LEARNING ACCURATE AND UNDERSTANDABLE RULES FROM SVM CLASSIFIERS

by

Fei Chen

B.Sc., Nanjing University of Aeronautics and Astronautics, 1999

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE
in the School
of
Computing Science

SIMON FRASER UNIVERSITY
July 2004

APPROVAL

Name: Fei Chen

Degree: Master of Science

Title of thesis: LEARNING ACCURATE AND UNDERSTANDABLE RULES FROM SVM CLASSIFIERS

Examining Committee: Dr. Tiko Kameda
Chair

Dr. Martin Ester, Senior Supervisor

Dr. Anoop Sarkar, Supervisor

Dr. Ramesh Krishnamurti, SFU Examiner

Date Approved: July 15, 2004

SIMON FRASER UNIVERSITY

Partial Copyright Licence

Bennett Library
Simon Fraser University
Burnaby, BC, Canada

Abstract

Despite of their impressive classification accuracy in many high dimensional applications, Support Vector Machine (SVM) classifiers are hard to understand because the definition of the separating hyperplanes typically involves a large percentage of all features. In this paper, we address the problem of understanding SVM classifiers, which has not yet been well-studied. We formulate the problem as learning models to approximate trained SVMs that are more understandable while preserve most of the SVM's classification quality. Our method learns a set of If-Then rules that are generally considered to be understandable and that allow an explicit control of their complexity to meet user-supplied requirements. The adoption of the unordered rule learning paradigm, along with exploiting the trained SVMs helps overcome the weakness of standard rule learners in high dimensional feature spaces. A pruning method is employed to maximize the accuracy of the resulting rule set for some user-specified complexity. Experiments demonstrate that the accuracy of the rule set is close to that achieved by SVMs and keeps stable even with substantial decreases of the rule complexity.

To my family

Acknowledgments

My greatest gratitude goes to my senior supervisor Dr. Martin Ester, who has been an excellent advisor and mentor. Martin has shared with me a great deal of his expertise in data mining and valuable experiences in writing, speaking and numerous other important aspects of conducting a career in research. His efficient working style and patience provides me with the ideal balance of freedom to pursue my own ideas and consistent guidance to steer me in the right direction.

I would also like to thank my supervisor Dr. Anoop Sarkar, who has provided insightful feedback and helpful suggestions throughout my thesis work. My gratitude goes to Dr. Ramesh Krishnamurti and Dr. Tiko Kameda for serving on my examining committee. Thank you for your precious time in reviewing my thesis and providing thoughtful comments and suggestions. I truly thank for Dr. Fiona S.L. Brinkman, without whom there may be no such research at all. I also wish to thank Dr. Ke Wang, from whom I have received instrumental advices on improving the quality of this thesis work.

I would like to express my sincere thanks to Jennifier L. Gardy , Yudong Liu , Rong She and Byron Gao who have offered me great help on this research. Thanks to my fellow students in the Database and Data Mining lab: Sourav Chakraborty, Hongying Cui, Richard Frank, Benjamin Fung, Byron Gao, Wen Jin, Rong She, Alex Wang, Ping Wang, Yabo Xu, Xiang Zhang and Sengqiang Zhou, for their extensive discussions on my research work and wonderful friendship.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation

Knowledge Discovery and Data Mining (KDD) is the process of discovering knowledge or interesting data patterns hidden in a large amount of data. With massive amounts of data being continuously collected in databases, there are great demands for analyzing such data and transforming them into useful knowledge [14]. Classification is a form of data analysis that can be used to built models describing important data classes. Many classification algorithms have been proposed by researchers and have been applied to numerous difficult, real-world problems [30] [29] [34] [15].

The application of classification algorithms is usually driven by two underlying goals: performance and discovery. In the first case, the goal is to use a classification algorithm to learn a model that can be used to perform some task of interest. For example, in a spam email filter system, a classifier is built to distinguish spam emails from other emails; in a webpage categorization system, a classifier is built to classify various web pages into different categories such as sports, news etc. Another reason to make use of classifications is for the purpose of gaining insight into a collection of data by learning a descriptive model that is humanly comprehensible and can lead to a better understanding of the problem domain. Of course, it is often the case that a classification method is applied in a given domain for both of these purposes: to construct a system that can perform a useful task, and to get a better understanding of the available data.

Given the goals of performance and discovery, two of the criteria that are most often used to evaluate learning systems are the *predictive accuracy* and the *understandability* of

the learned models. Predictive accuracy, which is usually the predominant criterion, refers to how well a given model accounts for examples that were not used in learning the model. Understandability refers to how easily we can inspect and understand a model constructed by the learning system. It is often the case that the classification methods which construct the model with the highest predictive accuracy are not the methods which construct the most understandable model. For example, neural networks provide good predictive accuracy in a wide variety of problem domains, but produce models that are notoriously difficult to understand[31] [33] [25] [20]. Support Vector Machines (SVMs), as another example, have achieved impressive classification accuracy in many challenging applications such as text classification [21], classification of proteins [32], and face detection in images [28]. Unfortunately, SVMs, like the neural networks, generate black box models in the sense that they do not have the ability to explain, in an understandable form for a domain expert, the reasoning behind the predictions.

There are, however, many applications where the understandability of the classification model is as important as its accuracy. In biological applications, for example, the scientists want to gain some insight into the underlying biological processes. In medical applications, as another example, the classification model must be checked by a physician before making crucial decisions. In spite of the good predictive accuracy of neural networks and SVMs, their inability to produce plausible explanations compromises their applicability. In order to deal with this limitation, it is desirable to be able to understand the models learned by these algorithms.

In the last decade, a proliferation of methods for understanding trained neural networks has been presented in the literature [31] [33] [25] [20]. These methods apply the strategy of translating a trained neural network into a more understandable model, such as a set of If-Then rules or a decision tree. Nevertheless, in the case of SVMs, few research works have been published. Most of the application domains where SVMs significantly outperform other classification methods involve high dimensional (>1000) feature spaces, which often causes severe problems for building accurate classifiers. Essentially the amount of training data to sustain an accurate classifier increases exponentially with the dimensionality of the feature space. Based on the statistical learning theory, SVMs implements the structural risk minimization principle with the purpose of obtaining a good generalization from limited training data [34]. In order to preserve this virtue of SVMs , it is necessary to develop an approach that is capable of translating SVMs into understandable yet accurate models

in the scenario of high dimensional feature spaces. Even though SVMs can be viewed as an extension of perceptron models (single level neural networks), existing methods of understanding neural networks can not effectively be applied to the problem of understanding SVMs, because they are not scalable to high dimensional spaces.

To overcome this limitation, this thesis investigates the following problem: in the scenario of high dimensional feature spaces, can we take an arbitrary, incomprehensible SVM classification model, and closely approximate it in a language that better facilitates understandability. We propose an approach for approximating the prediction behavior of trained SVMs by a set of If-Then rules. The approach has the following advantages.

- It produces understandable yet accurate descriptions of trained SVMs.
- It can scale to high dimensional feature spaces very well.
- It can apply to a broad class of SVMs.

1.2 Organization of the Thesis

The rest of this thesis is organized as follows.

Chapter 2 provides background material for the rest of the thesis. Chapter 3 surveys related work. In Chapter 4, we formulate the problem addressed by this thesis and present a two-phase approach to extract a set of If-Then rules from SVMs. Section 5 reports the results of an experimental evaluation and comparison with existing methods. Section 6 concludes the thesis and discusses directions for future research.

Chapter 2

Background

This chapter provides background information for the remainder of the thesis. The first section introduces main concepts and terminologies of classification. The next section gives a brief overview of decision tree learning methods and rule induction methods. This material is relevant since the Boost Unordered Rule Learner (BUR) algorithm, presented in chapter 4, learns a set of inference (If-Then) rules to approximate the trained SVMs. Moreover, the experimental evaluation of our proposed approach presented in chapter 5 involves experimental comparison with a standard decision tree learning method and a rule induction method. The last section in this chapter provides a brief introduction to SVMs , and discusses the difficulties in understanding the SVM classifiers.

2.1 Introduction to classification

Classifying examples into a discrete set of possible categories are referred to as *classification.* Classification is a two-step process. In the first step, a model (*hypothesis*) is learned describing a predetermined set of data classes. The model is constructed by analyzing data described by attributes. Each data is assumed to belong to a predefined class, as determined by its *class label.* In the context of classification, data analyzed to build the model collectively form the *training data set.* The individual data making up the training set are referred as *training examples* and are randomly selected from the example population. Since the class label of each training example is provided, this step is also known as *supervised learning* (i.e. the learning of the model is "supervised" in that it is told to which class each

training example belongs). It contrasts with *unsupervised learning*(also known as *clustering*), in which the class label of each training example is not known, and the number of set of classes to be learned may not be known in advanced.

Typically, the learned model is represented in the form of inference rules, decision trees, or a function or distribution that is learned.

In the second step, the model is used for classification. First the predictive accuracy of the model (or classifier) is estimated. There are several ways to estimate the accuracy. The most common method is often referred to as *holdout* approach which estimates the predictive accuracy of a model by measuring its accuracy on a set of examples that it is not allowed to access when constructing the model. Such a set is called a *test data set* or a *holdout data set.* These examples are randomly selected and are independent of the training data. The *accuracy* of a model on a given test set is the percentage of test set examples that are correctly classified by the model. For each test example, the known class label is compared with the learned model's class prediction for that example. The motivation of this approach is: even though the learning algorithm may be misled by random errors and coincidental regularities within the training data set, the validating data set is unlikely to exhibit the same random fluctuations. Therefore, the validation data set can be expected to provide a safety check against *overfitting* the spurious characteristics of the training data set (that is, the learned model may have incorporated some particular anomalies of the training data that are not present in the overall example population). Of course, it is important that the validation set be large enough to itself provide a statistically significant sample of the examples. To serve this purpose, a preferred method for accuracy estimation is to use *cross-validation.* In k-fold cross validation, the available data is partitioned into k separate sets of approximately equal size. The cross-validation procedure involves k iterations in which the learning method is given k-1 of the subsets to use as training data, and is tested on the set left out. Each iteration leaves out a different subset so that each is used as the test set exactly once. The *cross-validation accuracy* of the given algorithm is simply the average of the accuracy measurement from the individual folds.

If the accuracy of the model is considered acceptable, the model can be used to classify future examples for which the class label is not known. Such data are also referred to in the machine learning literature as "unknown" or "previously unseen" data.

Classification methods can be compared and evaluated according to the following criteria:

- *Classification Quality:* This refers to the ability of the model to correctly predict the class label of unseen data.
- *Speed:* This refers to the computation costs involved in generating and using the model.
- *Robustness:* This is the ability of the model to make correct predictions given noisy data and data with missing values.
- *Scalability:* This refers to the ability to construct the model efficiently given large amounts of data.
- *Interpretability:* This refers to the level of understanding and insight that is provided by the model.

In this thesis, we are particularly interested in the classification quality and interpretability issues.

The standard classification quality measures are *overall accuracy*, *recall* and *precision*. They are defined based on a confusion matrix as shown in table 2.1. We refer to the classes in a binary classification problem as "positive" and "negative" class respectively.

Table 2.1: Confusion matrix in classificaion

# Examples	Classified as positive class	Classified as negative class
Actual positive class	n_{00}	n_{01}
Actual negative class	n_{10}	n_{11}

$accuracy = \frac{n_{00}+n_{11}}{n_{00}+n_{01}+n_{10}+n_{11}}$; $recall = \frac{n_{00}}{n_{00}+n_{01}}$; $precision = \frac{n_{00}}{n_{00}+n_{10}}$

The Interpretability is often measured in the size of the classifiers.

2.2 Decision Trees

Decision tree induction algorithms are among the most widely used classification methods in data mining. Whereas neural networks and SVMs are perhaps the most popular representatives of the non-symbolic class of learning algorithms, decision-tree methods are the

most widely used symbolic algorithms. In this section, the strategy of classifying examples by decision tree is described, and the decision tree learning algorithm is discussed.

2.2.1 Decision-Tree Classification

A *decision tree* is a flow-chart-like tree structure, where each *internal node* denotes a test on a feature, each *branch* represents an outcome of the test, and *leaf nodes* represent classes or class distributions. The topmost node in a tree is the *root* node. A typical decision tree is shown in figure 2.1 [19]. It represents the concept *buys_computer*, that is it predicts whether or not a customer at *AllElectronics* is likely to purchase a computer. Internal nodes are denotes by rectangles, and leaf nodes are denoted by oval.

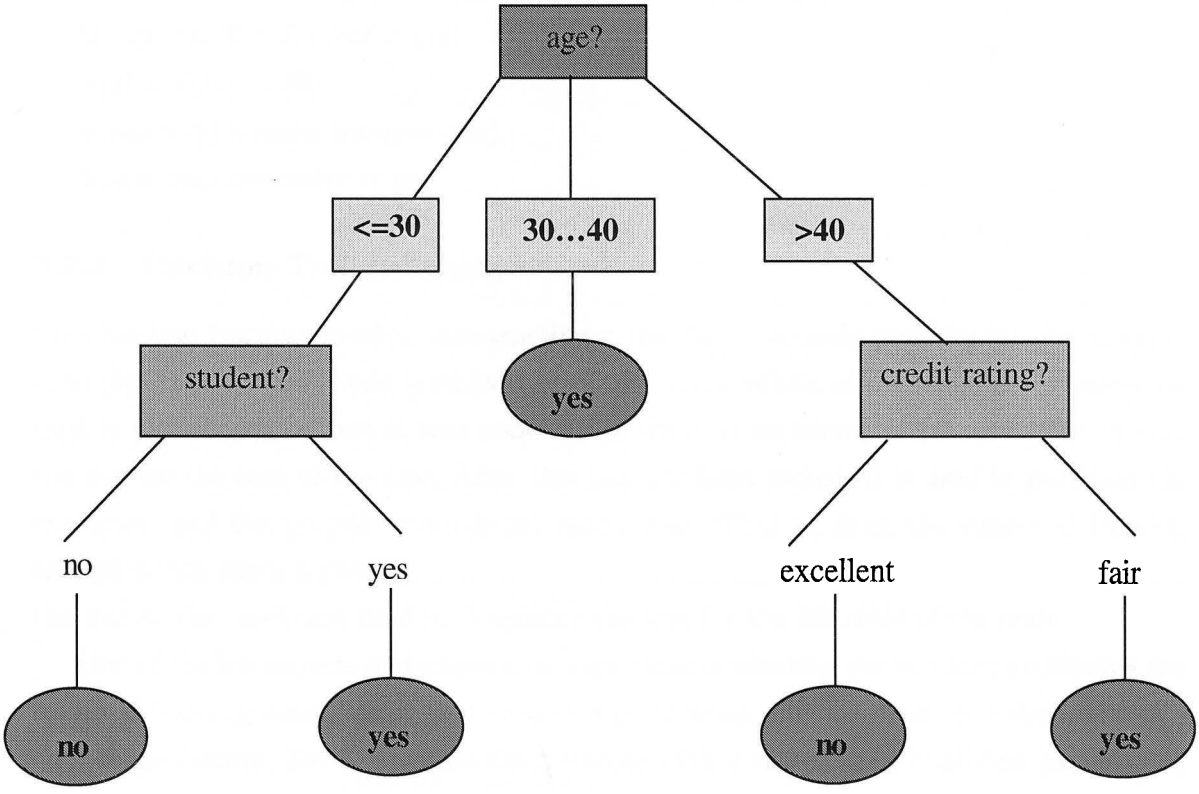


Figure 2.1: A decision tree for the concept *buys_computer*, indicating whether or not a custmor at *AllElectronics* is likely to purchase a computer. Each internal (nonleaf) node represents a test on a feature. Each leaf node represents a class (either *buys_computer* =*yes* or *buys_computer* =*no*)

In order to classify an unknown example, the attribute values of the examples are tested

against the decision tree. A path is traced from the root to a leaf node that holds the class prediction for that example. For instance, the example

$< age = 25, student = no, credit\_rating = fair >$

would be sorted down the leftmost branch of the decision tree in figure 2.1 and would therefore be classified as a negative example, i.e. the customer is not likely to purchase a computer.

