

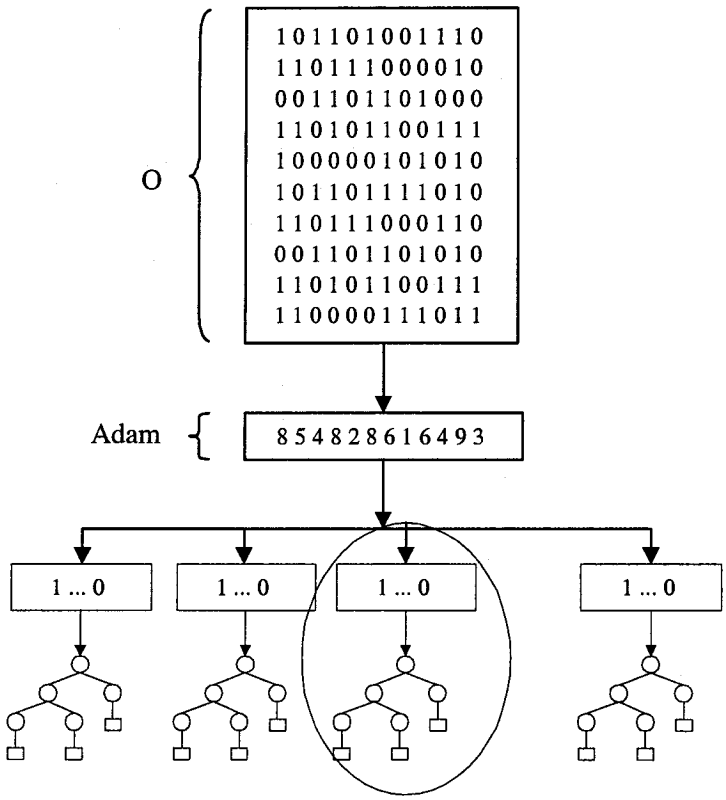

Figure 3.2: Graphical representation of the the FortalFS Algorithm.

be used as underlying algorithm in FortalFS. First, the algorithm must be *non-deterministic*, otherwise, the k best subsets would be the same and FortalFS would consequently select this same subset all the time. For instance, Focus is not a good candidate because of its deterministic behaviour. Second, it should be ideally an *any-time algorithm*, that is, being able to output several partial results during processing. This way, one can obtain the k best results in one single run of the algorithm. LVF is an example of such algorithms. Finally, the algorithm should in fact be a *selection algorithm*, not a weighting algorithm such as the original Relief[1]. However, FortalFS

[1]The Relief version we presented previously and use in our experiments is a "selection" version of the original "weighting" Relief algorithm.

can be modified to work with feature weights directly. We present and evaluate this modification later on.

CalculateAdam(O) uses the following equation:

$$Adam = \{a_i, 1 \leq i \leq n\}$$

where: $a_i = \sum o_{ji}$, with $1 \leq j \leq k$ and $1 \leq i \leq n$.

to create the *Adam* vector, which stores the number of occurrences of each feature in O.

GenerateSubset(*Adam*) generates a new subset of features S in a stochastically guided fashion using *Adam* as a seed. The generation process works as described below. Let i denote a particular feature in *Adam*. Let S be a vector of n elements where n is the total number of features in O. Element S_i (of S) $= 1$ if feature i is included in the subset of features represented by S. $S_i = 0$, otherwise. Vector S is computed as follows:

$S_i = 1$, if $a_i > random(k)$ and $S_i = 0$ otherwise,

where $random(k)$ returns a random number between 0 and k.

This procedure is such that features with high frequency have a better chance of being selected than those with a low one at each iteration.

ErrorRate(S, D) makes use of a learning algorithm, inputting the subset S to generate a prediction model and receiving the error rate calculated for this model over dataset D.

3.3 Computational Complexity Analysis

The time complexity of the FortalFS algorithm is dependent on a) the complexity of the underlying feature selections algorithm and b) the complexity of the learning algorithm to be applied for evaluating the subsets of features.

First note that both the procedures to calculate the *Adam* vector, generate a new subset of features and calculate the cardinality of a particular subset are linear in the number of features in the original dataset. Thus, these procedures will not influence the time complexity of FortalFS since more complex processes are present.

Now note that FortalFS initially executes another selection algorithm, say f. Let $O(f)$ be the time complexity of f. If we have to execute this algorithm k times to get the k best results needed to calculate *Adam*, the final complexity of this procedure will be $O(k \cdot O(f))$.

Next, our feature selection approach will train and evaluate $NumIter$ models using a certain learning algorithm l. Let $O(l)$ be the time complexity of algorithm l for both training and testing a new model. Since this process is repeated $NumIter$ times, the complexity for this whole procedure is $O(NumIter \cdot O(l))$.

Consequently, the time complexity of FortalFS can be computed as $max(O(k \cdot O(f)), O(NumIter \cdot O(l)))$, where max is maximum function.

3.4 Expected Behaviour

Based on the design of the FortalFS algorithm, we can make several assumptions about the behaviour of this algorithm in practice. In this section, we will describe and discuss what kind of practical behaviour we expect from our feature selection solution. Each claim described below outlines one expected practical characteristic of FortalFS and includes a justification of why such behaviour is anticipated.

Claim 1: *FortalFS selects subsets of features which generate learning models with predictive accuracy comparable to those from subsets selected by wrappers.*

Justification: Since FortalFS uses the same evaluation method as wrappers and its search strategy concentrates on subsets likely to be among the best ones, we predict that the subsets selected by our algorithm will generate learning models as good as those generated from subsets selected by wrappers in terms of predictive accuracy.

Claim 2: *FortalFS operates more time efficiently than wrappers.*

Justification: Because of the reduced search space considered by FortalFS, even though both FortalFS and wrappers apply the same time consuming evaluation method, we expect that FortalFS will be able to operate more efficiently in term of time consumption.

Claim 3: *FortalFS filters both irrelevant, redundant and randomly class-correlated features.*

Justification: As discussed earlier, the presence of both irrelevant, redundant and randomly class-correlated features in a dataset tend to hurt the ability of learning algorithms to train classification models. Considering that the FortalFS algorithm employs the model accuracy as evaluation measure of subsets of features, we expect

that given sufficient time these types of non-relevant features will be filtered out by the evaluation process.

Claim 4: *FortalFS can have its time performance drastically improved through parallelism.*

The FortalFS algorithm is a perfect candidate for parallelization due to some particular characteristics (easy partitioning, independent partitioning and easy load balancing). A good parallel design should then allow for high speedup levels.

In the next several chapters of this thesis, we will be examining each one of these claims in order to better understand our new feature selection algorithm and its practical behaviour. In chapter 4, we will evaluate the FortalFS algorithm using real-world datasets and compare its performance with other selection systems both in terms of accuracy of the selected subset and time efficiency. That will allow us to draw conclusions regarding claims 1 and 2. Claim 3 will be examined in chapter 5, where specially designed artificial datasets will be used. Finally, in chapter 6 we will apply parallelism in the attempt to speed up our solution, our claim 4.

3.5 FortalFS in the Feature Selection Framework

Some researches in the feature selection field have focused on the development of hybrid solutions. Researchers have been trying to combine different evaluation functions and learning algorithm biases in order to find a good match that will improve selection, as exemplified in the framework for hybrid features selection algorithms described in the previous chapter.

The FortalFS algorithm, as described in this chapter, allows the use of any evaluation

criterion as well as any learning system in a way that different combinations can be applied. This flexibility permits us to shift the FortalFS behaviour toward different categories under the hybrid feature selection framework. In fact, FortalFS is a *general* hybrid solution that can be configured to assume any position in the framework just by attaching different evaluation methods.

3.6 FortalFS with Feature Weighting Algorithms

We have presented above our novel feature selection solution. We have shown that FortalFS uses an underlying algorithm which should have a few characteristics to be adequate. Such underlying feature selection solution must be *non-deterministic*, ideally an *anytime algorithm* and be in fact a *selection algorithm*, not a weighting algorithm. These few restrictions make FortalFS general enough to work with several feature selection systems. However, the third restriction prevents the direct use of highly successful weighing algorithms such as Relief. In fact, it could be advantageous to be able to receive from such underlying algorithms the weights of the features instead of a selected subset. With that in mind, we have developed a variant of FortalFS designed to work with weighting algorithms on this basis. This new algorithm will be referred to as FortalFS-W and will be presented next.

3.6.1 The FortalFS-W Algorithm

FortalFS-W (Figure 3.3) starts by running a feature weighting algorithm and storing the weights in the array *Adam*. The new generation method will give features with higher weight a better chance to be selected. The process repeats and the best subset is returned.

FortalFS-W$(D, NumIter)$

$Adam$ = FeatureSelectorW(D)
for $i = 1$ **to** $NumIter$
  S = GenerateSubsetW$(Adam)$
  if ErrorRate(S, D) < ErrorRate(S_{best}, D) **then**
    $S_{best} = S$
  else
    if ErrorRate(S, D) = ErrorRate(S_{best}, D) **and**
      $Card(S) < Card(S_{best})$ **then**
        $S_{best} = S$
return S_{best}

---

where:
  D - dataset.
  $NumIter$ - number of iterations.

Figure 3.3: The FortalFS-W Algorithm.

Next, we present a more detailed description of the methods used in the FortalFS-W algorithm.

FeatureSelectorW(D) runs a feature weighting system, for example Relief, storing the resulting weights in $Adam = \{a_1, a_2, ..., a_n\}$.

GenerateSubsetW$(Adam)$ generates a new subset of features S based on $Adam$. S is computed as follows:

$S_k = 1$, if $a_k > random(2 \cdot average(Weights))$ and $S_k = 0$ otherwise,

where $random(2 \cdot average(Weights))$ returns a random number between 0 and $2 \cdot average(Weights)$[2].

3.7 Conclusion

We believe that this design will make FortalFS able to select accurate subsets of features in a reasonable time mainly because the wrapper part on this new algorithm will have its search space reduced by the *Adam* vector. This way, only subsets with high probability of being accurate will be considered. Yet, the use of the learner bias will make the evaluation of these subsets highly precise. In addition, the flexibility present in FortalFS allows one to select a particular combination of a subset evaluation criterion (present in a particular selection system) and a learning algorithm as to optimize its performance to specific situations.

In the next chapter we will describe a number of experiments designed to evaluate the performance of FortalFS over several datasets and compare such performance with the ones of other feature selection algorithms. There, we will be analyzing both accuracy and time consumption.

A possible drawback of this approach is its running time, that is hurt by the use of a learning algorithm. In order to improve the time performance of our hybrid solution, we propose in Chapter 6 a parallel version of FortalFS designed to achieve high scalability.

[2]The value $2 \cdot average(Weights)$ is used as seed instead of the higher weight to avoid the selection of a very small number features when only a few features receive very high weights compared to all others.

Chapter 4

FortalFS Evaluation with Real-World Datasets

The great tragedy of Science - the slaying of a beautiful hypothesis by an ugly fact.

Thomas H. Huxley

4.1 Introduction

In order to evaluate FortalFS, several feature selection algorithms were implemented and their performances compared to our new hybrid algorithm. The algorithms used are Best-First Search [Xu et al., 1989], Genetic Search (GA) [Vafaie and De Jong, 1992], LVF [Liu and Setiono, 1996b], Relief [Kira and Rendell, 1992b], Focus [Almuallim and Dietterich, 1991], Forward Wrapper [John et al., 1994], Backward Wrapper [John et al., 1994] and a Random Wrapper (adapted from [John et al., 1994]).

The comparisons between FortalFS and classical filter and wrapper feature selection algorithms are designed to help us introduce and understand our novel approach. In our view, such comparisons are essential to new methods to get situated in the feature selection research and differentiated from more classical solutions. Even though comparisons with more recent selection methods (hybrid and SVM-based solutions, for instance) would have a higher practical value, we feel that we first need to address more basic questions about the FortalFS behaviour and, thus, we do not perform such additional comparisons in this thesis.

4.2 Methodology

Performance measures such as the accuracy of the selected subset, the time used for selection and the number of features selected were obtained in each experiment.

The accuracy for each selected subset was obtained as follows: the original dataset was randomly and equally split into a selection set and a testing set. The feature selection in all cases was performed considering only the selection set. For the wrappers we evaluated the

subsets of features with 5-fold cross validation on the selection set. Finally, the selected subset was then evaluated using 5-fold cross validation on the testing set. All methods are compared using this independent evaluation to avoid the overfitting problem discussed in [Reunanen, 2003]. This paper studies the overfitting problem when comparing feature selection methods. In light of experimental evidence, the author warns that the cross-validation on the data used to guide the search for subsets of features in wrappers should not be used to compare methods, since it may suffer from overfitting. In order to avoid this problem, it is suggested that a separate testing set is to be used when comparing methods.

Different induction algorithms have different biases, each of which guides a learning algorithm towards better hypotheses. This fact impacts the performance of feature selection algorithms, that consequently may be more suitable for some induction algorithms than others. With that in mind, we decided to run our experiments over the same datasets using three different learning algorithms (C4.5, Naive Bayes and k-Nearest Neighbour).

The reported selection times consider the selection phase only, that is, the time of the final evaluation of the selected subset is not included. For FortalFS, the reported times include the execution of the underlying selection algorithm.

Both the filters and the wrappers as well as FortalFS were implemented in Java by using the Weka library [Garner, 1995] as a starting point and the datasets used were obtained from the UCI Repository [Blake and Merz, 1998]. These particular datasets were chosen mainly because they have been used extensively to benchmark machine learning algorithms as well as their diversity it terms of number of features, number of instances, number of classes and type of features. The following 13 datasets were used:

Credit This dataset deals with the task of distinguishing from credit-worthy to non credit-worthy customers of an Australian credit company. 15 features and 690 instances.

Labor The data includes all collective agreements reached in the business and personal services sector for locals with at least 500 members in Canada in 1987 and first quarter of 1988. The data was used to learn the description of an acceptable and unacceptable contract. 16 features and 57 instances.

Vote Instances in this dataset describe how members of the U.S. congress voted on various bills. The task is to predict what political party the member belongs to (Democrat or Republican) based on the way they voted. 16 features and 435 instances.

Primary Tumor The task is focused in the prediction of the location of a primary tumor. 17 features and 339 instances.

Lymph The task is to distinguish healthy patients from those with metastases or malignant lymphoma. 18 features and 148 instances.

Mushroom The task is to distinguish edible from poisonous mushrooms. 22 features and 8124 instances.

Colic The task is to distinguish surgical lesions in horses. 23 features and 368 instances.

Autos It deals with the prediction of risk ratings of cars. 25 features and 205 instances.

Ionosphere It discriminates radar returns from the ionosphere that show or not evidence of structure. 34 features and 351 instances.

Soybean It discriminates soybean diseases based on symptoms. 35 features and 683 instances.

Splice The task in this dataset is to recognize two types of splice junctions in DNA sequences. 60 features and 3190 instances.

Sonar The task is to discriminate between sonar signals bounced off a metal cylinder and those bounced off a roughly cylindrical rock. 60 features and 208 instances.

Audiology This dataset deals with the diagnosis of ear dysfunctions. 69 features and 226 instances.

4.3 Experimental Settings

Table 4.1 describes the parameters used during the experiments for each one of the feature selection algorithms. Next, we present a short discussion on why such parameters were used.

For the Best-First algorithm, we use 5 as the number of non-improving expansions before termination simply because we did not have any studies showing that improved results could be obtained with a different value, so the Weka default, 5, was used.

The parameters used in our Genetic implementation were the ones suggested in [De Jong, 1975].

The inconsistency threshold in LVF was set to be the initial inconsistency of the dataset following the suggestion of LVF's authors in [Liu and Setiono, 1996b]. That value allows LVF to work properly with noisy data. The number of iterations was obtained by experiment by the authors of LVF in its original paper.

For Relief, we set the number of near hits and misses to be considered to 10, as suggested in [Kononenko et al., 1996] and the selection threshold suggested in [Hall, 2000]. In addition, we use all instances of the dataset to make the algorithm use as much information as possible.

For the random wrapper, we first run it with $10 \cdot N$ iterations, the maximum number used by FortalFS. N is the number of features in the original dataset. However, since Fort-

Best-First
Number of non-improving expansions before termination = 5
Genetic
Number of populations = 200
Size of population = 50
Mutation probability = 0.001
Crossover probability = 0.6
LVF
Inconsistency threshold = initial inconsistency of the dataset
Number of iterations = $77 \cdot N^5$
Relief
Number of iterations = number of instances in the dataset
Number of $NearHits$ and $NearMisses$ considered = 10
Selection threshold = 0.01
Random Wrappers
Number of iterations = $10 \cdot N$ and N^2
FortalFS
Number of iterations = $6 \cdot N$, $8 \cdot N$ and $10 \cdot N$
Underlying selection algorithm = LVF
$k = 10$

Table 4.1: Experimental Settings. N is the number of features in the original dataset.

alFS has an heuristic search inserted in it, we decided to check the influence of this heuristic by running the random wrapper with a considerably higher number of iterations, N^2. In particular, we wanted to check whether FortalFS's performance is caused by increased search-time alone or whether the heuristic itself does make a considerable difference.

Finally, for FortalFS, we try three different settings, $6 \cdot N$, $8 \cdot N$ and $10 \cdot N$. LVF was used as underlying selection algorithm mainly for its capacity to generate a set of results with one execution. In addition, we empirically selected k to be ten. To get to this number, we have tried different values for several datasets of small and medium sizes (up to 69 features) and the results showed that the FortalFS performance is hurt, in several cases, if we use less than ten results. Furthermore, using more than ten does not improve its performance in most situations. Thus, since the computational cost of obtaining each one of these results may be high, we concluded that ten would be the optimal value.

A detailed description of the results can be found in Appendixes A, B and C for C4.5, Naive Bayes and k-Nearest Neighbour, respectively, including the statistical significance of the accuracy difference between FortalFS and all other algorithms for each dataset and learning algorithm.

4.4 Experimental Results and Analysis

In the next section, we will present and discuss the results obtained in our experiments with FortalFS and other feature selection algorithms in a general manner. The following acronyms will be used on the tables/figures to refer to each algorithm: NFS (No feature Selection - C4.5, Naive Bayes or k-Nearest Neighbour with original dataset), FFS (best FortalFS result among the three settings), FS10 (FortalFS($10 \cdot N$)), FS8 (FortalFS($8 \cdot$

N)), FS6 (FortalFS($6 \cdot N$)), FWR (Forward Wrapper), BWR (Backward Wrapper), W10N (Random Wrapper($10 \cdot N$)), WN2 (Random Wrapper(N^2)), B-F (Best-First Search), Gen (Genetic Search), LVF (LVF), Rel (Relief), Foc (Focus).

Following, we will focus on specific pairwise comparisons between FortalFS and some of the other algorithms. In each pairwise comparison, we will present a figure that will show the number of datasets each algorithm performed better at each relevance level. The relevance levels were calculate using the students t-test. We consider the levels <0.01, <0.05, <0.1, and >0.1 (not statistically relevant).

The total number of experiments is 39, since we have 13 datasets where we employ 3 different learning algorithms.

4.4.1 Overall Results and Analysis

As shown in Figure 4.1, the three FortalFS settings (FS10, FS8 and FS6) are among the best algorithms in terms of accuracy. FortalFS($10 \cdot N$) had the best performance in 12 cases, FortalFS($8 \cdot N$) and Relief in 7, the Backward Wrapper in 6, and FortalFS($6 \cdot N$) and the Forward Wrapper in 5. When considering the best FortalFS result in each case (FFS), FortalFS performs at least as well as all other algorithms in 24 out of the 39 experiments.

As expected, the FortalFS performance was proportional to the number of subsets considered, that is, FortalFS($10 \cdot N$) performed better than FortalFS($8 \cdot N$) that was better than FortalFS($6 \cdot N$).

The Random Wrappers did worse than FortalFS, Forward and Backward Wrappers in most cases. An important and expected conclusion that can be extracted from this result

is that the strength of FortalFS, Forward and Backward Wrappers come also from the search heuristic they apply and not only from their strong evaluation method.

In the next sections, we will examine with more details the differences obtained in the experiments between FortalFS and the three filters (Focus, Relief and LVF) along with the wrappers (Random, Forward and Backward).

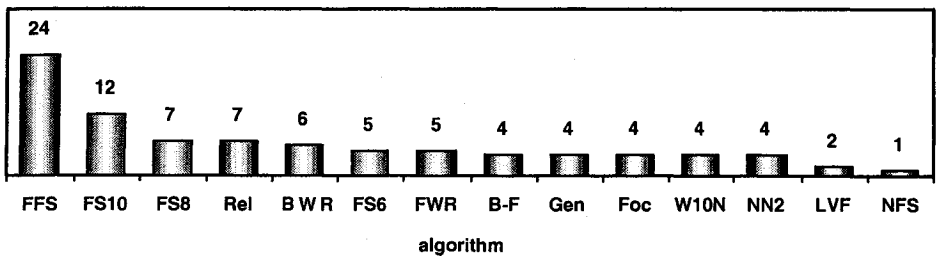

Figure 4.1: Overall performance of all algorithms in terms of accuracy. Number of experiments which each algorithm performed the best or tied with the best.

In terms of time consumption (Figure 4.2), as expected, the wrappers and FortalFS are down in the list, which shows the impact of the evaluation method. However, the three FortalFS settings were faster than all wrappers.

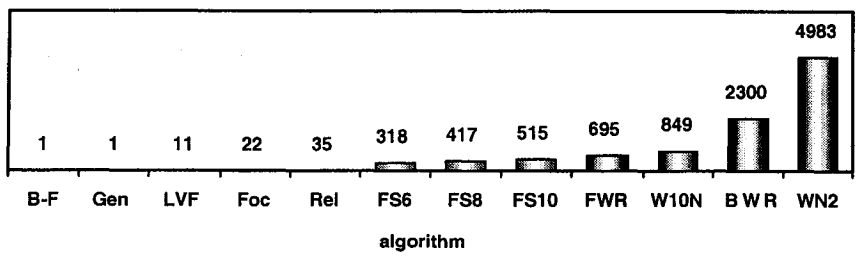

Figure 4.2: Overall time consumption (in minutes) for each algorithm considering all experiments.

Figure 4.3 shows the percentage of the features selected by each algorithm. The Forward

Wrapper, Best-First and Genetic algorithms selected the smallest number of features overall choosing respectively 16.91%, 19.76% and 21.95% of all features. FortalFS was able to achieve a dimensionality reduction of over 60% and still select very accurate subsets. The Backward Wrapper, with 1072 features selected (87.15%), is on the top of the list.

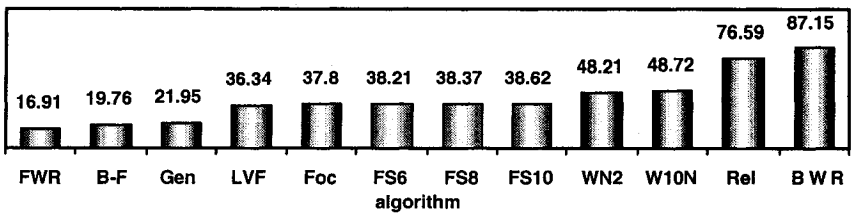

Figure 4.3: Percentage of the features selected by each algorithm in all experiments from a total number of 1230 features.

Finally, we should add that we could not find any relevant difference in the performance of FortalFS when separating the results from the three learning algorithms.

4.4.2 FortalFS vs LVF

By comparing the results obtained with FortalFS versus LVF, we can get a good measure of the ability of the first algorithm to improve the performance of the second (since we used LVF in our FortalFS implementation as underlying feature selector).

Figure 4.4 shows us that FortalFS significantly (at least within the 0.1 significance level) outperformed LVF in 27 out of the 39 experiments and it is significantly outperformed only once.

