FEATURE SELECTION AND DISCRIMINANT ANALYSIS IN DATA MINING

By

EUN SEOG YOUN

A DISSERTATION PRESENTED TO THE GRADUATE SCHOOL OF THE UNIVERSITY OF FLORIDA IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF DOCTOR OF PHILOSOPHY

UNIVERSITY OF FLORIDA

2004
This dissertation is dedicated to my parents, Jaebyung Youn and Guisoon Yoo.
ACKNOWLEDGMENTS

I would like to thank all five professors, Drs. Li Min Fu, Stanley Su, Su-Shing Chen, Anand Rangarajan, and Mark Yang, who served on my dissertation committee. Their comments and suggestions were invaluable. My special thanks go to Dr. Li Min Fu for being my supervisor and introducing me to this area of research. He provided ideas, financial support, assistance, and active encouragement over the years.

I would like to thank my parents, Jaebyung Youn and Guisoon Yoo, for their spiritual and financial support, and brother-in-law, Eungkil, for his thoughts and prayers. I would like to thank my wife, Jongjin, and my son, Sungho, for their continuous encouragement and prayers. I apologize for not being able to spend more time with them while I was pursuing my graduate studies. They always have been a great joy for me.

At last, but not least, I thank God for allowing me the joy of writing this dissertation.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ACKNOWLEDGMENTS</td>
<td>iv</td>
</tr>
<tr>
<td></td>
<td>LIST OF TABLES</td>
<td>viii</td>
</tr>
<tr>
<td></td>
<td>LIST OF FIGURES</td>
<td>x</td>
</tr>
<tr>
<td></td>
<td>NOTATION</td>
<td>xi</td>
</tr>
<tr>
<td></td>
<td>ABSTRACT</td>
<td>xiii</td>
</tr>
<tr>
<td></td>
<td>CHAPTER</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>1.1 Feature Selection Problem</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>1.2 Dissertation Overview</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>SUPPORT VECTOR MACHINES</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>2.1 Introduction</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>2.2 Linearly Separable Case</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>2.3 Linearly Inseparable Case</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>2.4 Dual Problem</td>
<td>9</td>
</tr>
<tr>
<td></td>
<td>2.5 Kernel Methods and Support Vector Classifiers</td>
<td>12</td>
</tr>
<tr>
<td></td>
<td>2.6 Least Squares SVM</td>
<td>14</td>
</tr>
<tr>
<td></td>
<td>2.6.1 LS-SVM Formulation</td>
<td>14</td>
</tr>
<tr>
<td></td>
<td>2.6.2 Comparison with the L1-norm SVM</td>
<td>15</td>
</tr>
<tr>
<td>3</td>
<td>MODEL SELECTION IN SVM</td>
<td>18</td>
</tr>
<tr>
<td></td>
<td>3.1 Introduction</td>
<td>18</td>
</tr>
<tr>
<td></td>
<td>3.2 Generalization Error Bound</td>
<td>18</td>
</tr>
<tr>
<td></td>
<td>3.3 Cross-Validation</td>
<td>18</td>
</tr>
<tr>
<td></td>
<td>3.4 $\xi\alpha$ Bound</td>
<td>19</td>
</tr>
<tr>
<td></td>
<td>3.5 GACV Bound</td>
<td>20</td>
</tr>
<tr>
<td></td>
<td>3.6 Other Methods</td>
<td>20</td>
</tr>
<tr>
<td>4</td>
<td>RELATED WORKS</td>
<td>21</td>
</tr>
<tr>
<td></td>
<td>4.1 SVM Gradient Method</td>
<td>21</td>
</tr>
<tr>
<td></td>
<td>4.1.1 Algorithm: Linear Kernel</td>
<td>21</td>
</tr>
<tr>
<td></td>
<td>4.1.2 Algorithm: General Kernels</td>
<td>22</td>
</tr>
</tbody>
</table>
4.2 SVM-RFE ........................................ 23
  4.2.1 Algorithm: Linear Kernel ....................... 23
  4.2.2 Algorithm: General Kernels .................... 24
4.3 SVM Gradient-RFE ................................... 25
  4.3.1 Linear Kernel .................................. 26
  4.3.2 General Kernels ................................ 26
4.4 SVM Projection-RFE ................................ 27
  4.4.1 Idea ........................................... 27
  4.4.2 Linear Kernel .................................. 29
  4.4.3 General Kernels ................................ 31
  4.4.4 SVM Projection-RFE Algorithm ................... 33
4.5 The M-fold SVM .................................... 34
  4.5.1 Binary-class M-fold-SVM ......................... 35
  4.5.2 Multi-class M-fold-SVM ......................... 36
4.6 Linear Programming Methods .......................... 37
  4.6.1 Feature Selection via Concave Minimization .... 37
  4.6.2 SVM $\| \cdot \|_p$ Formulation .................... 38
4.7 Correlation Coefficient Methods ..................... 40
4.8 Feature Scaling Methods ............................ 41
4.9 Dimension Reduction Methods ....................... 41
  4.9.1 Principal Component Analysis ................... 41
  4.9.2 Projection Pursuit .............................. 43

5 SUPPORT VECTOR FEATURE SELECTION .................. 44
  5.1 Datasets ........................................... 44
  5.2 Support Vector Feature Selection Using FLD ........ 46
    5.2.1 Fisher’s Linear Discriminant ................... 46
    5.2.2 Support Vector Feature Selection with FLD .... 47
    5.2.3 Experiments ................................... 50
    5.2.4 Discussion .................................... 51
  5.3 SV-RFE SVM ........................................ 52
    5.3.1 Experiments ................................... 53
    5.3.2 Discussion .................................... 57

6 NAIVE BAYES AND FEATURE SELECTION ................... 59
  6.1 Naive Bayes Classifier ............................. 59
  6.2 Naive Bayes Classifier Is a Linear Classifier .... 61
  6.3 Improving Naive Bayes Classifier ................... 63
    6.3.1 Imbalanced Data ............................... 63
    6.3.2 Training with Only One Feature ................. 65
    6.3.3 A Vector of Zeros Test Input .................. 66
  6.4 Text Datasets and Preprocessing .................... 66
  6.5 Feature Selection Approaches in Text Datasets .... 70
    6.5.1 Information Gain .............................. 70
6.5.2  $t$-test ......................................................... 70
6.5.3 Correlation Coefficient ........................................ 71
6.5.4 Fisher Score ...................................................... 71
6.5.5 Odds Ratio ......................................................... 72
6.5.6 NB Score ......................................................... 72
6.5.7 Note on Feature Selection ....................................... 72
6.6 Class Dependent Term Weighting Naive Bayes .................. 73
6.6.1 Motivation for Class Dependent Term Weighting ............. 74
6.6.2 CDTW-NB-RFE Algorithm .................................... 79
6.7 Experimental Results ............................................. 81
6.7.1 Feature Ranking Works ....................................... 81
6.7.2 Experimental Results on the Newsgroup Datasets ........... 83
6.8 Naive Bayes for Continuous Features ........................... 85
6.9 Discussion ......................................................... 88

7 FEATURE SELECTION AND LINEAR DISCRIMINANTS .......... 90
7.1 Feature Selection with Linear Discriminants .................. 90
7.2 Linear Discriminants and Their Decision Hyperplanes .......... 92
7.3 Visualization of Selected Features ............................... 95
7.4 Compact Data Representation and Feature Selection .......... 98
7.5 Applications of Feature Selection ............................... 100

8 CONCLUSIONS ....................................................... 102

REFERENCES ......................................................... 104

BIOGRAPHICAL SKETCH ............................................. 112

vii
LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2–1</td>
<td>Common Kernel Functions</td>
<td>14</td>
</tr>
<tr>
<td>4–1</td>
<td>Common Kernel Functions and Their Derivatives</td>
<td>22</td>
</tr>
<tr>
<td>5-1</td>
<td>Training/Test Data Splits Four Datasets</td>
<td>46</td>
</tr>
<tr>
<td>5-2</td>
<td>Simple All vs. Simple SV Performance on MNIST Data (FLD)</td>
<td>50</td>
</tr>
<tr>
<td>5-3</td>
<td>Simple All vs. Simple SV Performance on Sonar Data (FLD)</td>
<td>50</td>
</tr>
<tr>
<td>5-4</td>
<td>All-RFE vs. SV-RFE Performance on MNIST Data (FLD)</td>
<td>51</td>
</tr>
<tr>
<td>5-5</td>
<td>All RFE vs. SV-RFE Performance on Sonar Data (FLD)</td>
<td>51</td>
</tr>
<tr>
<td>5-6</td>
<td>Guyon RFE vs. SV-RFE Accuracy on Leukemia Data (SVM)</td>
<td>54</td>
</tr>
<tr>
<td>5-7</td>
<td>Guyon RFE vs. SV-RFE Time on Leukemia Data (SVM)</td>
<td>54</td>
</tr>
<tr>
<td>5-8</td>
<td>Guyon RFE vs. SV-RFE Performance on Colon Cancer Data (SVM)</td>
<td>55</td>
</tr>
<tr>
<td>5-9</td>
<td>Guyon RFE vs. SV-RFE Performance on MNIST Data (SVM)</td>
<td>56</td>
</tr>
<tr>
<td>5-10</td>
<td>Guyon RFE vs. SV-RFE Performance on Sonar Data (SVM)</td>
<td>56</td>
</tr>
<tr>
<td>6–1</td>
<td>Imbalanced Training Data Example</td>
<td>64</td>
</tr>
<tr>
<td>6–2</td>
<td>Probability Estimates</td>
<td>64</td>
</tr>
<tr>
<td>6–3</td>
<td>Keywords Used to Collect the Text Datasets</td>
<td>67</td>
</tr>
<tr>
<td>6–4</td>
<td>20 Newsgroup Dataset Distribution</td>
<td>69</td>
</tr>
<tr>
<td>6–5</td>
<td>Illustrative Data for the Class Dependent Term Weighting</td>
<td>74</td>
</tr>
<tr>
<td>6–6</td>
<td>Top 10 Words Selected from the TextAB Dataset</td>
<td>81</td>
</tr>
<tr>
<td>6–7</td>
<td>Top 10 Words Selected from the TextCD Dataset</td>
<td>82</td>
</tr>
<tr>
<td>6–8</td>
<td>Top 10 Words Selected from the TextCL Dataset</td>
<td>82</td>
</tr>
<tr>
<td>6–9</td>
<td>CDTW-NB-RFE vs. Original NB-RFE</td>
<td>83</td>
</tr>
<tr>
<td>6–10</td>
<td>Original NB vs. CDTW-NB</td>
<td>84</td>
</tr>
<tr>
<td>6–11</td>
<td>Accuracy Results for Ranking Schemes</td>
<td>84</td>
</tr>
</tbody>
</table>
6–12 Comparison between FLD, SVM, and Discretized NB . . . . . . . . . . 88
7–1 Performance of Various Ranking Schemes on Linear SVM (%) . . . . . . . 99
7–2 Compact Data Representation on SV-RFE (SVM) . . . . . . . . . . . . . 100
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2–1</td>
<td>Linear Discriminant Planes</td>
<td>7</td>
</tr>
<tr>
<td>2–2</td>
<td>Geometric Margin in the Maximum Margin Hyperplane</td>
<td>8</td>
</tr>
<tr>
<td>2–3</td>
<td>Linearly Inseparable Data</td>
<td>9</td>
</tr>
<tr>
<td>2–4</td>
<td>A Nonlinear Kernel Support Vector Machine</td>
<td>12</td>
</tr>
<tr>
<td>4–1</td>
<td>Motivation for the SVM Projection-RFE</td>
<td>28</td>
</tr>
<tr>
<td>4–2</td>
<td>Linear Kernel Case for Projection-RFE</td>
<td>29</td>
</tr>
<tr>
<td>4–3</td>
<td>Nonlinear Kernel Case for Projection-RFE</td>
<td>31</td>
</tr>
<tr>
<td>5–1</td>
<td>Images of Digits 2 and 3.</td>
<td>45</td>
</tr>
<tr>
<td>5–2</td>
<td>Recursive FLD Idea.</td>
<td>48</td>
</tr>
<tr>
<td>5–3</td>
<td>Effect of the Parameter C in SVM.</td>
<td>57</td>
</tr>
<tr>
<td>6–1</td>
<td>Pictorial Illustration of the Bag-Of-Words Model.</td>
<td>69</td>
</tr>
<tr>
<td>6–2</td>
<td>Test Accuracy with Best-ranked Features on 20 Newsgroup Datasets.</td>
<td>85</td>
</tr>
<tr>
<td>7–1</td>
<td>Linear Discriminants-Separable, Balanced Data.</td>
<td>93</td>
</tr>
<tr>
<td>7–2</td>
<td>Linear Discriminants-Separable, Imbalanced Data.</td>
<td>94</td>
</tr>
<tr>
<td>7–3</td>
<td>Linear Discriminants-Inseparable, Balanced, Diff. Var. Data.</td>
<td>94</td>
</tr>
<tr>
<td>7–4</td>
<td>Images with the Best-ranked Features.</td>
<td>96</td>
</tr>
<tr>
<td>7–5</td>
<td>Leukemia Data Plot with the Best Two Features.</td>
<td>96</td>
</tr>
<tr>
<td>7–6</td>
<td>Test Accuracy with Best-ranked Features on MNIST Data</td>
<td>97</td>
</tr>
<tr>
<td>7–7</td>
<td>Test Accuracy with Best-ranked Features on Three Datasets.</td>
<td>98</td>
</tr>
</tbody>
</table>
NOTATION

\( x_i \) training instance
\( y_i \) target corresponding to instance \( x_i; y_i \in \{-1, 1\} \)
\( g(x) \) decision hyperplane trained from SVM
\( w \) weight vector (normal) when \( g(x) \) is linear
\( b \) threshold in \( g(x) \)
\( \alpha \) Lagrange multipliers
\( \phi \) mapping function to feature space
\( K(x, y) \) kernel \( \langle \phi(x) \cdot \phi(y) \rangle \)
\( l \) number of training instances
\( n \) dimension of input space
\( \langle x \cdot y \rangle \) inner product between \( x \) and \( y \)
\( \| \cdot \| \) 2-norm
\( \| \cdot \|_p \) \( p \)-norm
\( \| \cdot \|' \) dual norm of \( \| \cdot \| \)
\( p(x, g(x)) \) projection of \( x \) onto \( g(x) \)
\( \xi \) slack variable
\( C \) upper bound for \( \alpha \)
\( L \) primal Lagrangian function
\( x' \) transpose of vector \( x \)
\( SV \) support vectors
\( |x| \) component-wise absolute value of vector \( x \)
\( |SV| \) number of support vectors
\( \angle(x, y) \) angle between \( x, y \)
\( \mu_i \)  mean for feature \( i \) over training instances

\( \sigma_i \)  standard deviation for feature \( i \) over training instances

\( V(x) \)  variance of \( x \)

\( E(x) \)  expectation \( x \)

\( \Sigma \)  covariance matrix

\( S_W \)  total *within-class* covariance matrix

\( P(y|x) \)  the posterior probability of \( y \) with respect to \( x \)

\( P(x|y) \)  the likelihood of \( y \) with respect to \( x \)
The problem of feature selection is to find a subset of features for optimal classification. A critical part of feature selection is to rank features according to their importance for classification. This dissertation deals with feature ranking. The first method is based on support vectors (SVs). Support vectors refer to those sample vectors that lie around the boundary between two different classes. We show how we can do feature ranking with only support vectors. While SV-based feature ranking can be applied to any discriminant analysis, here we considered two linear discriminants, SVM and FLD. Feature ranking is based on the weight associated with each feature or determined by recursive feature elimination (RFE). Both schemes are shown to be effective in various domains. The second method of feature ranking is based on naive Bayes. The naive Bayes classifier has been extensively used in text categorization. Despite its successful application to this problem, naive Bayes has not been used for feature selection. We have developed a new feature scaling method, called class-dependent-term-weighting (CDTW). A new feature selection method, CDTW-NB-RFE, combines CDTW and RFE. Our experiments showed that CDTW-NB-RFE
outperformed any of the five popular feature ranking schemes used on text datasets. This method has also been extended to continuous data domains.
CHAPTER 1
INTRODUCTION

Feature selection is an important problem in machine learning [12, 26, 56]. In this chapter, we present the nature of the feature selection and show why it is needed. We also provide a glossary of notations and an overview of this dissertation.