Decision trees can be re-presented as a set of If-Then rules to improve understandability. In general, decision trees represent a disjunction of conjunction of constraints on the feature values of instances. Each path from the tree root to a leaf corresponds to a conjunction of feature tests, and the tree itself to a disjunction of these conjunctions. For example, the decision tree shown in figure 2.1 corresponds to the expression

If$(age <= 30 \wedge student = yes)$

$\vee(31 < age <= 40)$

$\vee(age > 40 \wedge credit\_rating = fair)$

Then $buys\_computer = yes$

2.2.2 Decision-Tree Learning

Decision-tree learning involves constructing a tree by recursively partitioning the training examples. Each time a node is added to the tree, some subset of the training examples are used to pick the logical test at that node. All of the training examples are used to determine the rest for the root of the tree. After this test has been picked, it is used to partition the examples, and the process is continued recursively. That is, from the subset of training examples that reach a given internal node, only the examples that have the ith outcome for the test at that node are used to determine the test for the ith child of the node.

One of the key aspects of decision tree algorithms is selecting the test for partitioning the subset of training examples, S, that reaches a given node. C4.5[30] uses tests that are based on a single feature. For a discrete-valued feature with v values, C4.5 considers partitioning based on a test with v outcomes - one for each possible value. For a real-valued feature, C4.5 considers binary tests that compare the feature against various thresholds. The outcomes in this case are either that (1) the value is less than or equal to the threshold, or (2) the value is greater than the threshold. The thresholds considered by C4.5 for a real-valued split at a node are determined by the values that occur in the training examples that reach that node.

In order to pick a splitting test from a set of candidates, C4.5 uses an evaluation measure called *information gain.* In order to define information gain precisely, a measure commonly used in information theory, called *entropy* is defined as:

$$entropy(S) = -\sum_{j=1}^{k} \frac{freq(C_j,S)}{|S|} log_2(\frac{freq(C_j,S)}{|S|})$$

where j ranges over the classes and $freq(C_j, S)$ is the number of examples in S that belong to class C_j. Entropy characterizes the (im)purity of an arbitrary collection of examples and the information needed to identify the class of an example. Given the partition T, entropy is defined as the expected value of the entropy over the subsets induced by T:

$$entropy_T(S) = \sum_{j=1}^{n} \frac{|S_i|}{|S|} entropy(S_i)$$

Here i ranges over the outcomes of T and S_i is the subset of examples in S that have the ith outcome.

Given entropy as a measure of the impurity in a collection of training examples, the information gain picks the test, T, that maximizes the expected reduction in entropy caused by partitioning the examples according to this test:

$$gain(T) = entropy(S) - entropy_T(S)$$

where $entropy$(S) is the amount of information needed to identify the class of an example in S, and $entropy_T(S)$is the corresponding measurement after S has been partitioned according to T.

Another key aspect of a decision tree algorithm is determining when to stop growing a tree.C4.5 uses several stopping criteria to decide when to make a node into a leaf. First, if the subset of examples that reaches a node are all members of the same class then C4.5 will not split the subset any further. Second, if C4.5 cannot find a test that result in at least two outcomes having a minimum number of examples in them, then it will stop splitting at this node. Finally, if the list of candidate tests available to use at a node is empty, then C4.5 will not partition this node further.

After C4.5 has grown a tree, it then tries to simplify it by pruning away various subtrees and replacing them with leaves. This strategy is a method for avoiding *over- fitting.* C4.5's pruning method considers replacing each internal node by either a leaf or one of the node's branches. In order to decide if a change should be made, C4.5 computes a confidence interval

around the resubstitution rate for the node. A change is made to a subtree if the resulting resubstitution error rate for the modified subtree is within a $C\%$ confidence interval of the unmodified subtree's error rate, where C is a parameter of the algorithm that determines how conservative the pruning process should be.

2.3 Rule Induction

One of the most expressive and human readable representation for learned models is sets of If-Then rules. This section explores the algorithms for learning such sets of rules. In many cases it is useful to learn the target function represented as a set of If-Then rules that jointly define the function. As discussed in section 2.2, one way to learn sets of rules is to first learn a decision tree, then translate the tree into an equivalent set of rules - one rule for each leaf node in the tree. In this section we overview a variety of algorithms that directly learn rule sets.

Most algorithms for learning rule sets are based on the strategy of learning one rule, removing the data it covers, then iterating this process. Such algorithms are called *sequential covering* algorithms. To elaborate, imaging we have a subroutine LEARN_ONE_RULE that accepts a set of positive and negative training examples as input, then outputs a single rule that covers many of the positive examples and few of the negative examples. We require that this output rule have high accuracy, but not necessarily high coverage. By high accuracy, we mean the prediction it makes should be correct. By accepting low coverage, we mean it need not make prediction for every training example. Given this LEARN_ONE_RULE subroutine for learning a single rule, one obvious approach to learning a set of rules is to invoke LEARN_ONE_RULE on all the available training examples, remove any positive examples covered by the rule it learns, then invoke it again to learn a second rule based on the remaining training examples. This procedure can be iterated as many times as desired to learn a disjunctive set of rules that together cover any desired fraction of the positive examples. This is called a *sequential covering* algorithm because it sequentially learns a set of rules that together cover the full set of positive examples. The final set of rules can then be sorted so that more accurate rules will be considered first when a new example must be classified. Unlike a decision tree, a rule set may not fully cover the data space. Some examples cannot be covered by any rule in the rule set and thus the rule set cannot make a decision on these examples. In these cases, a default rule which labels the examples as

the majority class is applied. A prototypical sequential covering algorithm is described in figure 2.2.

Algorithm 2.1: SEQUENTIAL_COVERING

Input: training examples $\vec{X}$ and their labels Y.
Output: a disjunctive set of rules.
begin
1 $Learned\_rules = \{\}$
2 $Rule = LEARN\_ONE\_RULE(\vec{X}, Y)$
3 **while** $PERFORMANCE(Rule, \vec{X}) > Threshold$ **do**
4 $Learned\_rules = Learned\_rules \cup \{Rule\}$
5 $\vec{X} = \vec{X}$- {examples correctly classified by Rule}
6 $Y = \{$ labels of examples in $\vec{X}\}$
7 $Rule = LEARN\_ONE\_RULE(\vec{X}, Y)$
end
8 $Learned\_rules$ = sort $Learned\_rules$ accord to $PERFORMANCE$ over the whole training data set
9 **return** $Learned\_rules$
end

Figure 2.2: The sequential covering algorithm for learning a disjunctive set of rules.

This sequential covering algorithm is one of the most widespread approaches to learning disjunctive sets of rules. It reduces the problem of learning a disjunctive set of rules to a sequence of simpler problems, each requiring that a single conjunctive rule be learned. Because it performs a greedy search, formulating a sequence of rules without backtracking, it is not guaranteed to find the smallest or best set of rules that cover the training examples.

One effective approach to implementing LEARN-ONE-RULE is to organize the hypothesis space search in the same general fashion as the decision tree algorithm, but to follow only the most promising branch in the tree at each step. The search begins by considering the most general rule precondition possible (the empty test that matches every example), then greedily adding the feature test that most improves rule performance measured over the training examples. Once this test has been added, the process is repeated by greedily adding a second feature test, and so on. Like the decision tree algorithm, this process grows the hypothesis by greedily adding new feature tests until the hypothesis reaches an acceptable level of performance. Unlike the decision tree algorithm, this implementation of LEARN_ONE_RULE follows only a single descendant at each search step – the feature - value pair yielding the best performance – rather than growing a subtree that covers all

possible values of the selected feature.

This approach to implementing LEARN_ONE_RULE performs a general-to-specific search through the space of possible rules in search of a rule with high accuracy, though perhaps with incomplete coverage of the data. As in decision tree learning, there are many ways to define a measure to select the "best" descendant. To follow the lead of decision tree algorithms let us for now define the best descendant as the one whose covered examples have the lowest entropy.

The general-to-specific search suggested above for the LEARN_ONE_RULE algorithm is a greedy depth-first search with no backtracking. As with any greedy search, there is danger that a suboptimal choice will be made at any step. To reduce this risk, we can extend the algorithm to perform a *beam search*; that is, a search in which the algorithm maintains a list of the k best candidates at each step, rather than a single best candidate. On each search step, descendants (specializations) are generated for each of these k best candidates, and the resulting set is again reduced to the k most promising members. Beam search keeps track of the most promising alternatives to the current top-rated hypothesis, so that all of their successors can be considered at each search step. This general to specific beam search algorithm is used by the CN2 described in figure 2.3.

2.4 Support Vector Machines

2.4.1 Introduction to Support Vector Machines

SVMs have a solid theoretical foundation based on the statistical learning theory [34]. Consider a binary classification task with the training data set $\{\vec{x_i}, y_i\}_{i=1}^{N}$,$\vec{x_i} \in R^m$, $y_i \in \{-1, +1\}$. SVMs find a hyperplane that correctly separates the training data of the two different classes while maximizing the distance of either class from the hyperplane (maximizing the margin). Thus, they can generalize well to the test data even in the presence of a large number of features, as long as the training data can be separated by a sufficiently wide margin. SVMs can also deal with linear non-separable data sets by either using a kernel function K to map the original data vectors into a much higher dimensional space, called *feature space* ,where the data points are linearly separable, or by using soft margin separation hyperplanes that allow some degree of training errors in order to obtain a large margin. The direction of the maximal margin hyperplane is determined through a set of *support vectors* SV, which are data vectors lying on the margin. A new example $\vec{x}$ is

Algorithm 2.2: LEARN_ONE_RULE

Input: training examples $\vec{X}$ and their labels Y ; number of best candidates at each step, k .

Output: a single best rule.

begin

1 Initialize *Best_hypothesis* to the most general hypothesis *null*

2 Initialize *Candidate_hypothesis* to the set { *Best_hypothesis*}

3 **while** *Candidate_hypothesis is not empty* **do**

/*`Generate the next more specific` *Candidate_hypotheses* */

4 *All_constraints* ⟵ the set of all constraints of the form $(a = v)$, where a is a feature , and v is a value of a that occurs in the current set of $\vec{X}$

5 **foreach** *h in Candidate_hypotheses* **do**

6 **foreach** *c in All_constraints* **do**

7 *New_candidate_hypothesis* ⟵ create a specification of h by adding the constraint c

end

end

8 Remove from *New_candidate_hypotheses* any hypotheses that are duplicates, inconsistent, or not maximally specific

/*`Update` *Best_hypothesis* */

9 ForEach h in *New_candidate_hypotheses*

10 **if**

$PERFORMANCE(h, \vec{X}, Y) > PERFORMANCE(Best\_hypothesis, \vec{X}, Y)$

then

11 *Best_hypothesis* ⟵ h

end

/*`Update` *Candidate_hypotheses* */

12 *Candidate_hypothesis* ⟵ the k best members of *New_candidate_hypotheses*,according to the $PERFORMANCE$ measure

end

13 **return** *a rule of the form "**IF** Best_hypothesis **THEN** prediction",where prediction is the most frequent value of Y among those examples in $\vec{X}$ that match Best_hypothesis*

end

Algorithm 2.3: PERFORMANCE

Input: a hypothesis h; training examples $\vec{X}$ and their labels Y .

Output: the value of h's performance.

begin

1 *h_examples* ⟵ the subset of $\vec{X}$ that match h

2 **return** *Entropy(h_examples), where entropy is with respect to the labels*

end

Figure 2.3: The sequential covering algorithm for learning a disjunctive set of rules.

classified depending on the sign of the following decision function:

$$f(\vec{x}) = \sum_{x_i \in SV} \alpha_i y_i K(\vec{x_i}, \vec{x}) + b \tag{2.1}$$

where $K(\vec{x_i}, \vec{x})$ is the kernel function ,y_i is the class label of the support vectors and α_i are the weights associated with the support vectors, obtained by solving the following convex Quadratic Programming (QP) problem:

$$\underset{\vec{\alpha}}{\text{maximize}} \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{i=1}^{N} \alpha_i \alpha_j \cdot y_i y_j \cdot K(\vec{x}_i, \vec{x}_j) \tag{2.2}$$

$$\text{subject to} \quad \alpha_i \geq 0, i = 1, 2, ..., N$$
$$\textstyle\sum_{i=1}^{N} \alpha_i y_i = 0.$$

Once the weights α_i are determined, support vectors are just those input vectors who have non-zero weights. In equaion 2.2 , N is the number of input vectors, i.e., number of training examples.

Typical kernel functions include:

- Linear Kernel Function:

$$K(\vec{x_i}, \vec{x}) = \vec{x_i} \cdot \vec{x} \tag{2.3}$$

- Polynomial Kernel Function:

$$K(\vec{x_i}, \vec{x}) = (\vec{x_i} \cdot \vec{x} + 1)^d \tag{2.4}$$

- Radial Basic Function(RBF):

$$K(\vec{x_i}, \vec{x}) = exp(-\gamma \|\vec{x_i} - \vec{x}\|^2) \tag{2.5}$$

- Sigmoid Function:

$$K(\vec{x_i}, \vec{x}) = tanh(\gamma \vec{x_i} \cdot \vec{x} + \theta) \tag{2.6}$$

The parameter d in equation 2.4 is the *degree* of the polynomial function; when $d = 1$, the function will revert into the linear kernel function.