The obvious drawback of FortalFS can be seen in Table 4.2: it was in average about 47 times more time-consuming in our experiments when running $10 \cdot N$ iterations, and about 38 and 29 times slower for $8 \cdot N$ and $6 \cdot N$ iterations, respectively. Such high values can

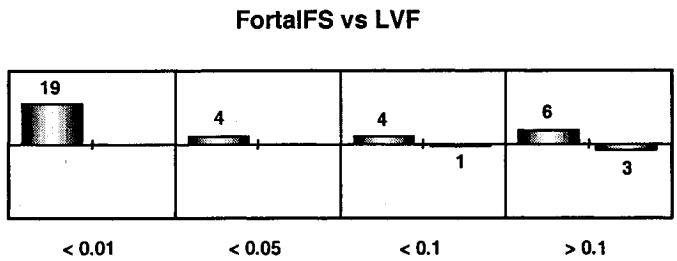

Figure 4.4: Number of experiments (out of 39) each algorithm performed better within each relevance level. Bars over the line indicate a better performance for FortalFS. Bars under the line indicate a better performance for LVF.

be attributed, at least in part, to the fact that LVF is one of the least computationally expensive algorithms among the ones considered in this paper. However, the high decrease in speed is a direct result of the time-consuming evaluation method based on a learning algorithm used by FortalFS, and should be expected whenever this algorithm is applied.

In terms of number of features selected, the differences do not seem to represent any relevant change.

FortalFS Setting	Time consume	Total # Fea. Selected
$10 \cdot N$	$46.91x$	$1.06x$
$8 \cdot N$	$37.98x$	$1.05x$
$6 \cdot N$	$29.00x$	$1.05x$

Table 4.2: Performance comparison in terms of time consumption and number of features selected between FortalFS and LVF. Proportion between total time consumed and total number of features selected by FortalFS and LVF.

4.4.3 FortalFS vs Focus

Specifically compared to Focus, FortalFS outperformed this algorithm in 30 cases (significantly in 25 of them) and it is outperformed only in 6 (Figure 4.5).

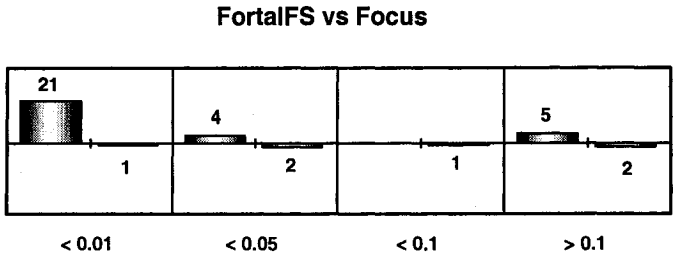

Figure 4.5: Number of experiments (out of 39) each algorithm performed better within each relevance level. Bars over the line indicate a better performance for FortalFS. Bars under the line indicate a better performance for Focus.

FortalFS Setting	Time consume	Total # Fea. Selected
$10 \cdot N$	$22.86x$	$1.02x$
$8 \cdot N$	$18.51x$	$1.01x$
$6 \cdot N$	$14.13x$	$1.01x$

Table 4.3: Performance comparison in terms of time consumption and number of features selected between FortalFS and Focus. Proportion between total time consumed and total number of features selected by FortalFS and Focus.

Even being twice slower than LVF, Focus is still substantially faster than FortalFS, about 18 times faster in average. Yet, for the number of features selected by the two algorithms, we almost have a tie (Table 4.3).

4.4.4 FortalFS vs Relief

Relief was the filter algorithm that performed the best in terms of accuracy. It was able to perform as well as FortalFS($8 \cdot N$). However, when one considers the best FortalFS result, Relief is significantly outperformed about half of the time.

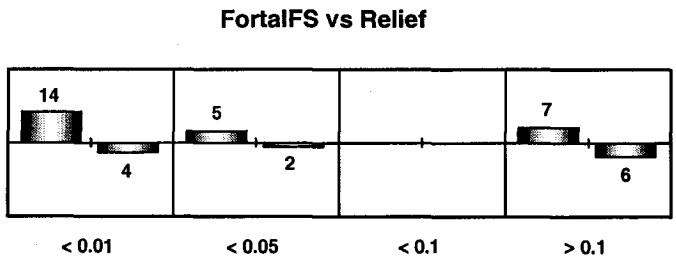

Figure 4.6: Number of experiments (out of 39) each algorithm performed better within each relevance level. Bars over the line indicate a better performance for FortalFS. Bars under the line indicate a better performance for Relief.

FortalFS Setting	Time consume	Total # Fea. Selected
$10 \cdot N$	$14.52x$	$0.78x$
$8 \cdot N$	$11.76x$	$0.78x$
$6 \cdot N$	$8.98x$	$0.78x$

Table 4.4: Performance comparison in terms of time consumption and number of features selected between FortalFS and Relief. Proportion between total time consumed and total number of features selected by FortalFS and Relief.

In Table 4.4, one can see that FortalFS required on average about 12 times more time to select its best subset. In terms of number of features selected, Relief returned 22% more features than FortalFS.

4.4.5 FortalFS vs Forward Wrapper

The importance of the comparison between FortalFS and both Forward and Backward Wrappers relies on the fact that all these algorithms take advantage of the same evaluation method. Such a fact gives us a common background to analyse exclusively the performance differences between the search heuristics. With that in mind one can conclude, from Figure 4.7, that the FortalFS search method performs better in most cases. In fact, FortalFS outperformed the Forward Wrapper algorithm within the 0.01 significance level in 14 experiments (out of 39), and in other 5 within 0.1. On the other hand, the wrapper did significantly better in 4 cases.

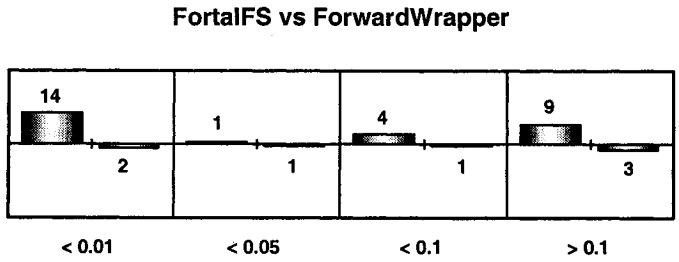

Figure 4.7: Number of experiments (out of 39) each algorithm performed better within each relevance level. Bars over the line indicate a better performance for FortalFS. Bars under the line indicate a better performance for Forward Wrapper.

As far as the time consumed by each algorithm, FortalFS performs better. In particular FortalFS($6 \cdot N$) is as much as 55% faster than the Forward Wrapper. For the number of selected features, the Forward Wrapper prefers smaller results. This algorithm indeed selected the smallest subsets of features among all feature selection systems studied here. FortalFS suggested subsets about 2.2 times as large as (Table 4.5).

FortalFS Setting	Time consume	Total # Fea. Selected
$10 \cdot N$	$0.74x$	$2.28x$
$8 \cdot N$	$0.60x$	$2.26x$
$6 \cdot N$	$0.45x$	$2.25x$

Table 4.5: Performance comparison in terms of time consumption and number of features selected between FortalFS and Forward Wrapper. Proportion between total time consumed and total number of features selected by FortalFS and Forward Wrapper.

4.4.6 FortalFS vs Backward Wrapper

When we compare the performance of FortalFS with the one of the Backward Wrapper, we find somehow similar results to what was found with the Forward Wrapper. Indeed, FortalFS outperformed significantly the Backward Wrapper 18 times and it is outperformed only 3.

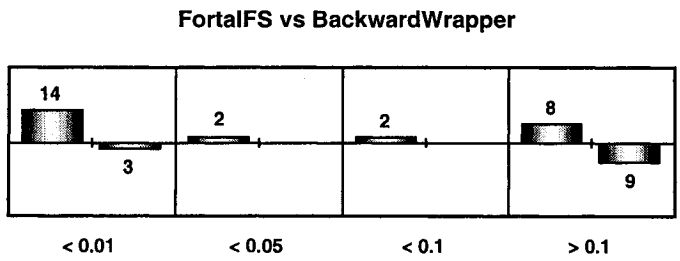

Figure 4.8: Number of experiments (out of 39) each algorithm performed better within each relevance level. Bars over the line indicate a better performance for FortalFS. Bars under the line indicate a better performance for Backward Wrapper.

In terms of time consumption and number of features selected, FortalFS($10 \cdot N$) needed only 22% of the total time used by the Backward Wrapper to achieve its good results and,

in addition, selected only about half of the total number of features picked by the Backward Wrapper algorithm. The time used by FortalFS decreases to 18% and 13% of the time used by the wrapper as we decrease the number of FortalFS iterations to $8 \cdot N$ and $6 \cdot N$, respectively. However, the total number of features selected by these two settings remain at about half of the features returned by the Backward Wrapper, see Table 4.6.

FortalFS Setting	Time consume	Total # Fea. Selected
$10 \cdot N$	$0.22x$	$0.51x$
$8 \cdot N$	$0.18x$	$0.48x$
$6 \cdot N$	$0.13x$	$0.52x$

Table 4.6: Performance comparison in terms of time consumption and number of features selected between FortalFS and Backward Wrapper. Proportion between total time consumed and total number of features selected by FortalFS and Backward Wrapper.

4.4.7 FortalFS vs Random Wrapper

By comparing FortalFS and a Random Wrapper, we are able to analyse the relative effect of the search heuristic applied by FortalFS in the selection process. Here, we decided to study the performance differences between FortalFS and our best random wrapper, that uses N^2 iterations. From Figure 4.9 we can see that even when using a much smaller number of iterations, FortalFS($10 \cdot N$) outperforms the Random Wrapper in 27 out of the 39 cases and is outperformed in 8 others.

Table 4.7 shows us a huge difference in time consumed by the two systems. FortalFS($10 \cdot N$) will save as much as 90% of the time required by the Random Wrapper and yet outperform this algorithm in most cases. FortalFS will also select smaller subsets.

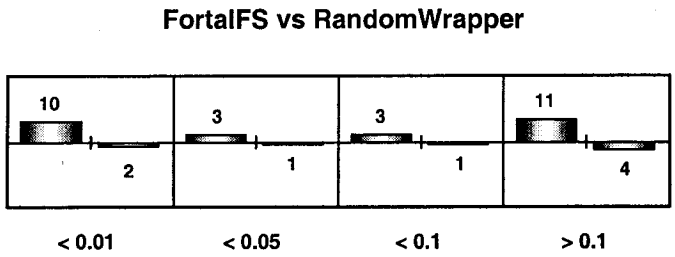

Figure 4.9: Number of experiments (out of 39) each algorithm performed better within each relevance level. Bars over the line indicate a better performance for FortalFS. Bars under the line indicate a better performance for Random Wrapper(N^2).

FortalFS Setting	Time consume	Total # Fea. Selected
$10 \cdot N$	$0.10x$	$0.80x$
$8 \cdot N$	$0.08x$	$0.79x$
$6 \cdot N$	$0.06x$	$0.79x$

Table 4.7: Performance comparison in terms of time consumption and number of features selected between FortalFS and Random Wrapper(N^2) . Proportion between total time consumed and total number of features selected by FortalFS and Random Wrapper(N^2).

4.5 Trying Different Underlying Algorithms

The search strategy applied by FortalFS uses the *Adam* vector as a seed. It is clear that the quality of this vector directly influences the efficiency of the search and consequently the FortalFS performance.

Up to this point, we have been creating our *Adam* vector based on a single feature selection algorithm, LVF. However, since other filters such as Focus and Relief performed

really well in our experiments, we decided to try new *Adam* vectors now based on a) Relief alone as well as b) a combination of results from LVF, Focus and Relief. For Relief, we simply run a non-deterministic version of this algorithm 10 times and combine the results in *Adam*, the same way we did previously with LVF. For the combination, we consider the four best subsets returned by LVF and 3 repetitions of the best subsets from Focus and Relief[1]. The FortalFS setting using Relief alone to generate *Adam* will be referred to as FortalFS-R and the one that uses a combination of results from LVF, Focus and Relief, FortalFS-C (Combined).

In order to try these new approaches, we ran FortalFS($10 \cdot N$) with the new *Adam* vectors with all 13 datasets using C4.5 as induction algorithm. The results are summarized in Figures 4.10 and 4.11 and described in Appendix D.

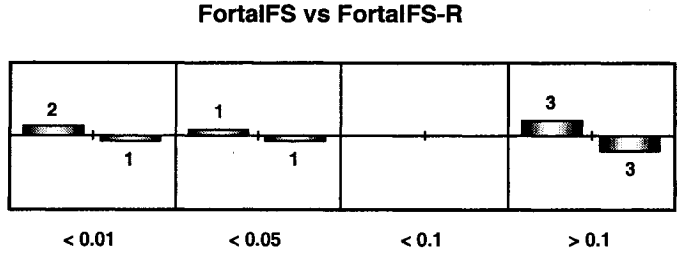

Figure 4.10: Number of datasets (out of 13) each algorithm performed better within each relevance level. Bars over the line indicate a better performance for FortalFS. Bars under the line indicate a better performance for FortalFS-R.

The results were not conclusive. The original strategy outperformed FortalFS-R in 6 cases and it was outperformed in 5. For FortalFS-C, the original FortalFS significantly outperformed this setting for 2 datasets. Even though FortalFS-R seems to do better

[1]We use repetitions for Focus and Relief because such algorithms output only one subset. This way we avoid repeated executions.

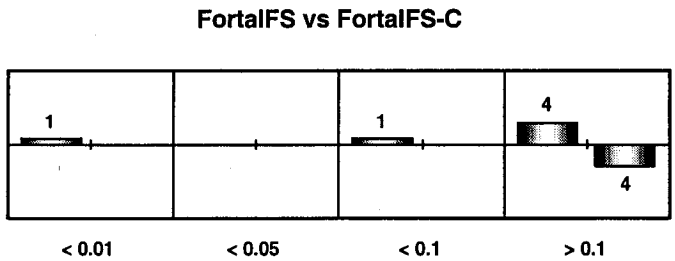

Figure 4.11: Number of datasets (out of 13) each algorithm performed better within each relevance level. Bars over the line indicate a better performance for FortalFS. Bars under the line indicate a better performance for FortalFS-C.

than FortalFS-C, such results suggest that these more elaborated *Adam* vectors may not result in direct improvements in FortalFS performance, which would make the additional computational effort not worthwhile. For FortalFS-C, the fact that a single result from Focus and Relief is repeated three times in *Adam* may be giving too much weight to these two solutions in a way that the final result is too dependent on the performance of these underlying algorithms.

4.6 FortalFS-W Evaluation

In order to evaluate the FortalFS-W variant of FortalFS, which was described in Chapter 3, we ran a number of experiments with C4.5 and the 13 datasets used earlier in this chapter. We used Relief as underlying algorithm. A detailed description of the results can be found in appendix D.

The experimental results (Figure 4.12) show that FortalFS-W is not able to perform better than FortalFS with LVF in most cases. In fact, FortalFS outperformed its variant in

4 cases within the 0.1 significance level. In addition, FortalFS-W was outperformed other 8 times (not significantly) and did better than FortalFS only once.

This result can be, at least in part, explained by the fact that when using the weights directly, FortalFS-W still needs to consider all features, what would maintain the original large search space. Yet when we use a set of subsets returned by another system, the features never selected (probably irrelevant) are not considered at all by FortalFS.

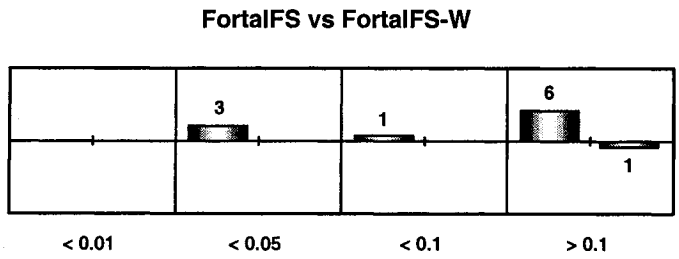

Figure 4.12: Number of datasets (out of 13) each algorithm performed better within each relevance level. Bars over the line indicate a better performance for FortalFS. Bars under the line indicate a better performance for FortalFS-W.

4.7 Evaluation on Larger Datasets

The results presented previously in this chapter give us a very good indication of the efficiency of the FortalFS algorithm in selecting relevant features. However, since practical feature selection usually requires for algorithms to deal with much larger datasets in terms of dimensionality, we thought necessary to extend our experiments to include datasets with more features. For that purpose, we have selected four datasets:

Clean1 This data (from UCI Repository [Blake and Merz, 1998]) induces a classifier to predict whether new molecules are musks or non-musks. 166 features and 476 instances.

Clean2 Same as Clean1. 166 features and 6598 instances.

Arrhythmia The aim is to distinguish between the presence and absence of cardiac arrhythmia and to classify it in one of the 16 groups, also from UCI [Blake and Merz, 1998]. 279 features and 452 instances.

Madelon Extracted from the NIPS2003 feature selection challenge, this dataset is used to classify random data. It contains 500 features and 2000 training instances.

We ran the FortalFS-W variant of FortalFS using Naive Bayes as induction algorithm and 5-fold cross-validation as evaluation method. Again, a separate testing set was kept aside to be used in the final evaluation of the selected subsets. The *Adam* vector was filled with values returned by the Relief weighting method. Since some of the feature selection algorithms we compared against are not practical for such large datasets, we only compare the performance of FortalFS against the Naive Bayes algorithm using all features. The following tables show the experimental results.

	Error Rate (%)	FS Time (sec)	# Fea. Selected
NaiveBayes	29.3172	-	-
FortalFS $(10 \cdot N)$	26.4447	569	82
FortalFS $(8 \cdot N)$	27.1291	459	91
FortalFS $(6 \cdot N)$	27.2075	348	87

Table 4.8: Experimental results for the Clean1 Dataset (166 features and 476 instances)

	Error Rate (%)	FS Time (sec)	# Fea. Selected
NaiveBayes	15.4348	-	-
FortalFS $(10 \cdot N)$	12.4341	10310	49
FortalFS $(8 \cdot N)$	13.5220	7715	57
FortalFS $(6 \cdot N)$	14.1559	5533	47

Table 4.9: Experimental results for the Clean2 Dataset (166 features and 6598 instances)

	Error Rate (%)	FS Time (sec)	# Fea. Selected
NaiveBayes	43.4421	-	-
FortalFS $(10 \cdot N)$	36.5946	877	35
FortalFS $(8 \cdot N)$	37.8867	705	38
FortalFS $(6 \cdot N)$	41.1239	536	40

Table 4.10: Experimental results for the Arrhythmia Dataset (279 features and 452 instances)

	Error Rate (%)	FS Time (sec)	# Fea. Selected
NaiveBayes	40.1664	-	-
FortalFS $(10 \cdot N)$	38.6000	15982	160
FortalFS $(8 \cdot N)$	39.7262	12319	127
FortalFS $(6 \cdot N)$	39.7428	9381	146

Table 4.11: Experimental results for the Madelon Dataset (500 features and 2000 instances)

The FortalFS-W algorithm was able to significantly improve the classification performance of all datasets after feature selection. For two of them (Clean2 and Arrhythmia), FortalFS performed better within the 0.01 significance level and within the 0.05 level for the other two datasets. However, as the number of features in the original dataset increases, the time required by FortalFS-W becomes more and more prohibitive.

4.8 Conclusion

The large number of experiments presented here gives us a fair portrait of the performance of the new FortalFS algorithm for feature selection.

This new algorithm relevantly outperformed well-known filters in our study, namely LVF, Focus and Relief. Specifically compared to LVF, FortalFS improved the performance of this system in all but 6 cases (breaking even in 2 of them). Such improvements were obtained, however, with certain increase in time consumption. For the other filters, on the other hand, we observed less relevant increases in time-consumption.

Since the FortalFS, Forward, Backward and Random Wrapper algorithms share the same evaluation technique, the experiments give us a good background to compare the different search heuristics. The experimental results allow us to conclude, first, that the absence of a non-random search heuristic hurts the selection process. In addition, we could find a clear difference in terms of performance between the FortalFS search strategy and the search method employed by forward and backward wrappers in favor of FortalFS. The most relevant conceptual difference between the FortalFS and forward and backward search strategies is the fact that the last two are greedy methods. As such, both sequential methods result is nested feature subsets without any chance to correct a decision later on the process, which can cause performance problems.

We have also evaluated in this chapter a FortalFS variant designed to work with feature weighing algorithms directly. Even though the experimental results showed no improvements compared to our original FortalFS algorithm, other weighing solutions may generate better results in specific situations. One could, for example, apply FortalFS-W by weighting features based on information gain when it is known that the final subset will be used to generate a decision tree. One could also generate *Adam* based on weighing algorithms that

apply different and somehow complementary (in a specific setting) evaluation functions (distance + information gain, consistency + information gain, etc.).

In addition, the use of a single execution of a feature weighting algorithm speeds up the performance of FortalFS and have allowed us to try our new solution on much larger datasets. We have shown that our algorithm significantly improved the predictive ability of four relatively large dataset.

In a nutshell, the results obtained in our experiments demonstrated the power of FortalFS in selecting relevant features. The fact that FortalFS selected more accurate subsets than any other algorithm and it achieves that faster than all wrappers proves the potential of this new hybrid solution for feature selection. Thus, the results confirm the two initial claims made in Chapter 3 about the expected practical behaviour of FortalFS concerning both accuracy and time performance.

Chapter 5

FortalFS Evaluation with Artificial Datasets

The most exciting phrase to hear in science, the one that heralds new discoveries, is not 'Eureka!' (I found it!) but 'That's funny ...'

Isaac Asimov

5.1 Introduction

The experiments described and analyzed in the previous chapter allow us to evaluate the performance of FortalFS by comparing it to other well-known feature selection algorithms. However, since those experiments used real-world datasets where the relevant features are not known in advance, we were not able to measure directly how well our new feature selection algorithm can deal with irrelevant, redundant and randomly class-correlated features.

Irrelevant features are those that do not contribute to the prediction accuracy of a particular target concept. *Redundant* features refer to those that, even when relevant to a target concept, provide mostly information already present in another feature and, in fact, do not contribute to getting better predictors. Feature relevance and redundancy have been formally defined in Chapter 1. *Randomly class-correlated* features are correlated to the target class most of the time, and random otherwise.

Several studies have shown the negative effects of these types of features to common learning algorithms. For the k-Nearest Neighbour algorithm, for instance, the sample complexity grows exponentially with the number of irrelevant features, whether for conjunctive concepts or more sophisticated ones [Aha et al., 1991, Langley and Sage, 1997]. For C4.5 and Naive Bayes, sample complexity seems to grow in general linearly with the number of irrelevant features. But when features interact (as in parity concepts) the sample complexity for C4.5 exhibits exponential grown [Aha et al., 1991, Blum and Langley, 1997]. The Naive Bayes classifier can be adversely affected by redundant features due to its assumption that features are independent given the class [Langley and Sage, 1994]. Decision Tree algorithms such as C4.5 can sometimes overfit training data when redundancy is present, resulting in large trees [John et al., 1994]. Randomly class-correlated features have been shown [John et al., 1994] to mislead both learning algorithms (C4.5 for instance) as well

as feature selection algorithms, specially filters.

Well known feature selection algorithms perform very differently in identifying and removing these types of features. Feature weighting algorithms such as Relief [Kira and Rendell, 1992b, Kira and Rendell, 1992a], for instance, usually cannot identify redundant features since they evaluate features individually, not in sets of features like other feature selection algorithms. However, they are often very efficient in estimating feature relevance. Relief also suffers with randomly class-correlated features. In [Dash and Liu, 1997], the authors report that Relief preferred a correlated feature rather than a relevant one in the CorrAL dataset [John et al., 1994]. The Focus algorithm [Almuallim and Dietterich, 1991, Almuallim and Dietterich, 1994], on the other hand, deals really well with both irrelevant, redundant and class-correlated features since it looks for subsets that generate no inconsistency. Unless the redundant or class-correlated features perfectly duplicate their pairs (another feature or class label, respectively), they will be eventually eliminated by Focus. The LVF algorithm [Liu and Setiono, 1996b] presents a similar behaviour, since it will search for more consistent subsets. For small datasets or given enough time, LVF tends to get rid of undesirable features. Other results regarding the ability of feature selection algorithms in dealing with these problematic features can be found in [Dash and Liu, 1997]. In this paper, the authors report results of several well known selection algorithms over three datasets (CorrAL, Parity3+3 and Monk3) containing together irrelevant, redundant and randomly class-correlated features.

In this chapter, we will examine the FortalFS performance over several artificial datasets. For the next section, we will create new data containing irrelevant, redundant and randomly class-correlated features to be used by FortalFS. Next, we will apply our algorithm over other artificial datasets that have been used previously in the feature selection literature.