1.1 Feature Selection Problem

Feature selection is a problem of selecting input variables (features) that are most predictive of a given output. In the classification problem setting, given a set of input-output pairs \( \{(x_i, y_i) : i = 1 \ldots l, x_i \in \mathbb{R}^d, y_i \in \{-1, +1\}\} \), we attempt to construct a decision function \( f \), which maps the input \( x_i \) onto its label \( y_i \). Then the goal is to find an \( f \) which minimizes the error \( (f(x) \neq y) \) for every possible \( x \). The feature selection identifies a small subset of features (variables) so that the classifier constructed with the selected features minimizes the error and the selected features also better explain the data. The most important part of the feature selection is to rank the features according to their importance for classification. While this dissertation deals with feature ranking, we will use the terms, feature selection and feature ranking interchangeably. But most of the time, it is meant to be the feature ranking. There are several motivations for the feature selection [94].

First, the feature selection identifies relevant features. For example, in a microarray dataset, researchers strongly believe that only a small subset of genes (features) is responsible for a disease. Our previous experimental results [99] and other researchers’ findings [44] support this conclusion. When the feature selection is applied to text datasets, the relevance of the selected features is obvious. They represent the keywords of the documents collection. If it is applied to handwritten digit image
datasets, the selected features are the subset of the most discriminating pixels in the image.

Second, the feature selection gives a better generalization error. Typically, only a small number of features of \( x \) give sufficient information for the classification. The feature selection problem finds the small subset of features that are relevant to the target concept. A small subset of relevant features gives more discriminating power than using more features. This is counter-intuitive since more features give more information and therefore should give more discriminating power. But if a feature is irrelevant, then that feature does not affect the target concept. If a feature is redundant, then that feature does not add anything new to the target concept [60]. This justifies our counter-intuition.

Third, the feature selection reduces the dimension of the input space. Reducing the dimension has many implications. Processing (collecting, storing, computing, transferring, and so forth) several thousand feature values is much more expensive than processing just several dozen or several hundred feature values.

Benefits of the feature selection therefore include cutting the processing time short, giving a better discriminating hyperplane, and a better understanding of the data.

As an easy attempt for a feature selection solution, we may consider all the possible subsets. This may be acceptable only when the number of features is small. The microarray data, however, have the number of features in the order of several thousands. Simply trying all the possible subsets of features is prohibitive in this case. In our experiments, we have a colon cancer microarray dataset and a leukemia microarray dataset. In the colon cancer dataset, each instance consists of 2,000 components (gene expression level), and the leukemia dataset 7,129 components. For example, there are \( 2^{2000} \approx \infty \) subsets for colon cancer data. And high dimensionality is typical in datasets we will consider, such as text datasets and image datasets.
Many methods are available for solving the feature selection problem of the high dimensional data [26, 56]. One part of this dissertation concerns a feature selection using support vectors. Support vectors refer to those sample vectors that lie around the boundary between two different classes. Classifiers built on only support vectors have more discriminating power. This has been proven in Support Vector Machines (SVM) and Fisher’s linear discriminant (FLD). We show how we can do feature ranking with only support vectors. While the support vector feature ranking can be applied to any discriminant analysis, we particularly considered two linear discriminants, SVM and FLD. The main idea for support vector based feature ranking is to identify support vectors and then apply feature ranking schemes on only those support vectors. The feature ranking schemes can be a simple weight based ranking or a recursive feature elimination (RFE). In this dissertation, both are shown to be effective on SV based feature ranking on various data domains.

Another part of this dissertation concerns feature selection and categorization on text document datasets. The text categorization problem has been studied extensively [54, 55, 61, 64, 65, 66, 75]. One of the most popular text categorization methods is the naive Bayes (NB) method [54, 65, 66, 75]. The naive Bayes method is popular as a text data classification tool for its implementation and usage simplicity, and linear time complexity of the method. Despite its success as a classification tool, the naive Bayes has not been utilized as a feature selection method. We have developed a new feature scaling method, called class-dependent-term-weighting (CDTW). A new feature selection method, CDTW-NB-RFE, combines CDTW and NB-RFE. For our experiments on text data, we have used three mini datasets and real world datasets. The three mini datasets were collected from PubMed using some keywords. By applying the feature selection methods, we could identify the discriminating keywords between two different groups of text document datasets. In contrast to other data domains, the text data domain needs some preprocessing, which is unique in
text document representation. We will discuss how to preprocess the text documents so that they are suitable for a supervised learning. We have used 20 Newsgroup datasets as the real world datasets. The effectiveness of CDTW-NB-RFE has been shown by comparison with the other five most popular feature selection methods on text datasets. CDTW-NB-RFE outperformed the most popular feature selection methods.

We apply the feature selection methods on linear discriminants. In fact, we make use of the weight vector (normal direction) of the linear discriminants as a feature ranking criterion. We show how the weight vector of any linear discriminants can serve as a ranking criterion. Conversely, a ranking score for a feature can be utilized to be a weight of a linear discriminant. We address these issues, too.

1.2 Dissertation Overview

We begin this dissertation by defining the feature selection problem, and then we discuss why the feature selection problem is important, especially the feature selection using linear discriminants such as SVM, FLD, and NB. The SVM is introduced as background knowledge in Chapter 2. Derivations for linear SVM, for linearly separable data, linearly inseparable data, and nonlinear SVM are covered together. Least square SVM (LS-SVM) is briefly introduced and compared to the regular SVMs. In Chapter 3, model selection in SVMs is reviewed. Model selection concerns tuning appropriate parameter findings for SVM. Therefore, model selection is important when we attempt to get the best possible decision function using SVM. In Chapter 4, we present some existing feature selection algorithms in the SVM domain and other domains. Several of the related methods are reviewed with their ideas and algorithms. In Chapter 5, we introduce a new feature selection algorithm, SV-based feature selection. Two linear discriminants, SVM and FLD, are considered with SV-based feature selection. The motivation of the SV-based feature selection is discussed. The effectiveness of the SV-based method is shown with experimental results on four different
datasets. In Chapter 6, we show a new feature selection method based on a naive Bayes method, CDTW-NB-RFE. Other issues related to the NB, such as systemic problems arising when NB is used for a small number of features, are discussed on the way of development of the CDTW-NB-RFE idea. We also show how we apply the feature selection on discretized data using NB. In Chapter 7, we show the generalization of the feature selection with linear discriminants. In this discussion, we show that the weight vector of any linear discriminants can serve as a feature ranking score. Conversely, some feature ranking scores can be used as a weight of a linear discriminant for classification. The effectiveness of the selected features from our methods is shown by visualizing the data with the those selected features. Two possible feature selection applications, compact data representation and cluster labeling, are discussed. This dissertation concludes by showing the intended contribution of our research and possible future research.
CHAPTER 2
SUPPORT VECTOR MACHINES

2.1 Introduction

Support Vector Machines (SVMs) are a new classification method developed by Vapnik and his group at AT&T Bell Labs [89]. The SVMs have been applied to classification problems [45, 73] as alternatives to multilayer networks. Classification is achieved by a linear or nonlinear separating surface in the input space of the data set. The goal of SVM is to minimize the expectation of the output of sample error. Support Vector Machines map a given set of binary labeled training data to a high-dimensional feature space and separate the two classes of data with a maximum margin of hyperplane. To understand the SVM approach, we need to know two key ideas: duality and kernels [5, 25, 84]. We examine these concepts for the simple case and then show how they can be extended to more complex tasks.

2.2 Linearly Separable Case

Consider a binary classification problem with training instances and its class label pairs \((x_i, y_i)\), where \(i = 1, \cdots, l\) and \(y_i \in \{-1, 1\}\). The \(x_i\) is called a positive instance if the corresponding label is +1; otherwise, it is a negative instance. Let \(P\) denote the set of positive instances and \(N\) the set of negative instances. Figure 2–1 shows some possible linear discriminant planes which separate \(P\) from \(N\). In Figure 2–1 [5] an infinite number of discriminating planes exist and \(P1\) and \(P2\) are among them. \(P1\) is preferred since \(P2\) more likely misclassifies an instance if there are small perturbations in the instance [5]. A maximum distance or margin plane is a discriminant plane which is furthest from both \(P\) and \(N\). This gives an intuitive explanation about why maximum margin discriminant planes are better. \(P3\) and \(P4\) are called supporting planes, and points lying on the supporting planes are called support vectors. The
maximum margin is the distance between two supporting planes, and the geometric margin is the normalized distance by weight between supporting planes.

For $x_i \in P$, suppose we want to find $w$ and $b$ such that $\langle w \cdot x_i \rangle + b \geq 0$. Suppose $k = \min_i |\langle w \cdot x_i \rangle + b|$. Then, $|\langle w \cdot x_i \rangle + b| \geq k, \forall x_i$. For the points in the other class, we require $\langle w \cdot x_i \rangle + b \leq -k$. Note that $w$ and $b$ can be rescaled so we can always set $k$ equal to 1. To find the plane furthest from both sets, we maximize the distance between the supporting planes for each class. The support vectors are shown inside the dotted circle in Figure 2–2.

In Figure 2–2, the normalized margin, or geometric margin, between the supporting planes $\langle w \cdot x_1 \rangle + b = +1$, and $\langle w \cdot x_2 \rangle + b = -1$ is $\gamma = 2/\|w\|$ [6, 83]. Since we want to maximize the margin for the reason we intuitively justified, we can formulate the problem as an optimization problem [25].

$$\max_{w, b} \frac{2}{\|w\|^2}$$

s.t. $\langle w \cdot x_i \rangle + b \geq +1, \ y_i \in P$

$\langle w \cdot x_i \rangle + b \leq -1, \ y_i \in N$

(2.1)
Since maximizing the margin is equivalent to minimizing $\|w\|^2/2$ and the constraints can be simplified to $y_i((w \cdot x_i) + b) \geq 1, \forall i$, we can rewrite them as follows:

$$\min_{w,b} \quad \frac{\|w\|^2}{2}$$

s.t. $y_i((w \cdot x_i) + b) \geq 1, i = 1, \ldots, l. \quad (2.2)$

In mathematical programming, a problem such as (2.2) is called a convex quadratic problem. Many robust algorithms exist for solving the quadratic problems. Since the quadratic problem is convex, any local minimum found is always a global minimum [9].

**2.3 Linearly Inseparable Case**

Figure 2–3 [5] shows the two intersecting convex hulls. In Figure 2–3, if the single bad square is removed, then the same formulation we had would work. To this end, we need to relax the constraint and add a penalty to the objective function in (2.2).
Equivalently, we need to restrict the influence of any single point. Any point falling on the wrong side of its supporting plane is considered to be an error. We want to maximize the margin and minimize the error. To this end, we need to make some changes to the formulation in (2.2).

A nonnegative error variable, or slack variable $\xi_i$, is added to each constraint and added to the objective function as a weighted penalty [25]:

$$
\min_{w, b, \xi} \frac{||w||^2}{2} + C \sum_{i=1}^{l} \xi_i
$$

s.t. $y_i(\langle w \cdot x_i \rangle + b) + \xi_i \geq 1, \xi_i \geq 0,$

$$
\quad i = 1, \ldots, l,
$$

(2.3)

where $C$ is a constant. The constant $C$ enforces to reduce the influence of any particular point, such as the bad square in Figure 2–3. More explanation of $C$ is given in the next section.

2.4 Dual Problem

In this section, we show how to derive a dual representation of the problem. The dual representation of the original problem is important since its objective function and constraints are expressed in inner products. By replacing them with a kernel function, we can therefore benefit from an enormous computational shortcut. Dual representation involves the so-called Lagrangian theory [4, 7, 9, 22, 69]. The
Lagrangian theory characterizes the solution of an optimization problem. First we introduce the dual representation for a linearly separable problem (2.2).