Figure 2.4 illustrates the basic concept of SVMs.

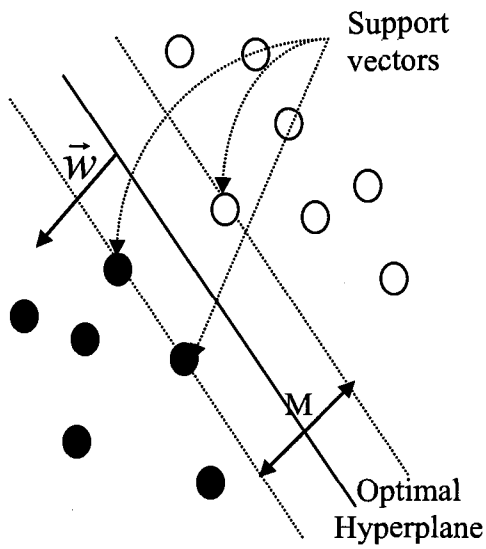


Figure 2.4: A linear SVM for a two-dimensional training set. M is the margin between the two classes. $\vec{w}$ is the norm vector of the separating hyperplane.

2.4.2 Support Vector Machines and Understandability

An SVM decision function defined in form of (2.1) is hard to understand because it implicitly involves (in varying degrees) all features that are relevant for the definition of the separating hyperplane. SVMs do not explicitly assign weights to all features, implicit feature weights can be derived from the weights of the training instances using the kernel functions. Nevertheless, we argue that it is still very hard for domain experts to understand the reasoning behind the SVM decision functions and to identify important features in the input spaces from these decision functions.

This is caused by the following two reasons. Firstly, if nonlinear kernel functions are used, the feature space will contain many more features than the input space. Generally, it is very hard for domain experts to understand the meaning of these new derived features.

Secondly, when linear kernel functions are used, typically a large percentage of all features are relevant to the SVM classifiers. This has been experimentally investigated in [7], which introduces another form of SVMs that can select very few relevant features as a result of the trained SVM models. Unlike the standard SVMs where the distance of either class from the separating hyperplanes is measured in 2-norm, the distance of this special form of SVMs (so called 1-norm SVMs) is measured in 1-norm. The vector norm $|\vec{x}|_p$ for

Table 2.2: Average number of features selected in the SVM classifiers (Asterisk* indicates that the full experiment had not been carried out because of excessive time hence results are averaged over folds completed [7].

DataSet	# Input Features	# Features Selected by 2-norm SVMs
Wisconsin Prognostic Breast Cancer(24 mo.)	32	32
Wisconsin Prognostic Breast Cancer(60 mo.)	32	32
Johns Hopkins University Innosphere	34	33*
Cleveland Heart Disease	13	13*
BUPA Liver Disorders	6	6*

$p = 1, 2, ...$ is defined as $|\vec{x}|_p = (\sum_i |x_i|^p)^{\frac{1}{p}}$. [7] experimentally investigates the average number of features selected by (i.e. number of features that are involved in) the standard linear SVM classifiers on 5 public data sets using 10-fold cross validation. The description of these datasets and results are listed in table 2.2. The results indicate that on five datasets almost all features in the input spaces are relevant to (involved in) the SVM classifiers.

We observe similar results on a biology sequence dataset, which appears to be the most difficult classification problem investigated in chapter 5. Figure 2.5 shows the distribution of all feature weights that are derived from a linear SVM trained to predict if a protein is an outer membrane protein from its amino acid sequence. The features are in an ascending order of the absolute value of their weights, which indicate their importance to the SVM classifier. Among 1425 features, the weights of 1420 features are non-zero. The distribution among features is more or less uniform, with a standard deviation of 0.0380 from the mean 0.0387. It seems to be difficult to make much sense of these weights in isolation.

To summarize, both linear and nonlinear SVM classifiers are hard to understand for a domain expert. In order to address the deficiency in understandability of SVMs, we propose an approach that can extract understandable rules from both linear and nonlinear SVMs in chapter 4.

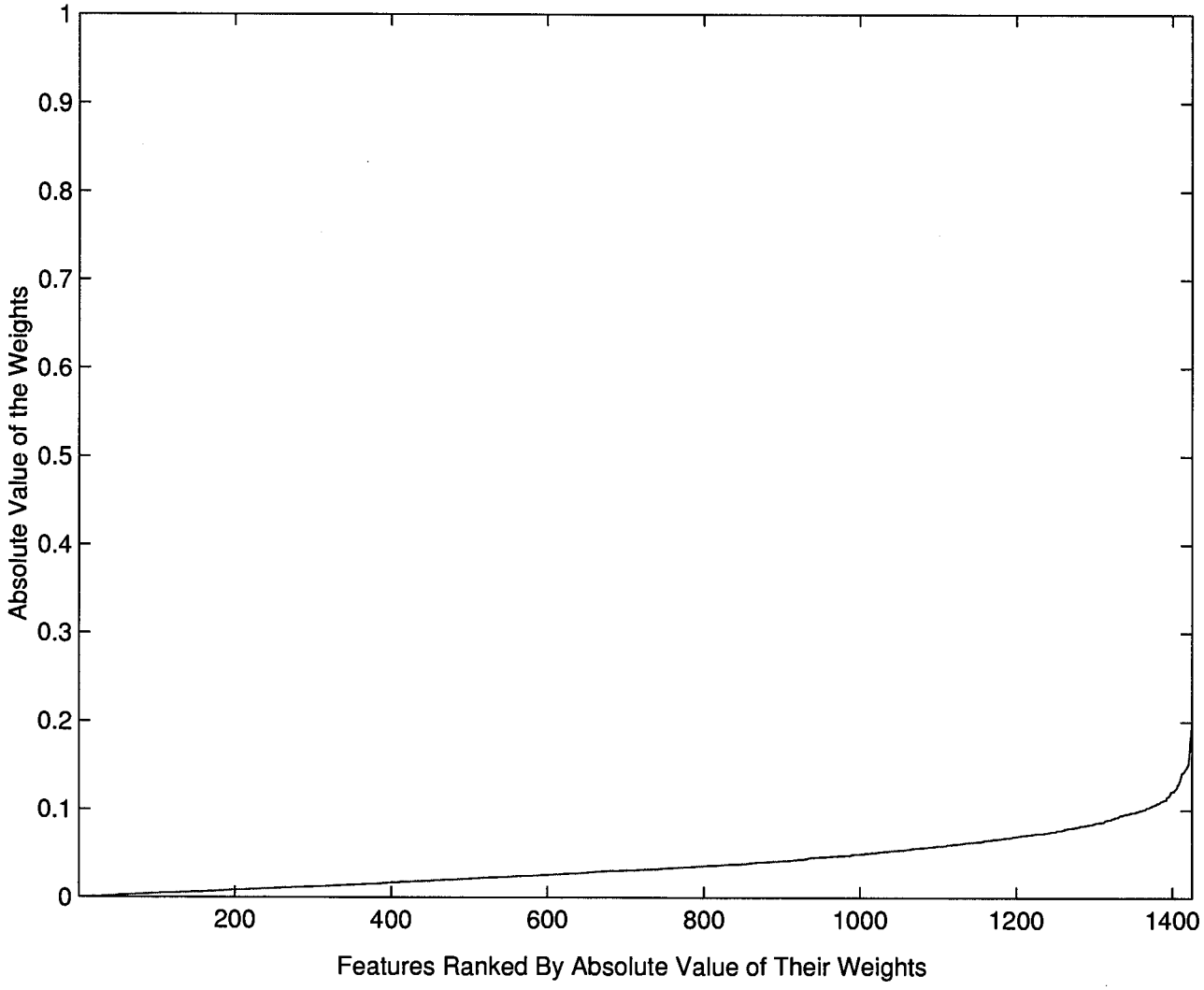


Figure 2.5: Feature weights distribution of a linear SVM.

Chapter 3

Related Work

In this chapter, I review previous work related to the problem of interpreting trained SVMs and techniques related to our proposed BUR algorithm. The purpose of this discussion is to provide suitable context for the novel work introduced in the subsequent chapter.

3.1 Related Problems

3.1.1 The Rule Extraction Task

A significant research effort has been expended in the last decade to address the deficiency in the understandability of neural networks [31] [33] [25] [20]. [26] presents a complete overview on this research. The generally used strategy to understand a model represented by a trained neural network is to translate the model into a more comprehensible language (such as a set of If-Then rules or a decision tree). This strategy is investigated under the rubric of *rule extraction.*

[25] defines the task of rule extraction from neural networks as follows:

> *" Given a trained neural network and the data on which it was trained, produce a description of the network's hypothesis that is comprehensible yet closely approximates the network's prediction behavior."*

Although the task of rule extraction has only been formally formulated in the context of interpreting neural networks, this formulation can be generalized to any other opaque models (such as SVMs).

A variety of concerns have been addressed to evaluate the proliferation and diversity of techniques for extracting rules from trained neural networks in literatures. However, the following criteria are considered most relevant for measuring schemes of general rule extraction:

- *Understandability:* The extent to which the representation of the extracted models are humanly comprehensible.
- *Accuracy:*The ability of extracted models to make accurate predictions on previously unseen cases.
- *Scalability:*The ability of the method to scale to large input space and more complicated incomprehensive models.

In the subsequent sections and chapters, the term "*understandability*" and "*comprehensibility*" are interchangeably used.

Whereas *understandability* and *accuracy* are prime concerns for a rule extraction method, *scalability* also plays an important role in evaluating the rule extraction methods. A desirable rule-extraction technology with a greater impact requires that the method can be applied to a wider range of learned models in a wide range of application domains. In part, this means the extraction methods can be applied to problems and learned models that are, in some degree "big". Thus, we would like to have rule extraction algorithms that are scalable. In particular, the following definition is of interest for the task rule extraction [26]:

> *Scalability* refers to how the running time of a rule extraction algorithm and the *comprehensibility* of its extracted models vary as a function of such factors as the trained incomprehensible model, feature set, and training set size.

This definition differs from the conventionally used definition in that it includes the *understandability* of extracted models as well as the running time of the method. Since comprehensibility is of fundamental importance in rule extraction, methods that scale well in terms of running time, but not in terms of understandability will be little value.

Scaling in terms of understandability is a hard problem. In cases where the problem domain has a large number of features, or the trained incomprehensible models are of very complicated structures (such as neural networks with "big" architectures or SVMs with

complicated kernel functions), the classification functions represented by the trained incomprehensible models are often too complicated for a succinct comprehensible model to closely approximate.

To address the issue of scalability in understandability, a strategy for controlling the understandability and accuracy tradeoff can be utilized with the rule extraction algorithms. This strategy is to improve the understandability of an extracted model by compromising on its accuracy of approximation to the large, complicated trained models.

In the subsequent subsections, we will review the state-of-art rule extraction algorithms and discuss the applicability of these methods to the task of interpreting SVMs. As we have discussed in section 1.1, we want to investigate the problem of interpreting SVMs in the scenario of high dimensional feature spaces where SVMs demonstrate substantial benefits over other classification methods. Thus we will focus on discussing the performance (in terms of the accuracy and understandability of the extracted models) of existing rule extraction methods in high dimensional feature spaces.

Specifically, the discussions are with respect to the following issues:

- What language is used to represent the extracted models?
- What extraction strategy is applied by the algorithm?
- Does the algorithm utilize any strategies to address the issue of scalability in understandability?
- Is the method (if it is a method of extracting rules from neural networks)applicable to the task of extracting rules from SVMs? Or is there any limitation of the existing rule extraction method(if it is a method of extracting rules from SVMs)?

3.1.2 Rule Extraction from Neural Networks

Existing rule extraction algorithms developed for neural network can be categorized into two families by their extraction strategy: algorithms that set up the task as a search problem and algorithms that view the problem as an inductive learning task. In this section, we survey well-known methods from both families.

The search-based rule extraction algorithms extract a set of conjunctive rules from trained neural networks. The extraction strategy involves exploring a space of candidate

conjunctive rules and testing individual candidates against the network to see if they are *valid* rules.

Most of these algorithms conduct their search through a space of conjunctive rules. Figure 3.1 shows a rule search space for a problem with three Boolean features. Each node in the tree corresponds to the antecedent of a possible rule, and the edges indicate specialization relationships between nodes. The node at the top of the graph represents the most general rule (i.e., a rule that covers all examples), and the nodes at the bottom of the tree represent the most specific rules. Unlike most search processes which continue until the first goal node is found, a rule-extraction search continues until all (or most) of the *maximally general* rules have been found. A rule is *valid* if the predictions of the rule agree with the perditions of the trained neural network on all those examples that match the rule's antecedent. A rule is *maximally general* if any one of the literals is dropped from the rule's antecedent, the rule is no longer *valid.*

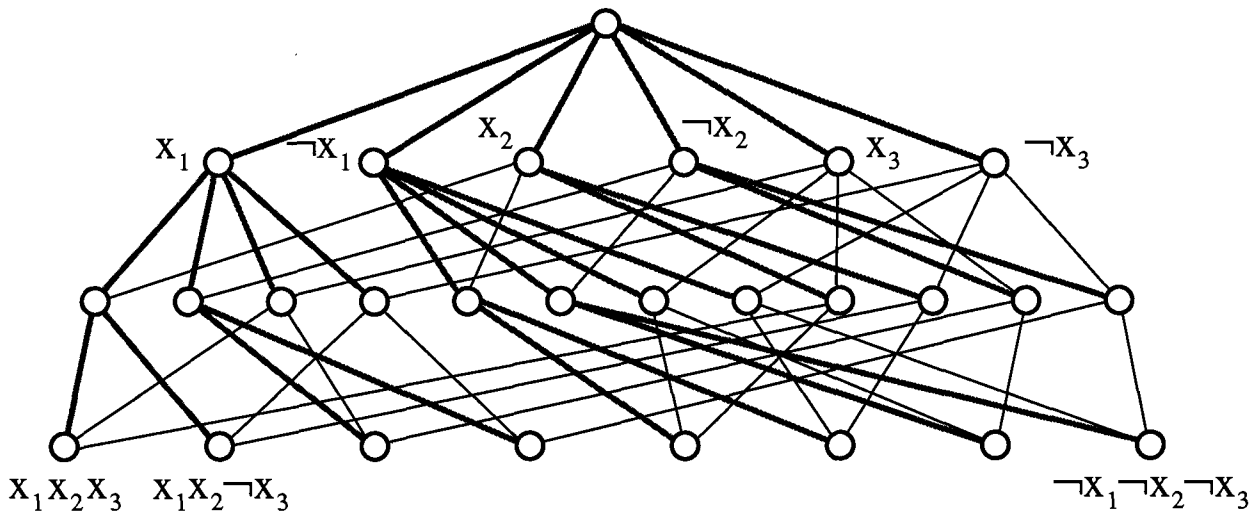


Figure 3.1: A rule search space.