5.2 New Artificial Datasets

In order to directly evaluate the efficiency of FortalFS in dealing with irrelevant, redundant and randomly class-correlated features we have created six new artificial datasets[1] based on the three target concepts (C1, C2 and C3) first introduced by Langley and Sage [Langley and Sage, 1997]. These concepts exhibit increasing complexity in terms of feature interaction. The Table 5.1 below describes the concepts:

Name	Target Concept
C1	$A \wedge B \wedge C$
C2	$(A \wedge B) \vee (A \wedge C) \vee (B \wedge C)$
C3	$(A \wedge B \wedge C) \vee (\overline{A} \wedge \overline{B} \wedge \overline{C})$

Table 5.1: Target concepts used in the new artificial datasets.

Based on these three target concepts, we have generated six datasets by introducing either irrelevant, redundant or randomly class-correlated features. Since redundant and class-correlated features are more related to each other, we have decided to add them together in our datasets and keep the irrelevant features separate.

In the next two sections, we will describe the new datasets and evaluate the performance of FortalFS on them.

For all artificial datasets, we have run FortalFS using LVF as underlying selection algorithm. We tried FortalFS with $6 \cdot N$ and $10 \cdot N$ iterations, where N refers to the number of features in the original dataset, in order to evaluate whether an increased number of iterations in FortalFS results in better performance in selecting only the relevant features.

[1] Similar datasets have been created in [Hall, 1998], but since class-correlated feature were not considered we have decided to create our own new data from scratch.

We used C4.5 as learning algorithm. We also show the results for LVF alone in the attempt to understand how this algorithm helps filtering some of the non-relevant features. In all cases, we have randomly selected half of the instances in each dataset for feature selection. The results shown in the following tables represent the number of times each feature in the dataset was selected in 100 executions of each algorithm.

5.2.1 Irrelevant Features

We have added 10 uniformly random, and consequently irrelevant, features to datasets based on the target concepts C1, C2 and C3 described earlier to create A1I, A2I, A3I, respectively. Thus, dataset A1I contains features A, B and C, target concept C1 and 10 new uniformly random features. Similarly, dataset A2I has features A, B and C, concept C2 and 10 random features. The same for dataset A3I. The Table 5.2 below specifies the features in each new artificial dataset:

Feature	Description
A,B,C	relevant features
I1,...,I8	uniformly random boolean features
I9	uniformly random 5-valued feature
I10	uniformly random 20-valued feature

Table 5.2: Features in artificial datasets A1I, A2I and A3I.

In order to verify whether the FortalFS behaviour changes depending on the number of values an irrelevant feature can be assigned, we have added both binary and multi-valued irrelevant features.

Figure 5.1 shows the number of times each feature was selected by both LVF, FortalFS(6·

N) and FortalFS($10 \cdot N$) on the dataset A1I.

First, the results show that LVF chose the relevant features A, B and C in the majority of the cases and it is able to eliminate most of the irrelevant features in most cases. Interestingly, this algorithm considers features I9 and I10 relevant on several occasions. One may explain this fact by considering that when calculating the inconsistency count for a particular subset, LVF numbers the pairs of instances that match except for their class labels. This way, since I9 and I10 can receive more values than other binary features, finding matching instances is harder when these two features are included. Consequently, LVF will prefer subsets with I9 and I10 rather than with other irrelevant features.

For FortalFS, both settings selected only the three relevant features in all cases. This result can be easily attributed to the fact that using only the relevant features will generate more accurate classification models. Therefore, this FortalFS evaluation criterion will eliminate all irrelevant features that passed through LVF.

Figure 5.2 presents the results for A2I. For this dataset, LVF continues to select irrelevant features in some cases. Nevertheless, the number of non-relevant features being picked for this dataset is in fact smaller than for A1I. For A1I, LVF selected 125 irrelevant features overall. Yet for A2I, only 94. As a matter of fact, the more complex concept used in this dataset seems to make LVF more capable of distinguishing between relevant and irrelevant features due to an increased degree of feature interaction. In addition, this higher degree of interaction may also explain the perfect number of relevant features being selected.

One more time, both FortalFS settings selected only A, B and C and not a single irrelevant feature in all 100 runs.

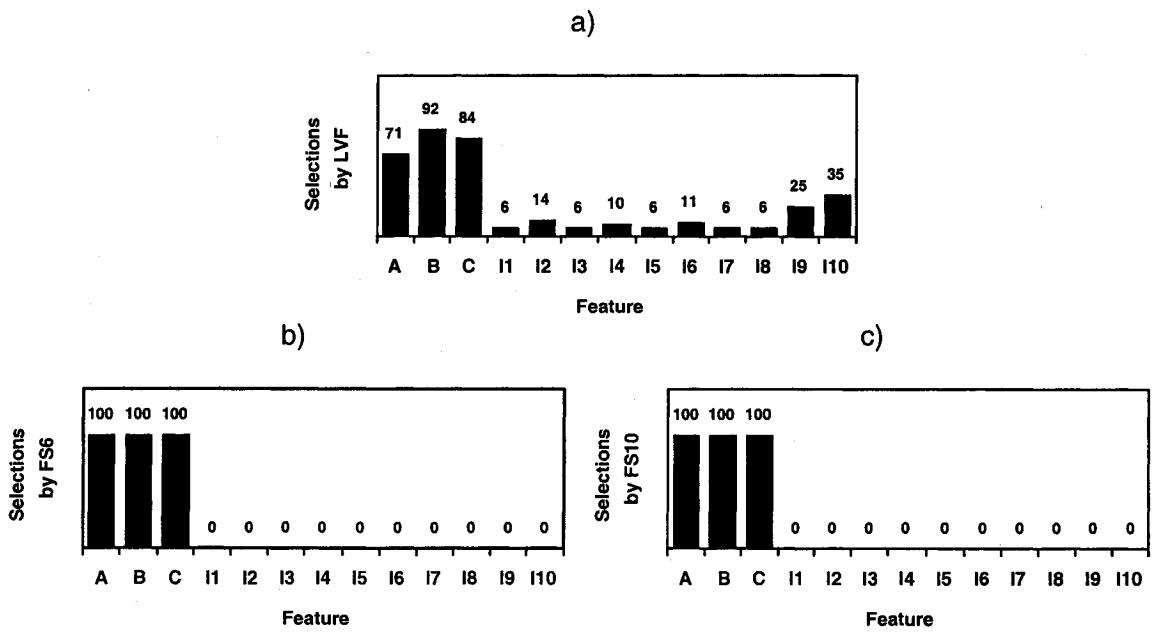

Figure 5.1: Results for A1I Dataset. Number of times each feature is selected in 100 runs of a) LVF b) FortalFS $(6 \cdot N)$ and c) FortalFS $(10 \cdot N)$.

Finally, the results for dataset A3I are described in Figure 5.3. One can observe that LVF remains selecting some irrelevant features but that number decreases as more complex concepts are used. For this datasets, only 85 non-relevant features were selected overall.

Both FortalFS$(6 \cdot N)$ and FortalFS$(10 \cdot N)$ kept their perfect performance by choosing only relevant features in all cases.

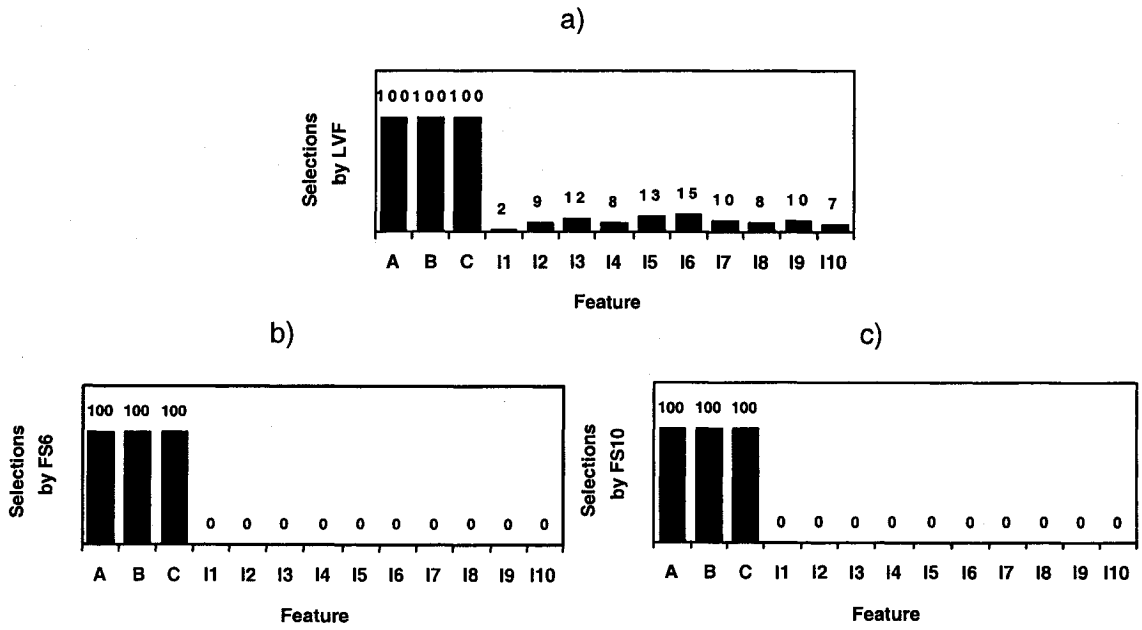

Figure 5.2: Results for A2I Dataset. Number of times each feature is selected in 100 runs of a) LVF b) FortalFS $(6 \cdot N)$ and c) FortalFS $(10 \cdot N)$.

5.2.2 Redundant and Randomly Class-correlated Features

Starting with the datasets generated with our three target concepts C1, C2 and C3, we have developed the artificial datasets A1RC, A2RC and A3RC by including both redundant and randomly class-correlated features. The added redundant features reproduce a relevant one 50 or 75% of the time and are uniformly random otherwise. The randomly class-correlated features were created in a similar way except that they relate to the target value. Table 5.3 describes all features present in datasets A1RC, A2RC and A3RC.

We can see from Figure 5.4 that LVF selected a total of 57 redundant features and 24 class-correlated when using dataset A1RC. However, we expect that this number will

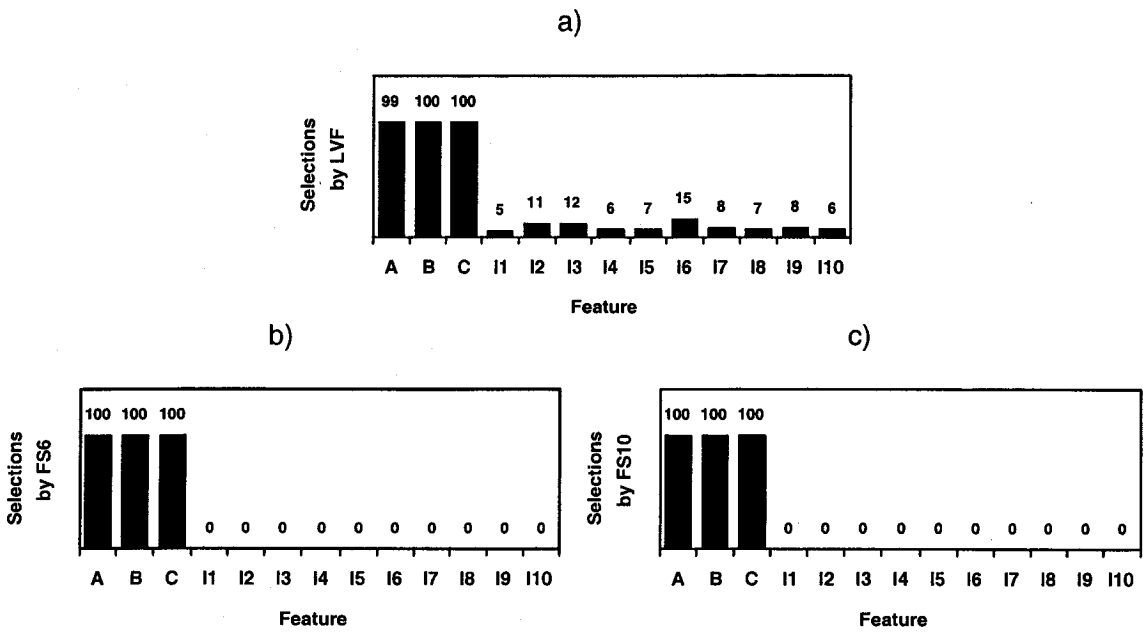

Figure 5.3: Results for A3I Dataset. Number of times each feature is selected in 100 runs of a) LVF b) FortalFS $(6 \cdot N)$ and c) FortalFS $(10 \cdot N)$.

Feature	Description
A,B,C	relevant features
A50,B50,C50	features 50% redundant with A, B and C, respectively uniformly random otherwise
A75,B75,C75	features 75% redundant with A, B and C, respectively uniformly random otherwise
C150	feature 50% correlated to target class
C175	feature 75% correlated to target class

Table 5.3: Features in artificial datasets A1RC, A2RC and A3RC.

fall as we increase the complexity of the target concept similarly to what happened in the previous section with irrelevant features.

Even though C4.5 can be tricked to select randomly class-correlated features [John et al., 1994], since subsets with only relevant features generate more accurate learning models than those containing additional randomly class-correlated features, both FortalFS settings removed all class-correlated features. Similarly, the inclusion of redundant features hurt the performance of ML models and are therefore excluded in the FortalFS process.

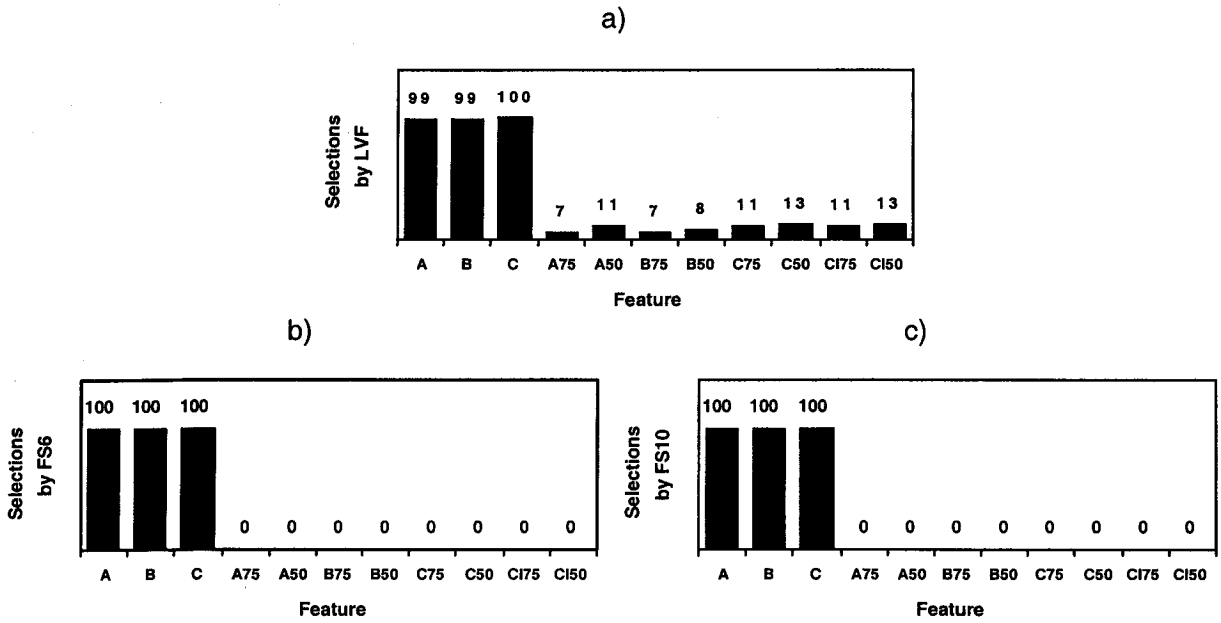

Figure 5.4: Results for A1RC Dataset. Number of times each feature is selected in 100 runs of a) LVF b) FortalFS $(6 \cdot N)$ and c) FortalFS $(10 \cdot N)$.

Figure 5.5 presents the experimental results for LVF and FortalFS when we consider the A2RC artificial dataset. As expected from the increase in feature interaction, the number of non-relevant features selected by LVF decreased to a total of 53. All of them

are eventually completely eliminated by both FortalFS settings that, once again, selected only relevant features in all runs.

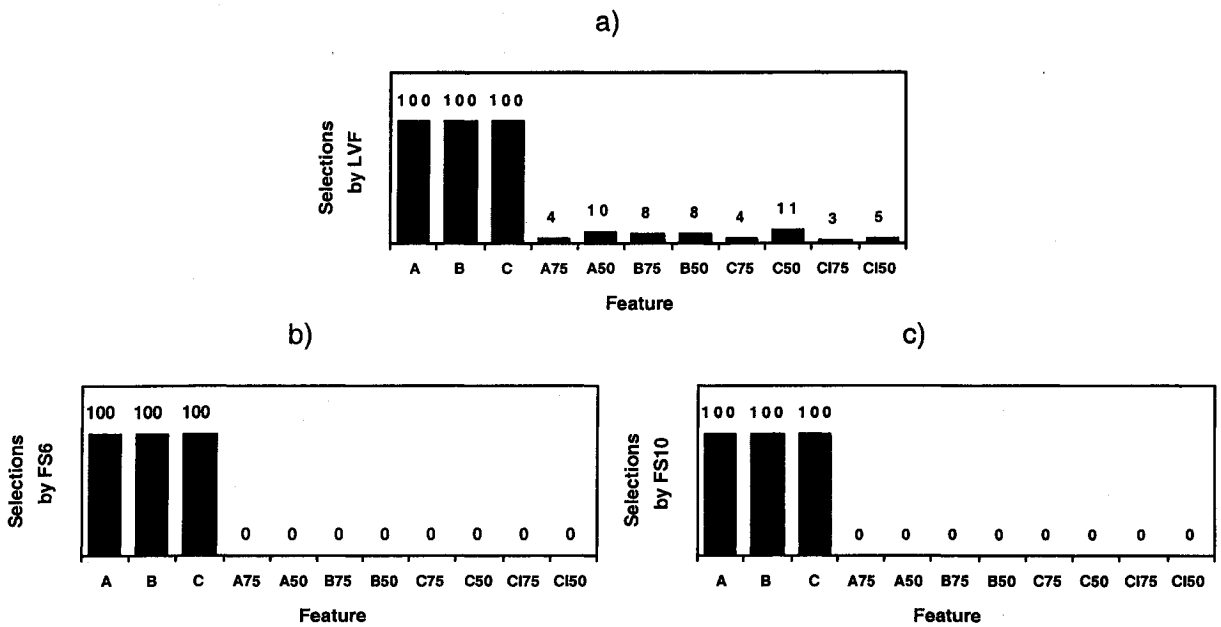

Figure 5.5: Results for A2RC Dataset. Number of times each feature is selected in 100 runs of a) LVF b) FortalFS $(6 \cdot N)$ and c) FortalFS $(10 \cdot N)$.

Once more, Figure 5.6 confirms the improved performance of LVF as we use more complex concepts and the perfect ability of FortalFS in removing both redundant and randomly class-correlated features.

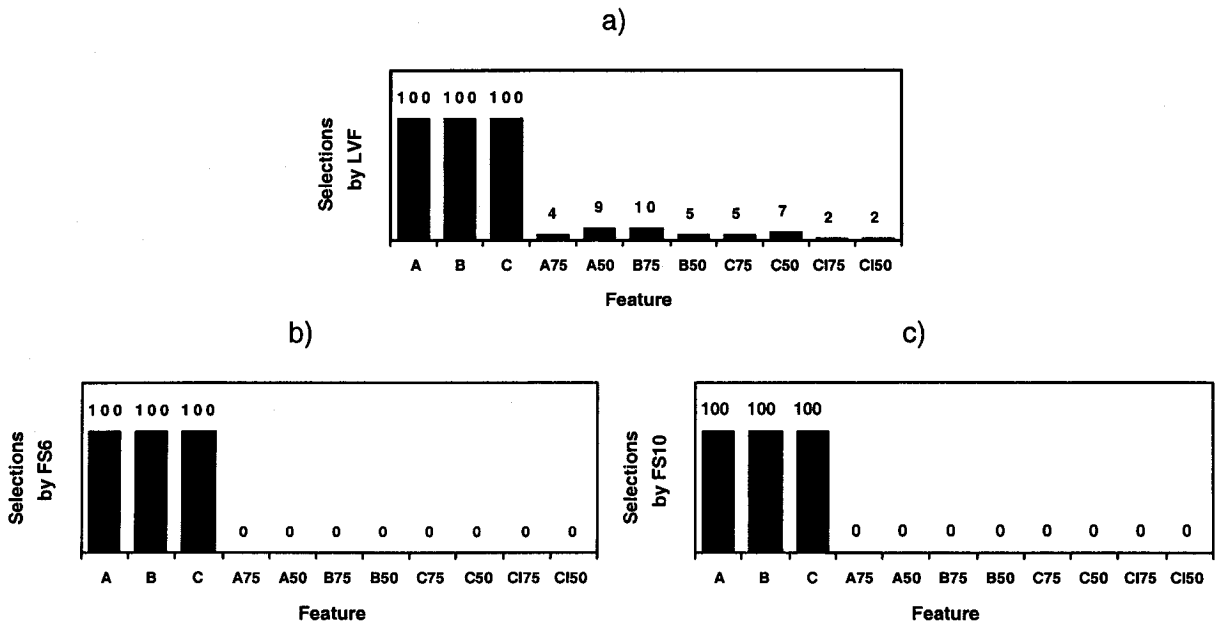

Figure 5.6: Results for A3RC Dataset. Number of times each feature is selected in 100 runs of a) LVF b) FortalFS $(6 \cdot N)$ and c) FortalFS $(10 \cdot N)$.

5.3 Other Artificial Datasets

In the previous section, we have created and used six new artificial datasets to study how well FortalFS can deal with irrelevant, redundant and randomly class-correlated features. In this section, we will expand our evaluation now employing other artificial datasets that have been proposed and studied before in the feature selection literature. The idea is that by applying FortalFS to well-known datasets we have yet another basis for comparison with other selection systems. The datasets to be used are CorrAL [John et al., 1994], Parity5+5 and Monk's problems [Thrun and et al., 1991].

Once again, we have tried these datasets with LVF, FortalFS$(6 \cdot N)$ and FortalFS$(10 \cdot N)$.

In all cases we have randomly selected half of the instances in each dataset for feature selection. The results presented next show the number of times each feature was selected in 100 executions of each algorithm.

5.3.1 CorrAL

The CorrAL dataset has been noted by previous researchers [John et al., 1994, Dash and Liu, 1997] as particularly difficult for some feature selection methods, especially filters. This artificial dataset generated by John and others [John et al., 1994] specially for research in feature selection contains 6 features (A, B, C, D, I, Co). The target concept is the boolean function $(A \wedge B) \vee (C \wedge D)$. I is a completely irrelevant feature and Co matches the class value 75% of the time and it is entirely random the other 25%.

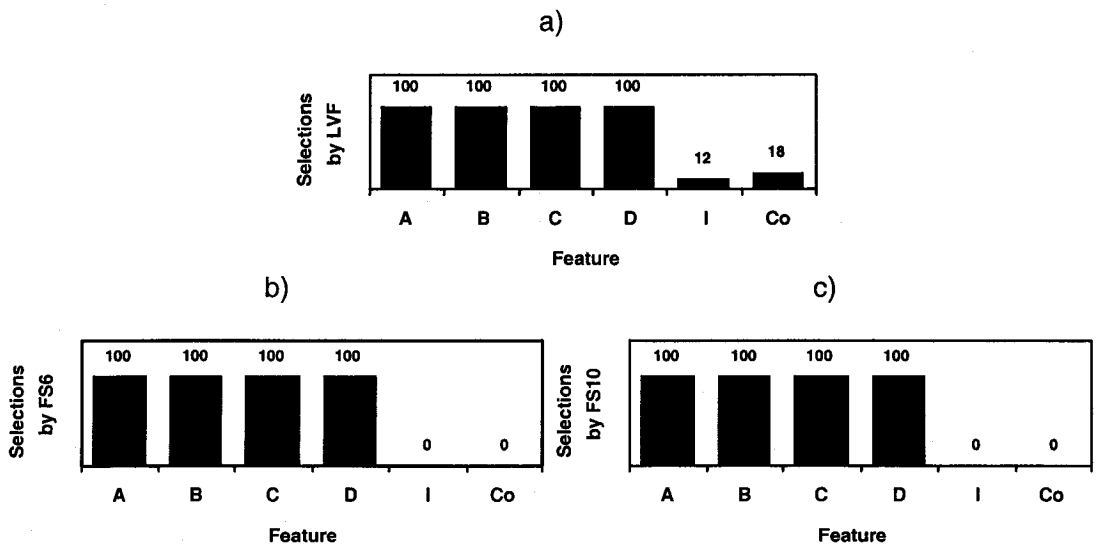

Figure 5.7: Results for CorrAL Dataset. Number of times each feature is selected in 100 runs of a) LVF b) FortalFS $(6 \cdot N)$ and c) FortalFS $(10 \cdot N)$.