The primal Lagrangian function for problem (2.2) is as follows [25]:

\[
L(w, b, \alpha) = \frac{1}{2} \langle w \cdot w \rangle - \sum_{i=1}^{l} \alpha_i [y_i (\langle w \cdot x_i \rangle + b) - 1] \tag{2.4}
\]

where \( \alpha_i \geq 0 \) are the Lagrange multipliers. At the extreme, the partial derivatives of the Lagrangian are 0, that is,

\[
\frac{\partial L(w, b, \alpha)}{\partial b} = \sum_{i=1}^{l} y_i \alpha_i = 0
\]

\[
\frac{\partial L(w, b, \alpha)}{\partial w} = w - \sum_{i=1}^{l} y_i \alpha_i x_i = 0
\]

or

\[
0 = \sum_{i=1}^{l} y_i \alpha_i, \quad (2.5)
\]

\[
w = \sum_{i=1}^{l} y_i \alpha_i x_i. \quad (2.6)
\]

Substituting the relations (2.5, 2.6) into the primal problem (2.4) leads to

\[
L(w, b, \alpha) = \sum_{i=1}^{l} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{l} y_i y_j \alpha_i \alpha_j \langle x_i \cdot x_j \rangle. \tag{2.7}
\]

Now we have the dual problem corresponding to the primal problem (2.2):

\[
\text{maximize} \quad \sum_{i=1}^{l} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{l} y_i y_j \alpha_i \alpha_j \langle x_i \cdot x_j \rangle \\
\text{s.t} \quad \sum_{i=1}^{l} y_i \alpha_i = 0, \\
\quad \alpha_i \geq 0, i = 1, \ldots, l. \tag{2.8}
\]

The primal problem (2.2) and corresponding dual problem (2.8) yield the same normal vector to the hyperplane \( \sum_{i=1}^{l} y_i \alpha_i x_i + b \).
We do the similar dual formulation for the linearly nonseparable problem. The primal Lagrangian for the problem (2.3) is as follows:

\[ L(w, b, \xi, \alpha, r) = \frac{1}{2} \langle w \cdot w \rangle + C \sum_{i=1}^{l} \xi_i^2 - \sum_{i=1}^{l} \alpha_i[y_i(\langle w \cdot x_i \rangle + b) - 1 + \xi_i] - \sum_{i=1}^{l} r_i \xi_i \]  

(2.9)

where \( \alpha_i \geq 0 \) and \( r_i \geq 0 \). Setting the partial derivatives of the primal Lagrangian (2.9) to 0, we derive the dual problem.

\[
\frac{\partial L(w, b, \xi, \alpha, r)}{\partial b} = \sum_{i=1}^{l} y_i \alpha_i = 0,
\]

\[
\frac{\partial L(w, b, \xi, \alpha, r)}{\partial \xi_i} = C - \alpha_i - r_i = 0,
\]

\[
\frac{\partial L(w, b, \xi, \alpha, r)}{\partial w} = w - \sum_{i=1}^{l} y_i \alpha_i x_i = 0
\]

or

\[
0 = \sum_{i=1}^{l} y_i \alpha_i, \quad (2.10)
\]

\[
C = \alpha_i + r_i, \quad (2.11)
\]

\[
w = \sum_{i=1}^{l} y_i \alpha_i x_i. \quad (2.12)
\]

Substituting the relations (2.10, 2.11, 2.12) into the primal problem (2.9) leads to

\[ L(w, b, \xi, \alpha, r) = \sum_{i=1}^{l} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{l} y_i y_j \alpha_i \alpha_j \langle x_i \cdot x_j \rangle. \]

(2.13)

Now we have the dual problem corresponding to the primal problem (2.3):

\[
\text{maximize} \quad \sum_{i=1}^{l} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{l} y_i y_j \alpha_i \alpha_j \langle x_i \cdot x_j \rangle
\]

s.t. \[
\sum_{i=1}^{l} y_i \alpha_i = 0,
\]

\[ C \geq \alpha_i \geq 0, i = 1, \ldots, l \]

(2.14)

where \( C \) is the upper bound for the Lagrange multipliers. That is, it gives the upper bound for the \( \alpha \) so it limits the influence of any point.
2.5 Kernel Methods and Support Vector Classifiers

How to construct the decision hyperplane from the solution of the problem (2.8) is the next topic. Let $\alpha$ solve the dual problem (2.8). Then by the relation (2.6), the weight $w$ is as follows:

$$w = \sum_{i=1}^{l} y_i \alpha_i x_i$$

and

$$b = \frac{\max_{y_i=-1}(\langle w \cdot x_i \rangle) + \min_{y_i=1}(\langle w \cdot x_i \rangle)}{2}$$

Its decision function therefore is

$$g(x) = \sum_{i=1}^{l} \alpha_i y_i \langle x_i \cdot x \rangle + b \quad (2.15)$$

$$= \sum_{i \in SV} \alpha_i y_i \langle x_i \cdot x \rangle + b. \quad (2.16)$$

For the example on the left of Figure 2–4, no simpler algorithm would work well. A quadratic function such as the circle pictured is needed. Figure 2–4 shows how we can make use of existing linear discriminant methods by mapping the original input data into a higher dimensional space. When we map the original input space, say $X$ using $\phi(x)$, we create a new space $F = \{\phi(x) : x \in X\}$. Then $F$ is called the feature space. We introduce the idea of the kernel with an example [82].
To produce a quadratic discriminant in a two-dimensional input space with attributes \( x_1 \) and \( x_2 \), map the two-dimensional input space \([x_1, x_2]\) to the three-dimensional feature space \([x_1^2, \sqrt{2}x_1x_2, x_2^2]\) and construct a linear discriminant in the feature vector space. Specifically, define: \( \phi(x) : \mathbb{R}^2 \to \mathbb{R}^3 \) then, \( x = [x_1, x_2] \), and \( \mathbf{w} \cdot \mathbf{x} = w_1x_1 + w_2x_2 \). \( \phi(x) = [x_1^2, \sqrt{2}x_1x_2, x_2^2] \), \( \mathbf{w} \cdot \phi(x) = w_1x_1^2 + w_2\sqrt{2}x_1x_2 + w_3x_2^2 \). The resulting decision function,

\[
g(x) = \mathbf{w} \cdot \phi(x) + b = w_1x_1^2 + w_2\sqrt{2}x_1x_2 + w_3x_2^2 + b
\]

is linear in the three-dimensional space, but it is nonlinear in the original two-dimensional space. This approach, however, may cause some problems. If the data are noisy, then SVM would try to discriminate all the positive examples from the negative examples by increasing the dimensionality of the feature space. This is the so-called overfitting. The growth of the dimensionality of the feature space would be exponential. The second concern is computing the separating hyperplane by carrying out the map into the feature space. The SVMs magically get around this problem by using the so-called kernel. The formal definition for a kernel is as follows [25]:

**Definition 2.5.1** A kernel is a function \( K \), such that for all \( \mathbf{x}, \mathbf{z} \in \mathcal{X} \)

\[
K(\mathbf{x}, \mathbf{z}) = \langle \phi(\mathbf{x}) \cdot \phi(\mathbf{z}) \rangle,
\]

where \( \phi \) is a mapping from \( \mathcal{X} \) to an (inner product) feature space \( \mathcal{F} \).

To change from a linear to a nonlinear classifier, we substitute only the inner product \( \langle \mathbf{x} \cdot \mathbf{y} \rangle \) by a kernel function \( K(\mathbf{x}, \mathbf{y}) \). In our example, \( \phi(\mathbf{x}) \cdot \phi(\mathbf{y}) = (x_1^2, \sqrt{2}x_1x_2, x_2^2)(y_1^2, \sqrt{2}y_1y_2, y_2^2)' = (\mathbf{x} \cdot \mathbf{y})^2 =: K(\mathbf{x}, \mathbf{y}) \). Note that the actual dot product in \( \mathcal{F} \) is computed in \( \mathbb{R}^2 \), the input space. Table 2–1 shows common kernel functions. By changing the kernels, we can also get different nonlinear classifiers, but no algorithm change is required. From a machine trained with an appropriately
defined kernel, we can construct a decision function:

\[ g(x) = \sum_{i \in SV} \alpha_i y_i K(x_i, x) + b. \]  

(2.17)

Table 2–1: Common Kernel Functions

<table>
<thead>
<tr>
<th>Kernel</th>
<th>(K(x, y))</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linear</td>
<td>((x \cdot y))</td>
</tr>
<tr>
<td>Polynomial</td>
<td>((1 + (x \cdot y))^\theta)</td>
</tr>
<tr>
<td>RBF</td>
<td>(\exp(-\frac{|x-y|^2}{\theta}))</td>
</tr>
</tbody>
</table>

2.6 Least Squares SVM

The least squares SVM (LS-SVM) has been developed by Gestel et al. [41]. They modified the traditional L1-norm SVM to formulate it as a regression problem targeting \(\pm 1\). In the following section, we briefly introduce the LS-SVM formulation, and in the next section show the comparison with L1-norm SVM.

2.6.1 LS-SVM Formulation

An error variable, or slack variable \(e_i\), is added to each constraint and its square is added to the objective function as a weighted penalty:

\[
\min_{w, b, \xi} \frac{\|w\|^2}{2} + \gamma \frac{1}{2} \sum_{i=1}^{l} e_i^2 \\
\text{s.t} \quad y_i(\langle w \cdot x_i \rangle + b) = 1 - e_i, \\
i = 1, \ldots, l, 
\]

(2.18)

where \(\gamma\) is a constant and balances the data fit and the complexity of the resulting classifier function. Note the equality constraint. Because of this equality constraint, the solution of the optimization problem 2.18 can be obtained from a system of linear equations. From 2.18, we construct the Lagrangian function

\[
L(w, b, e; \alpha) = \frac{1}{2} \langle w \cdot w \rangle + \gamma \frac{1}{2} \sum_{i=1}^{l} e_i^2 - \sum_{i=1}^{l} \alpha_i [y_i(\langle w \cdot x_i \rangle + b) - 1 + e_i] 
\]

(2.19)
where $\alpha_i$ are the Lagrange multipliers. Note that the Lagrange multipliers $\alpha_i$ can be either positive or negative because of the equality constraint in the 2.18. We can write down the optimality condition as follows:

$$\frac{\partial L(w, b, e; \alpha)}{\partial w} = 0 \rightarrow w = \sum_{i=1}^{l} y_i \alpha_i x_i$$

$$\frac{\partial L(w, b, e; \alpha)}{\partial b} = 0 \rightarrow \sum_{i=1}^{l} y_i \alpha_i = 0$$

$$\frac{\partial L(w, b, e; \alpha)}{\partial e_i} = 0 \rightarrow \alpha_i = \gamma e_i, i = 1, \ldots, l$$

$$\frac{\partial L(w, b, e; \alpha)}{\partial \alpha_i} = 0 \rightarrow y_i(\langle w \cdot x_i \rangle + b) - 1 + e_i = 0, i = 1, \ldots, l$$

or equivalently,

$$\begin{bmatrix}
I & 0 & 0 & -Z^T \\
0 & 0 & 0 & -Y^T \\
0 & 0 & \gamma I & -I \\
Z & Y & I & 0
\end{bmatrix}
\begin{bmatrix}
w \\
b \\
e \\
\alpha
\end{bmatrix}
= \begin{bmatrix} 0 \\ 0 \\ -1 \\ \bar{1} \end{bmatrix}$$

where $Z = [x_1^T y_1; \cdots; x_l^T y_l]$, $Y = [y_1; \cdots; y_l]$, $\bar{1} = [1; \cdots; 1]$, $e = [e_1; \cdots; e_l]$, and $\alpha = [\alpha_1; \cdots; \alpha_l]$. After eliminating $w$, and $e$, we have the following:

$$\begin{bmatrix}
0 & -Y^T \\
Y & ZZ^T + \gamma^{-1}I
\end{bmatrix}
\begin{bmatrix}
b \\
\alpha
\end{bmatrix}
= \begin{bmatrix} 0 \\ \bar{1} \end{bmatrix}$$

Hence the classifier is found from the solution of the system of equations instead of quadratic programming.

### 2.6.2 Comparison with the L1-norm SVM

It would be interesting to compare and contrast the LS-SVM with the L1-norm SVM. We start this discussion by recapping the idea of the L1-norm SVM. The idea of the L1-norm SVM is based on the structural risk minimization principle, and the risk bound is minimized by minimizing both the data fit measure (sum of $e_i$) and complexity of the resulting classifier function ($\|w\|^2/2$). The complexity of
the function can be measured by the margin of the decision hyperplane, and the maximum margin decision hyperplane has a simpler complexity. The LS-SVM is introduced by modifying the L1-norm SVM to formulate the classification problem as a binary target regression problem. Both the L1-norm SVM and LS-SVM have similar objective functions except for the sum of nonnegative slack variables in the L1-norm SVM and the sum of square of the slack variable in LS-SVM. In the constraints L1-norm SVM has an inequality and nonnegativity condition for the slack variable while LS-SVM has just equality constraint. A general way of solving the L1-norm SVM is by introducing the Lagrange multipliers for each constraint and convert them into a dual problem. In the dual problem, the objective function is a quadratic convex function (by the positive semi-definiteness of the kernel matrix) with the linear convex set constraint. This means the solution found is a global optimum, and the solution finding algorithms are well studied in the mathematical programming literature [9]. On the other hand, the LS-SVM solution can be found by solving the systems of linear equations. Also, the solution found here is global because of the same reason as in the L1-norm SVM.

It is worth mentioning the meaning of the $\alpha$, the Lagrange multiplier. In the L1-norm SVM, $\alpha_i$ measures an importance of a training example $x_i$ during the training process. $\alpha_i$ close to the decision hyperplane have non-zero values, and corresponding examples are called support vectors (SVs). Only SVs are involved in constructing the decision hyperplane and usually $|SVs|/l$, where $l$ is the total number of examples, is small. This achieves a sparse representation of the dataset and the decision hyperplane.

On the other hand, in LS-SVM, since the problem formulation is a regression formulation with binary target, almost all the examples have some adjustments, $e_i$ to the target($\pm 1$), and these $e_i$ are non-zero values. Since $\alpha_i = \gamma e_i$, many of $\alpha_i$ have non-zero values. Therefore, no sparseness is achieved in LS-SVM. And examples close
and far to the decision plane get the larger $\alpha_i$ and hence $\alpha_i$ do not have such meaning as in the L1-norm SVM.
CHAPTER 3
MODEL SELECTION IN SVM

3.1 Introduction

The model selection is getting more important when we consider SVM as a classification tool for a real-world problem. This is because parameters chosen with care give much better generalization performance. Despite the successful application of SVMs in various domains, actually using it as a classification tool requires a non-trivial tuning of SVM parameters. The parameters include the tradeoff parameter $C$ in the linear classifier case, and $C$ and a kernel function parameter. For example, when the kernel function is a radial basis function, the kernel parameter is the $\sigma$. In general, cross-validation is used to find good estimates of the SVM parameters, but it is computationally expensive. However, recent model selection research is making an important step by allowing the user to offer reasonably good estimates for the parameters without the use of some additional validation data. In this chapter, we survey some of the work in this spirit. The basic idea is to find a generalization error bound and try to minimize this bound with some combination of parameters.

3.2 Generalization Error Bound

The purpose of the generalization error bound is two-fold: 1) to estimate the true error for a decision hyperplane with a given training dataset; and 2) to tune the appropriate hyperparameters for given data. The hyperparameters are $C$, the tradeoff parameter between the training error and margin, and $\sigma$ in the radial basis function kernel, for example.