Notice that rules with more than one literal in their antecedent have multiple ancestors in the graph. Obviously when exploring a rule space, it is inefficient for the search procedure to visit a node multiple times. In order to avoid this inefficiency, we can impose an ordering on the literals thereby transforming the search graph into a tree. The thicker (red) lines in figure 3.1 depict one possible search tree for the given rule space.

One of the problematic issues that arise in search-based approaches is that the size of

the rule space can be very large. For an input feature space with n binary features, there are 3^n possible conjunctive rules (since each feature can be absent from a rule antecedent, or it can occur as a positive or a negative literal in the antecedent). To address this issue, a number of heuristics have been employed to limit the combinatorics of the rule-exploration process.

Several rule-extraction algorithms manage the combinatorics of the task by limiting the number of literals that can be in the antecedents of extracted rules [31] [18]. For example , Satio and Nakano's algorithm uses two parameters, k_{pos} and k_{neg} , that specify the maximum number of positive and negative of literals respectively that can be in an antecedent. By restricting the search to a depth of k, the rule space considered is limited to a size given by the following expression:

$$\sum_{i=0}^{k} \binom{k}{n} 2^k$$

For fixed k, this expression is polynomial in n, but obviously, it is exponential in depth k. This means that exploring a space of rules might still be intractable since, for some networks, it may be necessary to search deep in the tree in order to find valid rules.

The limited search space also improves the understandability of the extracted rules. Human experts often have difficulties in understanding rules with many (e.g 30) literals in the antecedents and a rule set with large number of rules (e.g 100). Limiting the number of literals that can be in the antecedents of extracted rules to a user desirable level will partially improve the understandability of the extracted rule sets. Therefore, algorithms that apply this heuristic can control the tradeoff between understandability and the accuracy of approximation by adjusting the parameter k_{pos} and k_{neg}. Enlarging these parameters (the extreme case is to set them to the maximal possible value, which means exploring the entire search space) will result in more accurate but less understandable rules. On the other hand, understandability of the extracted rules can be improved by compromising the search space of the maximal general rules.

The drawback of the method is that it can only be applied to discrete features and to low-dimensional data sets. Otherwise, the search space of all possible rules would become too large to be enumerated and explored. Even though the heuristic of limiting search space can alleviate the problem of inefficient search in high dimensional feature spaces, the algorithms still have difficulties in numeric feature spaces. Many, if not most, classification applications involve both numeric-valued and discrete-valued features. Furthermore, the algorithms will

return all or most rules that are maximally general. At the same time there is no mechanism to control the tradeoff between the understandability of the resultant rule sets in terms of number of rules and the accuracy of the approximation. Hence the understandability is not fully scalable to high dimensional spaces.

The alternative rule extraction algorithms are learning based methods, which are more efficient than the search based methods. In this learning task, the target concept is the function represented by the network, and the hypothesis produced by the learning algorithm is the model represented in a language used by the rule extraction algorithms to approximate the network.

TREPAN [25] takes a trained neural network and a set of training data as inputs. As output, it produces a decision tree that provides a close approximation to the function represented by the network. The task of TREPAN then, is to induce the function represented by the trained network.

In many respects, TREPAN is similar to conventional decision tree algorithms overviewed in section 2.2, which induce trees directly from training data. TREPAN's learning task differs in several key aspects, however. First, the target concept to be learned by TREPAN is the function represented by the network. This means that TREPAN uses the network to label (i.e. get the predicted output value of) all instances. Second, because TREPAN can use the network to label instances, it learns not just from a fixed set of training data, but from arbitrarily large samples. This can avoid the lack of examples for the splitting test in the lower levels of the tree and could alleviate the overfitting problem in the conventional decision tree algorithms.

Figure 3.2 provides a sketch of the TREPAN algorithm. The basic idea of the method is to progressively refine an extracted description of a neural network by incrementally adding nodes to a decision tree that characterizes the network. Initially, TREPAN's description of the network is a single leaf node that predicts the class that the network itself predicts most often. This crude description of the network is refined by iteratively selecting a leaf node of the tree to expand into an internal node with leaves as children.

In order to decide which node to expand next, TREPAN uses an evaluation function to rank all of the leaves in the current tree and then picks the node with the greatest potential to increase the accuracy of approximation to the network. The motivation for expanding an extracted tree in this best-first manner is that it gives the user a fine degree of control over the size of the tree to be returned: a tree of arbitrary size (in terms of number of

Algorithm 3.1: TREPAN

Input: the trained neural netowrk;training examples $\{\vec{x}_i, y_i\}_{i=1}^{N}$, where y_i is the class label predicted by the trained neural network on the training example $\vec{x}_i$; global stopping criteria.

Output: extracted decision tree.

begin

1 initialize the tree as a leaf node

2 **while** *global stopping criteria are not met and current tree can be further refined* **do**

3 pick the most promising leaf node to expand

4 draw a sample of examples

5 use the trained network to label these examples

6 select a splitting test for the node

7 for each possible outcome of the test make a new leaf node

end

end

Figure 3.2: TREPAN algorithm.

internal nodes) can be selected to describe a given network. The size of the decision tree could be viewed as a measure of the understandability of the extracted models. The best first manner to grow a tree allows the user to directly control the trade-off between the understandability and accuracy of approximation to neural networks. Hence the issue of scalability in understandability is addressed.

Expanding a node in the tree involves two key tasks: determining a logical test with which to partition the examples that reach the node, and determining the class labels for the leaves that are children of the newly expanded node. In order to make these decisions, TRPEAN ensures that it has a reasonably large sample of examples. It gets these examples from two sources. First, it uses the network's training examples that reach the node. Second, TREPAN constructs a model (using the training examples) of the underlying distribution of data in the domain, and uses this model in a generative manner to draw new examples. These examples are randomly drawn but are subject to the constraint that they would reach the node being expanded if they were classified by the tree. In both cases, TREPAN queries the trained neural networks to get the class label for the examples.

TREPAN can be applied directly to the task of interpreting SVMs by substituting neural networks with SVMs as an oracle. But the method is not expected to work well in a high dimensional feature space because the effectiveness of this method largely relies on accurate

density estimation, which also suffers from the "curse of dimensionality"[16]. In fact, the largest dimensionality of the datasets used in the experimental evaluation of TREPAN [25] is only 64. We will further discuss the experimental comparison of our proposed approach with TREPAN in chapter 5.

3.1.3 Rule Extraction from Support Vector Machines

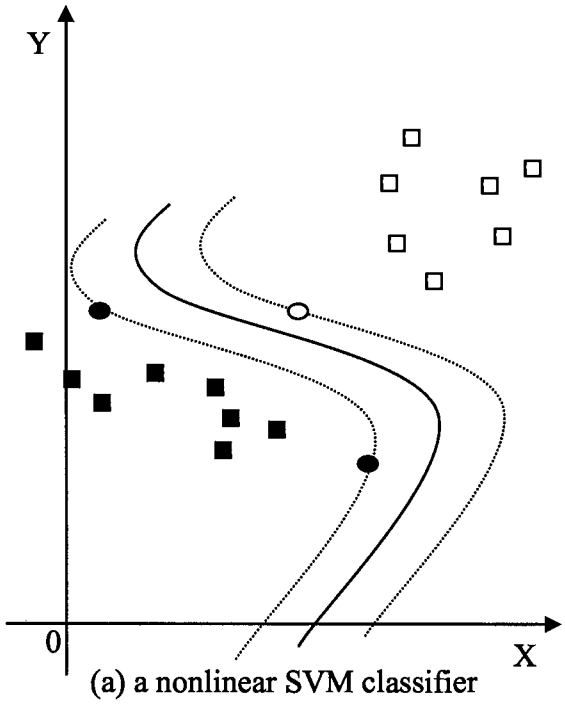
To date, few papers have focused on interpreting SVMs. [27] uses a geometry based method to extract a set of If-Then rules from a trained SVM. Given a trained SVM and its training dataset, a cluster is first obtained through a clustering algorithm for positive class and negative class respectively. For each cluster, a hyperrectangle is constructed in the full dimensional space using the following way. The center of each cluster, termed *prototype* vector, is used as the center of a hyperrectangle. In each cluster region, the support vector which is farthest from the cluster's prototype vector is chosen as the corner of that hyperrectangle. These hyperrectangles can later be translated into If-Then rules.

In order to define the number of hyperrectnagles per class, the algorithm follows an incremental scheme. Beginning with a single cluster per class, the associated hyperrectangle is generated. Next, a partition test is applied on this region. If it is negative, the region is translated into an If-Then rule. Otherwise, new regions are generated. This procedure is repeated while a region that fulfils the partition test exists or until the maximal number of iterations is reached. The process results in a desirable number of generated If-Then rules.

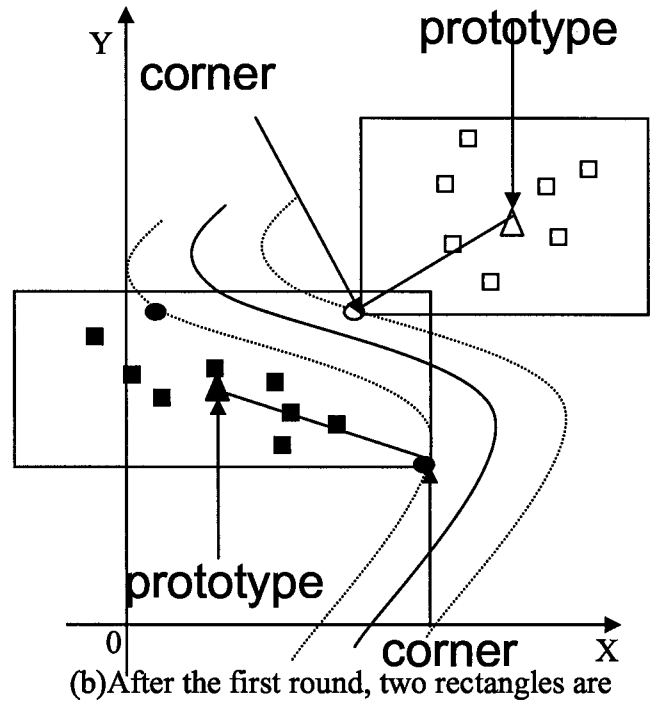
For each iteration, there are p regions with a negative partition test (they will be translated into If-Then rules) and m regions with a positive partition test. In the next iteration, data from these m regions are used to determine $m+1$ new clusters and the associated new hyperrectangles. If the maximum number of iterations is reached, all the regions (independently of the results of the partition test) are translated into rules.

The partition test attempts to diminish the level of overlapping between regions of different classes by apply several heuristics. Such heuristics could be, for example, the partition test is positive if the generated prototype belongs to another class, or if a support vector from another class exists within the region. Iteratively reducing the overlapping between regions of different classes and thus increasing the accuracy of resulting rule sets, the partition process gains a good tradeoff between the accuracy and size of resulting rule size.

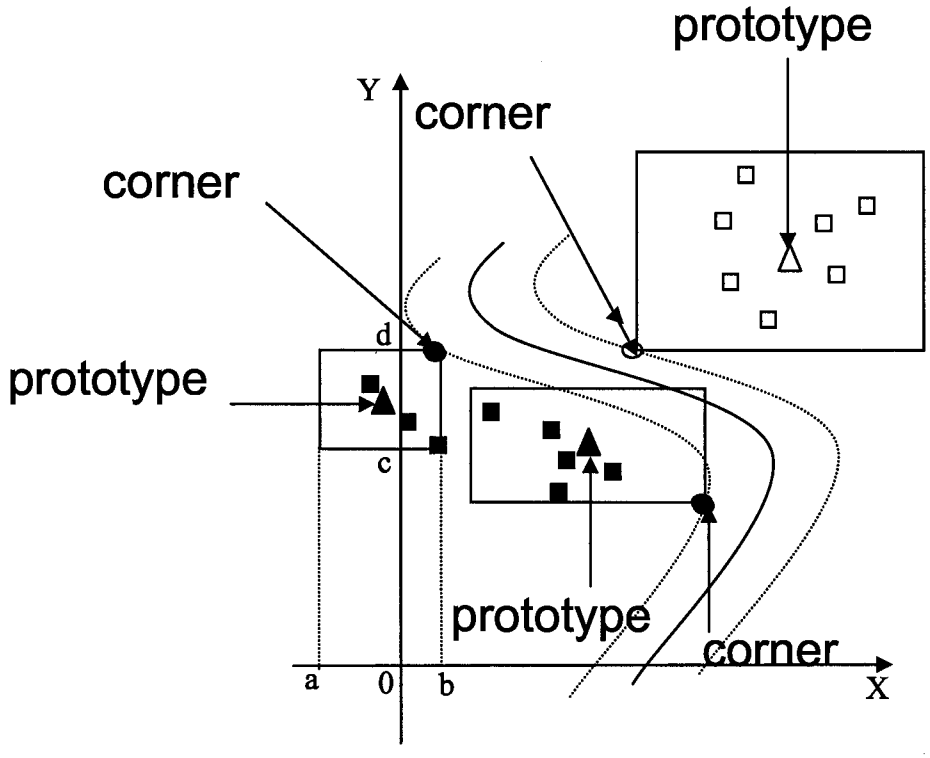
An example in figure 3.3 illustrates the essential idea of this algorithm. Figure 3.3 (a)



(a) a nonlinear SVM classifier



(b)After the first round, two rectangles are generated, with some overlapping between them.



(c)After the second round, three rectangles are obtained, without any overlapping between region of different classes.

Figure 3.3: An example of extracting If-Then rules from SVMs.

shows a trained nonlinear SVM that distinguishes black points from white pints in a two dimension space. The round points represent support vectors of both classes. Figure 3.3 (b) shows the result after the first iteration of the algorithm. The triangles represent the prototypes, i.e. centers of clusters for both classes. With these two prototypes and the furthest support vectors in respective clusters (indicated as "corner" in the figure), two rectangles are defined. Then a partition test is conducted, which results in a positive test, because a white support vector resides in the region of black points. At the second iteration, one more cluster is generated. Three rectangles are constructed using the new prototypes and support vectors. At last, the rectangles can be translated into If-Then rules. For example, the leftmost rectangle in 3.3 (c) can be translated as

$$\textbf{If} \quad a \leq X \leq b \text{ and } c \leq Y \leq d$$
$$\textbf{Then} \text{ it is a black point.}$$

By using support vectors which are the closest points to the separating hyperplane in the class, it is possible to build a set of rectangles that represent the class with minimum overlap between classes. This method can control the number of If-Then rules, but it cannot control the size of the individual rules which all involve every dimension. Therefore the understandability of the extracted rules can not scale to a very high dimensional feature space. In addition the performance depends heavily on the chosen clustering algorithm.