Figure 5.7 shows that LVF was able of always selecting the four relevant features as well as removing both the irrelevant and the randomly class-correlated feature most of the time. FortalFS, on the other hand, in addition to maintaining all relevant features also got rid of both I and Co in all runs.

5.3.2 Parity5+5

The target concept is the parity of 5 bits. Other 5 uniformly random and consequently irrelevant features are also present in the dataset.

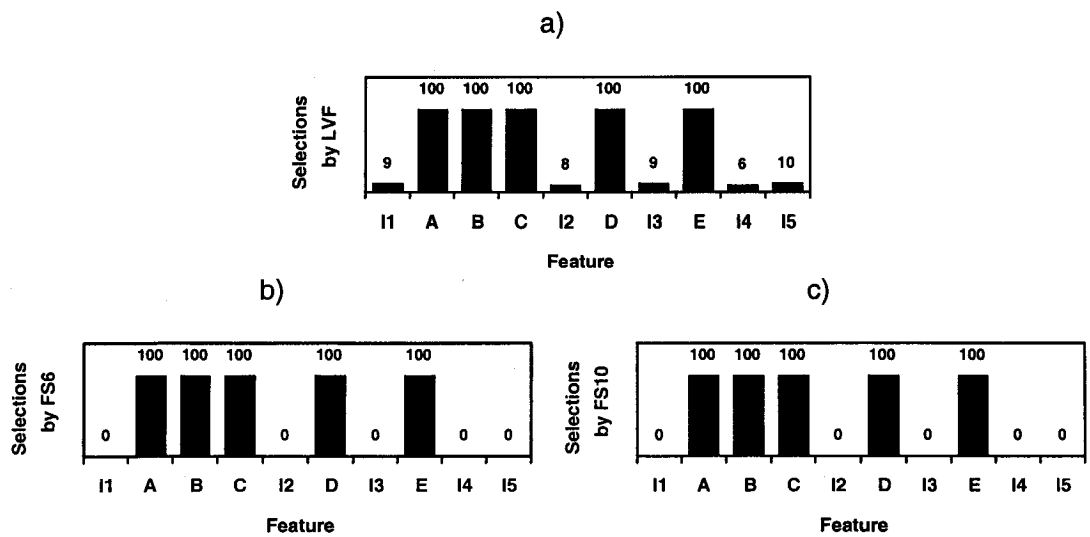

Figure 5.8: Results for Parity5+5 Dataset. Number of times each feature is selected in 100 runs of a) LVF b) FortalFS $(6 \cdot N)$ and c) FortalFS $(10 \cdot N)$.

Once again, as shown in Figure 5.8, LVF always selected all relevant features along with a few random ones. Overall, this algorithm picked 42 irrelevant features in 100 executions. One can also see from these results that both FortalFS settings removed all non-relevant

features in all cases.

5.3.3 Monk's Problems

First used to compare the performance of several learning algorithms [Thrun and et al., 1991], these three datasets involve irrelevant features, noise and a high degree of interaction among features. The three datasets contain 6 features each. For Monk1 and Monk3, 3 features are relevant (A, B and C) and 3 are irrelevant ($I1$, $I2$ and $I3$). For Monk2, all features are relevant to the target concept.

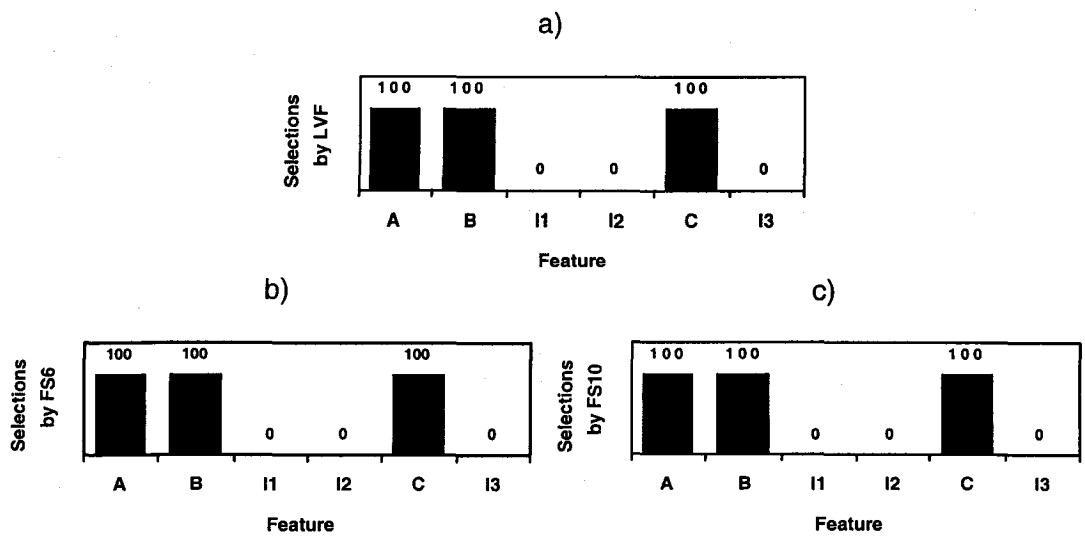

Figure 5.9: Results for Monk1 Dataset. Number of times each feature is selected in 100 runs of a) LVF b) FortalFS $(6 \cdot N)$ and c) FortalFS $(10 \cdot N)$.

Figures 5.9, 5.10 and 5.11 reveal that both LVF and the two FortalFS settings performed perfectly with the three datasets Monk1, Monk2 and Monk3. For Monk1 and Monk3, they have removed the three irrelevant features and kept the relevant ones in all executions.

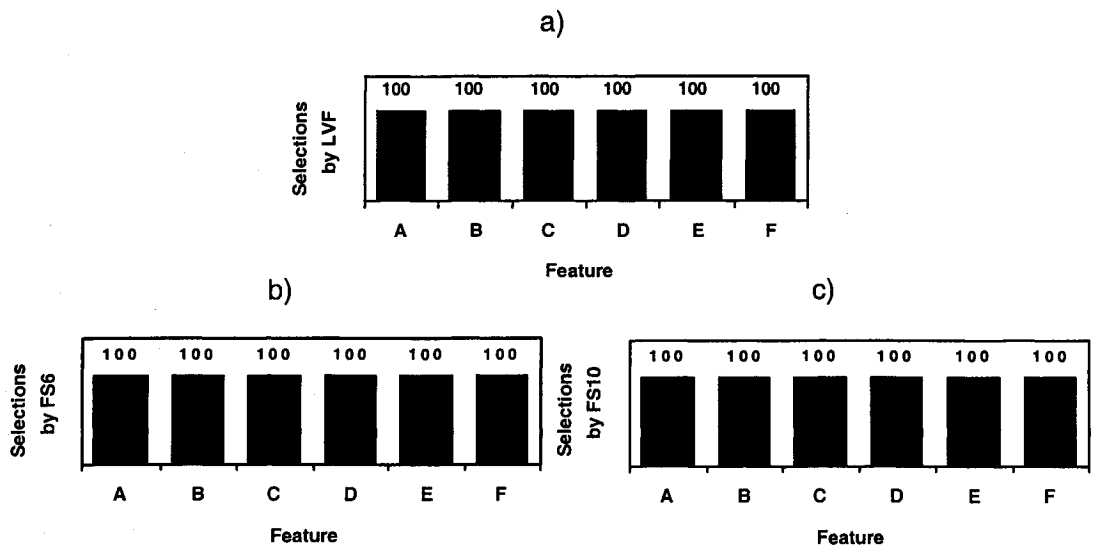

Figure 5.10: Results for Monk2 Dataset. Number of times each feature is selected in 100 runs of a) LVF b) FortalFS $(6 \cdot N)$ and c) FortalFS $(10 \cdot N)$.

And for Monk2, all six features have been kept. Note that since not a single non-relevant features have been selected by LVF, the FortalFS algorithm in fact only reproduced the result presented to him by LVF.

5.4 Conclusion

We have examined in this chapter how well our new feature selection algorithm can deal with irrelevant, redundant and randomly class-correlated features, since the presence of these types of features may in fact hurt the classification ability of a learning model.

In order to accomplish that we have first created six new artificial datasets based on increasingly complex target concepts. Additionally, we have applied FortalFS to previously

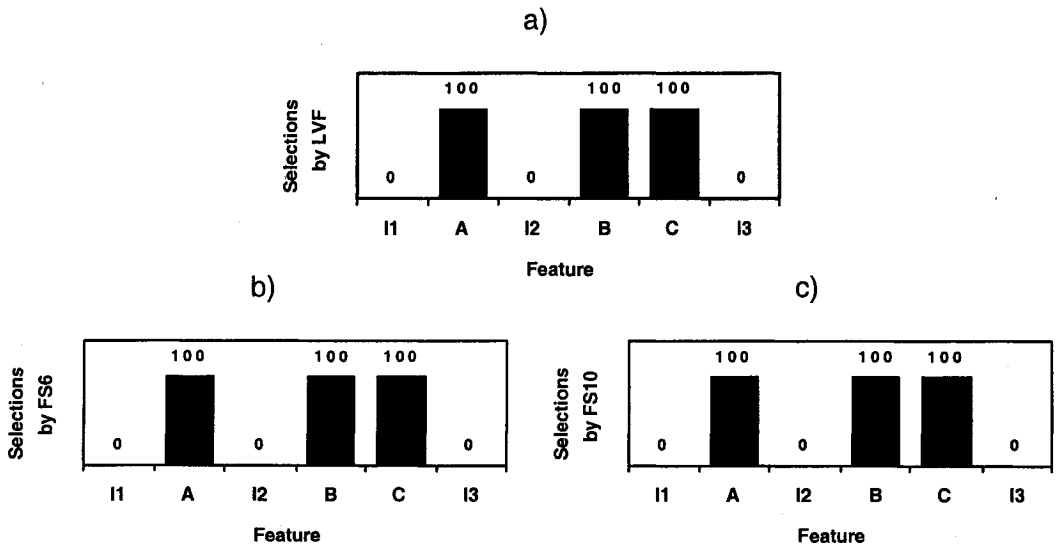

Figure 5.11: Results for Monk3 Dataset. Number of times each feature is selected in 100 runs of a) LVF b) FortalFS $(6 \cdot N)$ and c) FortalFS $(10 \cdot N)$.

proposed artificial datasets.

All results demonstrated that FortalFS is able to identify and remove both irrelevant, redundant and randomly class-correlated features. In addition, our experiments showed that the less time consuming setting of FortalFS using $6 \cdot N$ iterations is sufficient to consistently get rid of these undesirable features in a dataset.

The results presented in this chapter not only confirm our third claim about our new feature selection algorithm, which stated that FortalFS should filter both irrelevant, redundant and randomly class-correlated features, but can also help explain the good performance of this algorithm in the experiments with real-world datasets.

Chapter 6

Parallelizing Feature Selection

In a few minutes a computer can make a mistake so great that it would have taken many men many months to equal it.

Unknown

6.1 Introduction

Computer science researchers of the most diverse fields have recognized the usefulness of parallelism when dealing with complex problems. In artificial intelligence, new versions of standard algorithms and new parallel techniques have emerged to solve traditional AI problems [Kitano and Hondler, 1994]. Genetic algorithms [Pettey et al., 1987, Jog et al., 1990, Cantu-Paz, 2000] and image [Jochem and Baluja, 1993, Pissaloux, 2003] and natural language [Chung et al., 1993, Adriaens and Hahn, 1994] processing are among the first fields to benefit from this approach.

[Freitas, 1998] surveys several works on parallel data mining related to rule induction, instance-based learning, genetic algorithms and neural networks. In addition, his article proposes a set of principles for designing efficient parallel data mining systems. In [Zaki, 1999], a large number of parallel algorithms for mining association rules are reviewed and open problems in this field are discussed. Several further discussions and researches can be found in [Freitas and Lavington, 1998, Kargupta and Chan, 2000, Zaki and Ho, 2000]. From a more theoretical stand point, [Skillicorn, 1999] presents practical complexity measures for parallel programming used to assess different parallelization strategies for data mining algorithms.

In feature selection, surprisingly not many attempts have be made at applying parallelism. [Miller et al., 1995] propose a parallel algorithm based on the Sum of Squared Differences strategy designed to select features in images. Experimental results suggests the approach lends itself well to parallelization. Other researchers have used the inherently parallel characteristic of genetic algorithm to design new parallel feature selection algorithms. A parallel variant of genetic algorithm is used in [Punch et al., 1993]. Here, sets of individuals (representing subsets of features) of a single population are passed out

to individual processors for evaluation with K-nearest-neighbor. Since evaluation dominates the rest of the GA operations, the approach can achieve near linear speedup. In [Melab et al., 2002], parallelism is applied over a genetic algorithm to create a new wrapper feature selection system. Authors report improved accuracies when applying this approach on a near-infrared (NIR) spectroscopic application and linear speedup.

In this chapter, we discuss how parallelism can be used to improve the performance of feature selection algorithms. In particular, we present, discuss and evaluate a coarse-grained parallel version of our feature selection algorithm FortalFS.

6.2 Parallelizing Feature Selection Algorithms

In this section, we will study a few important aspects that should be considered when trying to create parallel solutions to the feature selection problem. First, we will describe some characteristics that are essential to help us generate parallel solutions from sequential algorithms. Next, we will study the potential of parallelization of some well-known feature selection algorithms discussed earlier.

6.2.1 Parallelizing a Sequential Algorithm

Parallelizing a sequential algorithm may constitute a hard task depending on what kind of problem such an algorithm solves and how the algorithmic solution is organized. However, we can identify a few clear characteristics that make certain algorithms lend themselves better to parallelism than others. They are:

Easy Partitioning The first step in parallelizing a sequential algorithm is to break the solution into discreet "chunks" of work that can be distributed to multiple tasks. This is known as decomposition or partitioning. Such partitioning is straight-forward for certain algorithms (for example, solutions based on a main loop that can be decomposed in smaller ones) and virtually impossible occasionally (in purely sequential solutions, for example.).

Independent Partitioning It refers to how independently a partition task can be executed from other tasks. That will determine the amount of communication that needs to be exchanged in order for all tasks to finish. Higher task dependency results in more communication and consequently more problems with communication overhead and task synchronization.

Easy Load Balancing It refers to the practice of distributing work among tasks so that all tasks are kept busy all of the time. It can be considered a minimization of task idle time. Such a balance has an obvious influence on performance.

Algorithms that can be easily partitioned in independent tasks which can be equally distributed to be executed by parallel processors are consequently great candidates for parallelization. Thus, parallel solutions generated from such algorithms tend to achieve very high levels of efficiency.

With that in mind, we study next the potential of parallelization of several feature selection algorithms described previously in this thesis. Each algorithm will be analyzed under the three characteristics above and classified for their potential in generating efficient parallel solutions. We will use the following classes in our classification:

Hard Parallelization Algorithms in this class present solutions that are sequential in nature, i.e., cannot be run before the execution of all previous commands. In this

cases, any attempt to parallelize such algorithms would generate a level of communication or synchronization that would make such a solution impractical.

Easy Parallelization The solution may be broken into smaller subtasks that can be execute in parallel requiring a relatively small amount of interaction with other subtasks. The parallelization of these algorithms can bring good gains in terms of speedup but communication may become relevant as the number of parallel nodes grows.

Obvious Parallelization Algorithms that are organized in smaller subtasks that can be executed completely independently from each other. Such tasks can then be placed in separate processors with an overall minimum amount of communication and synchronization required. In this case, high efficiency can be achieve even when a high number of parallel processors is used.

6.2.2 Classifying the Feature Selection Algorithms

In this section, we will analyse several feature selection algorithm presented easier including FortalFS for their potential in generating efficient parallel solutions. In order to do that, we will present and analyse an obvious strategy of parallelization for each algorithm. Such strategy will help us identify potential difficulties in this parallelization process and classify each algorithm according to the classes presented is the previous section. However, it is important to mention that the parallelization strategy described here should not be considered the only one, or the best one for that matter. In fact, different solutions can be designed. In addition, small modifications in the sequential version of some algorithms may help us adapt them for parallelization. Thus, our parallel strategy will serve only for analyzing the sequential original solution.

Best-First Search

This search strategy can be easily parallelized by decomposing the total number of subsets to be evaluated at each step in equal parts and passing this subsets to parallel processors. Each processor can then calculate its local best and return it to a central task that will compute the global best of that iteration. The straight-forward partitioning and load balancing make the design of this parallel algorithm very simple. However, the fact that the process described above needs to be repeated for each iteration brings both communications and some synchronization overheads (not so relevant because of the good load balancing).

Classification: Easy Parallelization

Genetic Search

Genetic algorithms are interesting candidates for effective parallelization since they are inspired by the principle of evolution, in parallel, of a population of individuals that represent solutions to a certain problem. In fact, several different parallelization strategies can be applied: in a global parallelization solution, the evaluation of the individual fitness is carried out in a parallel form. In a more coarse-grained solution, the population is divided in smaller subpopulations that evolve independently and simultaneously in different processors. The population can also be broken is very small subpopulations (sometimes with only one individual). This way, GA operations are applied by considering neighboring individuals.

Flexibility in designing a parallel version of a GA-based feature selection algorithm makes this solution an obvious candidate for parallelization.

Classification: Obvious Parallelization

LVF

Ever though LVF has a main loop that can be easily partitioned in equally smaller loops, since the evaluation of a subset generated in each iteration is dependent of previous iterations, a parallel version of this algorithm would require a considerable amount of communication and would equally suffer from synchronization problems.

Classification: Hard Parallelization

Relief

Since Relief updates the weights of the features independently of previous computation, one could partition the total number of iterations in equal parts and pass them to separate processors. At the end, a main process would compute the final weights based on the local values.

Classification: Obvious Parallelization

Focus

Just like Best-First Search, the computation in each iteration in Focus can be parallelized be dividing all subsets that need to be evaluated in equal groups that are computed in parallel. In fact, this solution would suffer from the same communication and synchronization

problems since all parallel computation of an iteration needs to be done before the next iteration can start.

Classification: Easy Parallelization

Forward Wrapper

With Forward Wrapper, a straightforward parallelization strategy would be to carry out the evaluation of the subsets in a parallel way. Since evaluation usually dominates the computation of wrapper algorithms (much more than with filters), the parallelization of this task has the potential to generate very efficient solution. One could in fact parallelize each iteration by passing the evaluation of equal fractions of the total number of subsets in that iteration to different processors. Here, both the communication and synchronization overheads would be relatively smaller than with filters, due to the time-dominating characteristic of the evaluation function.

Classification: Easy Parallelization

Random Wrapper

Differently from Forward Wrapper, all iterations in the Random Wrapper algorithm are carried out independently of previous computation. In fact, the computation of each parallel processor can be done from start to end without any interaction with other tasks simply by dividing the total number of subsets to be generated and evaluated in equal parts and passing this computation to other processors. At the end, a central process can receive local bests and compute a final result. Indeed, this design requires a minimum

amount of communication.

Classification: Obvious Parallelization

FortalFS

Since all FortalFS iterations can also be performed independently of results obtained from previous iterations, in addition to the fact that the exact number of subsets to be generated is known before hand, this algorithm is a perfect candidate for parallelization.

Classification: Obvious Parallelization

6.3 The Parallel FortalFS Algorithm

From the previous chapters, it is clear that FortalFS is a feature selection algorithm that performs extremely well compared to other solutions in terms of accuracy but that suffers somewhat from time efficiency. In addition, this algorithm lends itself very well to parallelization due to some particular characteristics. All that makes FortalFS a perfect candidate for parallelization.

In this section, we will present, discuss and evaluate a parallel version of the FortalFS feature selection algorithm. This parallel solution is designed to solve the time efficiency constrains of FortalFS without hurting its selection precision.

The design of ParallelFortalFS was based on the Master-Slave design pattern [Buschmann, 1995]. According to this pattern, a master process distributes work to identical slave components

and computes a final result from the results these slaves return. This design was picked over others (Producer-Consumer, Pipeline, Tree) first because of its simplicity and mainly because it is the paradigm that suits this particular situation best.

In ParallelFortalFS, the master process distributes the work among slave processes, that will calculate a local best subset, and computes the global best subset from these results. A more detailed description of both master and slave processes can be found in the next subsections.

This implementation of ParallelFortalFS will require a minimum number of messages being exchanged between processes. Master and slaves will communicate only once when the slaves are created and another time when slaves send their local best. In addition, slaves will not communicate among themselves. In fact, the exact number of messages (remote method calls) exchanged between processes will be $2 \times (NumProcessors - 1)$, since one slave will be created per processor (and one will be in the same processor as the master.). The reduced amount of communication provided by this implementation makes us expect and predict a high efficiency level in practice, particularly when larger datasets are used (when communication will represent an even smaller fraction of the total execution time).

6.3.1 The Master Process

The master process (Figure 6.1) starts by running a feature selection system and calculating the Adam vector. This first part is done sequentially. Next, this component is responsible for starting the slaves in their corresponding processors (hosts) and distributing the work equally among them. In order to accomplish that, it will simply divide the total number of iterations by the number of available processors (slaves) and assign this new number of iterations to each of the slave processes. Finally, the master will receive each slave's local

main$(D, Iter, NumP)$

1. O = FeatureSelector(D)
2. $Adam$ = CalculateAdam(O)
3. $IterP = Iter/NumP$
4. **for** $i = 1$ **to** $NumP$
5. *start* $Slave_i$
6. ***call*** $calcLocalBest(D, Adam, IterP)$ of $Slave_i$

calcGlobalBest$(lBestSet, lBestError)$

1. **if** $lBestError < gBestError$ **then**
2. $gBestSet = lBestSet$
3. $glBestError = lBestError$
4. $count++$
5. **if** $count = NumProc$ **then**
6. *quit*

---

where:

D - dataset.

$Iter$ - total number of iterations.

$NumP$ - number of processors available.

$lBestSet$ - local best subset of features.

$lBestError$ - error rate of local best subset.

Figure 6.1: ParallelFortalFS - the Master process.

best subsets and calculate the global optimum.

6.3.2 The Slave Process

The slave component of ParallelFortalFS will iteratively generate a new subset according to the Adam vector, evaluate this subset with a ML system and update local variables that will store the best subset and its accuracy, see Figure 6.2. The final local best subset and its corresponding accuracy will then be sent to the master process.

calcLocalBest($D, Adam, IterP$)

1. **for** $i = 1$ **to** $IterP$
2. S = GenerateSubset($Adam$)
3. **if** Error(S, D) < Error($lBestSet, D$) **then**
4. $lBestSet = S$
5. $lBestError = Error(S, D)$
6. ***call*** $calcGlobalBest(lBestSet, lBestError)$ of $Master$

---

where:

D - dataset.

$Adam$ - $Adam$ vector.

$IterP$ - number of iteration per processor.

Figure 6.2: ParallelFortalFS - the Slave process.

6.4 Evaluation

In order to measure how much the FortalFS performance can be improved by employing the parallel solution proposed in this chapter, we have implemented ParallelFortalFS and executed it over several datasets. This evaluation will additionally help us verify the claim that FortalFS lends itself well to parallelization.

The ParallelFortalFS algorithm was implemented in Java with RMI (Remote method Invocation)[1] [Sun Microsystems, 2003]. This implementation of ParallelFortalFS was run on a 100-Mbps local network composed of Pentium 4 PCs running Windows NT. We tried ParallelFortalFS with 1, 2, 4, 8 and 16 computers (processors), using LVF as underlying feature selection algorithm, $10 \cdot N$ iterations (N is the initial number of features in the dataset) and 5 UCI datasets (mushroom(22 features), ionosphere(34 features), splice(60 features), sonar(60 features) and audiology(69 features)).