3.3 Cross-Validation

Cross-validation (CV) is performed as follows [10]. One divides the training set randomly into $k$ disjoint subsets. One then trains the classifier using only $k-1$ subsets
and test its performance by calculating the accuracy using the left-out subset. This
process is repeated \( k \) times. Each time, a different subset is left out. The average of \( k \)
accuracies is the estimate of the generalization accuracy. One can extend this idea to
an extreme case by taking the \( k \) as the number of examples in the training data. Then
this process is called the Leave-One-Out (LOO) method. The disadvantage of CV is
that it requires \( k \) repeated training and testing. This is quite expensive, especially for
LOO. The advantage is that it gives a very accurate estimate for the generalization
bound [30] and is easy to perform the process. That is why this process is popular.
The basic idea behind error estimation is that the test set must be independent of the
training set, and the partition of a data into these two subsets should be unbiased.
The respective data size should be adequate, and the estimated error rate refers to
the test error rather than the training error rate [38].

### 3.4 \( \xi \alpha \) Bound

Joachims establishes the following LOO error bound, which is called \( \xi \alpha \) estimator
of LOO error rate [53]. This estimator is upper bounded by \(|\{i : 2\alpha_i R^2 + \xi_i \geq 1\}|/l|,
where \( \alpha_i \) is the Lagrange multiplier of example \( i \) in the dual problem, \( \xi_i \) is the slack
variable of example \( i \) in the primal problem, \( R^2 \) is the upper bound on \( K(x,x) \), and
\( K(x,x') \geq 0 \). The idea is the relationship between the training examples for which
the inequality \( (2\alpha_i R^2 + \xi_i \geq 1) \) holds and those training examples that can produce
an error in LOO testing. Specifically, if an example \((x_i, y_i)\) is classified incorrectly
by a LOO testing (that is, by a SVM trained on the examples without \((x_i, y_i)\)),
then that example must satisfy the inequality \( (2\alpha_i R^2 + \xi_i \geq 1) \) for a SVM trained
on the all examples. By this, the cardinality of \(|\{i : 2\alpha_i R^2 + \xi_i \geq 1\}|\) is an upper
bound on the number of LOO errors. Here, \( \xi_i \) can be computed from the relation
\( y_i(\sum_j \alpha_j (x_i, x_j) + b) \geq 1 - \xi_i \). Then, to find out a good choice of \( C \) (or kernel
parameter), first choose a \( C \) and then train SVM to find out those quantities needed
in the formula. From these quantities, we can compute the LOO error. Then \( C \)
(or kernel parameter) is increased or decreased in the direction to improve the LOO error. The advantage of this approach is to get a good choice of parameters without an additional validation set.

### 3.5 GACV Bound

Wahba et al. [92] used Generalized Approximate Cross-Validation (GACV) to computably approximate the Generalized Comparative Kullback-Liebler Distance (GCKL) for SVM. The GCKL for SVM is as follows:

\[
GCKL(\lambda) = \frac{1}{l} \sum_{i=1}^{l} \left\{ p_i (1 - f_{\lambda i})_+ + (1 - p_i) (1 + f_{\lambda i})_+ \right\}
\]

where \( f_\lambda(x) = wx + b \) is the decision function, \( f_{\lambda i} = f_\lambda(x_i), p_i = p(y_i|x_i) \), and \((f)_+ = f \) if \( f > 0 \) and 0 otherwise. \( \lambda \) is all the tunable parameters, and these parameters can be expressed inside the kernel function [21, 30]. The GACV is then defined as

\[
GACV(\lambda) = \frac{1}{l} \left\{ \sum_{i=1}^{l} \xi_i + 2 \sum_{y_i f_{\lambda i} < 1} \alpha_i K_{ii} + \sum_{y_i f_{\lambda i} \in [-1, 1]} \alpha_i K_{ii} \right\}
\]

### 3.6 Other Methods

Vapnik introduced VC bound [90] and Vapnik et al. [91] introduced a new concept called “span of support vectors.” Computationally feasible span bound is called “approximate span bound,” and a so-called “radius-margin bound” concerns the L1-norm SVM formulation. All of them are used for estimating generalization error and parameter tuning.
CHAPTER 4
RELATED WORKS

4.1 SVM Gradient Method

Training support vector machines provides optimal values for the Lagrange multipliers $\alpha_i$. From the $\alpha_i$’s, one constructs the decision hyperplane $g(x)$,

$$ g(x) = \sum_{i \in SV} \alpha_i y_i K(x_i, x) + b. $$

Researchers proposed a feature selection technique for SVMs based on the gradient [49]. To rank the features of a given $x$ according to their importance to the classification decision, compute the angles between $\nabla g(x)$ and unit vectors $e_j$, $j = 1, \cdots, n$, representing the indices of each feature. If the $j^{th}$ feature is not important at $x$, $\nabla g(x)$ is almost orthogonal to $e_j$. One does these computations for all the support vectors (SVs), that is, compute the angles for all the SVs, compute their averages and sort them in descending order. This gives a feature ranking for all the features. The SVM-Gradient algorithm is given below for a linear kernel and general kernels separately.

4.1.1 Algorithm: Linear Kernel

(a) Train SVM using all available data components to get $g(x)$.

(b) Compute the gradient $\nabla g(x)$ (or weights $w$), $\forall x \in SV$,

$$ w = \nabla g(x) = \sum_{i \in SV} \alpha_i y_i x_i. $$

Computation of $\nabla g(x)$ can be done easily since

$$ g(x) = \sum_{i \in SV} \alpha_i y_i K(x_i, x) = \sum_{i \in SV} \alpha_i y_i \langle x_i \cdot x \rangle $$

and

$$ \nabla_x K(x_i, x) = x_i. $$
Table 4–1: Common Kernel Functions and Their Derivatives

<table>
<thead>
<tr>
<th>Kernel</th>
<th>$K(x, y)$</th>
<th>$\nabla_y K(x, y)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linear</td>
<td>$x^T y$</td>
<td>$x$</td>
</tr>
<tr>
<td>Polynomial</td>
<td>$(1 + x^T y)^\theta$</td>
<td>$\theta(1 + x^T y)^{\theta-1}$</td>
</tr>
<tr>
<td>RBF</td>
<td>$\exp(-\frac{|x-y|^2}{\theta})$</td>
<td>$\frac{2(x-y)\exp(-\frac{|x-y|^2}{\theta})}{\theta}$</td>
</tr>
</tbody>
</table>

(c) Sort the $|w|$ in descending order. Then this gives the feature ranking.

4.1.2 Algorithm: General Kernels

(a) Train SVM using all available data components to get $g(x)$.

(b) Compute the gradient $\nabla g(x)$, $\forall x \in SV$,

$$\nabla g(x) = \sum_{i \in SV} \alpha_i y_i \nabla_x K(x_i, x).$$

Computation of $\nabla g(x)$ can be done easily again since $g(x) = \sum_{i \in SV} \alpha_i y_i K(x_i, x)$, and only the term $K(x_i, x)$ involves the variable $x$. Therefore take the derivative of the kernel function only $K$. Table 4–1 summarizes common kernel functions and their derivatives.

(c) Compute the sum of angles between $\nabla g(x)$ and $e_j$, $\gamma_j, j = 1, \cdots |s|$

$$\gamma_j = \sum_{x \in SV} \angle(\nabla g(x), e_j)$$

where

$$\angle(\nabla g(x), e_j) = \min_{\beta \in \{0, 1\}} \left\{ \beta \pi + (-1)^\beta \arccos \left( \frac{\langle \nabla g(x) \cdot e_j \rangle}{\|\nabla g(x)\|} \right) \right\}.$$

(d) Compute the averages of the sum of the angles

$$c_j = 1 - \frac{2}{\pi} \cdot \frac{\gamma_j}{|SV|}.$$

(e) Sort the $c$ in descending order. Then this gives the feature ranking.

The authors also proposed when the number of SVs is small compared to the number of training instances, include all the points within the $\epsilon$-region around the
borders, that is, $|g(x_i)| \leq 1 + \epsilon$.

4.2 SVM-RFE

The SVM-RFE [44] is an application of a recursive feature elimination based on sensitivity analysis for an appropriately defined cost function.

4.2.1 Algorithm: Linear Kernel

In the linear kernel case, define a cost function $J = (1/2)\|\mathbf{w}\|$. Then the least sensitive feature which has the minimum magnitude of the weight is eliminated first. This eliminated feature becomes ranking $n$. The machine is retrained without the eliminated feature and removes the feature with the minimum magnitude of weights. This eliminated feature becomes ranking $n - 1$. By doing this process repeatedly until no feature is left, we can rank all the features. The algorithms are as follows:

Given training instances $X_{all} = [x_1, \cdots, x_l]'$, and class labels $y = [y_1, \cdots, y_l]'$, initialize the subset of features $s = [1, 2, \cdots n]$ and $r = \text{an empty array}$. Repeat (a)-(e) until $s$ becomes an empty array.

(a) Construct new training instances $X = X_{all}(; s)$

(b) Train SVM$(X, y)$ to get $g(x)$.

(c) Compute the gradient $\mathbf{w} = \nabla g(x)$.

$$\mathbf{w} = \sum_{i \in SV} \alpha_i y_i x_i.$$

(d) Find the feature $f$ with the smallest $w_j, j = 1, \cdots |s|$

$$f = \arg\min_j(|w_j|).$$

(e) Update $r$ and eliminate the feature from $s$

$$r = [s(f), r],$$

$$s = s - \{s(f)\}.$$ The last eliminated feature is the most important one.
4.2.2 Algorithm: General Kernels

Define a cost function as follows:

\[ J = \frac{1}{2} \alpha^T H \alpha - \alpha^T e, \]

where \( H_{hk} = y_h y_k K(x_h, x_k), \) \( K \) is a kernel function, \( \alpha \) is a Lagrange multiplier and \( e \) is an \( l \) (number of training instances) dimensional vector of ones. To compute the change in \( J \) caused by removing the feature \( i \), one has to retrain a classifier for every candidate feature to be eliminated. This difficulty is avoided by assuming no change in \( \alpha \). Under this assumption, one recomputes the \( H \),

\[ H(-i)_{hk} = y_h y_k K(x_h(-i), x_k(-i)), \]

where \((-i)\) means that the component \( i \) has been removed. So the sensitivity function is defined as follows:

\[ DJ(i) = J - J(-i) = \frac{1}{2} \alpha^T H \alpha - \frac{1}{2} \alpha^T H(-i) \alpha \]

Given training instances \( X_{all} = [x_1, \cdots x_l]' \), and class labels \( y = [y_1, \cdots y_l]' \), initialize the subset of features \( s = [1, 2, \cdots n] \) and \( r \) = an empty array.

Repeat (a)-(e) until \( s \) becomes an empty array.

(a) Construct new training instances \( X = X_{all}(;, s) \).
(b) Train SVM(\( X, y \)) to get \( \alpha \).
\[ \alpha = \text{SVM - train} \]
(c) Compute the ranking criteria for all \( i \).
\[ DJ(i) = \frac{1}{2} \alpha^T H \alpha - \frac{1}{2} \alpha^T H(-i) \alpha \]
(d) Find the feature $k$ such that

$$k = \arg\min_i DJ(i).$$

(e) Eliminate the feature $k$.

$$r = [s(k), r],$$
$$s = s - \{s(k)\}.$$ 

In the linear kernel,

$$K(x_h, x_k) = \langle x_h \cdot x_k \rangle$$
and

$$\alpha^T H \alpha = \|w\|^2.$$ 

Therefore

$$DJ(i) = (1/2) (w_i)^2.$$

This matches the feature selection criterion in the linear kernel case.

### 4.3 SVM Gradient-RFE

The SVM Gradient-RFE combines two existing feature selection methods: SVM-RFE and SVM Gradient. When we say an algorithm, two factors are of concerns: prediction accuracy and computing time. The new method takes the merits of two existing methods so it is competitive to SVM-RFE in terms of prediction accuracy while maintaining speedy computation.

As in [49], this method uses the gradient for feature selection criteria, but in order to give a ranking for all the features, the machines are trained using all the features and then computing the feature selection criterion, that is, the feature with a minimum angle is eliminated. The ranking of this eliminated feature then becomes $n$. Train the machine now without the eliminated feature, and the feature with the minimum selection criterion is eliminated. This eliminated feature becomes ranking
$n - 1$. By recursively eliminating all the features, one ranks all the features. The following section describes the algorithm and computational details.

4.3.1 Linear Kernel

Given training instances $X_{all} = [x_1, \cdots, x_l]^t$, and class labels $y = [y_1, \cdots, y_l]^t$, initialize the subset of features $s = [1, 2, \cdots, n]$ and $r$ as an empty array. Repeat (a)-(e) until $s$ becomes an empty array.

(a) Construct new training instances $X = X_{all}(;, s)$

(b) Train SVM($X, y$) to get $g(x)$.

(c) Compute the gradient $w = \nabla g(x)$.

\[
w = \sum_{i \in SV} \alpha_i y_i x_i.
\]

(d) Find the feature $f$ with the smallest $w_j$, $j = 1, \cdots |s|$

\[f = \text{argmin}_j(|w_j|).
\]

(e) Update $r$ and eliminate the feature from $s$

\[
r = [s(f), r],
\]

\[s = s - \{s(f)\}.
\]

For the linear kernel, the decision hyperplane is linear, and hence its gradient at any support vector is a constant vector (normal vector). The normal vector of the $g(x)$ is the feature selection criterion.

4.3.2 General Kernels

Given training instances $X_{all} = [x_1, \cdots, x_l]^t$, and class labels $y = [y_1, \cdots, y_l]^t$, initialize the subset of features $s = [1, 2, \cdots, n]$ and $r$ as an empty array. Repeat (a)-(e) until $s$ becomes an empty array.

(a) Construct new training instances $X = X_{all}(;, s)$

(b) Train SVM($X, y$) to get $g(x)$. 
(c) Compute the gradient $\nabla g(x)$, $\forall x \in SV$,

$$\nabla g(x) = \sum_{i \in SV} \alpha_i y_i \nabla x_i K(x_i, x).$$

(d) Compute the sum of angles between $\nabla g(x)$ and $e_j$, $\gamma_j$, $j = 1, \cdots |s|$

$$\gamma_j = \sum_{x \in SV} \angle(\nabla g(x), e_j)$$

where

$$\angle(\nabla g(x), e_j) = \min_{\beta \in \{0, 1\}} \left\{ \beta \pi + (-1)^\beta \arccos \left( \frac{\langle \nabla g(x) \cdot e_j \rangle}{\| \nabla g(x) \|} \right) \right\}.$$

(e) Compute the averages of the sum of the angles

$$c_j = 1 - \frac{2}{\pi} \cdot \frac{\gamma_j}{|SV|}.$$

(f) Find the feature $f$ with the smallest $c_j$, $j = 1, \cdots |s|$

$$f = \arg\min_j c_j.$$

(g) Update $r$ and eliminate the feature from $s$

$$r = [s(f), r]$$

$$s = s - \{s(f)\}.$$

As we saw in Section 3.1, the $\nabla g(x)$ computation can be done easily.