3.2 Related Techniques

Although we are not aware of a learning strategy similar to our proposal, a few ideas that we discuss in this section hint that the proposed BUR algorithm, which learns a set of unordered rules to approximate the SVM classifiers by approximating their classification confidence in the training data, could be promising. In the following subsections, we will review the techniques related to our proposed method and discuss the rationale of our approach.

3.2.1 Choice of The Rule Learner-Decision Trees or Rule Induction Methods

Rule induction methods are the commonly used methods to produce If-Then rules. Also, in section 2.2, we have mentioned that a decision tree can be translated into a set of If-Then

rules. To decide on the most appropriate rule learner for the task of extracting accurate and understandable IF-Then rules from trained SVMs, we need to find out more about the properties of learning in high dimensional features spaces.

Previous work [22] indicates that high dimensional applications share the properties of dense concepts and sparse instances. Specifically speaking, most features seem equally relevant to class labels yet only few features are relevant to a certain example. Recently, researchers began to investigate deeply these properties of learning in high dimensional feature spaces. [5] [10], e.g., address the problem of clustering in high dimensional spaces by finding clusters in separate subspaces where a cluster is only required to be dense in its corresponding subspace. [13], as another example, evaluates a "sleeping experts" learning algorithm on a number of large text categorization problems, one of the typical high dimensional application domains. Given a pool of fixed "experts"- each of which is usually a simple, fixed, classifier (a sequence of words, possibly containing some "gaps", appearing in a fixed order but at any position in a document) -, sleeping experts build a *master*, which combines the classification of the experts in some manner. The master keeps a large pool of experts each associated with a weight, which are allocated in the learning period. In the classification period, whenever a new document is presented, only those experts (i.e. sequences of words) which appear in the document are "awakened" and extracted from the pool. The actual prediction given by the master combines the weights of these awakened experts. A large number of the rest "sleeping" experts have nothing to do with this prediction.

The success of these works suggests that we can reasonably assume that different subsets of features are relevant to (or are awakened for predicting) class labels for data in different regions of the data space. More precisely speaking, the target concept is a disjunction of many independent conjunction rules.

Comparing decision tree learners and rule induction learners in the light of the above discussion, rule induction learners are more appropriate for our task for several reasons. First of all, because their hierarchical structure only allows testing one condition at each level, decision trees are less efficient in learning a disjunction of a set of independent conjunction rules. Moreover, it is well-known that decision trees have the problems of overfitting and instability [30] [8] [9] [24]. Due to the nature of iteratively splitting the data space, decision trees are more likely to overfit in high dimensional spaces, wherein a node could have as many as 100 similar quality splits and a choice may be no better than an arbitrary guess. In contrast, rule induction approaches do not split the entire data space and, therefore, a

previous bad choice will not seriously affect all following choices as it does in the decision tree learning process.

3.2.2 Learning Rules in High Dimensional Feature Spaces

Most rule induction algorithms generate rules assembled in a particular order. During classification of a new example, each rule is tried in this order until one fires and that rule predicts which class the example belongs to. Understanding such rules is not easy since the meaning of any single rule is dependent on all rules that precede it in the rule list. An example in [11] illustrating this problem is as follows:

"**If** feather = yes
Then class = bird
ElseIf leg = two
Then class = human
ElseIf ..."

The rule "**If** leg = two **Then** class = human", when considered alone, is not correct as birds also have two legs. Therefore, to understand the rule in the rule list, all rules preceding the rule must be considered. This problem becomes acute with a large number of rules, making it difficult for an expert to understand the true meaning of a rule far down in the list.

In addition, we assume that separate subsets of features are relevant to class labels for data in different regions of the space. These feature subsets are equally important and should be applied simultaneously in order to decide the class of an example. However, if a rule list is used, the decision is made as soon as one rule in the rule list is fired. This could be an arbitrary decision, considering the number of possible instances existing in a high dimension space and the number of rules we rely on to make a single decision.

Because of these reasons, we adopt the paradigm of unordered rule learning in our proposed approach. [11] proposes a method for generating unordered rules using CN2 [12]. CN2 is a sequential covering rule learning algorithm. At each step, CN2 learns a single best rule by performing a beam search. This has been discussed in section 2.3 and the CN2 algorithm is outlined in figure 2.3. The original CN2 algorithm generates a rule list. The

main modification to this algorithm presented in [11] is to iterate the beam search for each class in turn. When it searches the rules for the positive class, only covered examples of the positive class are removed. Unlike for ordered rules (or rule list), the examples in the negative class remain because now each rule must independently stand against all negative examples. The covered positive class examples must be removed to prevent CN2 from finding the same rule again. The same strategy is applied to searching the rules for the negative class.

With unordered rules, each rule is associated with the class distribution of the covered training examples. At the classification stage, all rules are tried, those which fire are collected and the aggregation of the class distributions corresponding to those rules decides the most probable class for the example.

3.2.3 Framework for Function Estimation by Gradient Boosting Machine

[17] proposes a framework for approximating a function by a gradient boosting machine. Given training examples $\{\vec{x}_i, y_i\}_{i=1}^N$, where $y_i \in R$, the goal is to obtain an estimate $F(\vec{x})$ of the function $y = F^*(\vec{x})$ using the training data, so that the expected value of the loss function $L(y, F(\vec{x}))$ is minimized. The form of $F(\vec{x})$ is restricted to a special additive function as follows:

$F(\vec{x}; \{\beta_m, \vec{a}_m\}_0^M) = \sum_{m=0}^M \beta_m \cdot h(\vec{x}; \vec{a}_m)$, where $h(\vec{x}; \vec{a}_m)$ is a basic learner such as a decision tree, $\vec{a}_m$ is the learning parameters(e.g. define the shape of a decision tree), β_m is the *confidence* which indicates the reliability of the corresponding basic learner and M is the boosting round. This is a generalized form of boosting methods.

The goal is formulated as finding

$$\{\beta_m, \vec{a}_m\}_0^M = \underset{\{\beta'_m, \vec{a}'_m\}}{\operatorname{argmin}} \sum_{i=1}^N L(y_i, \sum_{m=0}^M \beta'_m \cdot h(\vec{x}_i; \vec{a}'_m))$$

In situations where it is very complicated to solve this optimization problem, we can do it by a "greedy-stagewise" approach as follows:

for $m = 0$ **to** M **do**

$$\{\beta_m, \vec{a}_m\}_0^M = \underset{\{\beta'_m, \vec{a}'_m\}}{\operatorname{argmin}} \sum_{i=1}^N L(y_i, F_{m-1}(\vec{x}_i) + \beta'_m \cdot h(\vec{x}_i; \vec{a}'_m))$$

$$F_m(\vec{x}) = F_{m-1}(\vec{x}) + \beta_m \cdot h(\vec{x}; \vec{a}_m)$$

end

Using the "steepest-descent" method, the steepest-descent to minimize the loss function at each $\vec{x}_i$ at step m is

$$\tilde{y}_i = -[\frac{\partial L(y_i, F(\vec{x}_i)}{\partial F(\vec{x}_i)}]_{F(\vec{x})=F_{m-1}(\vec{x})}.$$

The problem can be viewed as greedily finding the best step towards the estimate of $F^*(\vec{x})$ under the constraints that the "steepest direction" is a member of the parameterized class of functions $h(\vec{x};\vec{a})$. In the case that the "steepest direction" is infeasible, $\vec{a}$ is chosen so that $h(\vec{x};\vec{a})$ is most parallel to $\tilde{Y} \in R^N$.

Then a search for the optimal length of the step:

$$\beta_m = \underset{\beta}{\operatorname{argmin}} \sum_{i=1}^{N} L(y_i, F_{m-1}(\vec{x}_i) + \beta \cdot h(\vec{x}_i;\vec{a}_m))$$

is performed and the approximation function is updated as

$$F_m(\vec{x}) = F_{m-1}(\vec{x}) + \beta_m \cdot h(\vec{x};\vec{a}_m).$$

If the approximated function is a classifier, the classification is $sign(\sum_{m=0}^{M} \beta_m \cdot h(\vec{x};\vec{a}_m))$. In this thesis, we use a variant of Gradient Boosting to approximate a trained SVM classifier by a set of unordered If-Then rules, where a simple If-Then rule is treated as a basic learner. The detail of the method is discussed in chapter 4.

Chapter 4

The Algorithm

4.1 Problem Formulation And Overview of the Approach

We formulate the problem of extracting rules from SVMs as follows:

> *In the scenario of high dimensional feature spaces, given a trained SVM and the data on which it was trained, produce a description of the SVM's hypothesis that is understandable yet closely approximates the SVM's prediction behavior.*

In this thesis, we present and evaluate a novel algorithm for the task of extracting understandable description from hard-to-understand SVMs. We propose a method that takes a trained SVM and the data on which it was trained as input and learns a set of If-Then rules to approximate the prediction behavior of SVMs. If-Then rules are generally considered to be understandable and that allow an explicit control of their size. The learning algorithm advanced by this thesis research has the following properties.

First, the approach produces understandable yet accurate description of the SVMs in the scenario of high dimensional feature spaces. Both the understandability and classification accuracy of the rule sets learned by standard rule learners can not scale well to high dimensional feature spaces. This has been discussed in detail in section 3.2.1 and 3.2.2. To overcome these weaknesses of standard rule learners, we exploit two advanced concepts of rule learning.

1) In a very high-dimensional feature space, the amount of information contained in many alternative rules will be rather similar and the choice of a single rule will be somewhat random. Therefore, we adopt the paradigm of unordered rule learning where all applicable

rules are used for the purpose of classification. The resulting unordered rule sets also lead to a better understandability as a single rule in such rule sets can be understood in isolation. The understandability of the resulting rule sets will be discussed in section 4.3.

2) In classical rule learning, less and less training examples remain as the learning process proceeds, resulting in a high risk of overfitting. The trained SVMs can be exploited by our rule learner to alleviate this problem. The essential idea is as follows. A trained SVM will output a function value on every example. When SVMs are used for classification. The sign of this function value indicates the class label of this example (i.e. either positive class or negative class). The absolute value of this function value is called *confidence* which indicates the reliability of the prediction on this example. Classifiers that can produce such confidence value are called *confidence-rated* classifiers. The unordered rule sets can be easily modified into confidence-rated classifiers. A confidence which indicates the reliability of the prediction can be attached to each rule in the training process. The predictions of all applicable rules are combined in a way such that the predictions of more confident rules will play more important roles in the final decision. This final prediction is a value containing information on both the class label of the example and the confidence of this prediction, just like the function value output by SVMs. I will return to explaining this classification mechanism in section 4.3. To alleviate the problem we mentioned in the beginning of this paragraph, in the training process, we do not drop the positive class training examples until the unordered rules learned so far have a similar confidence in classifying this example as the SVM. In order to learn unordered rules approximating the confidences of an SVM, we adopt the method of function estimation by gradient boosting machine. The algorithm of approximating SVMs by the unordered rules will be discussed in section 4.2.

Secondly, the resulting unordered rule set provides both a global explanation of the entire model represented by the SVMs trained for the problem domain and an instance-based explanation of SVM's classification decision on a single example. All extracted rules collectively provide a global overview of the knowledge implied by the trained SVM classifiers. Collecting the meaning of every applicable rule on a particular example generates an instance-based explanation of the SVM classifiers.

Thirdly, the approach consists of two phases -a learning phase and a pruning phase, so that the tradeoff between understandability and classification accuracy is well controlled. In the learning phase, a rule set that maximally approximates the trained SVM is learned. In the pruning phase for some user-specified acceptable level of understandability (defined in

terms of rule set size), we want to obtain the most accurate model. This strategy allows the user to directly control the trade-off between understandability and classification accuracy. The learning phased is presented in section 4.2 and the pruning phase is presented in section 4.4.

Last but not least, the approach can be applied to a broad class of SVMs. Since the extracted rule sets approximate the prediction behavior of SVMs by approximating the SVMs' classification confidence on the training data, the rule learner queries the trained SVMs only one kind of information: the output function value (i.e the classification confidence) on each training data. The rule learner does not need to know the internal structures of the trained SVMs. Therefore the approach can extract understandable description from SVMs that use any kind of kernel functions.

The following sections provide a detail discussion of the approach.

4.2 Learning Phase

In this section, we will present the Boost Unordered Rule Learner algorithm that takes a trained SVM and the data on which it was trained as input, and produces a set of unordered rules that maximally approximate the prediction behavior of the SVM.

4.2.1 Boost Rule Learners

We approximate a trained SVM by a set of If-Then rules so that the classification confidence of the SVM and the If-Then rules are similar on the same training example. We achieve this goal by adopting the framework of gradient boosting machines. The input is a set of training examples $\{\vec{x}_i, y_i\}_{i=1}^N$, where $\{\vec{x}_i\}_{i=1}^N$ is exactly the same set of training examples on which the SVM was trained and $y_i \in R$ is the value (not the class label) output by the SVM classification function. The sign of y_i is the class label predicted by the SVM on example $\vec{x}_i$, and the absolute value of y_i indicates the confidence of this particular decision given by the SVM.

The estimator is a special additive function $\sum_{m=0}^{M} \beta_m \cdot h(\vec{x}; \vec{a}_m)$, where $h(\vec{x}; \vec{a}_m)$ is a basic learner, $\vec{a}_m$ is the learning parameters , β_m is the confidence and M is the boosting round. For better understandability, we choose a single If-Then (conjunctive) rule as the basic learner. Formally it is defined as:

$$r(\vec{x}) = h(\vec{x};\vec{a}) = \begin{cases} +1/-1 & \text{if the rule is applicable on example } \vec{x} \text{ and the majority class} \\ & \text{covered by this rule is the positive/negative class;} \\ 0 & \text{otherwise.} \end{cases} \tag{4.1}$$

where $\vec{a}$ denotes the antecedent of a rule.

At each boosting round, the basic learner and its corresponding confidence are chosen so that the loss function is minimized on the training examples.

for $m = 0$ **to** M **do**

$$\{\beta_m, \vec{a}_m\}_0^M = \underset{\{\beta_m', \vec{a}_m'\}}{\operatorname{argmin}} \sum_{i=1}^{N} L(y_i, F_{m-1}(\vec{x}_i) + \beta_m' \cdot h(\vec{x}_i; \vec{a}_m'))$$

$$F_m(\vec{x}) = F_{m-1}(\vec{x}) + \beta_m \cdot h(\vec{x}; \vec{a}_m)$$

end.