For each dataset, we calculated the following values according to the equations below:

$$Speedup_p = \frac{TotTime_1}{TotTime_p} \tag{6.1}$$

$$OptSpeedup_p = \frac{TotTime_1}{(SeqTime + \frac{ParTime_p}{p})} \tag{6.2}$$

$$ParEff_p = \frac{Speedup_p}{OptSpeedup_p} \tag{6.3}$$

[1]RMI (Remote method Invocation) is a mechanism that enables the development of Java parallel/distributed application in distributed-memory system. It allows methods of remote Java objects to be invoked from another JVM in a different host. RMI is the Java version of what is generally known as a remote procedure call (RPC). RMI uses object *serialization* to marshal and unmarshal parameters.

$$TotEff_p = \frac{Speedup_p}{p} \tag{6.4}$$

where p is the number of processors used, $TotTime_1$ is the total elapsed execution time (considering both the sequential and the parallel parts of the algorithm) when only one processor is used. $TotTime_p$ is the total execution time when p processors are used. $SeqTime$ is the execution time of the sequential part of the code alone and $ParTime_p$ the parallel execution time for p processors. Thus, $TotTime_p = SeqTime + ParTime_p$.

Intuitively, $Speedup_p$ is a measure that indicates how many times faster the parallel program runs compared to its sequential counterpart when p processors are used. The $OptSpeedup_p$ (optimum speedup) represents the maximum speedup that can be achieved when not taking into considering the execution time of the sequential part of the algorithm. Thus, a $Speedup_p = OptSpeedup_p$ means that the parallel part of the code reached its maximum performance. This $OptSpeedup_p$ measure is calculate based on Amdahl's Law [Amdahl, 1967]. Another important measure, called *linear speedup*, is used to represent the maximum speedup (p) a parallel program can achieve when running completely in parallel (with no sequential code). Thus, linear speedup is graphically represented by a 45% diagonal.

The $ParEff_p$ (parallel efficiency) value measures how effectively the p processors are being used by the parallel fraction of the program. $TotalEff_p$ (total efficiency) indicates efficiency as a whole. These two variables will receive values ranging from 0 to 1, 1 representing a perfect efficiency.

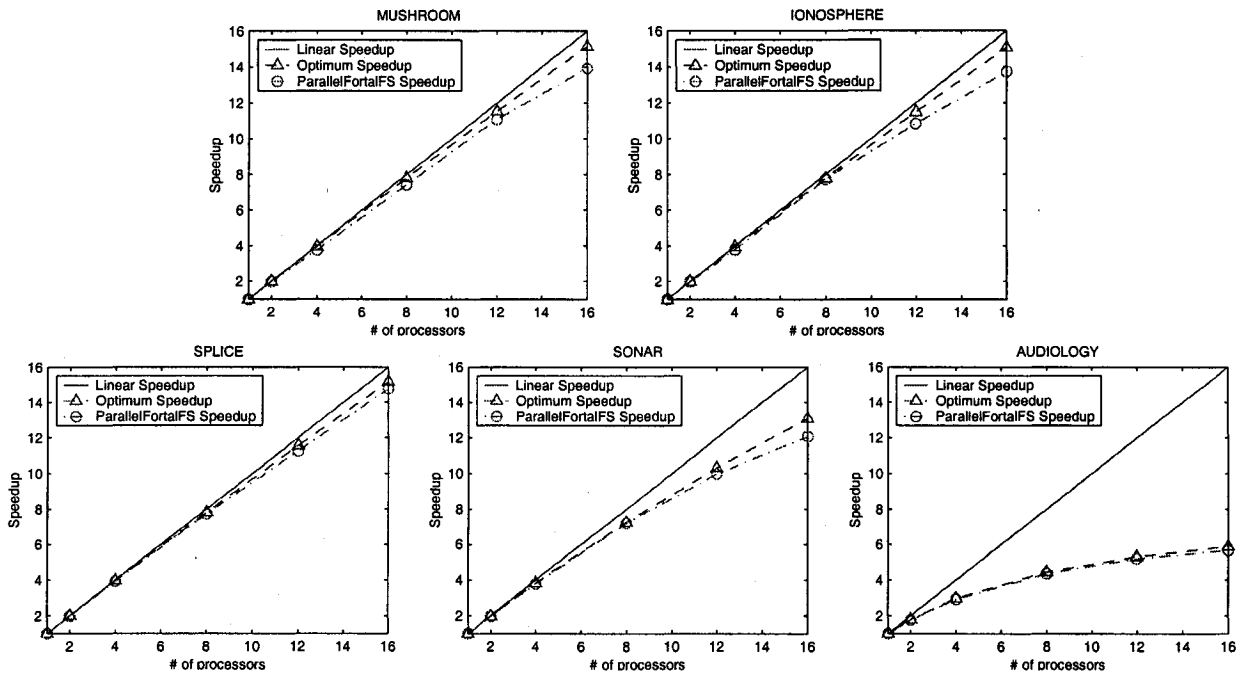

Figure 6.3: ParallelFortalFS experimental results - Speedup.

6.4.1 Results and Analysis

In terms of accuracy and number of features selected, note that the performance of ParallelFortalFS is identical to the performance of the sequential FortalFS. Thus, the results reported in chapter 4 (FortalFS Evaluation with Real-World Datasets) still hold.

The detailed results can be found is Appendix E. The graphs in figures 6.3 and 6.4 describe the speedups and efficiencies obtained for each dataset, respectively.

From the results obtained in our experiments, we can reach a few conclusions about the ability of ParallelFortalFS to speed up the FortalFS algorithm. We describe a few of these conclusions next:

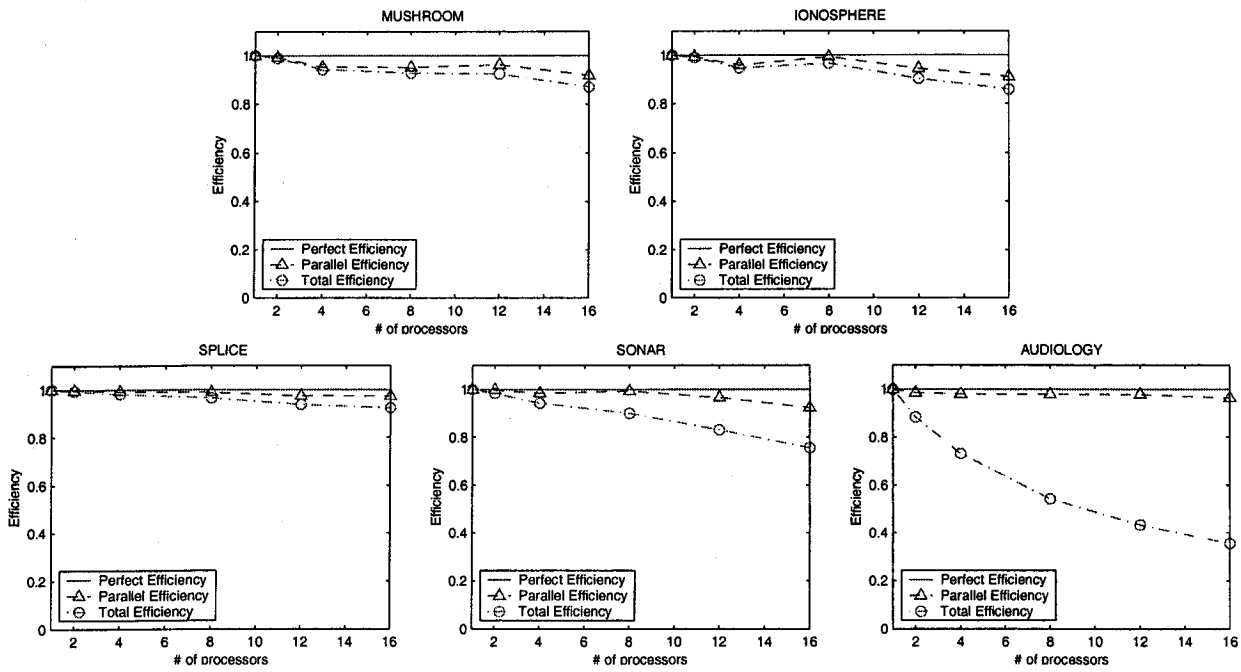

Figure 6.4: ParallelFortalFS experimental results - Efficiency.

1. ParallelFortalFS achieves a very high speedup when we use 2 processors, with an average parallel efficiency of 99.34% and total efficiency of 96.84% (Figure 6.3).

2. Total efficiency (Figure 6.4) drops as the number of processors being used increases. Despite that, ParallelFortalFS as a whole was able to run 12 times faster on average for the five datasets when using 16 processors.

3. The drop in the total efficiency can be attributed to the sequential fraction of the code, since the sequential execution time becomes more relevant as the parallel execution time decreases. For the Audiology dataset, for instance, total efficiency drops drastically due to the relatively high sequential execution time while parallel efficiency persists almost unaltered with the increased number processors being used.

4. If we consider the parallel efficiency alone, ParallelFortalFS is able to maintain a high performance in all cases, with averages of 99.34%, 97.34%, 98.10%, 96.44% and 93.74% for 1, 2, 4, 8 and 16 processors, respectively, see Figure 6.4. Thus, this parallel implementation achieves near optimal efficiency in most cases.

5. A few facts may help explain the small drop in parallel efficiency: First, the process of generating new subsets is non-deterministic. This fact can cause a particular slave to generate subsets that are on average larger that the ones from other slaves. Since larger subsets require more computation to be evaluated, slaves with larger subsets will take longer to finish. This execution time variance increases as the number of subsets being computed by each slave goes down, which is exactly what happens when we increase the number of processors. Second, the datasets must be read (at least once) by each slave process so that processing can start. As the number of slaves (processors) increases and consequently the parallel execution time decreases, the time required for reading the datasets become more and more relevant.

6. We were able to estimate the communication time to be 2, 6, 14, 22 and 30 milliseconds when 2, 4, 8, 12, and 16 processors are used, respectively. Thus, since we are dealing with relatively large datasets, communication does not seem to affect the ParallelFortalFS performance in any relevant way.

6.5 Evaluation on Larger Datasets

The results presented and described in the previous section give us a very good indication of the efficiency of our parallel design to speedup the performance of the feature selection algorithm discussed here. However, since practical feature selection usually requires for algorithms to deal with much larger datasets in terms of dimensionality, we thought it

necessary to extend our experiments to include datasets with more than 100 features. For that purpose, we have selected three datasets:

Clean1 This data (from UCI Repository [Blake and Merz, 1998]) induces a classifier to predict whether new molecules are musks or non-musks from 92 molecules of which 47 are judged by human experts to be musks and the remaining 45 molecules are judged to be non-musks. 168 features and 476 instances.

Arrhythmia The aim is to distinguish between the presence and absence of cardiac arrhythmia and to classify it in one of the 16 groups, also from UCI [Blake and Merz, 1998]. 279 features and 452 instances.

Madelon Extracted from the NIPS2003 feature selection challenge, this dataset is used to classify random data. It contains 500 features and 2000 training instances.

The FortalFS algorithm was able to improve the classification power of these three datasets after feature selection as described earlier in chapter 6.

The detailed results can be found is Appendix E. The figures 6.5 and 6.6 show respectively the speedup and efficiency of our parallel algorithm for this three datasets and up to 16 processors.

The results are consistent with the ones obtained with smaller datasets. In terms of speedup, our algorithm is able to archive a performance very similar to the optimum one is all cases, especially for our larger dataset Madelon.

As far as efficiency goes, the algorithm performance is again hurt by its sequential part, causing a decrease in total efficiency as we increase the number of processors, in especial for

Clean1 and Arrhythmia. However, the consistently high parallel efficiency demonstrates the power of our design.

The Madelon dataset, with 500 features and 2000 training examples, shows the potential of our parallel algorithm to be used with much larger datasets, when time allows. For this data, since feature evaluation dominates the execution time (the initial sequential processing is less relevant), we were able to maintain a very high efficiency as we add processors, with 87% and 96% of total and parallel efficiency for 16 processors, respectively.

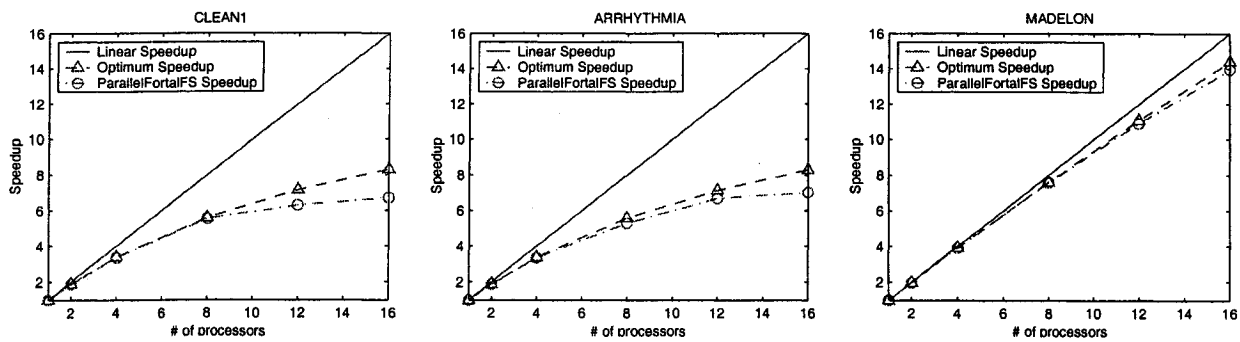

Figure 6.5: ParallelFortalFS experimental results (large datasets) - Speedup.

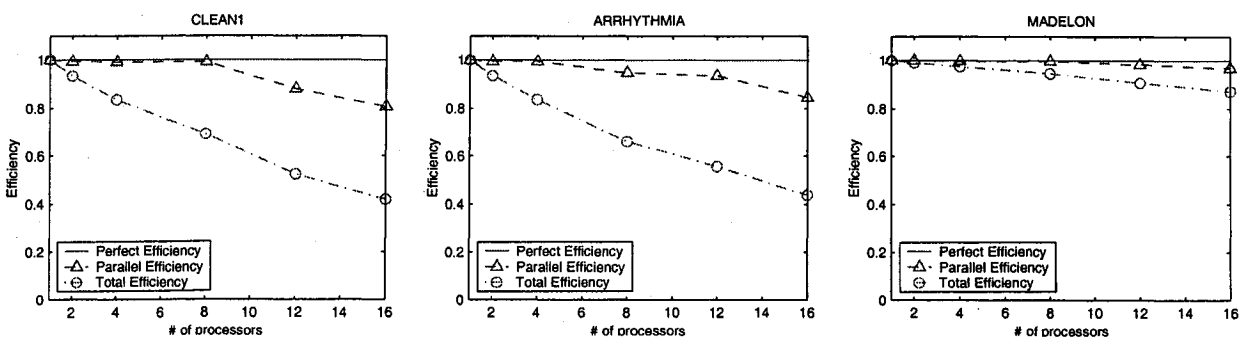

Figure 6.6: ParallelFortalFS experimental results (large datasets) - Efficiency.

6.6 Conclusion

FortalFS is a new feature selection algorithm that extracts and combines the best characteristics of filters and wrappers in one algorithm, namely, an efficient heuristic used to search through subsets of features and a precise evaluation criterion, respectively. It generates highly accurate subsets by computing a number of estimates of what the best features for a given problem are and combining them together. This additional work performed by FortalFS allows for better results but with extra computational cost. The analysis of the FortalFS algorithm has shown that this solution lends itself very well to parallelization due to particular characteristics of this algorithm.

We presented in this chapter a coarse-grained parallel version of FortalFS based on the Master-Slave design pattern. In our design, the master process is responsible for running another feature selection system and calculating the Adam vector, starting the slaves in their corresponding processors, distributing the work equally among them and calculating the final best subset from results received from the slaves. The slaves generate new subsets according to the Adam iteratively, evaluate them with a ML system and at the end send their local best subsets to the master process. This design allows for a very reduced amount of communication among processes. Thus, master and slaves will communicate only once when the slaves are created and another time when slaves send their local best and slaves will not communicate among themselves.

In order to evaluate this approach, we implemented this parallel algorithm in Java with RMI (Remote method Invocation) and ran it in a local network with up to 16 processors and 5 UCI datasets (mushroom, ionosphere, splice, sonar and audiology) and 3 extra larger datasets (clean1, arrhythmia and madelon). The experimental results allowed us to conclude that the parallel implementation of FortalFS presented here addresses, at the

practical level, the scalability of FortalFS, at least on datasets with up to 500 features. The results showed that we were able to achieve near optimum speedups in the context of the Amdahl's Law.

The results obtained with our FortalFS parallel implementation show that parallelization can considerably improve the time performance of FortalFS. This shows that, in addition to yielding a very good subset of features, FortalFS is able to run in very reasonable time. As an illustration of this performance gain, consider that our parallel implementation of FortalFS ran 12 times faster on average when using 16 processors and the five small and medium dataset used in our evaluation. By projecting this result to the experiments reported in Chapter 4 of this thesis and contrasting it with the famously time efficient filters, we can predict that, by using 16 processors, our slower FortalFS setting would perform about only 3.9 times slower than LVF, about twice slower than Focus and only 20% slower than Relief. If we double the number of processors to 32, we estimate that our algorithm would execute more efficiently than Focus and Relief. The time performance of FortalFS could virtually reach the performance of its underlying algorithm as the number of parallel nodes grows.

In addition, the results obtained with our FortalFS parallel implementation confirm our final claim about the FortalFS practical behaviour which predicted that parallelization could considerably improve the time performance of our algorithm.

Chapter 7

Conclusion

The danger from computers is not that they will eventually get as smart as men,
but that we will meanwhile agree to meet them halfway.
Bernard Avishai

7.1 Extended Summary

Classification is a key problem in machine learning. Algorithms for classification have the ability to predict the outcome of a new situation after having been trained on data representing past experience. Each example is described by a fixed number of attributes, or *features*, along with a label that denotes its class.

A number of factors influence the performance of induction algorithms, including the number and quality of features provided to describe the data, the training and testing data distribution, and others. The factor we focus on in this thesis is the number and quality of features present in the sample data. Intuitively, one could think that the more features there are available to describe some data, the more accurate the classifier created from such data should be. However what is seen in practice is that the more features there are the more training examples are needed to induce an accurate classification model. This problem is sometimes referred to as "the curse of dimensionality" [Bellman, 1961] and can be explained by the fact that the number of solutions (classifiers) that should be considered from a high dimensional dataset increases exponentially with the number of features, making it more difficult for a learning algorithm to find an accurate one. Also, as a consequence, when a large number of features is present, a greater computational effort is needed in order to compute these features.

The *Feature Selection* problem involves discovering a subset of features such that a classifier built only with this subset would have better predictive accuracy than a classifier built from the entire set of features. In practice, feature selection algorithms will discover and select features of the data that are *relevant* to the task to be learned.

In addition to irrelevant features, feature selection researchers have identifies other examples of problematic features which may have a negative impact on the performance

of learning systems such as redundant features and randomly class-correlated features. *Redundant* features refer to those that, even when relevant to a target concept, provide mostly information already present in another feature or set of features and, in fact, do not contribute to getting better predictors. *Randomly class-correlated* features are correlated to the target class most of the time and random otherwise.

A large number of algorithms have already been proposed for the feature selection problem. Although significantly different with regards to 1) the search strategy they use to determine the right subset of features and 2) how each subset is evaluated, feature selection algorithms are usually classified in three general groups: *Filters*, *Wrappers* and *Hybrid* solutions.

In this thesis, we proposed a new general hybrid system for the problem of feature selection in machine learning. The idea behind this new algorithm, FortalFS, is to extract and combine the best characteristics of filters and wrappers in one algorithm, namely, an efficient heuristic used to search through subsets of features and a precise evaluation criterion, respectively.

We started by extending a framework for studying and classifying different types of filter and wrapper feature selection algorithms. Under this framework, more than 50 algorithms are categorized and described. This framework is developed by taking into consideration the two main characteristics of any feature selection algorithm, namely a generation procedure and an evaluation function. The framework can be used to find unexplored areas that may generate new feature selection algorithms. We also generated a framework to help us organize and study hybrid feature selection methods by considering both the type of filter evaluation measure and the classifier used by hybrid solutions. The FortalFS algorithm is able to assume any position in the framework, hence the qualification of FortalFS as a general hybrid approach. To finalize our literature review, we summarized

a few feature selection methods for Support Vector Machine (SVM) learning after a short introduction of SVM.

Next, we presented FortalFS, a new hybrid solution for feature selection that uses results from another feature selection system as a starting point in the search through subsets of features that are evaluated by a machine learning algorithm. The idea is that, with an efficient heuristic, we can decrease the number of subsets of features to be evaluated by the learning algorithm, consequently decreasing computational effort and still be able to select an accurate subset. We have also designed a variant of the original algorithm in the attempt to work with feature weighting algorithm.

Based on the design of the FortalFS algorithm, we made several claims about the behaviour of this algorithm regarding its ability in selecting relevant features, its time efficiency, its power in dealing with some problematic kinds of features and, finally, its ability in having its performance drastically improved through parallelism. Each claim was experimentally verified in the thesis.

In order to evaluate our new algorithm, a number of experiments were run and the results compared to well-known feature selection filter and wrapper algorithms, such as Focus, Relief, LVF, and others. Such experiments were run over 13 datasets from the UCI Repository and three different learning algorithms. Results showed that FortalFS outperforms most of the algorithms significantly. However, it presents time-consuming performance similar to that of some wrappers. We have also tried FortalFS with different underlying algorithms as well as with larger datasets.

Additional experiments using specially designed artificial datasets demonstrated that FortalFS is able to identify and remove both irrelevant, redundant and randomly class-correlated features.

The FortalFS time-consumption issue was addressed through parallelism. A parallel version of FortalFS based on the master/slave design pattern was implemented and evaluated. In several experiments, we were able to achieve near optimal speedups.

7.2 Thesis Contributions

In this section we will examine how we achieved each one of the expected contributions described in Section 1.7.

A novel and general hybrid solution for feature selection.

The FortalFS algorithm is described in Chapter 3. This hybrid algorithm uses results from another feature selection system as a starting point in the search through subsets of features that are evaluated by a machine learning algorithm. Its unique general characteristic allows for the use of any kind of filter evaluation method and classifier. Also, in Section 3.6, we presented a variant of the FortalFS algorithm, named FortalFS-W, which makes possible the use with feature weighting algorithms as underlying algorithm.

An extensive experimental evaluation of several feature selection algorithms.

In Chapter 4, we presented a comprehensive empirical evaluation of some of the most important feature selection algorithms to date as they get pitted against FortalFS. This evaluation underlies the various strengths and weaknesses of each algorithm. We have used in our evaluation the following algorithms: Best-First Search, Genetic Search (GA), LVF, Relief, Focus, Forward Wrapper, Backward Wrapper and a Random Wrapper. For each experiment, we have obtained performance measures such as the accuracy of the selected subset, the time used for selection and the number

of features selected. The accuracy for each selected subset was obtained as follows: the original dataset was randomly and equally split into a selection set and a testing set. The feature selection in all cases was performed considering only the selection set. For the wrappers, we used 5-fold cross validation on the selection set as evaluation strategy. The selected subset was then evaluated using 5-fold cross validation on the testing set. Both the filters and the wrappers as well as FortalFS were implemented in Java by using the Weka library [Garner, 1995] as a starting point. The 13 datasets used in the experiments were obtained from the UCI Repository [Blake and Merz, 1998]. We also used three different learning algorithms (C4.5, Naive Bayes and k-Nearest Neighbour). We have reported a detailed description of the results in Appendixes A, B and C for C4.5, Naive Bayes and k-Nearest Neighbour, respectively, including the statistical significance of the accuracy difference between FortalFS and all other algorithms for each dataset and learning algorithm. In a nutshell, the experimental results showed that FortalFS relevantly outperformed well-known filters in our study, namely LVF, Focus and Relief. Such improvements were obtained, however, with certain increase in time consumption. In addition, we could find a clear difference in terms of performance between the FortalFS search strategy and the search method employed by forward and backward wrappers in favor of FortalFS.

The extension of a framework for studying and classifying filter and wrapper feature selection algorithms.

In our Literature Review, Chapter 2, we have expanded Dash and Liu's framework [Dash and Liu, 1997] to cover the most recent developments in filter and wrapper feature selection research. By taking into consideration the two main characteristics of a feature selection algorithm, namely a generation procedure and an evaluation function, we have classified and described more then fifty filter and wrapper algo-

rithms.

A framework for studying and classifying hybrid solutions in feature selection.

Also in Chapter 2, we introduced a new framework to help us organize and study feature selection hybrid methods. This framework was created by considering both the type of filter evaluation measure and the classifier used by hybrid feature selection algorithms. We have then classified and described five existing hybrid solutions. The framework helped us realize that the feature selection field could benefit from a general algorithm that could assume any position in the framework just by "plugging" different evaluation methods.