### 4.4 SVM Projection-RFE

In the SVM Projection-RFE, the magnitude of the distance between support vectors and its projections is the feature selection criterion. This criterion is combined with RFE so as to give birth to the SVM Projection-RFE.

#### 4.4.1 Idea

When we figured out this method, we identified what characteristics of important features in the SVM classifier are.

Consider the point $A = (x_1, x_2)$ in Figure 4–1.
When point $A$ is projected onto the $g(x) = 0$, let $P = (p_1, p_2)$ be the projected point, then

$$|A - P| = |(x_1, x_2) - (p_1, p_2)| = |(\Delta x_1, \Delta x_2)|.$$ 

Note that the larger magnitude of $\Delta x$ is more influential to the decision plane. In this example, $|\Delta x_1| > |\Delta x_2|$, and hence $x_1$ is more influential to the decision than $x_2$. This is true since the decision hyperplane is almost parallel to the $x_2$ axis, so that whatever the $x_2$ value is, it contributes little to the decision plane $g(x)$. We now have to answer how to efficiently compute $|A - P|$, or $\Delta x_i$, $i = 1, \cdots, n$. The idea is that the normalized distance between $P$ and $A$ is 1 since $A$ is a support vector and the normalized distance between SVs and the hyperplane is 1 by the SVM property. We can make use of this property to have efficient $|P - A|$ computation. Before we introduce the algorithm, we state a proposition [25] saying that the normalized distance (margin) between the hyperplane and SVs is 1.

**Proposition 4.4.1** Consider a linearly separable training sample

$$S = ((x_1, y_1), \cdots, (x_l, y_l)),$$
and suppose the parameters $\alpha^*$ and $b^*$ solve the following optimization problem:

\[
\begin{align*}
\text{maximize} & & \sum_{i=1}^l \alpha_i - \frac{1}{2} \sum_{i,j=1}^l y_i y_j \alpha_i \alpha_j \langle x_i \cdot x_j \rangle \\
\text{s.t.} & & \sum_{i=1}^l y_i \alpha_i = 0, \\
& & \alpha_i \geq 0, i = 1, \ldots, l.
\end{align*}
\]

Then $w = \sum_{i=1}^l y_i \alpha_i^* x_i$ realizes the maximal margin hyperplane with geometric margin

$$
\gamma = \frac{1}{\|w\|}
$$

4.4.2 Linear Kernel

Using the linear kernel means that we have a linear decision hyperplane $g(x) = \langle w \cdot x \rangle + b$. When $A$ is projected onto the $g(x) = 0$, the projected point $P$ is expressed in terms of $A$, and $w$, weight of $g(x)$, that is, point $P$ is the sum of point $A$ and some constant times the weight $w$. Figure 4–2 shows this.

$$
P = A + t \cdot w.
$$
where $t$ is a constant. By the proposition 4.4.1,

$$\|t \cdot w\| = \frac{1}{\|w\|},$$

and by solving this in terms of $t$, we have

$$t = \pm \frac{1}{\|w\|^2}.$$ 

Hence,

$$|P - A| = \frac{|w|}{\|w\|^2}.$$

Let $p(x_i, g(x))$ denote the projection of $x$ onto $g(x)$. Since $w$ is $\nabla g(x_i)$, we have,

$$|p(x_i, g(x)) - x_i| = \frac{|w|}{\|w\|^2} = \frac{|\nabla g(x_i)|}{\|\nabla g(x_i)\|^2}, \forall i \in SV.$$

Since $|p(x_i, g(x)) - x_i| = a$ constant vector (normal vector) $\forall i \in SV$, we only need to compute

$$\frac{|w|}{\|w\|^2}.$$

The following Lemma 4.4.2 summarizes what we have done above.

**Lemma 4.4.2** Consider a linearly separable training sample

$$S = ((x_1, y_1), \cdots, (x_l, y_l)),$$

and suppose the parameters $\alpha^*$ and $b^*$ solve the following optimization problem:

$$\maximize \sum_{i=1}^l \alpha_i - \frac{1}{2} \sum_{i,j=1}^l y_i y_j \alpha_i \alpha_j (x_i \cdot x_j) \quad (4.3)$$

s.t.

$$\sum_{i=1}^l y_i \alpha_i = 0,$$

$$\alpha_i \geq 0, i = 1, \cdots, l. \quad (4.4)$$

Then $w = \sum_{i=1}^l y_i \alpha_i^* x_i$. For any support vectors $x_i$

$$|p(x_i, g(x)) - x_i| = \frac{|w|}{\|w\|^2}.$$
Proof: We can write $p(x_i, g(x))$ in terms of $x_i$, and the gradient vector of the decision hyperplane $g(x)$ at $x_i$ as follows:

$$p(x_i, g(x)) = x_i + t \cdot w,$$

where $t$ is some constant. By the Proposition 4.4.1,

$$\|t \cdot w\| = \frac{1}{\|w\|},$$

Solving the above in terms of $t$ gives:

$$t = \pm \frac{1}{\|w\|^2}.$$ 

Hence,

$$|p(x_i, g(x)) - x_i| = \frac{|w|}{\|w\|^2}.$$ 

4.4.3 General Kernels

Using the nonlinear kernel, we have a nonlinear decision hyperplane in the input space. Consider Figure 4–3.
Let $g(x)$ be the hyperplane and $P$ be the projected point of $A$. Observe that
$\nabla g(A)$ is normal to the $g(x)$ at the point $P$. $\nabla g(A)$ can be computed easily as in the SVM Gradient since

$$g(x) = \sum_{i \in SV} \alpha_i y_i K(x_i, x),$$

and

$$\nabla g(x) = \sum_{i \in SV} \alpha_i y_i \nabla x_i K(x_i, x).$$

Hence, the projected point $P$ is expressed in terms of $A$ and $\nabla g(A)$, normal to the $g(x)$ at $P$, that is, the point $P$ is the sum of point $A$ and some constant $t$ times the normal vector $\nabla g(A)$. Figure 4–3 shows this.

$$P = A + t \cdot \nabla g(A)$$

where $t$ is a constant. But here we do not calculate $|P - A|$ exactly since its computation is complicated and the exact calculation is not needed. Since $|P - A|$ is a constant times $\nabla g(A)$, $|P - A|$ is proportional to $\frac{|\nabla g(A)|}{\|\nabla g(A)\|^2}$, that is,

$$|P - A| \propto \frac{|\nabla g(A)|}{\|\nabla g(A)\|^2}.$$  

For $\forall i \in SV$,

$$|p(x_i, g(x)) - x_i| \propto \frac{|\nabla g(x_i)|}{\|\nabla g(x_i)\|^2}.$$  

Since $|p(x_i, g(x)) - x_i|$ is not a constant vector unlike the linear case, we need to calculate

$$p_i = c|p(x_i, g(x)) - x_i| = \frac{|\nabla g(x_i)|}{\|\nabla g(x_i)\|^2}, \forall i \in SV,$$

where $c$ is a constant. Now we sum the $p_i, \forall i \in SV$, component-wise. Let $d$ denote the component-wise summation of $p_i, \forall i \in SV$. The feature corresponding to the largest magnitude of the $d$ will then be the most important feature for the classification of
the $g(x)$. This feature selection criterion is now combined to the recursive feature elimination.

### 4.4.4 SVM Projection-RFE Algorithm

We now describe how to rank all the features. Initially, the machine is trained with all the features and all the training data. Then compute the feature selection criterion, that is, $d$. The feature with the minimum magnitude component is eliminated. This eliminated feature becomes ranking $n$. Now train the machine without the eliminated feature, recompute the $d$, and eliminate the feature with the minimum selection criterion. This feature becomes ranking $n - 1$. Recursively doing this process, we can give a ranking for all the features. The SVM Projection-RFE algorithm and computational details are as follows:

**Linear kernel.** Given training instances $X_{all} = [x_1, \cdots x_l]'$, and class labels $y = [y_1, \cdots y_l]'$, initialize the subset of features $s = [1, 2, \cdots n]$ and $r = $ an empty array.

Repeat (a)-(e) until $s$ becomes an empty array.

(a) Construct new training instances $X = X_{all}(; s)$

(b) Train SVM($X, y$) to get $g(x)$.  

(c) Compute the gradient $w = \nabla g(x)$.

\[
    w = \sum_{i \in SV} \alpha_i y_i x_i.
\]

(d) Find the feature $f$ with the smallest $w_j, j = 1, \cdots |s|$

\[
    f = \arg\min_j (|w_j|).
\]

(e) Update $r$ and eliminate the feature from $s$

\[
    r = [s(f), r],
\]

\[
    s = s - \{s(f)\}.
\]

For the linear kernel case, $\nabla g(x_i)$ is a constant vector, that is, the normal vector of the $g(x)$, for any $i \in SV$. Only a one-time computation of the normal vector of the $g(x)$ is therefore needed for feature selection criterion computation for one feature.
elimination. This algorithm is therefore exactly the same as the SVM-RFE and SVM Gradient-RFE.

**General kernels.** Given training instances \( X_{all} = [x_1, \ldots, x_l]' \), and class labels \( y = [y_1, \ldots, y_l]' \), initialize the subset of features \( s = [1, 2, \ldots, n] \) and \( r \) = an empty array.

Repeat (a)-(g) until \( s \) becomes an empty array.

(a) Construct new training instances \( X = X_{all}(; , s) \)

(b) Train SVM(\( X, y \)) to get \( g(x) \).

(c) Compute the gradient \( \nabla g(x), \forall x \in SV \),

\[
\nabla g(x) = \sum_{i \in SV} \alpha_i y_i \nabla_x K(x_i, x).
\]

(d) Compute \( p_i = c|p(x_i, g(x)) - x_i| \)

\[
p_i = \frac{|\nabla g(x_i)|}{\|\nabla g(x_i)\|^2}, \forall i \in SV.
\]

(e) Compute the sum of \( p_i, \forall i \in SV \).

\[
d = \sum_{i \in SV} p_i.
\]

(f) Find the feature \( f \) with the smallest \( d_j, j = 1, \ldots, |s| \)

\[
f = \text{argmin}_j(|d_j|).
\]

(g) Update \( r \) and eliminate the feature from \( s \)

\[
r = [s(f), r],
\]

\[
s = s - \{s(f)\}.
\]

Again, the \( \nabla g(x) \) computation can be done easily, as we did in Section 3.1.

**4.5 The M-fold SVM**

Fu et al. proposed a new feature selection algorithm: M-fold-SVM [39]. They particularly used \( M = 10 \) in their implementation. Some of the characteristics of M-fold-SVM are that it improves the reliability of the selected features and it uses
an automatic threshold for the number of features to be selected. They have used it for gene selection in microarray datasets and keyword selection in text datasets.

4.5.1 Binary-class M-fold-SVM

The binary-class M-fold-SVM makes use of the SVM-RFE algorithm. First, it randomly divides the training dataset into 10 disjoint subsets. The \(i^{th}\) subset is left out for testing the selected features’ performance trained on the remaining nine subsets. The nine subsets are used to train and select features. This process repeats 10 times. Then they collect all the features obtained from the 10 iterations.

**Binary-class M-fold-SVM Algorithm**

Let \(X_{all}\) be the given training examples where \(X_{all} = [x_1, \ldots, x_l]'\).

- **input :** \(X_{all}, Y\) : Input samples and labels.
- **output :** A subset of features .

Randomly divide \(X_{all}\) into 10 disjoint subsets.

**for** \(i = 1\) **to** 10 **do**

\(X_{tst} = i^{th}\) subset.

\(X_{trn} = X_{all} - X_{tst};\)

\(s = [1, \ldots, \text{dim}], r = \text{an empty array}.\)

**While** (\(s\) is not) **do**

Construct a new training set \(X = X_{trn}(; , s)\)

Train SVM(\(X\)) to get \(g(x)\).

Compute the gradient \(w = \nabla g(x)\).

\[
w = \sum_{i \in SV} \alpha_i y_i x_i.
\]

Find the feature \(f\) with the smallest \(|w_j|, j = 1, \ldots, |s|\)

\(f = \text{argmin}_j(|w_j|).\)

Update \(r\) and eliminate the feature from \(s\)

\(r = [s(f), r],\)
\[ s = s - \{s(f)\}. \]

end while

Select \textbf{topGenes} from \( r \) which gives the best accuracy on the training examples \( X_{trn} \) using SVM\((X_{trn})\).

Compute CV accuracy using SVM\((X_{tst}, \text{topGenes})\).

end for

Collect all the features selected. Sort them by repeatability.

In practice, some modifications of the above algorithm are useful. In the \textbf{while} loop, instead of removing one feature at one time, chunk of features are eliminated at a time. Especially at the early stage of the feature elimination, more features are removed. One variation is that They remove the features so that the number of remaining features become the closest power of 2 to the original number of features.

In the later elimination step, eliminate just half of the remaining features. This step is repeated until one feature is left. Another variation is the same as before, but when the number of remaining features becomes 256, we start to eliminate the features one at a time. This variation is useful since it does not take much time to eliminate one feature at a time since then. A large number of features are eliminated during the first several elimination steps. Important features should supposedly be in the 256 features. They are eliminated one by one and hence ranked more correctly.

\subsection*{4.5.2 Multi-class M-fold-SVM}

When the instances are multi-class, we can assume that the class labels are 1, 2, \ldots, \( k \). Basically, we run the binary-class M-fold-SVM \( k \) times. First, class 1 instances are labeled as +1 and the rest are labeled as -1. Then they run the binary-class M-fold-SVM to select the features with the same training instances and the new labels. Next, class 2 instances are labeled as +1 and the rest are labeled as -1. Then they run the binary-class M-fold-SVM with the same training instances and the new labels. The algorithm repeats this procedure \( k \) times. Finally they collect all the
features which are selected in each binary-class M-fold-SVM.

**Multi-class M-fold-SVM Algorithm**

input : $X, Y$ : Input samples and labels.
output : A subset of features.

for $i = 1$ to $k$ do

$Y_{new}(j) = 1$ if $j^{th}$ instance’s label is $i$.

$Y_{new}(j) = -1$ if $j^{th}$ instance’s label is not $i$.

run the **Binary-class M-fold-SVM** ($X, Y_{new}$).

end for

Collect the features selected in each **Binary-class M-fold-SVM** run.