The loss function is defined as the absolute error, i.e. the absolute difference between the "correct" value as predicted by the SVM and the value predicted by the rules,i.e. $L(y, F(\vec{x})) = |y - F(\vec{x})|$. Using the "steepest-descent" method, the direction of the steepest-descent step at the training data $\vec{x}_i$ is

$$\tilde{y}_i = -[\frac{\partial L(y_i, F(\vec{x}_i))}{\partial F(\vec{x}_i)}]_{F(\vec{x})=F_{m-1}(\vec{x})} = sign(y_i - F(\vec{x}_i)).$$

where $y_i - F(\vec{x}_i)$ is termed "residuals".

The basic learner which is most parallel to the direction $\tilde{Y} \in \{\pm 1, 0\}_1^N$ is chosen at each boosting round. More specifically, we want to find the most accurate rule $r(\vec{x})$ that correctly labels $\vec{x}_i$ as the sign of the current residuals.

For a given rule $r_m(\vec{x})$, i.e. a fixed $\vec{a}_m$, its confidence β_m, i.e. the optimal length of the steepest-decent step is chosen to minimize the loss function.

$$\beta_m = \underset{\beta}{\operatorname{argmin}} \sum_{i=1}^{N} L(y_i, F_{m-1}(\vec{x}_i) + \beta \cdot r_m(\vec{x}_i)) \tag{4.2}$$

$$= \underset{\beta}{\operatorname{argmin}} \sum_{i=1}^{N} |y_i - F_{m-1}(\vec{x}_i) - \beta \cdot r_m(\vec{x}_i)| \tag{4.3}$$

$$= \underset{\beta}{\operatorname{argmin}} \sum_{\text{all } \vec{x}_i \text{ covered by rule } r_m} |y_i - F_{m-1}(\vec{x}_i) - \beta \cdot r_m(\vec{x}_i)| \tag{4.4}$$

$$= \underset{\beta}{\text{argmin}} \sum_{\text{all } \vec{x}_i \text{ covered by rule } r_m} |r_m(\vec{x}_i)| |\frac{y_i - F_{m-1}(\vec{x}_i)}{r_m(\vec{x}_i)} - \beta| \quad (4.5)$$

$$= \underset{\beta}{\text{argmin}} \sum_{\text{all } \vec{x}_i \text{ covered by rule } r_m} |\frac{y_i - F_{m-1}(\vec{x}_i)}{r_m(\vec{x}_i)} - \beta| \quad (4.6)$$

$$= \text{Median}\{\frac{y_i - F_{m-1}(\vec{x}_i)}{r_m(\vec{x}_i)} : \text{all } \vec{x}_i \text{ covered by rule } r_m\} \quad (4.7)$$

Equation 4.7 is proved by the following theorem.

Theorem 1 $\sum_{i=1}^{N} |w_i - d|$ *is minimized for* $d = d^*$ *whenever* d^* *is a median of* $\{w_i\}_{i=1}^{N}$.

Proof. (A similar proof can be found at M H De Groot, *Optimal Statistical Decisions*, McGraw-Hill 1970, section 11.4, Theorem 1, p. 232).

Let d^* be any median of $\{w_i\}_{i=1}^{N}$ and let d be any other number such that $d > d^*$. Then

$$|w_i - d| - |w_i - d^*| = \begin{cases} d^* - d & (w_i \geq d > d^*) \\ d + d^* - 2w_i & (d > w_i > d^*) \\ d - d^* & (d > d^* \geq w_i) \end{cases} \quad (4.8)$$

Since when $d > w_i > d^*$, so that $-2w_i \geq -2d$, we have $d + d^* - 2w_i > d^* - d$, and thus

$\sum_{i=1}^{N}\{|w_i - d| - |w_i - d^*|\} \geq (d^* - d)|\{w_i | w_i \leq d, 1 \leq i \leq N\}| + (d^* - d)|\{w_i | d > w_i > d^*, 1 \leq i \leq N\}| + (d - d^*)|\{w_i | w_i \leq d^*, 1 \leq i \leq N\}|$

with equality if and only of $|\{w_i | d > w_i > d^*, 1 \leq i \leq N\}| = 0$, which can only happen if and only if d is a median of $\{w_i\}_1^N$. We conclude that

$\sum_{i=1}^{N}\{|w_i - d| - |w_i - d^*|\} \geq (d - d^*)\{|\{w_i | w_i \leq d^*, 1 \leq i \leq N\}| - |\{w_i | w_i > d^*, 1 \leq i \leq N\}| \geq 0$ as d^* is a median of $\{w_i\}_1^N$. Hence

$$\sum_{i=1}^{N} |w_i - d| \geq \sum_{i=1}^{N} |w_i - d^*|$$

and there can be equality if and only if d is also a median. The case where $d < d^*$ can be dealt with similarly. □

Our algorithm specializing the gradient boosting method for approximating is presented in figure 4.1. Note that at step 7, we could adopt any existing rule learning method, e.g. the beam search strategy described in figure 2.3, to learn a single most accurate rule. The classification is given as $sign(\sum_{m=0}^{M} \beta_m \cdot r_m(\vec{x}))$.

Algorithm 4.1: BOOST_RULE_LEARNER

Input: training examples $\{\vec{x}_i, y_i\}_{i=1}^{N}$, where y_i is the value predicted by the SVM on the training example $\vec{x}_i$; number of iterations M.

Output: $F(\vec{x}) = \sum_{m=0}^{M} \beta_m \cdot r_m(\vec{x})$, where $r_m(\vec{x})$ is the function of a conjunctive rule defined as formula 4.1, and β_m is the confidence of the corresponding rule.

begin

1 $m \leftarrow 0$, $F_m(\vec{x}) \leftarrow 0$

2 **for** $m \leftarrow 1$ ***to*** M **do**

3 **foreach** $\vec{x}_i \in \vec{X}$ **do**

4 $\tilde{y}_i \leftarrow sign(y_i - F_{m-1}(\vec{x}_i))$

end

5 $\vec{X} \leftarrow \vec{X} - \{\vec{x}_i | \tilde{y}_i = 0\}$

6 $Y \leftarrow \{\tilde{y}_i | \vec{x}_i \in \vec{X}\}$

7 $r_m \leftarrow$ LEARN_ONE_RULE($\vec{X}$,Y)

8 $\beta_m \leftarrow$ Median$\{\frac{y_i - F_{m-1}(\vec{x}_i)}{r_m(\vec{x}_i)}$: all $\vec{x}_i$ covered by rule $r_m\}$

9 $F_m(\vec{x}) \leftarrow F_{m-1}(\vec{x}) + \beta_m \cdot r_m(\vec{x})$

end

end

Figure 4.1: Boost rule learner algorithm.

4.2.2 Improving Boost Rule Learners by Using Unordered Rules

However, there exists a problem in this simple method which is illustrated by the example in figure 4.2. A linear SVM is trained to distinguish rectangular objects (the positive class) from round objects (the negative class). Initially $F_0 = 0$, i.e. the initial rule learner estimates the SVM confidence on each example as 0. The signs of current residuals (i.e. the direction of the steepest-decent step) at each training data are shown in figure 4.2(a).We obtain the first most accurate rule $R1$ which correctly labels the residual signs of 5 rectangular objects. The confidence of rule $R1$ is set to the residual of the middle rectangle (the median) covered by Rule $R1$. After F is updated, the signs of the current residuals for the leftmost two rectangles covered by rule $R1$ become negative since they are closer to the separating hyper plane than the median of $R1$ and thus are assigned smaller values by the SVM. This is shown in 4.2(b). At the second round, the most accurate rule is $R2$, which correctly labels the residual signs of three objects. The problem with rule $R2$ is that it looks like "If an object can be covered by $R2$, then it's a round object.", but it covers more rectangular (positive class) than round (negative class) objects.

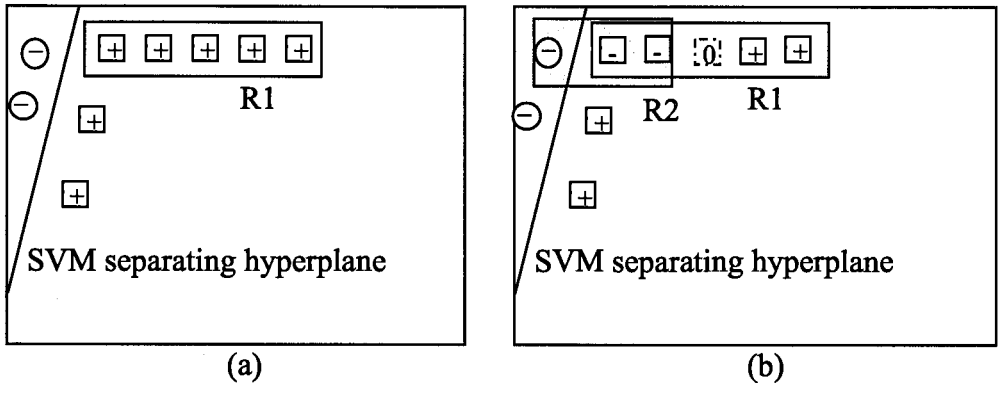


Figure 4.2: An example of boost rule learner algorithm

To remedy this problem, we borrow an idea from CN2 and generate unordered rules for each class in turn, i.e. first for the positive and then for the negative class. This is equivalent to only searching for the steepest positive/negative direction (thus only increasing /decreasing the value of F on the training examples) towards the approximation of SVMs at each turn. The advanced version of our algorithm is called "Boost Unordered Rule Learner"(BUR) and is presented in figure 4.3.

The algorithm works as follows: when it learns unordered rules for the positive class, the steepest direction to increase the value of F at each step is along every positive training example whose residuals > 0. Those positive training examples whose residuals ≤ 0 have been classified by the rules extracted as far in at least the same confidence as the SVM's.Thus we simply drop them from the current training examples (line 9). As in CN2, all examples in the negative class are kept (line 10), because each rule must independently stand against all negative examples. Then the optimal length of the step towards the approximation is found (line14) and the F is updated (line 15). A similar process proceeds to learn rules for the negative class.

4.2.3 Convergence of BUR

Unlike the algorithm presented in figure 4.1, there is no input parameter to restrict number of iterations in the BUR algorithm. However, we claim that the BUR algorithm will terminate in the end and the boost process will converge.This is informally proven as follows. In the process of learning rules for the positive class, every time a new rule is found, the confidence of this rule is chosen to be the median residual of all training examples covered. Half of the

Algorithm 4.2: BOOST_UNORDERED_RULE

Input: training examples $\{\vec{x}_i, y_i\}_{i=1}^{N}$, where y_i is the value predicted by the SVM on the training example $\vec{x}_i$.

Output: $F(\vec{x}) = \sum_{m=0}^{M} \beta_m \cdot r_m(\vec{x})$, where $r_m(\vec{x})$ is the function of a conjunctive rule defined as formula 4.1, β_m is the confidence of the corresponding rule, M is the total number of unordered rules discovered by this algorithm.

begin

1 $m \leftarrow 0$, $F_m(\vec{x}) \leftarrow 0$
2 $m \leftarrow m + 1$
3 **foreach** *class* $C \in \{C_{pos}$, $C_{neg}\}$ **do**
4 $\vec{X}_c \leftarrow$ {all training examples in class C}
5 $\vec{X}_o \leftarrow$ {all training examples in the other class}
6 **repeat**
7 **foreach** $\vec{x}_i \in \vec{X}_c$ **do**
8 $\tilde{y}_i \leftarrow sign(y_i - F_{m-1}(\vec{x}_i))$
end
9 $\vec{X}_c \leftarrow \vec{X}_c - \{\vec{x}_i | \vec{x}_i \in \vec{X}_c \wedge \tilde{y}_i \neq sign(y_i)\}$
10 $\vec{X}' \leftarrow \vec{X}_c \cup \vec{X}_o$
11 $Y' \leftarrow \{sign(y_i) | \vec{x}_i \in \vec{X}\}$
12 $r_m \leftarrow$ LEARN_ONE_RULE($\vec{X}'$,Y',C)
13 **if** r_m *is not null* **then**
14 $\beta_m \leftarrow$ Median$\{\frac{y_i - F_{m-1}(\vec{x}_i)}{r_m(\vec{x}_i)}$: all $\vec{x}_i$ covered by rule $r_m\}$
15 $F_m(\vec{x}) \leftarrow F_{m-1}(\vec{x}) + \beta_m \cdot r_m(\vec{x})$
16 $m \leftarrow m + 1$
end
until r_m *is null*
end
17 $M = m - 1$
end

Figure 4.3: Boost Unordered Rules algorithm

positive examples (whose residuals are smaller than the median's) covered by this rule will be removed since the unordered rules learned so far have at least the same confidence in classifying these examples as the SVM. Thus as the learning process proceeds, less and less training examples in the positive class are left, which leads to the termination of the search for rules for the positive class. The same argument stands for the process of learning rules for the negative class. Therefore the BUR algorithm will terminate in the end.

4.3 The Format of the Resulting Rules

To demonstrate the format of rules discovered by our method and their interpretation, we consider the following example that shows a small part of a set of unordered rules. The rule set is a result of approximating an SVM that predicts if a protein is an outer membrane protein from its amino acid sequence, which is one of the experiments in section 5.

" ...

If SLGG = no and QY= yes and QSQ = yes
Then class = outer_membrane [0.699955]

If QS = yes and FW =yes
Then class = non_outer_membrane [-0.999658]

...

Default Class = non_outer_membrane"

The rules indicate that absence of subsequence "SLGG" and presence of subsequences "QY" and "QSQ" in a protein sequence is a predictor of outer membrane proteins, while presence of subsequences "QS" and "FW" in a protein sequence predicts the protein is not an outer membrane protein. Furthermore, the absolute values of the weights associated with rules indicate the confidence for the corresponding decisions. In this case, the second rule has higher confidence than the first rule.

An important argument for the understandability of the unordered rules rests on the fact that the contribution of each rule can be understood in isolation. Summing these contributions generates the prediction/classification. This means that if we test the antecedent of every rule in a set of unordered rules, we accumulate evidence for or against the target

concept as we proceed. If none of the rules in the rule set is applicable, a default rule assigns the example to the majority class. Otherwise the absolute value of the sum can be understood as a measure of confidence of the classification. After gleaning the meaning of each rule in isolation, we can approximately understand the reasoning behind the SVMs decision.