The study of how parallelism can be used to speedup the performance of feature selection algorithms, in general, and our novel solution, in particular.

In Chapter 6, we studied how parallelism can be used to improve the performance of feature selection algorithms. First, we described some characteristics (easy partitioning, independent partitioning and easy load balancing) that are essential to help one generate efficient parallel solutions from sequential algorithms. Next, we studied the potential of parallelization of some well-known feature selection algorithms including Best-First Search, Genetic Search, LVF, Relief, Focus, Forward Wrapper, Backward Wrapper, Random Wrapper and finally FortalFS. Each algorithm was analyzed under the three characteristics above and classified for their potential in generating efficient parallel solutions. After identifying that our new hybrid algorithm lends itself very well to parallelization due to its basic characteristics, we have designed, presented, discussed and evaluated a parallel version of the FortalFS feature selection algorithm. The design of ParallelFortalFS was based on the Master-Slave design pattern [Buschmann, 1995]. According to this pattern, a master process distributes work to identical slave components and computes a final result from the results these slaves return. In order to measure how much the FortalFS performance can be improved by

employing the proposed parallel solution, we have implemented ParallelFortalFS and executed it over several datasets. The ParallelFortalFS algorithm was implemented in Java with RMI (Remote method Invocation) [Sun Microsystems, 2003] and run on a 100-Mbps local network composed of Pentium 4 PCs running Windows NT. We tried ParallelFortalFS with 1, 2, 4, 8 and 16 computers (processors), using LVF as underlying feature selection algorithm, $10 \cdot N$ iterations (N is the initial number of features in the dataset). We used five UCI datasets (mushroom(22 features), ionosphere(34 features), splice(60 features), sonar(60 features) and audiology(69 features)) and three extra larger datasets (clean1, arrythmia and madelon). The experimental results showed that we were able to achieve near optimum speedups in the context of the Amdahl's Law.

7.3 Future Research

We will indicate in this section a few directions for future researches related to this thesis.

7.3.1 More Experimental Analyses

We have designed and executed a considerable number of experiments in order to compare our new feature selection algorithm against well known filter and wrapper solutions. The results obtained from these experiments were used primarily to contrast FortalFS with the other methods. Thus, additional analyses of the results can be used to help discover the strengths and weaknesses of each algorithm. For instance, by comparing the performance of the Random Wrapper against Forward and Backward Wrappers, one could analyse what is the exact gain in performance for using a non-random search strategy.

7.3.2 Parameters Analysis

When considering the number of parameters associated with the FortalFS algorithm, we can observe that its behaviour can be changed by experimenting distinct settings. In fact, by trying different parameters we can not only better understand our algorithm but also learn how to make better use of it in practice.

Trying other Underlying Filters and Classifiers

The FortalFS algorithm allows for the use of different filters and classifiers. Different combinations may be, in practice, better in specific situations. By trying other underlying filters and classifiers, one can not only find a good combination for a particular dataset but also identify the best FortalFS setting to a given class of problems. For instance, one could try a combination of distance-based methods or try to find different evaluation strategies that complement each other.

Examining the Influence of k

For the FortalFS evaluation presented and discussed in this thesis, we have empirically selected the parameter k to be ten. To get to this particular number, we have tried different values for several datasets of small and medium sizes. Since most of our experiments used small and medium size datasets, this choice of k seems reasonable. However, for larger datasets, or even different or specific datasets, different values of k may in fact generate better selections.

Studying Overfitting

Another parameter that has an obvious influence in the final performance of FortalFS is the number of iterations. We have seen in our experiments that as the number of iterations increases, the accuracy of the final selected subset over unseen data

also increases. However, we have tried only three settings: $10 \cdot N$, $8 \cdot N$ and $6 \cdot N$ iterations, where N is the initial number of features in the dataset. One may expect that by drastically increasing the number of iterations, the predictive accuracy of the selected subsets may start decreasing due to overfitting. Thus, it would be interesting to study the overfitting problem in FortalFS. This study would also help in finding an optimal number of iterations that would generate the best final subset.

7.3.3 More Comparisons

We have compared the FortalFS algorithm with well-known filter and wrapper feature selection algorithms. These comparisons serve their purpose of helping introduce and understand our novel approach. However, in order to have a better justification for the practical use of FortalFS, it is essential to compare FortalFS with more recent and promising feature selection algorithms, such as hybrid solutions and SVM-based methods.

Hybrid Methods

The comparison between FortalFS and other hybrid feature selection algorithms is a very natural choice when considering that FortalFS itself employs a hybrid strategy. Such comparisons would be useful in identifying the advantages, and disadvantages for that matter, of having a general hybrid solution as opposed to more specific ones.

SVM Methods

Support Vector Machines (SVM) have received great attention by machine learning researchers and have in fact demonstrated an impressive ability in generating accurate predictive models. As previously described, some very recent SVM-based feature selection methods have been proposed. By comparing FortalFS with such methods,

we could estimate what kind of advantage, if any, SVM-specific methods would have over our general approach.

7.3.4 Instance Selection in FortalFS

It has been observed [Nock, 2004] that the FortalFS algorithm could benefit from instance selection. The idea is that, at each iteration, the instances filtered with the current subset would be evaluated and then unwanted instances would be removed. The instances would be evaluated with respect to their informative value. By improving data quality at each step, FortalFS could have both its time performance and its predictive power improved.

7.3.5 Generalizing FortalFS

FortalFS is a general hybrid solution since neither of its evaluation components are fixed. However, independently of which combination of evaluation measures is to be used in a particular FortalFS setting, the stochastic guided search strategy that links these two components remains the same. Thus, one could consider generalizing this linking step by allowing FortalFS to employ different methods to use the information obtained from the filter in searching the subset search space. This more general solution could be seen as a framework for hybrid feature selection algorithms.

Bibliography

[Adriaens and Hahn, 1994] Adriaens, G. and Hahn, U. (1994). *Parallel Natural Language Processing.* Ablex Publishing Corporation.

[Aha et al., 1991] Aha, D., Kibler, D., and Albert, M. (1991). Instance-based learning algorithms. *Machine Learning*, 6(1):37–66.

[Almuallim and Dietterich, 1991] Almuallim, H. and Dietterich, T. (1991). Learning with many irrelevant features. In *Proceedings of the Ninth National Conference on Artificial Intelligence (AAAI'91)*, volume 2, pages 547–552, Anaheim, CA. AAAI Press.

[Almuallim and Dietterich, 1994] Almuallim, H. and Dietterich, T. (1994). Learning boolean concepts in the presence of many irrelevant features. *Artificial Intelligence*, 69(1-2):279–305.

[Amdahl, 1967] Amdahl, G. (1967). Validity of the single processor approach to achieving large-scale computing capabilities. In *Proceedings of the AFIPS 1967 Spring Joint Computer Conference*, volume 30, pages 483–485, Atlantic City, New Jersey, USA.

[Bala et al., 1996] Bala, J., DeJong, K., Huang, J., Vafaie, H., and Wechsler, H. (1996). Using learning to facilitate the evolution of features for recognizing visual concepts. *Evolutionary Computation*, 4(3):297–311.

[Bellman, 1961] Bellman, R. (1961). *Adaptive Control Processes.* Princeton University Press.

[Blake and Merz, 1998] Blake, C. and Merz, C. (1998). UCI repository of machine learning databases. http://www.ics.uci.edu/~mlearn/MLRepository.html.

[Blum and Langley, 1997] Blum, A. and Langley, P. (1997). Selection of relevant features and examples in machine learning. *Artificial Intelligence*, 97(1-2):245–271.

[Bobrowski, 1988] Bobrowski, L. (1988). Feature selection based on some homogeneity coefficient. In *Proceedings of the Ninth International Conference on Pattern Recognition*, page 544546.

[Buschmann, 1995] Buschmann, F. (1995). *The Master-Slave Pattern.*, pages 133–142. Pattern Languages of Program Design. Addision-Wesley.

[Cantu-Paz, 2000] Cantu-Paz, E. (2000). *Efficient and Accurate Parallel Genetic Algorithms.* Kluwer Academic Publishers.

[Cardie, 1993] Cardie, C. (1993). Using decision trees to improve case–based learning. In *Proceedings of the Tenth International Conference on Machine Learning (ICML'93)*, pages 25–32. Morgan Kaufmann Publishers, Inc.

[Caruana and Freitag, 1994] Caruana, R. and Freitag, D. (1994). Greedy attribute selection. In *Proceedings of the Eleventh International Conference on Machine Learning (ICML'94)*, pages 28–36.

[Chung et al., 1993] Chung, S., Moldovan, D., and DeMara, R. (1993). A parallel computational model for integrated speech and natural language understanding. *IEEE Transactions on Computers*, 42(10):1171–1183.

[Craven and Shavlik, 1993] Craven, M. and Shavlik, J. (1993). Learning to represent codons: A challenge problem for constructive induction. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence*, pages 1319–1324.

[Das, 2001] Das, S. (2001). Filters, wrappers and a boosting-based hybrid for feature selection. In *Proceedings of the Eighteenth International Conference on Machine Learning (ICML'01)*.

[Dash and Liu, 1997] Dash, M. and Liu, H. (1997). Feature selection for classification. *Intelligent Data Analysis - An International Journal*, 1(3):131–156.

[Dash and Liu, 1998] Dash, M. and Liu, H. (1998). Hybrid search of feature subsets. In *Proceedings of the Pacific Rim International Conference on Artificial Intelligence*, pages 238–249.

[De Jong, 1975] De Jong, K. A. (1975). *An analysis of the behavior of a class of genetic adaptive systems.* PhD thesis, University of Michigan.

[Devijver and Kittler, 1982] Devijver, P. and Kittler, J. (1982). *Pattern Recognition: A Statistical Approach.* Prentice Hall Internacional.

[Doak, 1992] Doak, J. (1992). An evaluation of feature selection methods and their application to computer security. Technical report, University of California, Department of Computer Science, Davis, CA.

[Domingos, 1997] Domingos, P. (1997). Context-sensitive feature selection for lazy learners. *Artificial Intelligence Review*, (11):227–253.

[Foroutan and Sklansky, 1987] Foroutan, I. and Sklansky, J. (1987). Feature selection for automatic classification of non-gaussian data. *IEEE Transactions on Systems, Man, and Cybernatics*, SMC-17(2):187–198.

[Freitas, 1998] Freitas, A. (1998). A Survey of Parallel Data Mining. In Arner, H. and Mackin, N., editors, *Proceedings of the 2nd Int Conf on the Practical Applications of Knowledge Discovery and Data Mining*, pages 287–300, London. The Practical Application Company.

[Freitas and Lavington, 1998] Freitas, A. and Lavington, S. (1998). *Mining Very Large Databases with Parallel Processing.* Kluwer Academic Publishers, Boston.

[Garner, 1995] Garner, S. (1995). WEKA: The waikato environment for knowledge analysis. In *Proceedings of the New Zealand Computer Science Research Students Conference*, pages 57–64.

[Guyon and Elisseeff, 2003] Guyon, I. and Elisseeff, A. (2003). An introduction to variable and feature selection. *Journal of Machine Learning Research*, 3:1157–1182. Special Issue on Variable and Feature Selection.

[Guyon et al., 2002] Guyon, I., Weston, J., Barnhill, S., and Vapnik, V. (2002). Gene selection for cancer classification using support vector machines. *Machine Learning*, 46(1-3):389–422.

[Hall, 1998] Hall, M. (1998). *Correlation-based Feature Selection for Machine Learning.* PhD thesis, Waikato University, Department of Computer Science, Hamilton, NZ.

[Hall, 2000] Hall, M. (2000). Correlation-based feature selection for discrete and numeric class machine learning. In *Proceedings of the Seventeenth International Conference on Machine Learning (ICML'00).* Stanford University, CA, Morgan Kaufmann Publishers.

[Hearst et al., 1998] Hearst, M., Schölkopf, B., Dumais, S., Osuna, E., and Platt, J. (1998). Trends and controversies - support vector machines. *IEEE Intelligent Systems*, 13(4):18–28.

[Huber, 1981] Huber, P. (1981). *Robust Statistics.* John Wiley & Sons.

[Ichino and Sklansky, 1984a] Ichino, M. and Sklansky, J. (1984a). Feature selection for linear classifier. In *Proceedings of the Seventh International Confernce on Pattern Recognition*, volume 1, pages 124–127.

[Ichino and Sklansky, 1984b] Ichino, M. and Sklansky, J. (1984b). Optimum feature selection by zero-one programming. *IEEE Trans. on Systems, Man and Cybernetics*, SMC-14(5):737–746.

[Jain et al., 2000] Jain, A., Duin, R., and Mao, J. (2000). Statistical pattern recognition: A review. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(1):4–37.

[Jochem and Baluja, 1993] Jochem, T. and Baluja, S. (1993). Massively parallel, adaptive, color image processing for autonomous road following. Technical Report CMU-RI-TR-93-10, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA.

[Jog et al., 1990] Jog, P., Suh, J., and Gucht, D. (1990). Parallel Genetic Algorithms Applied to the Traveling Salesman Problem. Technical Report 314, Indiana University.

[John et al., 1994] John, G., Kohavi, R., and Pfleger, K. (1994). Irrelevant features and the subset selection problem. In *Proceedings of the Eleventh International Conference on Machine Learning (ICML'94)*, pages 121–129.

[Kargupta and Chan, 2000] Kargupta, H. and Chan, P., editors (2000). *Advance in Distributed and Parallel Knowledge Discovery,*. AAAI Press.

[Kira and Rendell, 1992a] Kira, K. and Rendell, L. (1992a). The feature selection problem: Traditional methods and new algorithm. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, pages 129–134. MIT Press.

[Kira and Rendell, 1992b] Kira, K. and Rendell, L. (1992b). A practical approach to feature selection. In *Proceedings of the Ninth International Workshop on Machine Learning (ML'92)*, pages 249–256, Aberdeen, Scotland. Morgan-Kaufmann.

[Kitano and Hondler, 1994] Kitano, H. and Hondler, J. A. (1994). *Massively Parallel Artificial Intelligence.* The AAAI Press / The MIT Press.

[Kohavi, 1995] Kohavi, R. (1995). The power of decision tables. In Lavrac, N. and Wrobel, S., editors, *Proceedings of the European Conference on Machine Learning (ECML'95)*, Lecture Notes in Artificial Intelligence 914, pages 174–189, Berlin, Heidelberg, New York. Springer Verlag.

[Koller and Sahami, 1996] Koller, D. and Sahami, M. (1996). Toward optimal feature selection. In *Proceedings of the Thirteenth International Conference on Machine Learning (ICML'96)*, pages 284–292.

[Kononenko, 1994] Kononenko, I. (1994). Estimating attributes: Analysis and extensions of RELIEF. In *Proceedings of the European Conference on Machine Learning (ECML'94)*, pages 171–182.

[Kononenko et al., 1996] Kononenko, I., Robnik-Sikonia, M., and Pompe, U. (1996). *ReliefF for estimation and discretization of attributes in classification.*, pages 31–40. Artificial Intelligence: Methodology, Systems, Applications. IOS Press.

[Laine, 2003] Laine, S. (2003). *Using visualization, variable selection and feature extraction to learn from industrial data.* PhD thesis, Helsinki University of Technology, Espoo, Finland.

[Langley and Sage, 1994] Langley, P. and Sage, S. (1994). Oblivious decision trees and abstract cases. In *Working Notes of the AAAI-94 Workshop on Case-Based Reasoning.* AAAI Press.

[Langley and Sage, 1997] Langley, P. and Sage, S. (1997). *Scaling to domains with irrelevant features*, volume 4 of *Computational Learning Theory and Natural Learning Systems*. MIT Press.

[Leite and Brazdil, 2002] Leite, R. and Brazdil, P. (2002). http://www.liacc.up.pt/ml/metal/consortium/doc/featsel_dtree.pdf.

[Liu et al., 1998] Liu, H., Motoda, H., and Dash, M. (1998). A monotonic measure for optimal feature selection. In *Proceedings of the European Conference on Machine Learning (ECML'98)*, pages 101–106.

[Liu and Setiono, 1996a] Liu, H. and Setiono, R. (1996a). Feature selection and classification - a probabilistic wrapper approach. In Tanaka, T., Ohsuga, S., and Ali, M., editors, *Proceedings of the Ninth International Conference on Industrial and Engineering Applications of AI and ES*, pages 419–424, Fukuoka, Japan.

[Liu and Setiono, 1996b] Liu, H. and Setiono, R. (1996b). A probabilistic approach to feature selection - a filter solution. In *Proceedings of the Thirteenth International Conference on Machine Learning (ICML'96)*, pages 319–327.

[Liu and Setiono, 1998] Liu, H. and Setiono, R. (1998). Scalable feature selection for large sized databases. In Cantu, F., Soto, R., Liebowitz, J., and Sucar, E., editors, *Proceedings of the World Congress on Expert Systems*, volume 2, pages 521–528, Mexico-New-York.

[Liu and Yu, 2002] Liu, H. and Yu, L. (2002). Feature selection for data mining.

[Matheus, 1991] Matheus, C. (1991). The need for constructive induction. In *Proceedings of the Eighth International Workshop on Machine Learning*, pages 173–177, Evanston, IL.

[Mehra et al., 1989] Mehra, P., Rendell, L., and Wah, B. (1989). Principled constructive induction. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 651–656.

[Melab et al., 2002] Melab, N., Cahon, S., Talibi, E., and Duponchel, L. (2002). Parallel GA-Based wrapper feature selection for spectroscopic data mining. In Werner, B., editor, *Proceedings of the 16th International Parallel and Distributing Processing Symposium (IPDPS'02)*, pages 201–201, Ft. Lauderdale, Florida, USA.

[Michalski et al., 1983] Michalski, R., Carbonell, J., and Mitchell, T., editors (1983). *Machine Learning — An Artificial Intelligence Approach.* Tioga Publishing Company, Palo Alto, CA.

[Miller et al., 1995] Miller, B., Papanikolopoulos, N., and Carlis, J. (1995). A parallel feature selection algorithm. Technical Report UMSI 95/55, University of Minnesota Supercomputing Institute.

[Modrzejewski, 1993] Modrzejewski, M. (1993). Feature selection using rough sets theory. In Brazdil, P., editor, *Proceedings of the European Conference on Machine Learning (ECML'93)*, pages 213–226.

[Moore and Lee, 1994] Moore, A. and Lee, M. (1994). Efficient algorithms for minimizing cross validation error. In *Proceedings of the Eleventh International Conference on Machine Learning (ICML'94)*, pages 190–198.

[Mucciardi and Gose, 1971] Mucciardi, A. and Gose, E. (1971). A comparison of seven techniques for choosing subsets of pattern recognition properties. *IEEE Transaction on Computers*, (20):1023–1031.

[Müller et al., 2001] Müller, K., Mika, S., Rätsch, G., and Tsuda, K. (2001). An introduction to kernel-based learning algorithms. *IEEE Transactions on Neural Networks*, 12(2):181–201.

[Narendra and Fukunaga, 1977] Narendra, P. and Fukunaga, K. (1977). A branch and bound algorithm for feature subset selection. *IEEE Transactions on Computers*, C-26(9):917–922.

[Ng, 1998] Ng, A. (1998). On feature selection: Learning with exponentially many irrelevant features as training examples. In *Proceedings of the Fifteenth International Conference on Machine Learning (ICML'98)*.

[Nock, 2004] Nock, R. (2004). Private communication.

[Oliveira and Vincentelli, 1992] Oliveira, A. and Vincentelli, A. (1992). Constructive induction using a non-greedy strategy. In *Proceedings of the Ninth International Conference on Machine Learning (ICML'92)*, page 355360, Aberdeen, Scotland. Morgan Kaufmann.

[O'Sullivan et al., 2000] O'Sullivan, J., Langford, J., Caruna, R., and Blum, A. (2000). FeatureBoost: A meta-learning algorithm that improves model robustness. In *Proceedings of the Seventeenth International Conference on Machine Learning (ICML'00)*, pages 703–710.

[Pagallo and Haussler, 1990] Pagallo, G. and Haussler, D. (1990). Boolean feature discovery in empirical learning. *Machine Learning*, 5(1):71–99.

[Pawlak, 1991] Pawlak, Z. (1991). *Rough Sets, Theoretical Aspects of Reasoning About Data*. Kluwer.

[Pettey et al., 1987] Pettey, C., Leuze, M., and Grefenstette, J. (1987). A Parallel Genetic Algorithm. In Grefenstette, J., editor, *Proceedings of the Second International Conference on Genetic Algorithms and Their Applications*, pages 155–161. Lawrence Erlbaum Associates.

[Pfahringer, 1995] Pfahringer, B. (1995). Compression-based feature subset selection. In *Proceedings of the IJCAI-95 Workshop on Data Engineering for Inductive Learning.*

[Pissaloux, 2003] Pissaloux, E. (2003). Parallel vision processing and dedicated parallel architectures. In *Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS'03)*, Nice, France.

[Press et al., 1988] Press, W. et al. (1988). *Numerical Recipes in C.* Cambridge University Press.

[Pudil et al., 1994] Pudil, P., Novovičová, J., and Kittler, J. (1994). Floating search methods in feature selection. *Pattern Recogn. Lett.*, 15(11):1119–1125.

[Punch et al., 1993] Punch, W., Goodman, E., Pei, M., Chia-Shun, L., Hovland, P., and Enbody, R. (1993). Further research on feature selection and classification using genetic algorithms. In Forrest, S., editor, *Proceedings of the Fifth Int. Conf. on Genetic Algorithms*, pages 557–564, San Mateo, CA. Morgan Kaufmann.

[Queiros and Gelsema, 1984] Queiros, C. and Gelsema, E. (1984). On feature selection. In *Proceedings of the Seventh International Conference on Pattern Recognition*, volume 1, pages 128–130.

[Quinlan, 1986] Quinlan, J. (1986). Induction of decision trees. *Machine Learning*, (1):81–103.

[Rakotomamonjy, 2003] Rakotomamonjy, A. (2003). Variable selection using svm based criteria. *Journal of Machine Learning Research*, 3:1157–1182. Special Issue on Variable and Feature Selection.

[Reunanen, 2003] Reunanen, J. (2003). Overfitting in making comparisons between variable selection methods. *Journal of Machine Learning Research*, 3:1371–1382. Special Issue on Variable and Feature Selection.

[Richeldi and Lanzi, 1996] Richeldi, M. and Lanzi, P. (1996). ADHOC: A tool for performing effective feature selection. In *Proceedings of the International Conference on Tools with Artificial Intelligence*, pages 102–105.

[Rissanen, 1978] Rissanen, J. (1978). Modeling by shortest data description. *Automatica*, 14:465–471.

[Rozsypal and Kubat, 2003] Rozsypal, A. and Kubat, M. (2003). Selecting representative examples and attributes by a genetic algorithm. *Intelligent Data Analysis*, 7(4):291–304.

[Scherf and Brauer, 1997] Scherf, M. and Brauer, W. (1997). Feature selection by means of a feature weighting approach. Technical Report FKI-221-97, Forschungsberichte kunstliche Intelligenz, Institut fur Informatik, Technische Universitat Munchen.

[Sebban and Nock, 2002] Sebban, M. and Nock, R. (2002). A hybrid filter/wrapper approach of feature selection using information theory. *Pattern Recognition*, (35):835 846.

[Segen, 1984] Segen, J. (1984). Feature selection and constructive inference. In *Proceedings of the Seventh International Conference on Pattern Recognition*, page 13441346.

[Skalak, 1994] Skalak, D. (1994). Prototype and feature selection by sampling and random mutation hill climbing algorithms. In *Proceedings of the Eleventh International Conference on Machine Learning (ICML'94)*, pages 293–301.

[Skillicorn, 1999] Skillicorn, D. (1999). Strategies for parallel data mining. *IEEE Concurrency*, 7(4):26–35.

[Stracuzzi and Utgoff, 2002] Stracuzzi, D. and Utgoff, P. (2002). Randomized variable elimination. In *Proceedings of the Nineteenth International Conference on Machine Learning (ICML'02)*.

[Sun Microsystems, 2003] Sun Microsystems (2003). Java Remote Method Invocation (RMI). http://java.sun.com/products/jdk/rmi/.