### 4.6 Linear Programming Methods

Linear programming (LP) is a mathematical formulation to solve an optimization problem whose constraints and objective function are linear [4, 12, 13, ?, 69]. The LP formulations were suggested to solve the feature selection problem [12, 13]. Given two point sets, $A$ and $B$ in $R^n$ are represented by $A \in R^{m \times n}$ and $B \in R^{k \times n}$. This problem is formulated as following robust linear programming (RLP) [13].

\[
\min_{w, \gamma, y, z} \frac{e^T y}{m} + \frac{e^T z}{k} \quad (4.5)
\]

\[-Aw + e\gamma + e \leq y, \]

s.t \quad $Bw - e\gamma + e \leq z, \quad (4.6)$

$y \geq 0, z \geq 0.$

### 4.6.1 Feature Selection via Concave Minimization

A subset of features is obtained by attempting to suppress as many components of the normal vector $w$ to the separating plane $P$, which is consistent with obtaining an acceptable separation between sets $A$ and $B$ [13]. This can be achieved by introducing an extra term with the parameter $\lambda \in [0, 1)$ into the objective in RLP while weighting...
the original objective by \((1 - \lambda)\).

\[
\begin{align*}
\min_{w, y, z, v} & \quad (1 - \lambda)(\frac{e^T y}{m} + \frac{e^T z}{k}) + \lambda e^T v_* \\
\text{s.t} & \quad -A w + e\gamma + e \leq y, \\
& \quad B w - e\gamma + e \leq z,
\end{align*}
\]  

(4.7)

Because of the discontinuity of \(e^T v_*\), this term is approximated by a concave exponential on the nonnegative real line:

\[
v_* \approx t(v, \alpha) = e - \epsilon^{-\alpha v}, \alpha > 0
\]

This leads to the Feature Selection Concave minimization (FSV):

\[
\begin{align*}
\min_{w, y, z, v} & \quad (1 - \lambda)(\frac{e^T y}{m} + \frac{e^T z}{k}) + \lambda e^T (e - \epsilon^{-\alpha v}) \\
\text{s.t} & \quad -A w + e\gamma + e \leq y, \\
& \quad B w - e\gamma + e \leq z,
\end{align*}
\]  

(4.8)

4.6.2 SVM \(\| \cdot \|_p\) Formulation

Bradley et al. [12, 13] have tried several LP formulations which vary depending on what norm to measure the distance between two bounding planes. The distance, measured by some norm \(\| \cdot \|\) on \(R^n\), is \(\frac{2}{\|w\|}\). Add the reciprocal of this term, \(\|w\|^{-2}\), to
the objective function of RLP.

\[
\min_{w, \gamma, y, z} \left( 1 - \lambda \right) (e^T y + e^T z) + \frac{\lambda}{2} \| w \|^2
\]

\[ (4.11) \]

\[-A w + e \gamma + e \leq y,\]

\[ \text{s.t} \]

\[ B w - e \gamma + e \leq z, \quad (4.12) \]

\[ y \geq 0, z \geq 0. \]

One uses the \( \infty \)-norm to measure the distance between the planes. Since the dual norm of the \( \infty \)-norm is 1-norm, we call the LP formulation as SVM \( \| \cdot \|_1 \). The SVM \( \| \cdot \|_1 \) formulation is as follows:

\[
\min_{w, \gamma, y, z, s} \left( 1 - \lambda \right) (e^T y + e^T z) + \frac{\lambda}{2} e^T s
\]

\[ (4.13) \]

\[-A w + e \gamma + e \leq y,\]

\[ \text{s.t} \]

\[ B w - e \gamma + e \leq z, \quad (4.14) \]

\[ -s \leq w \leq s, \]

\[ y \geq 0, z \geq 0. \]

In the SVM \( \| \cdot \|_p \) formulation, the objective functions attempt to balance between the number of misclassified instances and the number of non-zero \( w \)’s while minimizing the sum of these two values.

Similarly, if one uses the 1–norm to measure the distance between the planes, then the dual to the norm is the \( \infty \)-norm and SVM \( \| \cdot \|_\infty \) formulation as follows:

\[
\min_{w, \gamma, y, z} \left( 1 - \lambda \right) (e^T y + e^T z) + \frac{\lambda}{2} \nu
\]

\[ (4.15) \]

\[-A w + e \gamma + e \leq y,\]

\[ \text{s.t} \]

\[ B w - e \gamma + e \leq z, \quad (4.16) \]

\[-e \nu \leq w \leq e \nu, \]

\[ y \geq 0, z \geq 0. \]
The authors also attempted SVM $\| \cdot \|_2$. They reported that only FSV and SVM $\| \cdot \|_1$ gave small subsets of features and all other formulations ended up with no feature selections.

Solving the linear programming feature selection problem gives a subset of features with which the separating plane is linear in input space, and in SVM recursive feature elimination methods it could be nonlinear in input space, as well as linear in input space. Also, one does not have a choice for the number of features one desires since the solution for SVM $\| \cdot \|_p$ simply gives a subset of features. The SVM recursive feature elimination methods give ranking for all the features so one can choose top $k$ features if one wanted to choose a subset which consists of $k$ features.

### 4.7 Correlation Coefficient Methods

Evaluating how well an individual feature contributes to the separation can produce a simple feature ranking. Golub et al. [42] used

$$w_i = \frac{\mu_i(+) - \mu_i(-)}{\sigma_i(+) + \sigma_i(-)}.$$  

Pavlidis [74] used

$$w_i = \frac{(\mu_i(+)-\mu_i(-))^2}{\sigma_i(+)^2 + \sigma_i(-)^2}.$$  

The $\mu_i$ and $\sigma_i$ in the above are the mean and standard deviations regarding feature $i$. A large positive value in $w_i$ indicates a strong correlation with the (+) class and a large negative value with the (-) class. Each coefficient $w_i$ is computed with information about a single feature and does not consider mutual information between features. These methods assume the orthogonality assumption implicitly [44]. What it means is following. Suppose one wants to find two features which give a best classifier error rate among all combinations of two features. In this case, the correlation coefficient method finds two features which are individually good, but those two features may not be the best two features cooperatively. The SVM recursive feature elimination methods certainly do not have this problem.
4.8 Feature Scaling Methods

Some authors suggest using the leave-one-out (LOO) bounds for SVMs as feature selection criteria [94]. The methods search all possible subsets of $n$ features which minimize the LOO bound, where $n$ is the total number of features. But this minimum finding requires solving a combinatorial problem, which is very expensive when $n$ is large. Instead, it scales each feature by a real variable and computes this scaling via a gradient descent on the LOO bound. One keeps the features with the largest scaling variables. The authors incorporate scaling factors into the kernel:

$$K_{\sigma}(x, y) = K((x * \sigma), (y * \sigma))$$

where $x * \sigma$ is an element-wise multiplication. The algorithm is:

1. Solve the standard SVM to find $\alpha$’s for a fixed $\sigma$.
2. Optimize $\sigma$ for fixed $\alpha$’s by a gradient descent.
3. Remove features corresponding to small elements in $\sigma$ and return to step 1.

This method may end up with a local minimum depending on the choice of the gradient step size [44].

4.9 Dimension Reduction Methods

In this section, we examine methods in dimension reduction. Two representative methods are principal component analysis (PCA) [57, 2] and projection pursuit (PP) [35, 51, 56, 58]. The PCA aims to minimize the error by doing some transformation to another coordinate system, and PP aims to find interesting low-dimensional linear projections of high-dimensional data.

4.9.1 Principal Component Analysis

Let us start by introducing how to calculate the PCs. Consider a sample $x = [x_1, x_2, \cdots, x_n]'$. Assume without loss of generality $E[x_i] = 0, \forall i = 1, \cdots, n$. The covariance matrix $\Sigma$ of $x$ is $E(xx')$. The first principal component (PC): $Y_{(1)} = \alpha_1'x$. $\alpha_1$ can be found by maximizing the variance of the first PC $\alpha_1'x$ under the constraint
\( \alpha_1^t \alpha_1 = 1 \). I.e,

\[
\max_{\alpha_1} \quad V(\alpha_1^t x) \\
\text{s.t} \quad \alpha_1^t \alpha_1 = 1
\]  

(4.17)

Since

\[
V(\alpha_1^t x) = E((\alpha_1^t x)(\alpha_1^t x)^t) \\
= \alpha_1^t E(xx^t) \alpha_1 \\
= \alpha_1^t \Sigma \alpha_1
\]

the problem (4.17) can be rewritten as follows:

\[
\max_{\alpha_1} \quad \alpha_1^t \Sigma \alpha_1 \\
\text{s.t} \quad \alpha_1^t \alpha_1 = 1
\]  

(4.18)

By using the Lagrangian function, problem (4.18) becomes

\[
\max_{\alpha_1} \quad \alpha_1^t \Sigma \alpha_1 - \lambda(\alpha_1^t \alpha_1 - 1)
\]  

(4.19)

where \( \lambda \) is a Lagrange multiplier. The stationary condition states

\[
\Sigma \alpha_1 - \lambda \alpha_1 = (\Sigma - \lambda I) \alpha_1 = 0
\]

Since \( \alpha_1 \neq 0 \), \( \det(\Sigma - \lambda I) = 0 \). Hence \( \lambda \) is an eigenvalue of \( \Sigma \). Let \( \alpha_1 \) be an eigenvector associated with \( \lambda \). Then \( \max \alpha_1^t \Sigma \alpha_1 = \max \alpha_1^t \lambda \alpha_1 = \max \lambda \alpha_1^t \alpha_1 = \max \lambda \). So \( \lambda \) is the largest eigenvalue of \( \Sigma \), and \( \alpha_1 \) is the eigenvector associated with \( \lambda \). By similar formulations, one can find all the PCs and can discard the variables with small variance so that one can achieve the dimension reduction. But this is different from feature selection. In feature selection, we want to find a small number of features from the original input space. But the dimension reduction can be done by transformation using the largest PCs. But to find the PCs, one has to make use
of all the features. So there can not be a subset with small number of features in the original input space.

4.9.2 Projection Pursuit

The PP aims to discover an interesting structure projection of a multivariate data set, especially for the visualization purpose of high-dimensional data. In basic PP, one tries to find the direction $w$ so that the $x^T x$ has an interesting distribution. We do not go into computational details for PP, but just mention that like the PCA, PP cannot be a feature selection method by the same reason as PCA.
CHAPTER 5
SUPPORT VECTOR FEATURE SELECTION

Support vectors are the samples which are close to the decision boundary in SVM. In SVM, only those samples are involved in the decision hyperplane computation. The SVM produces the same decision hyperplane without non-support vectors. The support vector is now overloaded to mean those points which are close to the decision boundary in any discriminant analysis. In this chapter, we show how to do feature selection with only support vectors. While support vector feature selection can be applied to any discriminant analysis, in this dissertation we considered two particular linear discriminants, SVM and Fisher’s linear discriminant.

We present two algorithms based on these two linear discriminants. One is termed SV-RFE SVM, Support Vector Recursive Feature Elimination SVM. A similar naming can be done on FLD. The idea for SV feature ranking/selection is to first identify the support vectors and to apply the ranking schemes, such as recursive feature elimination or simple non-recursive ranking, on these support vectors and build a classifier using the SVs and features from the ranking.

5.1 Datasets

We have used several datasets, obtained from public domains. They are varied in terms of the number of samples and the dimension of the sample.

The colon cancer data consist of 62 instances, each instance with 2,000 genes [1, 21, 44, 86]. The dataset has 22 normal colon tissue examples and 40 colon tumor examples. It is analyzed with an Affymetrix oligonucleotide array complementary to more than 6,500 human genes and expressed sequence tags. In general, this kind of data and analysis is called a microarray data analysis. Statistical methods [3, 32, 52, 71] and SVM [16, 40, 68, 98] have been extensively used as the microarray analysis.
tool. In the colon cancer dataset, the 2,000 genes selected have the highest minimal intensity across the examples. The colon cancer data are $62 \times 2,000$ matrix. Each entry of the matrix is a gene expression level. Since the dataset is not set aside for a training set or a testing set, we randomly split them into two 31 examples. One of the two 31-example sets is used for training the machine, and the other 31-example set is used for a testing set.

The leukemia dataset consists of a $72 \times 7,129$ matrix [42]. The feature selection chooses the genes from 7,129 genes. Since this dataset is already divided into training and test sets by the original authors, we follow their split. Out of 72 examples, 38 of them were used as training examples and the rest (34 examples) as testing.

The MNIST dataset of handwritten digits ([http://yann.lecun.com/exdb/mnist](http://yann.lecun.com/exdb/mnist)) is publicly available. This dataset contains 60,000 training samples. Among them, digits 2 and 3 are extracted. Again among them, only 300 samples for each are selected in our experiments to become a training dataset. And 150 samples for each are selected as a test dataset, that is, 600 and 300 samples as a training dataset and test dataset, respectively. Each sample is a 28x28 image, or equivalently, a 784-dimensional row vector. Each pixel in an image has a value in [0,255]. Figure 5–1 shows sample images of digits 2 and 3.

The sonar dataset was used by Gorman and Sejnowski in their research on classification using neural network [43]. The classification task is to assign a new sample as a metal cylinder (-1) or a rock (+1). Each pattern is a set of 60 numbers in the range 0.0 to 1.0. There are 208 samples, with 60 features in each sample. Among
Table 5–1: Training/Test Data Splits Four Datasets

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Dimension</th>
<th># Training Samples</th>
<th># Test Samples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Leukemia</td>
<td>7,129</td>
<td>38</td>
<td>34</td>
</tr>
<tr>
<td>Colon Cancer</td>
<td>2,000</td>
<td>31</td>
<td>31</td>
</tr>
<tr>
<td>MNIST</td>
<td>784</td>
<td>600</td>
<td>300</td>
</tr>
<tr>
<td>Sonar</td>
<td>60</td>
<td>104</td>
<td>104</td>
</tr>
</tbody>
</table>

them, 97 belong to +1 and 111 belong to -1. In our experiment, we randomly split them evenly into training and test data. Also, data are class balanced, that is, 97 +1 samples are almost evenly divided into training and test data. Table 5–1 summarizes training and test data splits for four datasets.

5.2 Support Vector Feature Selection Using FLD

Fisher’s linear discriminant has long been used as a simple linear classifier [10, 31, 48]. Its performance was often outperformed by many other classifiers such as SVM [44]. A study, however, shows the equivalence between linear SVM and Fisher’s linear discriminant on the support vectors found by SVM [81]. This result has been of theoretical interest but not utilized or developed for practical algorithms. Recently Cooke [23] showed that Fisher’s linear discriminant can do better classification by working only on support vectors, and he showed how support vectors can be identified by applying Fisher’s linear discriminant recursively. Chakrabarti et al. applied multiple Fisher’s linear discriminants for text classification [18]. In the subsequent sections, we show how Fisher’s linear discriminant on only support vectors can give a better feature ranking/selection.