4.4 Pruning Phase

The objective of algorithm Boost Unordered Rule Learner is to find unordered rules that optimally approximate SVMs. However, the level of understandability of the resulting unordered rules may not necessarily satisfy the user's requirements. The understandability of a model is often measured in the model size. In the pruning phase, the size of the unordered rules is trimmed to a user defined limit while maintaining accuracy as much as possible.

We measure the size of a rule set by the number of rules and the number of features involved in each rule. For unordered rules, each rule must independently stand against all examples in the other class and will have a similar level of abstraction. Thus all rules have similar size in terms of number of features involved, which is also confirmed by our experimental evaluation in section 5. We also observe that the average number of features involved in the unordered rules is around 2-3. A single rule of this size is comprehensible to a human expert. Therefore, the pruning method focuses on controlling the number of rules.

Since the number of unordered rules is usually too large to test every possible combination of rules, we apply a greedy strategy. In essence, the rule that has the least effect on the overall performance of the current rule set is removed. Then we repeat evaluating the resulting smaller rule set until the size of the rule set meets the user's understandability requirement.

Typically, the performance of a classifier is estimated using a separate test data set or using only the training data set and adjusting the raw error estimate by a statistical correction to reflect the bias [30]. Since in most application domains labeled data is not abundant, we adopt the latter approach and evaluate the performance in the pruning phase by the apparent error rate on the training data set. For large rule sets, the overall error rate may not change if only a single rule is eliminated. To break ties, a second measure based on the loss function presented in section 4.2.1 is introduced. Our pruning algorithm

Algorithm 4.3: PRUNE_UNORDERED_RULES

Input: training examples $\{\vec{x}_i, y_i\}_{i=1}^{N}$, where y_i is the value predicted by the SVM on the training example $\vec{x}_i$; a set of unordered rules and their corresponding confidences discovered by algorithm BUR $\{r_m; \beta_m\}_{m=0}^{M}$; desirable number of rules S.

Output: a set of unordered rules and their corresponding confidence $\{r_m; \beta_m\}_{m=0}^{S-1}$.

begin

1 $R' \leftarrow \{r_m; \beta_m\}_{m=0}^{M}$

2 **while** $|R'| > S$ **do**

3 $\{r; \beta\} \leftarrow$ LEAST_SIGNIFICANT_RULE$(R', \vec{X}, Y)$

4 $R' \leftarrow R' - \{r; \beta\}$

end

5 **return** R'

end

Algorithm 4.4: LEAST_SIGNIFICANT_RULE

Input: training examples $\{\vec{x}_i, y_i\}_{i=1}^{N}$, where y_i is the value predicted by the SVM on the training example $\vec{x}_i$; a set of unordered rules and their corresponding confidences $\{r_m; \beta_m\}_{m=0}^{L}$.

Output: Rule r that has the least effect on the overall performance of $\{r_m; \beta_m\}_{m=0}^{L}$.

begin

1 $R \leftarrow \{r_m; \beta_m\}_{m=0}^{L}$

2 $F(\vec{X}) \leftarrow \sum_{m=0}^{L} \beta_m \cdot r_m(\vec{X})$

3 **for** $m \leftarrow 0$ ***to*** L **do**

4 $F(\vec{X})' \leftarrow F(\vec{X}) - \beta_m \cdot r_m(\vec{X})$

5 LossAccuracy[m] $\leftarrow$ ESTIMATED_ERROR$(F(\vec{X})', \vec{X}, Y)$- ESTIMATED_ERROR$(F(\vec{X}), \vec{X}, Y)$

6 LossResidual[m] $\leftarrow \sum_{i=1}^{N} |F'(\vec{x}_i) - y_i)| - \sum_{i=1}^{N} |F(\vec{x}) - y_i|$

end

7 find the least significant rule(s) with the smallest LossAccuracy, and break a possible tie by returning the rule r_i with the smallest LossResidual

8 **return** $\{r_i; \beta_i\}$

end

Figure 4.4: Pruning algorithm

is presented in figure 4.4.

Chapter 5

Experimental Evaluation

5.1 Experiment Design

We evaluated the quality of extracted unordered rules in terms of classification quality and understandability. Decision trees, rule induction learners and SVMs were chosen as competitors of our proposed algorithm. Classification quality is evaluated in *recall* and *precision* for the target class and overall *accuracy*, which have been defined in section 2.1. Understandability is evaluated in the size of the learned models. The C4.5 [3] implementation of decision trees and SVM^{light}[2] implementation of SVMs were chosen because they are well known and have been used extensively in previous research. For rule induction learners, we chose the CN2 implementation [4] which also learns a set of unordered rules. This is to study separately the consequences of exploiting unordered rules and SVM confidence in our proposed algorithm. We adopted the same procedure of learning a single rule in [4] as in our algorithm for the sake of fairness. Default learning parameters were used in C4.5, CN2 and SVM^{light}. In all experiments we performed 5-fold cross validation.

5.2 Dataset Description

For testing the effectiveness of our algorithm in high dimensional feature space, we performed experiments on two such application domains: biology sequence classification and text classification. The protein dataset was produced by the Department of Molecular Biology and Biochemistry at Simon Fraser University for tackling the outer membrane protein (OMP) identification problem [32]. Outer membrane proteins- which are exposed on the

surface of the cell- represent potential drug and vaccine targets, and the ability to identify such potential targets from sequence information alone would allow researchers to quickly prioritize a list of proteins for further study. [32] studies several learning methods on this challenging dataset, wherein SVMs outperform all other competing methods significantly, and by far the best result for OMP classification that has been reported. Therefore this dataset provides a good chance to evaluate the classification quality of the extracted rule sets.

The text dataset used in our experiments is bundled with SVM^{light} and can be downloaded from [2].The task is to learn which Reuter's articles are about "corporate acquisitions". This dataset is chosen because of its public accessibility. Also it is part of the Reuters dataset which has been widely used in text classification research.

The important properties of these two datasets are shown in table 5.1.

Table 5.1: Description of the evaluation datasets. "Exs" stands for "Examples".

DataSet	#. Exs. In Target/Contrasting Class	Feature Type	#. Features
OMP	427/1132	binary	1458
Reuters	1300/1300	numeric	9947

5.3 Experimental Results

The first group of experiments investigates the quality of rule sets produced by the BUR algorithms without pruning. Table 5.2 and table 5.3 show the classification quality and model size of the classifiers constructed by our method BUR, and its competitors on the two datasets. The classification quality is measured in terms of recall and precision of the target class and overall accuracy. The number of nodes in a decision tree is used to describe the size of the tree. The size of a rule set is defined in terms of number of rules and average number of features (components) involved in the antecedent of each rule. Note that the sets of unordered rules include both rules for the target class and rules for the contrasting class. Thus we also count the total number of components in the rule sets to make their size comparable with decision trees.

For all classification quality measures, the ones achieved by our method are within 4%

Table 5.2: Classification quality and classifier size on OMP dataset

	C4.5	CN2	BUR	SVMs
Recall(%)	65	56	65	81
Precision(%)	70	90	89	88
Accuracy(%)	83	86	88	92
# Rules	-	62	116	-
Ave. # Components	-	3.0	3.0	-
Total # Components	134	183	358	-

Table 5.3: Classification quality and classifier size on Reuters dataset

	C4.5	CN2	BUR	SVMs
Recall(%)	91	93	95	97
Precision(%)	89	91	94	96
Accuracy(%)	89	92	95	97
# Rules	-	140	265	-
Ave. # Components	-	1.8	1.9	-
Total # Components	160	249	498	-

from the ones achieved by SVMs, except the recall on the OMP dataset. BUR consistently outperforms C4.5 on both datasets and only loses to CN2 1% in precision on the OMP dataset while winning in all other cases.

We also observe that CN2 and our method achieve much higher precision than C4.5 on both datasets, in particular on the OMP dataset, due to the fact that each rule in the set of unordered rules stand against all examples in the other class. Comparing CN2 and BUR, BUR achieves much higher recall while still preserving very high precision. Since we do not drop the target class training examples until the unordered rules learned so far have a similar confidence in classifying these examples as the SVM, more useful rules are discovered by BUR and the overfitting problem caused by lack of training examples as the rule induction process proceeds is alleviated.

Both CN2 and our method construct more complicated classifiers than C4.5 as a tradeoff for higher classification quality. However, on both datasets, the average number of components in each rule is ≤ 3, i.e. a single rule is easy for human experts to understand.

We conclude that 1) Unordered rules perform very well in high dimensional feature spaces. 2) By approximating the SVM's classification confidence, the classification quality of unordered rules is improved.

The second groups of experiments compare BUR with another two rule extractors TREPAN and BRL.

As discussed in section 3.1.2, TREPAN can be directly applied to the problem of interpreting SVMs by replacing neural network oracles with trained SVM oracles. In this way we tried to compare BUR with TREPAN. The implementation of TREPAN can be downloaded from [1]. TREPAN builds a decision tree by using both the supplied training examples and randomly generated examples whose labels are predicted by the oracle. At each node to be expanded, TREPAN estimates a local sample distribution function, compares it with the global sample distribution function and chooses one that is more statistically significant. Then the chosen distribution function is used to generate new examples. With the growth of the decision tree, less and less supplied training examples remain and more and more examples need to be generated in order to keep a sufficient number of examples for finding a test. Both datasets used in our experimental evaluation involve very high dimensional feature spaces, which results in sparsely distributed training examples. A significant number of examples were generated at each node. Therefore the learning process of TREPAN was very slow. In the five-fold cross validation experiments on the OMP dataset, the experiment

of just one fold had not finished in two days on an 866MHz Pentium PC machine with 512M main memory, running Microsoft Windows XP. Consequently, the experiments of comparing BUR with TREPAN had not been carried out because of excessive time.

Table 5.4 shows the comparison results between BRL and BUR on OMP and Reuters datasets. Since we wanted to compare the classification quality of the similar size rule sets extracted by BRL and BUR respectively, we set the number of boosting rounds in BRL to the number of rules in the rule set constructed by BUR. The results demonstrate that BUR significantly outperforms BRL in all classification quality measures on both datasets. Since BRL keeps learning new rules until the resulting rule sets have exactly the same confidence in classifying the training examples as the SVMs, it is very likely that it will generate overfitting rules. This may lead to the inferior classification performance of BRL.

Table 5.4: Comparision of classification quality and classifier size of rule sets constructed by BRL and BUR.

	OMP		Reuters	
	BRL	BUR	BRL	BUR
Recall(%)	54	65	90	95
Precision(%)	66	89	82	94
Accuracy(%)	80	88	85	95
# Rules	116	116	265	265
Ave. # Components	3.2	3.0	2.1	1.9
Total # Components	384	358	564	498

We also conduct experiments of pruning the rule sets produced by BUR to observe the variation of classification quality. In particular, we prune the rule sets so that their size is similar to that of the classifiers produced by CN2 and C4.5.

Figure 5.1 and 5.2 show the classification quality at various rule set size. The classifier size refers to the total number of components in the rule sets. The results show that all classification qualities are fairly stable with substantial decrease of size (total number of components). Especially the accuracy only drops no more than 1% on both datasets when the rule sets are pruned to half of original size.

Table 5.5 and table 5.6 compare the extracted rule sets with the classifiers produced by CN2 and C4.5 at the similar size. The pruned rule sets are presented as "Pruned BUR",

followed by their size. For example, the item "Pruned BUR(182)" in table 5.5 means the pruned rule set totally contain 182 components and is compared to the rule set produced by CN2, which totally contain 183 components.

Compared with the unordered rule sets at the same size, the rule sets extracted from SVMs still preserve the benefit of higher recall and higher accuracy. Compared with decision trees at the same size, the rule sets extracted from SVMs achieve higher accuracy and precision while achieve at least the same level of recall.

To conclude, the accuracy of the pruned boosted unordered rules is very robust to the decrease of number of rules and BUR achieves higher accuracy compared with the similarly complicated decision trees and unordered rules generated by CN2.

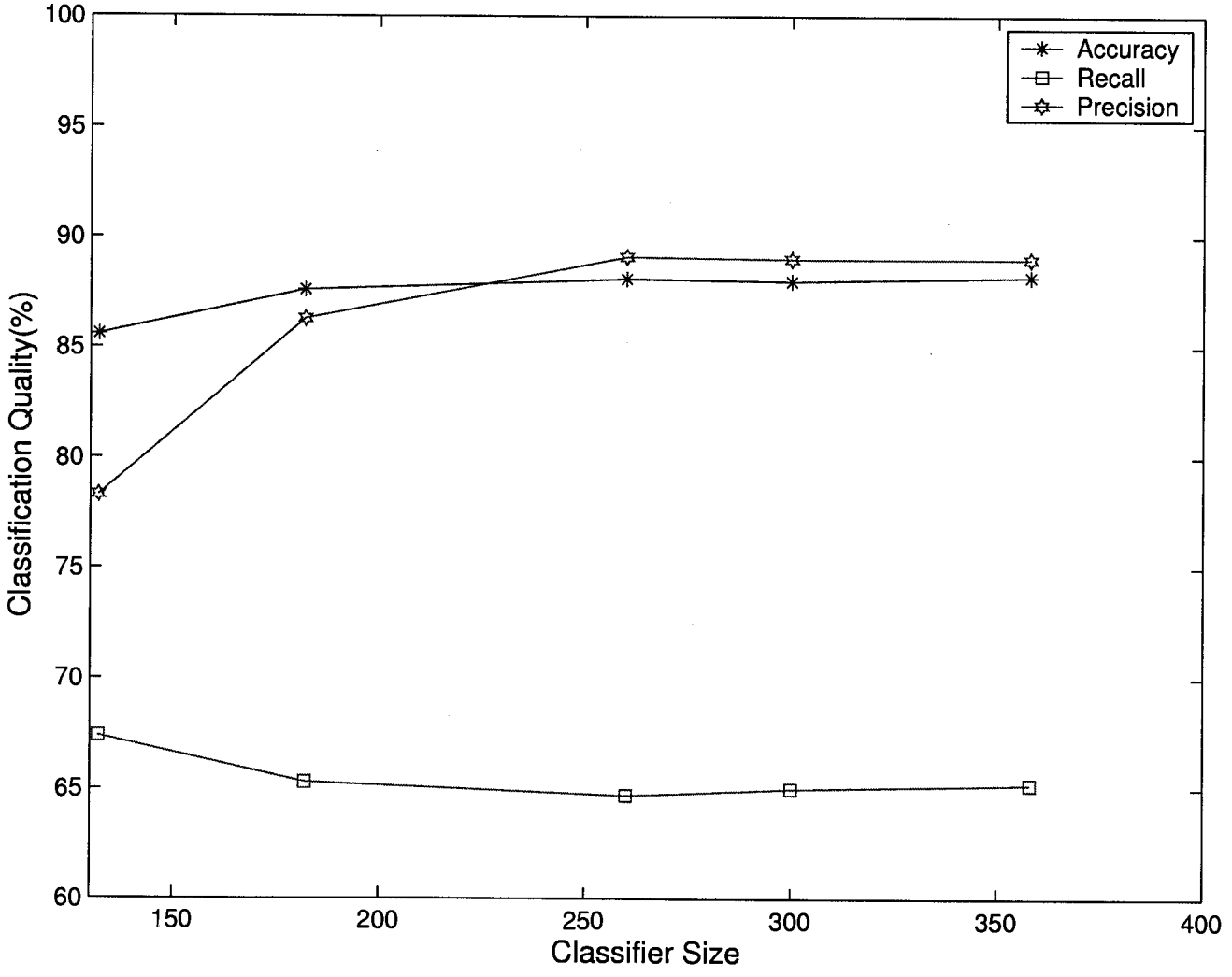


Figure 5.1: Classifier size and classification quality of the pruned rule set on OMP dataset.