[Thrun and et al., 1991] Thrun, S. and et al. (1991). The MONK's problems, a performance comparison of different learning algorithms. Technical Report CMU-CS-91-197, Carnegie Mellon University.

[Vafaie and De Jong, 1992] Vafaie, H. and De Jong, K. (1992). Genetic algorithms as a tool for feature selection in machine learning. In *Proceedings of the Fourth International Conference on Tools with Artificial Intelligence*, pages 200–204, Arlington, VA.

[Valiant, 1984] Valiant, L. (1984). A theory of the learnable. *Communications of the ACM*, pages 1134–1142.

[Vapnik, 1998] Vapnik, V. (1998). *Statistical Learning Theory*. Wiley.

[Weston et al., 2003] Weston, J., Elisseeff, A., Schlkopf, B., and Tipping, M. (2003). Use of the zero-norm with linear models and kernel methods. *Journal of Machine Learning Research*, 3(7-8):1439–1461. Special Issue on Variable and Feature Selection.

[Weston et al., 2000] Weston, J., Mukherjee, S., Chapelle, O., Pontil, M., Poggio, T., and Vapnik, V. (2000). Feature selection for SVMs. In Leen, T. K., Dietterich, T. G., and Tresp, V., editors, *Advances in Neural Information Processing Systems*, volume 13. MIT Press, Cambridge, MA.

[Xing et al., 2001] Xing, E., Jordan, M., and Karp, R. (2001). 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. Morgan Kaufmann.

[Xu et al., 1989] Xu, L., Yan, P., and Chang, T. (1989). Best first strategy for feature selection. In *Proceedings of the Ninth International Conference on Pattern Recognition*, pages 706–708. IEEE Computer Society Press.

[Yu and Liu, 2002] Yu, L. and Liu, H. (2002). Feature selection for high-dimensional data: A fast correlation-based filter solution. In *Proceedings of the Twentieth International Conference on Machine Learning (ICML'03)*, pages 856–863.

[Yu and Liu, 2004] Yu, L. and Liu, H. (2004). Redundancy based feature selection for microarray data. In *Proceedings of the CM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD 2004)*, Seattle, Washington.

[Zaki, 1999] Zaki, M. (1999). Parallel and distributed association mining: A survey. *IEEE Concurrency*, 7(4):14–25.

[Zaki and Ho, 2000] Zaki, M. and Ho, C.-T., editors (2000). *Large-Scale Parallel Data Mining, LNAI State-of-the-Art Survey*, volume 1759. Springer-Verlag, Berlin.

Appendix A

Experimental Results for C4.5

Experimental results for all feature selection algorithms with C4.5 as induction algorithm. The symbol beside a particular error rate indicates the t-test significance level of the accuracy difference between this algorithm and the best result of the three FortalFS settings. Filled circles($\bullet$) indicate that the difference falls within the 0.01 significance level, empty circles($\circ$) within 0.05 and empty diamonds($\diamond$) within 0.1. In addition, a bar over the symbol($\overline{\bullet}$, $\overline{\circ}$ or $\overline{\diamond}$) indicates that FortalFS performed better in that case and a bar under the symbol($\underline{\bullet}$, $\underline{\circ}$ or $\underline{\diamond}$) that FortalFS did worse.

	Error Rate (%)	FS Time (sec)	# Fea. Selected
C4.5	17.2609 $\bar{\bullet}$	-	-
FortalFS $(10 \cdot N)$	12.2560	68	10
FortalFS $(8 \cdot N)$	12.2560	55	10
FortalFS $(6 \cdot N)$	12.2560	46	10
Best-First	13.8647 $\bar{\diamond}$	1	1
Genetic	13.8647 $\bar{\diamond}$	1	2
LVF	12.6280	13	11
Relief	17.1304 $\bar{\bullet}$	3	10
Focus	16.7150 $\bar{\bullet}$	2	13
ForWrapper	13.8647 $\bar{\diamond}$	4	1
BackWrapper	12.6280	36	14
RWrapper $(10 \cdot N)$	15.7440 $\bar{\bullet}$	41	9
RWrapper (N^2)	15.4348 $\bar{\bullet}$	63	8

Table A.1: Experimental results for the Credit Dataset (15 features and 690 instances)

	Error Rate (%)	FS Time (sec)	# Fea. Selected
C4.5	26.6188	-	-
FortalFS $(10 \cdot N)$	30.2395	4	2
FortalFS $(8 \cdot N)$	30.2395	4	2
FortalFS $(6 \cdot N)$	30.2395	3	1
Best-First	34.2529 $\circ$	1	3
Genetic	34.2529 $\circ$	1	3
LVF	30.2395	1	3
Relief	26.6188	1	14
Focus	30.2395	1	2
ForWrapper	30.2395	1	1
BackWrapper	20.6609 $\bullet$	3	14
RWrapper $(10 \cdot N)$	30.9291	4	4
RWrapper (N^2)	30.2395	4	3

Table A.2: Experimental results for the Labor Dataset (16 features and 57 instances)

	Error Rate (%)	FS Time (sec)	# Fea. Selected
C4.5	03.1395 $\diamond$	-	-
FortalFS $(10 \cdot N)$	02.1002	12	5
FortalFS $(8 \cdot N)$	02.1002	11	6
FortalFS $(6 \cdot N)$	02.5122	11	8
Best-First	02.1002	1	1
Genetic	02.1002	1	1
LVF	03.1395 $\diamond$	2	6
Relief	03.1395 $\diamond$	1	16
Focus	03.1395 $\diamond$	1	11
ForWrapper	02.1002	2	1
BackWrapper	02.1002	17	12
RWrapper $(10 \cdot N)$	02.1002	12	8
RWrapper (N^2)	02.5122	19	7

Table A.3: Experimental results for the Vote Dataset (16 features and 435 instances)

	Error Rate (%)	FS Time (sec)	# Fea. Selected
C4.5	57.8235 $\underline{\diamond}$	-	-
FortalFS $(10 \cdot N)$	60.1176	82	14
FortalFS $(8 \cdot N)$	60.1176	71	14
FortalFS $(6 \cdot N)$	60.1176	54	14
Best-First	62.7059 $\overline{\diamond}$	1	8
Genetic	61.9902	1	11
LVF	57.8235 $\underline{\diamond}$	10	17
Relief	59.2157	1	14
Focus	57.8235 $\underline{\diamond}$	1	17
ForWrapper	61.1176	21	7
BackWrapper	61.6176	30	16
RWrapper $(10 \cdot N)$	59.0588	29	10
RWrapper (N^2)	56.4510 $\underline{\circ}$	52	7

Table A.4: Experimental results for the Primary Tumor Dataset (17 features and 339 instances)

	Error Rate (%)	FS Time (sec)	# Fea. Selected
C4.5	35.8679 $\bullet$	-	-
FortalFS $(10 \cdot N)$	22.4204	8	6
FortalFS $(8 \cdot N)$	23.1982	8	4
FortalFS $(6 \cdot N)$	27.2312	6	6
Best-First	33.9790 $\bullet$	1	10
Genetic	33.9790 $\bullet$	1	10
LVF	21.5946	1	6
Relief	35.8679 $\bullet$	1	16
Focus	22.1351	1	7
ForWrapper	32.2162 $\bullet$	4	3
BackWrapper	30.5015 $\bullet$	9	16
RWrapper $(10 \cdot N)$	26.1201 $\circ$	9	11
RWrapper (N^2)	23.4685	13	7

Table A.5: Experimental results for the Lymph Dataset (18 features 148 instances)

	Error Rate (%)	FS Time (sec)	# Fea. Selected
C4.5	00.0000	-	-
FortalFS $(10 \cdot N)$	00.0000	305	7
FortalFS $(8 \cdot N)$	00.0000	246	7
FortalFS $(6 \cdot N)$	00.0000	189	8
Best-First	01.4856 $\bullet$	1	1
Genetic	00.0464 $\circ$	2	4
LVF	00.0000	17	5
Relief	00.0000	471	21
Focus	00.0000	24	7
ForWrapper	00.0000	191	5
BackWrapper	00.0000	69	22
RWrapper $(10 \cdot N)$	00.0000	341	10
RWrapper (N^2)	00.0000	792	9

Table A.6: Experimental results for the Mushroom Dataset (22 features and 8124 instances)

	Error Rate (%)	FS Time (sec)	# Fea. Selected
C4.5	15.1684	-	-
FortalFS $(10 \cdot N)$	14.7444	40	6
FortalFS $(8 \cdot N)$	14.7444	31	6
FortalFS $(6 \cdot N)$	14.7444	27	9
Best-First	21.4091 $\bullet$	1	1
Genetic	21.4091 $\bullet$	1	2
LVF	15.1684	7	8
Relief	15.1684	2	18
Focus	15.1684	3	14
ForWrapper	19.7953 $\bullet$	6	3
BackWrapper	20.8235 $\bullet$	32	21
RWrapper $(10 \cdot N)$	19.3713 $\bullet$	38	6
RWrapper (N^2)	19.9040 $\bullet$	94	7

Table A.7: Experimental results for the Colic Dataset (23 features and 368 instances)

	Error Rate (%)	FS Time (sec)	# Fea. Selected
C4.5	53.1158 ◇	-	-
FortalFS $(10 \cdot N)$	49.3827	37	8
FortalFS $(8 \cdot N)$	48.8002	30	11
FortalFS $(6 \cdot N)$	48.9476	23	12
Best-First	55.3793 ○	1	5
Genetic	55.3793 ○	1	5
LVF	54.0271 ◇	5	10
Relief	49.9412	1	21
Focus	51.6405 ◇	1	9
ForWrapper	45.8608	45	6
BackWrapper	50.0419	233	13
RWrapper $(10 \cdot N)$	49.2708	37	11
RWrapper (N^2)	46.1130	99	9

Table A.8: Experimental results for the Autos Dataset (25 features and 205 instances)

	Error Rate (%)	FS Time (sec)	# Fea. Selected
C4.5	16.0739 $\bullet$	-	-
FortalFS $(10 \cdot N)$	10.0050	91	8
FortalFS $(8 \cdot N)$	10.3980	74	10
FortalFS $(6 \cdot N)$	09.1166	51	8
Best-First	11.4626 $\circ$	1	9
Genetic	15.2828 $\bullet$	1	8
LVF	09.3315	2	8
Relief	12.2297 $\circ$	2	33
Focus	13.8878 $\bullet$	2	5
ForWrapper	14.6625 $\bullet$	48	4
BackWrapper	16.3620 $\bullet$	476	29
RWrapper $(10 \cdot N)$	15.9399 $\bullet$	136	16
RWrapper (N^2)	13.4687 $\circ$	501	17

Table A.9: Experimental results for the Ionosphere Dataset (34 features and 351 instances)

	Error Rate (%)	FS Time (sec)	# Fea. Selected
C4.5	15.1008 $\bullet$	-	-
FortalFS $(10 \cdot N)$	11.1226	318	15
FortalFS $(8 \cdot N)$	12.8730	201	18
FortalFS $(6 \cdot N)$	12.1566	160	19
Best-First	14.0654 $\bullet$	1	18
Genetic	14.3418 $\bullet$	1	18
LVF	28.7612 $\bullet$	25	12
Relief	14.5760 $\bullet$	13	34
Focus	45.1846 $\bullet$	13	10
ForWrapper	14.1852 $\bullet$	319	12
BackWrapper	10.4411	397	31
RWrapper $(10 \cdot N)$	09.0935 $\circ$	188	21
RWrapper (N^2)	10.3635	669	23

Table A.10: Experimental results for the Soybean Dataset (35 features and 683 instances)

	Error Rate (%)	FS Time (sec)	# Fea. Selected
C4.5	07.4754	-	-
FortalFS $(10 \cdot N)$	08.6458	831	16
FortalFS $(8 \cdot N)$	08.0491	718	18
FortalFS $(6 \cdot N)$	07.8610	549	19
Best-First	08.5893 $\circ$	2	6
Genetic	08.5893 $\circ$	5	6
LVF	18.1850 $\bullet$	9	14
Relief	07.2832	187	58
Focus	44.6113 $\bullet$	534	9
ForWrapper	08.1755	2383	12
BackWrapper	07.3824	5626	53
RWrapper $(10 \cdot N)$	07.4754	2720	31
RWrapper (N^2)	06.7126 $\underline{\bullet}$	16553	30

Table A.11: Experimental results for the Splice Datasets (60 features and 3190 instances)

	Error Rate (%)	FS Time (sec)	# Fea. Selected
C4.5	40.9921 $\bullet$	-	-
FortalFS $(10 \cdot N)$	35.6532	328	23
FortalFS $(8 \cdot N)$	35.6990	279	26
FortalFS $(6 \cdot N)$	32.3993	233	22
Best-First	37.2161 $\bullet$	1	12
Genetic	37.2161 $\bullet$	2	12
LVF	38.8156 $\bullet$	117	26
Relief	33.5684	1	30
Focus	24.3864 $\circ$	33	35
ForWrapper	33.4646	37	4
BackWrapper	38.5440 $\bullet$	270	59
RWrapper $(10 \cdot N)$	43.9499 $\bullet$	261	20
RWrapper (N^2)	45.7509 $\bullet$	1390	30

Table A.12: Experimental results for the Sonar Dataset (60 features and 208 instances)

	Error Rate (%)	FS Time (sec)	# Fea. Selected
C4.5	29.5997 ○	-	-
FortalFS $(10 \cdot N)$	26.3115	139	21
FortalFS $(8 \cdot N)$	29.9276	123	27
FortalFS $(6 \cdot N)$	29.9395	96	27
Best-First	36.9779 ●	1	6
Genetic	36.1084 ●	2	8
LVF	26.9981	25	23
Relief	29.8195 ○	2	29
Focus	29.8836 ○	23	16
ForWrapper	34.1934 ●	17	4
BackWrapper	29.8967 ○	529	64
RWrapper $(10 \cdot N)$	26.9981	164	33
RWrapper (N^2)	26.9981	1060	31

Table A.13: Experimental results for the Audiology Dataset (69 features and 226 instances) with C4.5

Appendix B

Experimental Results for Naive Bayes

Experimental results for all feature selection algorithms with Naive Bayes as induction algorithm. The symbol beside a particular error rate indicates the t-test significance level of the accuracy difference between this algorithm and the best result of the three FortalFS settings. Filled circles($\bullet$) indicate that the difference falls within the 0.01 significance level, empty circles($\circ$) within 0.05 and empty diamonds($\diamond$) within 0.1. In addition, a bar over the symbol($\overline{\bullet}$, $\overline{\circ}$ or $\overline{\diamond}$) indicates that FortalFS performed better in that case and a bar under the symbol($\underline{\bullet}$, $\underline{\circ}$ or $\underline{\diamond}$) that FortalFS did worse.

	Error Rate (%)	FS Time (sec)	# Fea. Selected
NaiveBayes	21.5411 $\bullet$	-	-
FortalFS $(10 \cdot N)$	12.1063	20	8
FortalFS $(8 \cdot N)$	12.8551	18	10
FortalFS $(6 \cdot N)$	14.1739	17	10
Best-First	13.8647 $\circ$	1	1
Genetic	14.1304 $\bullet$	1	2
LVF	22.0290 $\bullet$	12	13
Relief	17.0338 $\bullet$	3	10
Focus	22.3333 $\bullet$	1	13
ForWrapper	12.3865	4	4
BackWrapper	11.7826	13	10
RWrapper $(10 \cdot N)$	16.8261 $\bullet$	6	7
RWrapper (N^2)	12.5459	9	7

Table B.1: Experimental results for the Credit Dataset (15 features and 690 instances)

	Error Rate (%)	FS Time (sec)	# Fea. Selected
NaiveBayes	23.0268	-	-
FortalFS $(10 \cdot N)$	23.7165	3	4
FortalFS $(8 \cdot N)$	21.0824	3	3
FortalFS $(6 \cdot N)$	23.7165	2	4
Best-First	29.8276 $\circ\bullet$	1	3
Genetic	29.8276 $\circ\bullet$	1	3
LVF	30.2395 $\circ\bullet$	1	2
Relief	15.2586 $\bullet$	1	14
Focus	30.2395 $\circ\bullet$	1	2
ForWrapper	28.7165 $\circ\bullet$	1	6
BackWrapper	20.3927	3	14
RWrapper $(10 \cdot N)$	28.7165 $\circ\bullet$	1	6
RWrapper (N^2)	28.7165 $\circ\bullet$	2	7

Table B.2: Experimental results for the Labor Dataset (16 features and 57 instances)

	Error Rate (%)	FS Time (sec)	# Fea. Selected
NaiveBayes	07.9604 $\bullet$	-	-
FortalFS $(10 \cdot N)$	02.1002	6	4
FortalFS $(8 \cdot N)$	02.6903	5	6
FortalFS $(6 \cdot N)$	03.1728	5	5
Best-First	02.1002	1	1
Genetic	02.1002	1	1
LVF	04.1524 $\circ$	3	7
Relief	07.9604 $\bullet$	1	16
Focus	06.6712 $\bullet$	1	11
ForWrapper	02.1002	1	1
BackWrapper	07.1908 $\bullet$	6	11
RWrapper $(10 \cdot N)$	04.6720 $\bullet$	3	7
RWrapper (N^2)	04.3477 $\bullet$	5	5

Table B.3: Experimental results for the Vote Dataset (16 features and 435 instances)

	Error Rate (%)	FS Time (sec)	# Fea. Selected
NaiveBayes	54.0784	-	-
FortalFS $(10 \cdot N)$	54.0882	19	15
FortalFS $(8 \cdot N)$	54.0882	15	15
FortalFS $(6 \cdot N)$	54.0882	15	15
Best-First	59.5588 $\bullet$	1	8
Genetic	54.6471	1	11
LVF	53.1667	9	14
Relief	52.5196 $\circ$	1	14
Focus	54.0784	1	17
ForWrapper	54.7549	3	5
BackWrapper	52.8824	6	15
RWrapper $(10 \cdot N)$	57.9902 $\bullet$	5	9
RWrapper (N^2)	54.5392	8	16

Table B.4: Experimental results for the Primary Tumor Dataset (17 features and 339 instances)

	Error Rate (%)	FS Time (sec)	# Fea. Selected
NaiveBayes	22.7057 ○	-	-
FortalFS $(10 \cdot N)$	17.3243	4	8
FortalFS $(8 \cdot N)$	24.6426	3	7
FortalFS $(6 \cdot N)$	23.5946	2	8
Best-First	25.5646 ●	1	10
Genetic	25.5646 ●	1	10
LVF	23.5315 ○	1	6
Relief	22.7057 ○	1	16
Focus	25.3093 ●	1	7
ForWrapper	25.8018 ●	2	4
BackWrapper	22.7057 ○	4	13
RWrapper $(10 \cdot N)$	25.1502 ○	2	11
RWrapper (N^2)	24.5165 ○	5	9

Table B.5: Experimental results for the Lymph Dataset (18 features 148 instances)

	Error Rate (%)	FS Time (sec)	# Fea. Selected
NaiveBayes	04.8664 •	-	-
FortalFS $(10 \cdot N)$	00.9327	101	4
FortalFS $(8 \cdot N)$	00.9327	78	4
FortalFS $(6 \cdot N)$	00.9274	60	7
Best-First	01.4856 •	1	1
Genetic	03.3120 •	2	4
LVF	11.4784 •	11	6
Relief	04.8664 •	524	21
Focus	01.5418 •	22	7
ForWrapper	01.5622 •	67	4
BackWrapper	00.9856	184	16
RWrapper $(10 \cdot N)$	00.9102	96	11
RWrapper (N^2)	00.9856	255	7

Table B.6: Experimental results for the Mushroom Dataset (22 features and 8124 instances)

	Error Rate (%)	FS Time (sec)	# Fea. Selected
NaiveBayes	23.3304 $\bullet$	-	-
FortalFS $(10 \cdot N)$	15.3437	8	7
FortalFS $(8 \cdot N)$	14.7395	8	7
FortalFS $(6 \cdot N)$	16.8488	7	7
Best-First	21.4091 $\bullet$	1	1
Genetic	22.3658 $\bullet$	1	2
LVF	23.1130 $\bullet$	4	8
Relief	20.2830 $\bullet$	2	18
Focus	21.6980 $\bullet$	2	14
ForWrapper	19.0002 $\bullet$	4	5
BackWrapper	19.0403 $\bullet$	16	15
RWrapper $(10 \cdot N)$	20.4788 $\bullet$	5	6
RWrapper (N^2)	18.9581 $\bullet$	11	8

Table B.7: Experimental results for the Colic Dataset (23 features and 368 instances)

	Error Rate (%)	FS Time (sec)	# Fea. Selected
NaiveBayes	57.6781 $\bar{\bullet}$	-	-
FortalFS $(10 \cdot N)$	44.1969	9	11
FortalFS $(8 \cdot N)$	40.4340	8	9
FortalFS $(6 \cdot N)$	45.0317	7	10
Best-First	61.5473 $\bar{\bullet}$	1	5
Genetic	61.5473 $\bar{\bullet}$	1	5
LVF	51.0411 $\bar{\bullet}$	3	9
Relief	57.6781 $\bar{\bullet}$	1	21
Focus	45.6441	1	9
ForWrapper	45.5023 $\bar{\diamond}$	9	8
BackWrapper	49.7414 $\bar{\bullet}$	20	21
RWrapper $(10 \cdot N)$	44.3146	8	13
RWrapper (N^2)	45.6312 $\bar{\diamond}$	17	14

Table B.8: Experimental results for the Autos Dataset (25 features and 205 instances)

	Error Rate (%)	FS Time (sec)	# Fea. Selected
NaiveBayes	17.9942 $\bullet$	-	-
FortalFS $(10 \cdot N)$	12.6530	14	13
FortalFS $(8 \cdot N)$	14.5466	11	4
FortalFS $(6 \cdot N)$	16.5800	10	13
Best-First	15.0900 $\circ$	1	9
Genetic	16.2964 $\bullet$	1	8
LVF	22.2749 $\bullet$	2	8
Relief	17.9942 $\bullet$	2	33
Focus	15.3009 $\bullet$	2	5
ForWrapper	14.4027 $\diamond$	8	4
BackWrapper	15.9847 $\bullet$	51	30
RWrapper $(10 \cdot N)$	21.0838 $\bullet$	98	13
RWrapper (N^2)	14.3297 $\diamond$	65	13

Table B.9: Experimental results for the Ionosphere Dataset (34 features and 351 instances)

	Error Rate (%)	FS Time (sec)	# Fea. Selected
NaiveBayes	09.4656	-	-
FortalFS $(10 \cdot N)$	09.7112	57	19
FortalFS $(8 \cdot N)$	10.4514	46	15
FortalFS $(6 \cdot N)$	11.1601	39	17
Best-First	09.6888	1	18
Genetic	09.5106	1	18
LVF	29.8414 $\bullet$	21	12
Relief	10.1144	12	34
Focus	39.4940 $\bullet$	10	10
ForWrapper	09.9840	51	12
BackWrapper	09.5212	92	31
RWrapper $(10 \cdot N)$	09.4107	31	23
RWrapper (N^2)	09.8083	106	27

Table B.10: Experimental results for the Soybean Dataset (35 features and 683 instances)

	Error Rate (%)	FS Time (sec)	# Fea. Selected
NaiveBayes	04.3584 •	-	-
FortalFS $(10 \cdot N)$	06.0878	125	21
FortalFS $(8 \cdot N)$	07.8903	108	15
FortalFS $(6 \cdot N)$	06.4305	84	24
Best-First	05.9822	2	6
Genetic	05.9822	4	6
LVF	20.7712 ◦	9	13
Relief	04.6092 •	182	58
Focus	39.7419 ◦	250	9
ForWrapper	05.7116 ○	309	16
BackWrapper	04.0031 •	206	59
RWrapper $(10 \cdot N)$	05.1985 •	178	32
RWrapper (N^2)	04.9843 •	1057	33

Table B.11: Experimental results for the Splice Datasets (60 features and 3190 instances)

	Error Rate (%)	FS Time (sec)	# Fea. Selected
NaiveBayes	37.1734	-	-
FortalFS $(10 \cdot N)$	35.3938	132	28
FortalFS $(8 \cdot N)$	35.5647	127	23
FortalFS $(6 \cdot N)$	36.8559	120	20
Best-First	34.0781	1	12
Genetic	34.0781	2	12
LVF	40.9463 ◇	102	25
Relief	36.5720	2	30
Focus	26.6758 ○	31	35
ForWrapper	33.0678	7	3
BackWrapper	38.5562	384	38
RWrapper $(10 \cdot N)$	34.8138	32	24
RWrapper (N^2)	37.5031	225	25

Table B.12: Experimental results for the Sonar Dataset (60 features and 208 instances)

	Error Rate (%)	FS Time (sec)	# Fea. Selected
NaiveBayes	40.7163 •	-	-
FortalFS $(10 \cdot N)$	36.2165	93	28
FortalFS $(8 \cdot N)$	31.2960	87	25
FortalFS $(6 \cdot N)$	31.5170	80	24
Best-First	35.0570 •	1	6
Genetic	33.5389 •	2	8
LVF	35.1580 •	58	24
Relief	30.6783	4	29
Focus	34.5474 •	66	16
ForWrapper	32.0907	107	12
BackWrapper	40.5393 •	457	65
RWrapper $(10 \cdot N)$	39.1079 •	143	38
RWrapper (N^2)	30.3885	324	30

Table B.13: Experimental results for the Audiology Dataset (69 features and 226 instances)

Appendix C

Experimental Results for k-Nearest Neighbour

Experimental results for all feature selection algorithms with k-Nearest Neighbour as induction algorithm. The symbol beside a particular error rate indicates the t-test significance level of the accuracy difference between this algorithm and the best result of the three FortalFS settings. Filled circles($\bullet$) indicate that the difference falls within the 0.01 significance level, empty circles($\circ$) within 0.05 and empty diamonds($\diamond$) within 0.1. In addition, a bar over the symbol($\overline{\bullet}$, $\overline{\circ}$ or $\overline{\diamond}$) indicates that FortalFS performed better in that case and a bar under the symbol($\underline{\bullet}$, $\underline{\circ}$ or $\underline{\diamond}$) that FortalFS did worse.