5.2.1 Fisher’s Linear Discriminant

Fisher’s linear discriminant [10, 31] tries to find one-dimensional projection. Suppose the data are binary class, and all the data points are transformed to one dimension using an unknown vector \( \mathbf{w} \). The method attempts to find a vector which maximizes the difference of the \textit{between-class} averages from the transformed data while minimizing the sum of the \textit{within-class} variances from the transformed data. This is
an unconstrained optimization problem and the $w$ can be obtained from
\[ w \propto S_W^{-1}(m_{-1} - m_{+1}), \]
where $m_{+1}(m_{-1})$ is the mean vector of class $+1(-1)$ and $S_W$ is the total within-class covariance matrix and given by
\[ S_W = \sum_{i \in +1} (x_i - m_{+1})(x_i - m_{+1})^T + \sum_{i \in -1} (x_i - m_{-1})(x_i - m_{-1})^T. \]
If $S_W$ is singular, the pseudo-inverse [46] of $S_W$ is computed. Although Fisher’s linear discriminant is not strictly a discriminant function, by devising appropriately, Fisher’s criterion can serve as a linear discriminant function. To devise a linear decision function we still need a bias term. The bias term can be the transformed point of the overall mean of the data points. Hence, we have a Fisher’s linear discriminant
\[ g(x) = w \cdot (x - m) = \sum_{i=1}^{d} w_i \cdot x_i + \sum_{i=1}^{d} w_i \cdot ((m_i(+)) + m_i(-))/2). \]
Here, the larger absolute value of $w_i$ has more impact on the classification decision. We use this $w$ as a ranking score criterion. This issue will be reviewed in detail in Chapter 7.

5.2.2 Support Vector Feature Selection with FLD

The support vector-based feature selection was hinted by the success of SVM [5, 16, 40, 44, 55, 99], the proof of equivalence between SVM and FLD on support vectors, and successful applications of FLD on support vectors by Cooke [23] and Chakrabarti et al. [18]. These imply that discriminants can do better classification by constructing the classifier based on support vectors. The SVM has been proven to be one of the most powerful discriminants. Cooke showed that a variation of FLD also can do competitive classification as SVM. Since discriminants using only support vectors have more discriminating power, feature ranking/selection obtained from more discriminating classifiers should give a better feature selection. Shashua
[81] showed the equivalence between SVM and FLD on support vectors by proving that for \( S \), the set of support vectors and \( w = \sum \alpha_i y_i x_i \) in (2.8), then \( w \) is the null space of \( S_W \) (\( S_W \cdot w = 0 \)), where \( S_W \) is the total within-class covariance matrix of \( S \). Cooke did not mention the relationship of his work to Shashua’s proof. It, however, seems that the better performance of Cooke’s recursive FLD is theoretically supported by Shashua’s proof. Figure 5–2 shows the idea of recursive FLD suggested by Cooke [18]. In Figure 5–2, the upper figure shows the original data and the lower figure shows projections of the data points into one-dimensional space using FLD.

There are three regions in the lower figure, left, middle, and right. Points in the left region are only from class \(-1\), and points in the right region are only from class \(-1\). After removing these points, the points in the middle are run on FLD to find a one-dimensional projection. This repeats in a recursive way. We show the recursive FLD
and SV-RFE FLD algorithms. The recursive FLD essentially is based on Cooke’s.

**SV-RFE FLD Algorithm**

1. Identify the support vectors using the recursive **FLD**

   **input**: 
   
   \[ T : \text{Surviving threshold} \]
   
   \[ S : \text{Decreasing rate on number of surviving samples} \]
   
   \[ X, Y : \text{Input samples and labels} \]

   **output**: \(svX, svY\) : Support vectors and corresponding labels

   **initialization**: \(svX := X, svY := Y\)

   **while** the fraction of surviving samples \((svX) > T\) **do**

   Find \(w\) by running \(\text{FLD}(svX, svY)\)

   Find one-dimensional points by projecting \(svX\) on \(w\)

   Keep \(S\) percent of the points from each class which are closest to the decision boundary

   Set \(svX, svY\) as surviving samples and corresponding labels

   **end while**

2. Rank the features by applying RFE on the support vectors found

   **input**: \(svX, svY\) : Input samples and labels

   **output**: rank

   **initialization**: rank = empty list, \(s = (1, 2, \ldots, j, \ldots, \text{dim})\)

   **while** \(s\) is not empty **do**

   Construct new samples, \(\text{newX}\) whose features contain only \(s\) from \(svX\)

   Find \(w\) by applying \(\text{FLD}(\text{newX}, svY)\)

   Find the feature \(j\) such that \(w_j\) has the minimum value among \(|w|\)

   Update \(s\) by removing the feature \(j\) from \(s\)

   Update rank by adding the feature \(j\) on top of the rank

   **end while**
5.2.3 Experiments

In the first experiment, we applied FLD and found ranking based on the absolute values of weight of Fisher’s linear discriminant. We call this as Simple All. In our approach, we first found the support vectors using FLD and then applied FLD on only those support vectors. When we found the support vectors, we used $T = 0.25$ and $S = 0.9$. We used the same parameters for all the experiments. We did not apply any normalization on the MNIST dataset. For Simple SV, we first found 144 support vectors. Compared to Simple All in terms of time, Simple SV took about 150% more than Simple All. Here, time includes finding support vectors, ranking computation and testing time. But, spending this extra time is worthwhile. When we look at the accuracy comparison, Simple SV gave about 30% more accuracy than that of Simple All. This is shown in the second row (Avg. Acc.) in Table 5–2.

The same experimental setting was applied to the sonar dataset. As in MNIST, we did not apply any normalization on the sonar dataset. Simple SV gave 3.25% more accuracy than that of Simple All at the expense of about 20% more time. Most of this extra time was initially due to finding support vectors.

We then applied the recursive feature elimination technique. We applied SV-RFE on the MNIST dataset. This resulted in 144 support vectors out of 600. In Table 5–4, the Time row includes the time taken to find the support vectors, recursive
Table 5–4: All-RFE vs. SV-RFE Performance on MNIST Data (FLD)

<table>
<thead>
<tr>
<th></th>
<th>All RFE (1)</th>
<th>SV-RFE (2)</th>
<th>[(1)-(2)]/(1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Avg. Acc.</td>
<td>82.41%</td>
<td>84.36%</td>
<td>-2.37%</td>
</tr>
<tr>
<td>#Samples</td>
<td>600</td>
<td>144</td>
<td>76.00%</td>
</tr>
<tr>
<td>Time</td>
<td>289.52 sec</td>
<td>594.22 sec</td>
<td>-105.24%</td>
</tr>
</tbody>
</table>

Table 5–5: All RFE vs. SV-RFE Performance on Sonar Data (FLD)

<table>
<thead>
<tr>
<th></th>
<th>All-RFE (1)</th>
<th>SV-RFE (2)</th>
<th>[(1)-(2)]/(1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Avg. Acc.</td>
<td>66.81%</td>
<td>68.39%</td>
<td>-2.37%</td>
</tr>
<tr>
<td>#Samples</td>
<td>104</td>
<td>22</td>
<td>78.85%</td>
</tr>
<tr>
<td>Time</td>
<td>84.51 sec</td>
<td>85.39 sec</td>
<td>-1.04%</td>
</tr>
</tbody>
</table>

ranking, and training and testing time. In the table, SV-RFE took almost twice as much as All-RFE because identifying the support vectors is time-consuming due to its repeated inverse computation. At the expense of this extra time spending, the accuracy gain on SV-RFE is about 2.37%.

The next experiment was on the sonar dataset. We first found 22 support vectors. Despite about an 80% training data reduction, the total time taken for SV-RFE was slightly more than that of All-RFE because finding the support vectors takes time. The accuracy gain of SV-RFE is 2.37% over All-RFE.

5.2.4 Discussion

We have shown the comparison of performance between feature selection using all the samples and one with only support vectors. Whether it is simple ranking or recursive ranking, feature selection on only support vectors gave much better accuracy. The only drawback of the support vector-based feature ranking is just a little more time spent because we needed to decide which samples are support vectors. The performance difference was sometimes significant when we applied the simple feature ranking on the MNIST dataset. At the expense of a little extra time spent, we achieved almost a 30% accuracy gain. When applying FLD on a dataset, we need to find the inverse of $S_W$, the total within-class covariance matrix. The order of the matrix is the dimension of the sample in the dataset. Among our datasets, samples
in two microarray datasets are very high dimensional. The order of the covariance matrix will be several thousand. Its inverse computation is very time-consuming and hence impractical. This is the reason why we did not apply FLD on the two microarray datasets. However, we are aware that there are linear time (in the size of input data) algorithms for computing the inverse for FLD. One of the successful algorithms is the hill-climbing one [18]. It would be necessary to use such linear time algorithms for FLD to be practical for large data. But we did not explore this issue in this dissertation.

5.3 SV-RFE SVM

The recursive feature elimination technique has long been used as a feature subset selection method [48, 88]. Guyon et al. [44] successfully applied the RFE with SVM (Guyon RFE). In order to apply the SV feature selection, it is necessary to identify the support vectors for the given training dataset. The SVM, however, finds support vectors in the process of building a classifier. So our first job is done by SVM. Then we apply the RFE technique on these SVs. We consider only RFE on all samples and support vectors. Unlike FLD, we do not consider simple ranking on all samples and support vectors since SVM always find support vectors first. Hence, simple rankings on all samples and support vectors are exactly the same. The SV-RFE SVM algorithm is introduced as follows.

SV-RFE SVM Algorithm

input : X,Y : Input samples and labels
output : rank

1. Identify the support vectors
   
   Identify support vectors by running SVM(X,Y)

   Set svX, svY as the support vectors and corresponding labels

2. Rank the features by applying RFE on the support vectors found
\textbf{while} \textbf{s} is not empty \textbf{do}

Construct new samples, \textbf{newX} whose features contain only \textbf{s} from \textbf{svX}

Find \textbf{w} by applying SVM(\textbf{newX},\textbf{svY})

Find the feature \(j\) such that \(w_j\) has the minimum value among \(|\textbf{w}|\)

Update \(\textbf{s}\) by removing the feature \(j\) from \(\textbf{s}\)

Update \textbf{rank} by adding the feature \(j\) on top of the \textbf{rank}

\textbf{end while}

5.3.1 Experiments

We applied the SV-RFE SVM on four different datasets and compared the performances with Guyon RFE. The performance measures are accuracy for test datasets and training time (including feature ranking) and testing time.

Typically features are eliminated one by one at each new training. In the leukemia dataset, each sample has 7,129 features. Hence, one-by-one recursive feature elimination may not be feasible. In practice, some modifications are useful. Instead of eliminating one feature at a time, chunks of features are eliminated at one time. Especially at the early stage of the feature elimination, more features are eliminated. One variation is that we eliminate the features so that the number of remaining features becomes the closest power of two to the original number of features. In the later elimination step, we eliminate just half of the remaining features. This step is repeated until one feature is left. Another variation is the same as before, but when the number of remaining features becomes 256, we start to eliminate the features one at a time. This variation is practical since it does not take much time to eliminate one feature at a time since then. A large number of features are eliminated during the first several elimination steps. Supposedly important features should be in the 256 features, and they are eliminated one by one, and therefore ranked more correctly.
Table 5–6: Guyon RFE vs. SV-RFE Accuracy on Leukemia Data (SVM)

<table>
<thead>
<tr>
<th>#Features</th>
<th>Guyon RFE (%)</th>
<th>SV-RFE (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>94.12</td>
<td>94.12</td>
</tr>
<tr>
<td>4</td>
<td>94.12</td>
<td>94.12</td>
</tr>
<tr>
<td>8</td>
<td>97.06</td>
<td>97.06</td>
</tr>
<tr>
<td>16</td>
<td>91.18</td>
<td>97.06</td>
</tr>
<tr>
<td>32</td>
<td>97.06</td>
<td>100.00</td>
</tr>
<tr>
<td>64</td>
<td>97.06</td>
<td>94.12</td>
</tr>
<tr>
<td>128</td>
<td>94.12</td>
<td>94.12</td>
</tr>
<tr>
<td>256</td>
<td>91.18</td>
<td>91.18</td>
</tr>
<tr>
<td>512</td>
<td>91.18</td>
<td>91.18</td>
</tr>
<tr>
<td>1024</td>
<td>94.12</td>
<td>94.12</td>
</tr>
<tr>
<td>2048</td>
<td>94.12</td>
<td>94.12</td>
</tr>
<tr>
<td>4096</td>
<td>94.12</td>
<td>94.12</td>
</tr>
<tr>
<td>7129</td>
<td>94.12</td>
<td>94.12</td>
</tr>
<tr>
<td><strong>Acc. Avg</strong></td>
<td><strong>94.12</strong></td>
<td><strong>94.57</strong></td>
</tr>
</tbody>
</table>

Table 5–7: Guyon RFE vs. SV-RFE Time on Leukemia Data (SVM)

<table>
<thead>
<tr>
<th></th>
<th>Guyon RFE (1)</th>
<th>SV-RFE (2)</th>
<th>[(1)-(2)]/(1)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>#Samples</strong></td>
<td>38</td>
<td>26</td>
<td>32.58%</td>
</tr>
<tr>
<td><strong>Time</strong></td>
<td>5.86 sec</td>
<td>4.00 sec</td>
<td>31.74%</td>
</tr>
</tbody>
</table>

We chose the latter on the leukemia dataset. Once ranking is done, accuracies on test dataset are computed using top 2, 4, 8, 32, ..., $2^k$, ..., 7129 features on the ranking. In this experiment, each sample is normalized so that the resulting sample becomes unit length. Table 5–6 shows the performance comparison between Guyon SVM and SV-RFE. In the experiment shown in Table 5–6, we used linear kernel SVM and $C = 100$, where $C$ is the upper bound for $\alpha$ in the SVM formulation. When we applied SV-RFE, we initially set $C = 1$ to get more support vectors. This resulted in 26 SVs and we started SV-RFE with these 26 samples while 38 SVs in Guyon RFE. These numbers are shown in the second row (#Samples) in Table 5–7. So we can expect that this difference $((38 - 26)/38 = 32.58\%)$ should be proportionally reflected in the training and testing time. The SV-RFE is 31.74% faster than Guyon RFE on the leukemia dataset. Table 5–7 shows almost exactly this proportionality in the last
Table 5–8: Guyon RFE vs. SV-RFE Performance on Colon Cancer Data (SVM)

<table>
<thead>
<tr>
<th></th>
<th>Guyon RFE (1)</th>
<th>SV-RFE (2)</th>
<th>(1)-(2)/ (1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Avg. Acc.</td>
<td>91.20%</td>
<td>91.38%</td>
<td>-0.2%</td>
</tr>
<tr>
<td>#Samples</td>
<td>31</td>
<td>29</td>
<td>6.45%</td>
</tr>
<tr>
<td>Time</td>
<td>5.55 sec</td>
<td>5.15 sec</td>
<td>7.21%</td>
</tr>
</tbody>
</table>

row. This is the exact savings of SV-RFE over Guyon RFE in training and testing time. There is more than time savings of SV-RFE. Table 5–6 shows that SV-RFE is better than Guyon RFE on average accuracy.