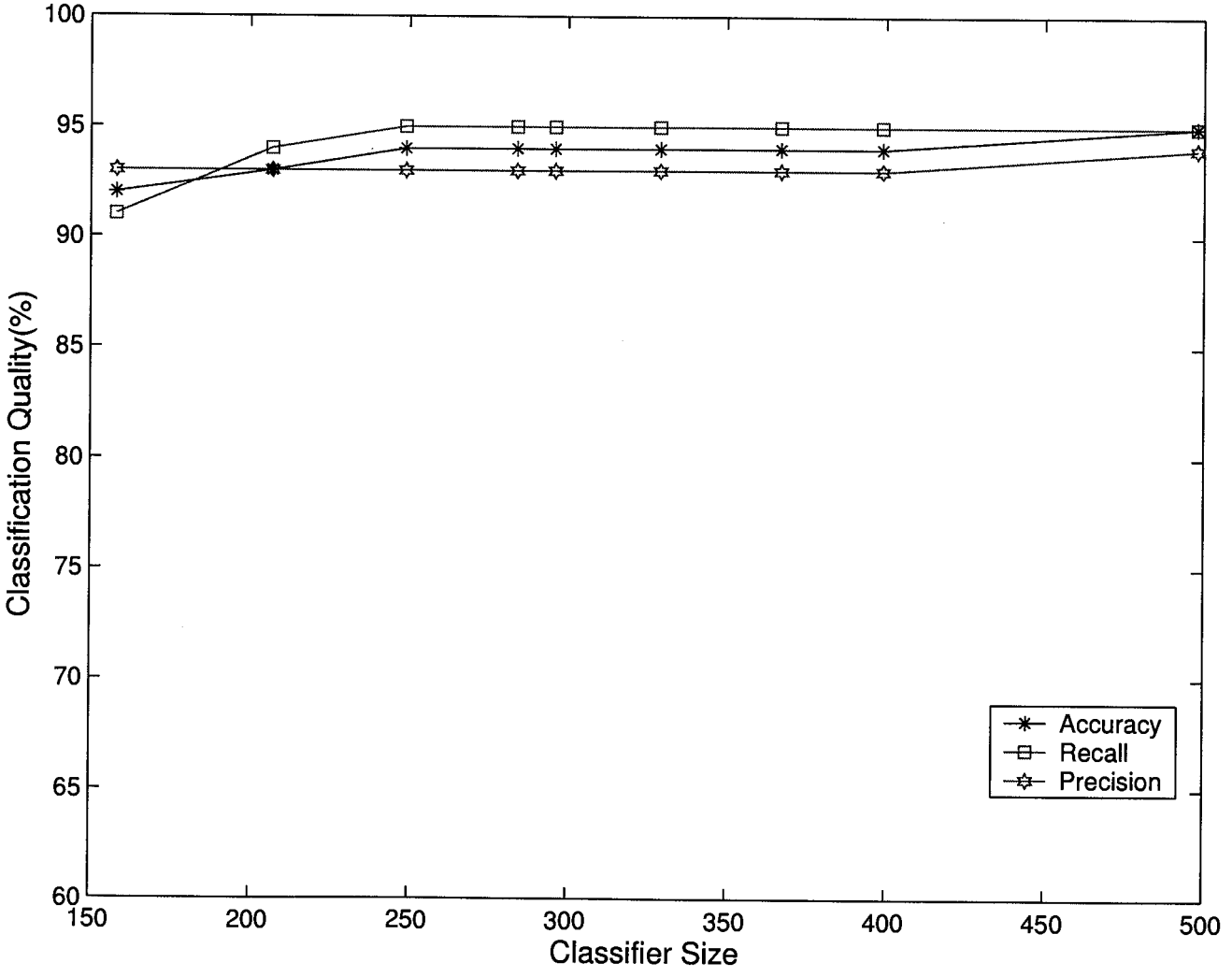


Figure 5.2: Classifier size and classification quality of the pruned rule set on Reuters dataset.

Table 5.5: Comparison of classification quality of similar size classifers on OMP dataset

	Pruned BUR(182)	CN2(183)	Pruned BUR(132)	C4.5(134)
Recall(%)	65	56	67	65
Precision(%)	86	90	78	70
Accuracy(%)	88	86	86	83
# Rules	62	62	45	–
Ave. # Components	3.1	3.0	2.9	–

Table 5.6: Comparison of classification quality of similar size classifers on Reuters dataset

	Pruned BUR(252)	CN2(249)	Pruned BUR(158)	C4.5(160)
Recall(%)	95	93	91	91
Precision(%)	93	91	93	89
Accuracy(%)	94	92	92	89
# Rules	141	140	88	–
Ave. # Components	1.8	1.8	1.8	–

Chapter 6

Conclusion

One of the main challenges in SVMs for data mining applications is to obtain explicit knowledge from the trained SVMs for explaining classification decisions. In this thesis, we address the problem of extracting understandable yet accurate rules from SVM classifiers in the scenario of high dimensional feature spaces and propose a two phase approach to produce a set of unordered If-Then rules that is within user supplied understandability limit while maximizing accuracy. In this chapter, we summarize the thesis and discuss future research directions.

6.1 Contributions

The contributions of this thesis are summarized as follows:

- We propose a novel algorithm to learn a set of If-Then rules to approximate the prediction behavior of the trained SVMs. In order to overcome the weaknesses of standard rule learners in high dimensional feature spaces, we adopt two advanced concepts of rule learning: the paradigm of unordered rule learning where all applicable rules are used for purpose of classification and forcing the extracted rule sets to classify the training examples in a confidence similar to the trained SVM's .

- The resulting rule set is a set of unordered rules, where each rule gets a confidence score derived directly from the SVM by a variant of gradient boosting machines. The contribution of each rule can be understood in isolation. After gleaning the meaning of every applicable rule in isolation during classification, we can approximately

understand the reasoning behind the SVMs decision on the example.

- Our approach consists of two phases - a learning phase and a pruning phase, which allows users to control the size of the extracted rule sets. In the learning phase, a rule set maximally approximating the trained SVM is learned. In the pruning phase the rule set is trimmed to a user desirable size while maximizing the classification accuracy. Thus the user can directly control the tradeoff between classification accuracy and understandability of the model.

- Experimental evaluation is conducted in two typical high dimensional application domains: biology sequence classification and text classification. On both classification tasks, the classification accuracy of the rule sets extracted by BUR is within 4% from the one gained by SVMs and much higher than the one achieved by decision trees and unordered rules learned by traditional rule induction methods. After applying the pruning strategies, the resultant rule sets are at the same level of understandability as decision trees or rule sets generated by CN2 while consistently achieving higher accuracy than the competitors of the same level of understandability.

6.2 Future Research Directions

Future extension of this study could be explored in the following directions:

- In this thesis, we have conducted several experiments to evaluate the performance of the proposed algorithm. Further research may theoretically investigate our proposed algorithm in depth. Such issues could include the lower bound and upper bound of loss in approximation on the training data set and the generalization of the extracted rule sets on unseen data. For example, the generalization of the exacted rule sets can be evaluated based on the difference between a learned distribution over samples by the original learners and the approximations. Various probability distribution measures of comparison include relative entropy, KL-distance and cross-entropy.

- Our proposed approach learns understandable rule sets that approximate the prediction behaviors of trained SVMs by approximating SVM's classification confidence on the training data. We could use the same strategy to approximate any confidence-rated classifiers, for example classifiers constructed by a boosting algorithm. In particular,

our approach is very promising to be applied to extracting understandable rules from these confidence-rated classifiers in the scenario of high dimensional feature spaces.

- Our proposed approach views the problem of extracting understandable yet accurate rules from trained SVMs as a learning task. However our learning task differs from the traditional learning tasks in several aspects. First, in the traditional setting for learning task, both the data distribution and the target concept are unknown. In contrast, the target concept is explicitly expressed in the rule extraction problem: the learned SVMs. Second, traditional learning algorithms and related theories focus on the generalization of the learning models, i.e. the classification quality on unseen data. But for rule extraction task, we not only consider the generalization of the extracted models, but also its understandability. How to integrate both goals into the search process of the learner? Is the rule extraction task easier than traditional learning tasks because of the first difference or is it more difficult than traditional learning tasks because of the second difference? Previous work [23][6] indicate that the knowledge of target concepts could make the learning problem easier. However, few theories have developed to address the understandability of a model. Thus there seems no direct answer to the question of how to integrate the understandability and generalization of a model. We believe it is crucial to develop theoretical models to further study these questions.

Bibliography

[1] *http://www.cmd.port.ac.uk/cmd/software.shtml.*

[2] *http://www.cs.cornell.edu/People/tj/svm_light/.*

[3] *http://www.cse.unsw.edu.au/~quinlan/.*

[4] *http://www.cs.utexas.edu/users/pclark/software.html.*

[5] Charu C. Aggarwal, Cecilia Magdalena Procopiuc, Joel L. Wolf, Philip S. Yu, and Jong Soo Park. Fast algorithms for projected clustering. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, pages 61–72. ACM Press, 1999.

[6] Endre Boros, Peter L. Hammer, Toshihide Ibaraki, and Kazuhiko Kawakami. Polynomial-time recognition of 2-monotonic positive boolean functions given by an oracle. *SIAM Journal on Computing*, 26:93–109, 1997.

[7] Paul S. Bradley and O. L. Mangasarian. Feature selection via concave minimization and support vector machines. In *Proceedings of the Fifteenth International Conference on Machine Learning (ICML 1998)*, pages 82–90. Morgan Kaufmann, 1998.

[8] L. Breiman. *Heuristics of instability in model selection.* Technique Report. Statistics Department,University of California at Berkeley, 1994.

[9] L. Breiman. Bagging predictors. *Machine Learning*, 24:123–140, 1996.

[10] P. Agarwal C. M. Procopiuc, M. Jones and T.M. Murali. A monte carlo algorithm for fast projective clustering. In *Proceedings ACM SIGMOD International Conference on Management of Data*, pages 418–427. ACM Press, 2002.

[11] P. Clark and R. Boswell. Rule induction with cn2: some recent improvements. In Yves Kodratoff, editor, *EWSL*, volume 482, pages 151–163. Springer, 1991.

[12] P. Clark and T. Niblett. The cn2 induction algorithm. *Machine Learning*, 3:261–283, 1989.

[13] William W. Cohen and Yoram Singer. Context-sensitive learning methods for text categorization. In *sigir 1996, Proceedings of the 19th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 307–315. ACM Press, 1996.

[14] Usama M. Fayyad, Gregory Piatetsky-Shapiro, Padhraic Smyth, and Ramasamy Uthurusamy. *Advances in Knowledge Discovery and Data Mining.* AAAI/MIT Press, 1996.

[15] Yoav Freund. Boosting a weak learning algorithm by majority. *Information and Computation*, 121:256 – 285, 1995.

[16] J.H Friedman. An overview of predictive learning and function approximation. In *Proceedings of NATO/ASI Workshop*, pages 1–61. Springer Verlag, 1994.

[17] J.H Friedman. Greedy function approximation: a gradient boosting machine. In *Technical Report.* Department of statistics,Stanford University, 1999.

[18] S.I. Gallant. *Neural network learning and expert systems.* MIT Press, 1993.

[19] Jiawei Han and Micheline Kamber. *Data Mining: Concepts and Techniques.* Morgan Kaufmann, 2000.

[20] J.C. Jackson and M.W.Craven. Learning sparse perceptrons. *Advances in Neural Information Processing Systems (NIPS)*, 8, 1996.

[21] T. Joachims. *Learning to Classify Text Using Support Vector Machines.* PhD thesis, Kluwer, 2002.

[22] Thorsten Joachims. Text categorization with support vector machines: Learning with many relevant features. In *Proceedings of the European Conference on Machine Learning*, pages 137–142. Springer, 1998.

[23] lya Shmulevich, Aleksey D. Korshunov, and Jaakko Astola. Almost all monotone boolean functions are polynomially learnable using membership queries. *Information Processing Letters*, 79:211 – 213, 2001.

[24] Tom M. Mitchell. *Machine learning.* McGraw-Hill, 1997.

[25] M.W.Craven. *Extracting Comprehensible Models from Trained Neural Networks.* PhD thesis, Department of Computer Science,University of Wisconsin-Madison, 1996.

[26] M.W.Craven and J. Shavlik. Using neural networks for data mining. *Future Generation Computer Systems*, 13:211–229, 1997.

[27] H. Núnez, C. Angulo, and A. Catalá. Rule extraction from support vector machines. In *Proceedings of European Symposium on Artificial Neural Networks*, pages 107–112, 2002.

[28] E. Osuna and R. I'eund nd F. Girosi. Training support vector machines: An application to face detection. In *Proceedings of EEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 130–136. IEEE Computer Society, 1997.

[29] Dean Pomerleau. Knowledge-based training of artificial neural networks for autonomous robot driving. In J. Connell and S. Mahadevan, editors, *Robot Learning*. 1993.

[30] J. Ross Quinlan. *C4.5: programs for machine learning*. Machine Learning. Morgan Kaufmann, 1993.

[31] K. Saito and R. Nakano. Medical diagnostic expert system based on pdp model. In *Proceedings of IEEE International Conference on Neural Networks*, pages 255–262. IEEE Press, 1988.

[32] R. She, F. Chen, K. Wang, M. Ester, J.L.Gardy, and F.S.L. Brinkman. Frequent-subsequence-based prediction of outer membrane proteins. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 436–445. ACM Press, 2003.

[33] S. Thrun. Extracting rules from artificial neural networks with distributed representations. *Advances in Neural Information Processing Systems (NIPS)*, 7, 1995.

[34] Vapnik V. *The Nature of Statistical Learning Theory*. Springer Verlag, 1995.