	Error Rate (%)	FS Time (sec)	# Fea. Selected
kNN	14.5169 ◇	-	-
FortalFS $(10 \cdot N)$	13.6087	181	9
FortalFS $(8 \cdot N)$	13.8068	144	10
FortalFS $(6 \cdot N)$	13.7826	110	9
Best-First	13.8647	1	1
Genetic	13.8647	1	2
LVF	14.5121 ◇	11	12
Relief	18.9227 ●	3	10
Focus	13.8792	1	13
ForWrapper	13.8647	23	1
BackWrapper	13.6618	153	13
RWrapper $(10 \cdot N)$	13.9034	82	7
RWrapper (N^2)	13.6329	128	5

Table C.1: Experimental results for the Credit Dataset (15 features and 690 instances)

	Error Rate (%)	FS Time (sec)	# Fea. Selected
kNN	16.4943 ○	-	-
FortalFS $(10 \cdot N)$	09.5690	3	6
FortalFS $(8 \cdot N)$	13.8697	2	5
FortalFS $(6 \cdot N)$	15.2586	2	3
Best-First	32.0402 ●	1	3
Genetic	32.0402 ●	1	3
LVF	41.1973 ●	1	3
Relief	13.1705	1	14
Focus	35.5077 ●	1	2
ForWrapper	28.2950 ●	2	4
BackWrapper	24.1284 ●	1	15
RWrapper $(10 \cdot N)$	22.3276 ○	2	5
RWrapper (N^2)	11.7816	2	6

Table C.2: Experimental results for the Labor Dataset (16 features and 57 instances)

	Error Rate (%)	FS Time (sec)	# Fea. Selected
kNN	05.7514	-	-
FortalFS $(10 \cdot N)$	05.9892	41	8
FortalFS $(8 \cdot N)$	05.6264	36	7
FortalFS $(6 \cdot N)$	06.0211	27	6
Best-First	02.1002 •	1	1
Genetic	02.1002 •	1	1
LVF	04.9870	3	6
Relief	05.3021	1	16
Focus	05.7501	1	11
ForWrapper	02.1002 •	8	1
BackWrapper	06.4279	42	15
RWrapper $(10 \cdot N)$	05.0961	36	7
RWrapper (N^2)	04.0074	59	7

Table C.3: Experimental results for the Vote Dataset (16 features and 435 instances)

	Error Rate (%)	FS Time (sec)	# Fea. Selected
kNN	57.6471 ○	-	-
FortalFS $(10 \cdot N)$	53.6961	51	15
FortalFS $(8 \cdot N)$	53.6961	41	15
FortalFS $(6 \cdot N)$	53.6961	33	15
Best-First	56.5686 ○	1	8
Genetic	57.8431 ●	1	11
LVF	57.6471 ○	9	17
Relief	63.3627 ●	1	14
Focus	57.6471 ○	1	17
ForWrapper	59.0000 ●	34	8
BackWrapper	52.9020	35	15
RWrapper $(10 \cdot N)$	60.6765 ●	24	12
RWrapper (N^2)	59.1471 ●	45	11

Table C.4: Experimental results for the Primary Tumor Dataset (17 features and 339 instances)

	Error Rate (%)	FS Time (sec)	# Fea. Selected
kNN	25.6426 $\bullet$	-	-
FortalFS $(10 \cdot N)$	15.6096	7	9
FortalFS $(8 \cdot N)$	18.0541	6	6
FortalFS $(6 \cdot N)$	21.3724	5	6
Best-First	27.4535 $\bullet$	1	10
Genetic	27.4535 $\bullet$	1	10
LVF	35.2012 $\bullet$	1	6
Relief	30.2312 $\bullet$	1	16
Focus	28.6126 $\bullet$	1	7
ForWrapper	30.7718 $\bullet$	5	4
BackWrapper	30.5646 $\bullet$	12	16
RWrapper $(10 \cdot N)$	30.1682 $\bullet$	9	8
RWrapper (N^2)	24.7868 $\circ$	11	12

Table C.5: Experimental results for the Lymph Dataset (18 features 148 instances)

	Error Rate (%)	FS Time (sec)	# Fea. Selected
kNN	00.0464	-	-
FortalFS $(10 \cdot N)$	00.0464	14476	10
FortalFS $(8 \cdot N)$	00.1879	11669	5
FortalFS $(6 \cdot N)$	00.1879	8732	4
Best-First	01.4856 $\bar{\bullet}$	1	1
Genetic	00.4122 $\bar{\bullet}$	2	4
LVF	00.5114 $\bar{\bullet}$	5	5
Relief	00.0000 $\underline{\circ}$	508	21
Focus	00.3954 $\bar{\bullet}$	22	7
ForWrapper	00.1879 $\bar{\bullet}$	30115	4
BackWrapper	00.0316	80252	18
RWrapper $(10 \cdot N)$	00.0098 $\underline{\diamond}$	23170	15
RWrapper (N^2)	00.0098 $\underline{\diamond}$	139643	15

Table C.6: Experimental results for the Mushroom Dataset (22 features and 8124 instances)

	Error Rate (%)	FS Time (sec)	# Fea. Selected
kNN	17.8848	-	-
FortalFS $(10 \cdot N)$	16.4933	48	4
FortalFS $(8 \cdot N)$	16.4933	41	6
FortalFS $(6 \cdot N)$	16.9810	33	7
Best-First	21.4091 $\overline{\circ}$	1	1
Genetic	23.3382 $\overline{\bullet}$	1	2
LVF	17.2170	3	8
Relief	19.9863 $\overline{\circ}$	1	18
Focus	14.3047 $\underline{\diamond}$	2	14
ForWrapper	21.3004 $\overline{\circ}$	26	3
BackWrapper	16.5942	262	13
RWrapper $(10 \cdot N)$	18.2530	52	11
RWrapper (N^2)	19.1089 $\overline{\diamond}$	121	9

Table C.7: Experimental results for the Colic Dataset (23 features and 368 instances)

	Error Rate (%)	FS Time (sec)	# Fea. Selected
kNN	50.7236 $\bullet$	-	-
FortalFS $(10 \cdot N)$	44.1559	25	11
FortalFS $(8 \cdot N)$	44.1559	21	11
FortalFS $(6 \cdot N)$	47.2314	17	9
Best-First	53.8104 $\bullet$	1	5
Genetic	53.8104 $\bullet$	1	5
LVF	50.2529 $\bullet$	5	10
Relief	43.8907	1	21
Focus	51.9523 $\bullet$	1	9
ForWrapper	57.6836 $\bullet$	30	8
BackWrapper	54.8621 $\bullet$	44	23
RWrapper $(10 \cdot N)$	62.4700 $\bullet$	21	13
RWrapper (N^2)	47.4481 $\bullet$	55	10

Table C.8: Experimental results for the Autos Dataset (25 features and 205 instances)

	Error Rate (%)	FS Time (sec)	# Fea. Selected
kNN	22.1951 $\bullet$	-	-
FortalFS $(10 \cdot N)$	14.7010	53	9
FortalFS $(8 \cdot N)$	16.4244	47	7
FortalFS $(6 \cdot N)$	13.4924	32	9
Best-First	18.1840 $\bullet$	1	9
Genetic	22.9230 $\bullet$	1	8
LVF	18.3817 $\bullet$	1	7
Relief	16.0555 $\bullet$	2	33
Focus	16.8541 $\bullet$	2	5
ForWrapper	15.3536 $\diamond$	24	2
BackWrapper	19.7074 $\bullet$	296	30
RWrapper $(10 \cdot N)$	21.0838 $\bullet$	98	13
RWrapper (N^2)	21.3576 $\bullet$	324	16

Table C.9: Experimental results for the Ionosphere Dataset (34 features and 351 instances)

	Error Rate (%)	FS Time (sec)	# Fea. Selected
kNN	22.7776 $\underline{\bullet}$	-	-
FortalFS $(10 \cdot N)$	26.4937	341	18
FortalFS $(8 \cdot N)$	29.0986	260	16
FortalFS $(6 \cdot N)$	26.6089	200	15
Best-First	28.4480 $\overline{\circ}$	1	18
Genetic	29.0727 $\overline{\circ}$	1	18
LVF	33.3934 $\overline{\bullet}$	20	13
Relief	09.4525 $\underline{\bullet}$	9	34
Focus	46.3880 $\overline{\bullet}$	9	10
ForWrapper	24.3609 $\underline{\diamond}$	996	16
BackWrapper	20.5906 $\underline{\bullet}$	1060	32
RWrapper $(10 \cdot N)$	24.8234 $\underline{\circ}$	343	22
RWrapper (N^2)	25.1812	1242	20

Table C.10: Experimental results for the Soybean Dataset (35 features and 683 instances)

	Error Rate (%)	FS Time (sec)	# Fea. Selected
kNN	16.9394 ◇	-	-
FortalFS $(10 \cdot N)$	15.5852	12489	18
FortalFS $(8 \cdot N)$	15.6855	10075	18
FortalFS $(6 \cdot N)$	16.7168	7770	20
Best-First	14.0585 ○	2	6
Genetic	14.0585 ○	4	6
LVF	35.6405 ●	8	13
Relief	28.9530 ●	175	58
Focus	43.7544 ●	235	9
ForWrapper	15.4848	6782	5
BackWrapper	16.5183 ◇	45911	58
RWrapper $(10 \cdot N)$	17.0031 ◇	22297	35
RWrapper (N^2)	15.4848	132351	32

Table C.11: Experimental results for the Splice Datasets (60 features and 3190 instances)

	Error Rate (%)	FS Time (sec)	# Fea. Selected
kNN	30.4853 ◇	-	-
FortalFS $(10 \cdot N)$	27.5611	197	19
FortalFS $(8 \cdot N)$	25.1129	186	29
FortalFS $(6 \cdot N)$	29.6123	157	21
Best-First	28.0250	1	12
Genetic	28.0250	1	12
LVF	34.5085 ○	92	24
Relief	29.2308	1	30
Focus	35.0519 ●	29	35
ForWrapper	25.9615	28	3
BackWrapper	30.4396 ◇	372	57
RWrapper $(10 \cdot N)$	26.9597	108	26
RWrapper (N^2)	32.8755 ●	658	31

Table C.12: Experimental results for the Sonar Dataset (60 features and 208 instances)

	Error Rate (%)	FS Time (sec)	# Fea. Selected
kNN	53.6968 $\bar{\bullet}$	-	-
FortalFS $(10 \cdot N)$	44.6495	150	28
FortalFS $(8 \cdot N)$	46.3994	135	35
FortalFS $(6 \cdot N)$	46.2817	103	19
Best-First	43.4699	1	6
Genetic	44.4084	2	8
LVF	46.9731	30	24
Relief	30.8292 $\underline{\bullet}$	2	29
Focus	46.9530	19	16
ForWrapper	38.5662 $\underline{\bullet}$	19	2
BackWrapper	52.4245 $\bar{\bullet}$	408	65
RWrapper $(10 \cdot N)$	53.4557 $\bar{\bullet}$	138	34
RWrapper (N^2)	46.5811	967	30

Table C.13: Experimental results for the Audiology Dataset (69 features and 226 instances)

Appendix D

Experimental Results for FortalFS-W, FortalFS-R and FortalFS-C

Experimental results for the FortalFS variants FortalFS-W, FortalFS-R and FortalFS-C with C4.5 as induction algorithm. The symbol beside a particular error rate indicates the t-test significance level of the accuracy difference between this algorithm and FortalFS. Filled circles($\bullet$) indicate that the difference falls within the 0.01 significance level, empty circles($\circ$) within 0.05 and empty diamonds($\diamond$) within 0.1. In addition, a bar over the symbol($\overline{\bullet}$, $\overline{\circ}$ or $\overline{\diamond}$) indicates that FortalFS performed better in that case and a bar under the symbol($\underline{\bullet}$, $\underline{\circ}$ or $\underline{\diamond}$) that FortalFS did worse.

	Error Rate (%)	FS Time (sec)	# Fea. Selected
FortalFS $(10 \cdot N)$	12.2560	68	10
FortalFS-W $(10 \cdot N)$	13.5797	35	6
FortalFS-R $(10 \cdot N)$	13.8647	46	7
FortalFS-C $(10 \cdot N)$	16.2126 $\bar{\bullet}$	70	10

Table D.1: Experimental results for the Credit Dataset (15 features and 690 instances)

	Error Rate (%)	FS Time (sec)	# Fea. Selected
FortalFS $(10 \cdot N)$	30.2395	4	2
FortalFS-W $(10 \cdot N)$	26.6188	3	12
FortalFS-R $(10 \cdot N)$	30.9291	4	3
FortalFS-C $(10 \cdot N)$	30.2395	6	2

Table D.2: Experimental results for the Labor Dataset (16 features and 57 instances)

	Error Rate (%)	FS Time (sec)	# Fea. Selected
FortalFS $(10 \cdot N)$	02.1002	12	5
FortalFS-W $(10 \cdot N)$	03.1395 $\bar{\diamond}$	12	15
FortalFS-R $(10 \cdot N)$	03.1395 $\bar{\diamond}$	14	13
FortalFS-C $(10 \cdot N)$	02.5122	14	11

Table D.3: Experimental results for the Vote Dataset (16 features and 435 instances)

	Error Rate (%)	FS Time (sec)	# Fea. Selected
FortalFS $(10 \cdot N)$	60.1176	82	14
FortalFS-W $(10 \cdot N)$	60.3529	53	12
FortalFS-R $(10 \cdot N)$	57.2451 $\underline{\diamond}$	55	11
FortalFS-C $(10 \cdot N)$	61.5784	73	13

Table D.4: Experimental results for the Primary Tumor Dataset (17 features and 339 instances)

	Error Rate (%)	FS Time (sec)	# Fea. Selected
FortalFS $(10 \cdot N)$	22.4204	8	6
FortalFS-W $(10 \cdot N)$	24.7387 $\overline{\diamond}$	9	11
FortalFS-R $(10 \cdot N)$	29.4535 $\overline{\bullet}$	10	10
FortalFS-C $(10 \cdot N)$	22.1351	10	11

Table D.5: Experimental results for the Lymph Dataset (18 features 148 instances)

	Error Rate (%)	FS Time (sec)	# Fea. Selected
FortalFS $(10 \cdot N)$	00.0000	305	7
FortalFS-W $(10 \cdot N)$	00.0000	828	16
FortalFS-R $(10 \cdot N)$	00.0000	821	21
FortalFS-C $(10 \cdot N)$	00.0000	841	8

Table D.6: Experimental results for the Mushroom Dataset (22 features and 8124 instances)

	Error Rate (%)	FS Time (sec)	# Fea. Selected
FortalFS $(10 \cdot N)$	14.7444	40	6
FortalFS-W $(10 \cdot N)$	15.1684	36	13
FortalFS-R $(10 \cdot N)$	14.7444	49	11
FortalFS-C $(10 \cdot N)$	14.7444	48	10

Table D.7: Experimental results for the Colic Dataset (23 features and 368 instances)

	Error Rate (%)	FS Time (sec)	# Fea. Selected
FortalFS $(10 \cdot N)$	49.3827	37	8
FortalFS-W $(10 \cdot N)$	49.1418	51	16
FortalFS-R $(10 \cdot N)$	49.1418	67	20
FortalFS-C $(10 \cdot N)$	51.5639	48	11

Table D.8: Experimental results for the Autos Dataset (25 features and 205 instances)

	Error Rate (%)	FS Time (sec)	# Fea. Selected
FortalFS $(10 \cdot N)$	10.0050	91	8
FortalFS-W $(10 \cdot N)$	11.1471	236	25
FortalFS-R $(10 \cdot N)$	11.4675	277	30
FortalFS-C $(10 \cdot N)$	13.5572	200	9

Table D.9: Experimental results for the Ionosphere Dataset (34 features and 351 instances)

	Error Rate (%)	FS Time (sec)	# Fea. Selected
FortalFS $(10 \cdot N)$	11.1226	318	15
FortalFS-W $(10 \cdot N)$	14.1806 $\circ$	248	22
FortalFS-R $(10 \cdot N)$	15.1008 $\bullet$	341	34
FortalFS-C $(10 \cdot N)$	10.6931	219	20

Table D.10: Experimental results for the Soybean Dataset (35 features and 683 instances)

	Error Rate (%)	FS Time (sec)	# Fea. Selected
FortalFS $(10 \cdot N)$	08.6458	831	16
FortalFS-W $(10 \cdot N)$	07.2633 $\underline{\bullet}$	1564	37
FortalFS-R $(10 \cdot N)$	07.4754 $\underline{\bullet}$	1860	52
FortalFS-C $(10 \cdot N)$	09.1515 $\overline{\circ}$	1803	31

Table D.11: Experimental results for the Splice Datasets (60 features and 3190 instances)

	Error Rate (%)	FS Time (sec)	# Fea. Selected
FortalFS $(10 \cdot N)$	35.6532	328	23
FortalFS-W $(10 \cdot N)$	38.1013	288	33
FortalFS-R $(10 \cdot N)$	33.5562	339	40
FortalFS-C $(10 \cdot N)$	33.8614	372	31

Table D.12: Experimental results for the Sonar Dataset (60 features and 208 instances)

	Error Rate (%)	FS Time (sec)	# Fea. Selected
FortalFS $(10 \cdot N)$	26.3115	139	21
FortalFS-W $(10 \cdot N)$	28.6232 $\overline{\diamond}$	125	26
FortalFS-R $(10 \cdot N)$	28.0494	169	31
FortalFS-C $(10 \cdot N)$	26.3115	150	18

Table D.13: Experimental results for the Audiology Dataset (69 features and 226 instances)

Appendix E

ParallelFortalFS Evaluation

# Processors	1	2	4	8	12	16
Seq. Exec. Time(sec)	1.682					
Parallel Exec. Time(sec)	450.207	226.726	118.280	59.255	39.096	30.786
Total Exec. Time(sec)	451.889	228.408	119.962	60.937	40.778	32.468
Speedup	1	1.978	3.766	7.415	11.081	13.917
Optimum Speedup	1	1.993	3.956	7.797	11.528	15.154
Parallel Efficiency	1	0.992	0.952	0.951	0.961	0.918
Total Efficiency	1	0.989	0.942	0.927	0.924	0.870

Table E.1: ParallelFortalFS with the Mushroom Dataset.

# Processors	1	2	4	8	12	16
Seq. Exec. Time(sec)	1.422					
Parallel Exec. Time(sec)	348.181	175.037	90.931	43.843	30.865	24.055
Total Exec. Time(sec)	349.603	176.459	92.353	45.265	32.287	25.477
Speedup	1	1.981	3.786	7.723	10.828	13.722
Optimum Speedup	1	1.992	3.952	7.779	11.486	15.08
Parallel Efficiency	1	0.994	0.958	0.993	0.943	0.910
Total Efficiency	1	0.991	0.947	0.965	0.902	0.858

Table E.2: ParallelFortalFS with the Ionosphere Dataset.

# Processors	1	2	4	8	12	16
Seq. Exec. Time(sec)	8.112					
Parallel Exec. Time(sec)	2240.034	1123.344	564.462	283.227	191.786	144.087
Total Exec. Time(sec)	2248.146	1131.456	572.574	291.339	199.898	152.199
Speedup	1	1.987	3.926	7.717	11.246	14.771
Optimum Speedup	1	1.993	3.957	7.803	11.542	15.178
Parallel Efficiency	1	0.997	0.992	0.989	0.974	0.973
Total Efficiency	1	0.994	0.982	0.965	0.937	0.923

Table E.3: ParallelFortalFS with the Splice Dataset.

# Processors	1	2	4	8	12	16
Seq. Exec. Time(sec)	7.651					
Parallel Exec. Time(sec)	507.468	254.045	129.096	63.899	44.033	35.020
Total Exec. Time(sec)	515.119	261.696	136.747	71.55	51.684	42.671
Speedup	1	1.968	3.767	7.199	9.967	12.072
Optimum Speedup	1	1.971	3.829	7.247	10.315	13.085
Parallel Efficiency	1	0.998	0.984	0.993	0.966	0.923
Total Efficiency	1	0.984	0.942	0.900	0.831	0.755

Table E.4: ParallelFortalFS with the Sonar Dataset.

# Processors	1	2	4	8	12	16
Seq. Exec. Time(sec)	43.493					
Parallel Exec. Time(sec)	335.802	171.046	86.444	43.823	29.622	23.474
Total Exec. Time(sec)	379.295	214.539	129.937	87.316	73.115	66.967
Speedup	1	1.768	2.919	4.344	5.188	5.664
Optimum Speedup	1	1.794	2.976	4.438	5.307	5.882
Parallel Efficiency	1	0.986	0.981	0.979	0.978	0.963
Total Efficiency	1	0.884	0.730	0.543	0.432	0.354

Table E.5: ParallelFortalFS with the Audiology Dataset.

# Processors	1	2	4	8	12	16
Seq. Exec. Time(min)	1.220					
Parallel Exec. Time(min)	18.593	9.350	4.708	2.358	1.924	1.730
Total Exec. Time(min)	19.813	10.571	5.929	3.578	3.145	2.950
Speedup	1	1.874	3.341	5.534	6.295	6.710
Optimum Speedup	1	1.889	3.370	5.582	7.153	8.310
Parallel Efficiency	1	0.994	0.991	0.991	0.879	0.807
Total Efficiency	1	0.935	0.835	0.691	0.524	0.419

Table E.6: ParallelFortalFS with the Clean1 Dataset.

# Processors	1	2	4	8	12	16
Seq. Exec. Time(min)	2.450					
Parallel Exec. Time(min)	37.262	18.707	9.373	5.044	3.476	3.203
Total Exec. Time(min)	39.713	21.157	11.823	7.494	5.927	5.654
Speedup	1	1.870	3.354	5.292	6.697	7.020
Optimum Speedup	1	1.881	3.376	5.587	7.140	8.302
Parallel Efficiency	1	0.994	0.994	0.948	0.936	0.845
Total Efficiency	1	0.935	0.837	0.661	0.557	0.438

Table E.7: ParallelFortalFS with the Arrythmia Dataset.

# Processors	1	2	4	8	12	16
Seq. Exec. Time(min)	4.833					
Parallel Exec. Time(min)	650.639	325.664	163.139	81.585	55.221	42.128
Total Exec. Time(min)	655.472	330.497	167.973	86.418	60.054	46.961
Speedup	1	1.984	3.903	7.581	10.918	13.955
Optimum Speedup	1	1.982	3.915	7.605	11.095	14.404
Parallel Efficiency	1	1	0.997	0.997	0.983	0.968
Total Efficiency	1	0.990	0.975	0.947	0.909	0.871

Table E.8: ParallelFortalFS with the Madelon Dataset.

OK, so you've got a Ph.D. Now, don't touch anything.

Unknown