Now Guyon RFE and SV-RFE are applied to the colon cancer dataset. We have done our experiments with the linear kernel. Before conducting the experiment, all the examples again went through a unit-length normalization, that is, for an example x, a unit-length normalization is x/||x||. We have computed accuracies on top 1, 2, 3, · · ·, 49, 50, 100, 200, 300, · · ·, 1900, 2000 features. The complete accuracy results are not shown. Results on top-ranked features will be shown in Chapter 7. Table 5–8 shows the average accuracies, the number of samples, and training and testing time for both methods. Again, the proportionality in the #Samples is reflected almost exactly as the training and testing time difference between both methods. The SV-RFE is 7.2% faster than Guyon RFE while the average accuracy is slightly better (by 0.2%). In this colon cancer experiment, we used C = 30 on both methods. When we get the initial support vectors, we again used C = 1. This resulted in 29 support vectors before we started to apply RFE. Twenty nine out of 31 means that almost all the training samples are support vectors, and that is why SV-RFE gained a little training and testing time savings. Compared to the leukemia dataset, the colon cancer dataset is more difficult in terms of class separation. This difficulty was reflected in the much higher rate of support vectors than the one in the leukemia dataset.

The MNIST data are images. Each image consists of 28 by 28 pixels. This 28 by 28 pixel matrix is concatenated to become a 784-dimensional row vector. As with the previous two experiments, each sample is unit length normalized before conducting
Table 5–9: Guyon RFE vs. SV-RFE Performance on MNIST Data (SVM)

<table>
<thead>
<tr>
<th></th>
<th>Guyon RFE (1)</th>
<th>SV-RFE (2)</th>
<th>[(1)-(2)]/(1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Avg. Acc.</td>
<td>96.73%</td>
<td>96.69%</td>
<td>+ 0.04%</td>
</tr>
<tr>
<td>#Samples</td>
<td>600</td>
<td>87</td>
<td>85.50%</td>
</tr>
<tr>
<td>Time</td>
<td>404.89 sec</td>
<td>60.68 sec</td>
<td>85.01%</td>
</tr>
</tbody>
</table>

Table 5–10: Guyon RFE vs. SV-RFE Performance on Sonar Data (SVM)

<table>
<thead>
<tr>
<th></th>
<th>Guyon RFE (1)</th>
<th>SV-RFE (2)</th>
<th>[(1)-(2)]/(1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Avg. Acc.</td>
<td>76.17 %</td>
<td>76.20 %</td>
<td>- 0.04 %</td>
</tr>
<tr>
<td>#Samples</td>
<td>104</td>
<td>63</td>
<td>39.42 %</td>
</tr>
<tr>
<td>Time</td>
<td>8.85 sec</td>
<td>5.41 sec</td>
<td>38.88 %</td>
</tr>
</tbody>
</table>

experiments. In this experiment, accuracies for top 1, 2, 3, ⋯, 249, 250, 300, 350, ⋯, 750, 784 were computed. The accuracy for the top-ranked features are shown in Chapter 7. For the experiment on MNIST data, we used $C = 30$. Unlike the previous two experiments, we also used $C = 30$ to get initial support vectors. The number of initial support vectors was 87 and shown in the #Samples row in Table 5–9. Relatively very small portions of samples are support vectors. From this fact, we may guess the data are not hard in terms of linear separability, and higher generalization accuracy is expected. This justifies our choice of the later $C$ values. A very small rate of support vectors directly contributed to very large training and testing time savings by 85% while their average accuracies are no different (only 0.04%). Two factors contribute to the large savings on training and testing time: relatively easier data in terms of linear separability and large training data. Easier separability means that only a small number of samples lie around the decision boundary and become support vectors. So the rate of support vectors is high, and this effect is compounded with large training data size.

The last experiment is done on the sonar data. Unlike the previous experiments, no normalization was conducted on this dataset. We used $C = 4$. For SV-RFE, we used $C = 2$ to get the initial support vectors. So we started with 63 support vectors to apply RFE. On sonar data, SV-RFE gave a competitive test accuracy (actually
slightly better) while cutting the training and testing time by 40% over Guyon SVM.

5.3.2 Discussion

We have a couple of points to mention. First, we can reduce 7% to 85% in training and testing time by using SV-RFE, and this reduction rate is proportional to the number of support vectors. For the larger training data, the more linearly separable data, SV-RFE has the larger reduction in terms of ranking, training, and testing time.

Second, we had to choose the initial $C$, the tradeoff parameter between a good generalization error and a larger margin. For this purpose, we show how $C$ affects the number of support vectors in detail.

Figure 5–3 shows the effect of $C$ on support vectors. We remind that $C$ is the upper bound for the Lagrange multiplier $\alpha$, and each sample has a corresponding Lagrange multiplier $\alpha$ in SVM formulation. Thus, small $C$ value influences each
sample smaller so that we have larger non-zero $\alpha$’s. This means we have more support vectors. If we use an extremely small $C$ value, then all the samples have almost the same influence to build a decision boundary regardless of their closeness to the decision boundary so that almost all the samples become support vectors. In Figure 5–3, the upper figure is somehow close to this case. The large number of support vectors are due to a small $C$. The created decision boundary misclassifies a couple of training points while training data are apparently linearly separable. But the decision boundary looks more reflective to the training points distribution. If the $C$ is extremely small, then the resulting decision boundary might become an average case decision hyperplane. No maximum margin concept is observed then. Obviously, we want to avoid this situation.

On the other hand, if we use a large $C$, then some samples close to the boundary between two classes can get large $\alpha$’s and other samples get small $\alpha$’s so that the influence of those close points decides the decision boundary. This may result in a good training error, but may cause overfitting. In Figure 5–3, the lower figure is close to this case. It tries to get a separable hyperplane and those three support vectors (‘+’) got a high $\alpha$, and hence these support vectors decide a decision hyperplane. Although it got a separating hyperplane (a good training error), the hyperplane does not reflect overall training points (poor generalization).

For this reason $C$ is called a tradeoff parameter between good generalization error and good training error, or equivalently a tradeoff between good training error and a larger margin. From this discussion, we can say the initial $C$ should be less than or equal to the $C$ in the main training algorithm. In our experiments, we used the initial $C = \text{main } C$ or the initial $C = 1$, or 2, but not greater than the main $C$. 
The Bayesian method is one of the most extensively used machine learning methods \cite{66, 85}. Among them, the naive Bayes classifier is popular in text document classification. Its successful applications to text document datasets have been shown in many research papers \cite{54, 65, 66, 75}. In this chapter, we point out several systemic problems related to the naive Bayes. Some of these problems arise only when the naive Bayes is used with just a small subset of features, for example, when we try to use the naive Bayes with just one or two features. This problem was not observed before because they did not try to apply the naive Bayes as a feature selection method. We suggested how to resolve these problems. Although the naive Bayes classifier was successful as a simple and easy tool for classification, there was room for improvement. We propose an improved naive Bayes classifier and show its effectiveness by experimental results on text data.

Despite its successful applications to text document categorization problems, the feature selection using the naive Bayes classifier has not been given any attention. Here, we have developed a feature selection method based on the improved naive Bayes approach, and we have showed its application to text document datasets.

### 6.1 Naive Bayes Classifier

Before we start this section, we introduce the terminology which we will use in this chapter. We will denote a feature (input) variable by the symbol $X$ and the $i^{th}$ component of $X$ is written as $X_i$. A particular observed instance vector is denoted as $x$ and the $i^{th}$ component of $x$ is denoted as $x_i$. Class label variable is denoted as $Y$ and a particular value of $Y$ is written as $y$. We will use $w$ to represent the word vector corresponding to the feature vector. The $i^{th}$ component of $w$ is denoted as $w_i$. 
and $w_i$ is the word corresponding to the $i^{th}$ feature. Hence, $X_i$ is a random variable denoting the number of occurrences of the word $w_i$. We will simplify the notation as follows:

\[
P(Y = y|X = x)
= P(Y = y|(X_1, \ldots, X_d) = (x_1, \ldots, x_d))
= P(y|x_1, \ldots, x_d)
= P(y|x),
\]

\[
P(X_i = 1|y)
= P(w_i|y),
\]

where $P(w_i|y)$ is the probability that a randomly drawn word from a document in class $y$ will be the word $w_i$. Now let us begin this section with the Bayes formula, which is fundamental theorem underlying our Bayesian feature selection. Bayes theorem :

\[
P(x|y) = \frac{P(x|y)P(y)}{P(x)}.
\]

We take an example of an application of the Bayes theorem to a classification problem. Suppose we have a two category ($Y = +1$ or $-1$) classification problem. We can do the classification using the Bayes theorem. Given a test instance $x$, we compute

\[
P(+1|x) = \frac{P(x|+1)P(+1)}{P(x)},
\]

\[
P(-1|x) = \frac{P(x|-1)P(-1)}{P(x)}.
\]

Then we assign $x$ as $+1$ class if $P(+1|x) > P(-1|x)$; otherwise $-1$. The probability $P(y|x)$ is called a posteriori probability (or posterior) of $y$ and $P(x|y)$ is called the likelihood of $y$ with respect to $x$. The naive Bayes classifier tries to assign a class label which maximizes a posteriori probability (MAP) for a given test instance. That is why this classifier is often called the MAP naive Bayes classifier. We turn to how to
compute *a posteriori* probability of $y$, given an instance $x$. Using the Bayes theorem,

$$P(y|x) = \frac{P(y|(x_1, x_2, \cdots x_d))}{P(x_1, x_2, \cdots x_d)} = \frac{P((x_1, x_2, \cdots x_d)|y) \cdot P(y)}{P(x_1, x_2, \cdots x_d)} \quad (6.1)$$

Since we only want to compare the posterior probabilities for different $y$'s, and the denominator is common for different $y$'s, we can simply ignore the denominator and compute only the numerator in (6.1). In the numerator, $P(y)$, called the prior probability, can be computed by simply counting the number of instances whose class labels are $y$, and the fraction of this number over the total number of training instances is $P(y)$. But computing $P((x_1, x_2, \cdots x_d)|y)$ by the same fashion is not feasible unless the data are big enough [66]. The naive Bayes approach tries to get around this problem by a simplifying assumption regarding the relationship between features [66, 87, 47, 85]. The naive Bayes approach thus introduces the class conditional independence assumption between the features. Hence, the numerator becomes

$$P((x_1, x_2, \cdots x_d)|y) \cdot P(y) = \prod_{i=1}^{d} P(x_i|y) \cdot P(y) \quad (6.2)$$

In summary, the naive Bayes approach classifies an instance $x$ as $c$ where $c = \arg\max_y \prod_{i=1}^{d} P(x_i|y) \cdot P(y)$. We will explain how to estimate the class conditional probabilities in subsequent sections.

### 6.2 Naive Bayes Classifier Is a Linear Classifier

In this chapter, we apply the naive Bayes approach to text data. When it is applied particularly to the text data, the probability $P(x_i|y)$, describing the probability that the word (feature) $w_i$ occurs $x_i$ times, provided that $x$ belongs to class $y$, is estimated [66]:

$$P(x_i|y) = \left( \frac{a \cdot p + N^y_i}{a + N^y} \right)^{x_i}. \quad (6.3)$$
where \( a \) is the equivalent sample size parameter, \( p \) is the prior probability, \( N^y \) is the total number of words in all documents in class \( y \), counting duplicate words multiple times, and \( N^y_i \) is the number of times the word \( w_i \) occurs in all documents in class \( y \). We will give a little more explanation about the equivalent sample size parameter shortly. Now we consider a binary classification problem using the naive Bayes. That is, \( Y \) is either +1 or -1. We first show that the binary naive Bayes classifier is a linear classifier \([18, 15, 72, 75]\). The naive Bayes classifies \( x \) as +1 if \( P(+1|x) > P(-1|x) \), and otherwise -1. That is, the decision function \( g(x) = P(+1|x) - P(-1|x) \).

\[
P(+1|x) - P(-1|x)
\]

\[
\iff \log P(+1|x_1, x_2 \cdots x_d) - \log P(-1|x_1, x_2 \cdots x_d)
\]

\[
\iff \log \left( \frac{P(+1|x_1, x_2 \cdots x_d)}{P(-1|x_1, x_2 \cdots x_d)} \right)
\]

\[
\iff \log \frac{\prod_{i=1}^{d} P(x_i|+1) \cdot P(+1)}{\prod_{i=1}^{d} P(x_i|-1) \cdot P(-1)}
\]

\[
\iff \log \prod_{i=1}^{d} \frac{P(x_i|+1)}{P(x_i|-1)} + \log \frac{P(+1)}{P(-1)}
\]

\[
\iff \sum_{i=1}^{d} \log \frac{P(x_i|+1)}{P(x_i|-1)} + \log \frac{P(+1)}{P(-1)}
\]

\[
\iff \sum_{i=1}^{d} x_i \left( \log \frac{a \cdot p + N^y_i + 1}{a + N^y} - \log \frac{a \cdot p + N^y_i - 1}{a + N^y} \right) + \log \frac{P(+1)}{P(-1)}
\]

\[
\iff \sum_{i=1}^{d} c_i \cdot x_i + b
\]

\[
\iff c \cdot x + b
\]

In our experiment, we have chosen the \( a \cdot p = 1 \) where \( a = \) total number of features (unique words) in the training samples. If we estimate the likelihood using only the frequency counts, instead of (6.3), then \( P(w_i|y) = \frac{N^y_i}{N^y} \). Here, a problem is that if the estimated probability for one of the features is zero, then the whole posterior probability having this feature (word) would be zero. To resolve this problem, a
modified probability estimate is used as follows [87, 66, 59]:

\[
P(w_i|y) = \frac{a \cdot p + N_i^y}{a + N^y}
\]  

(6.4)

where \(a, p, N^y\) and \(N_i^y\) are defined as in (6.3). Even when \(N_i^y\) is zero because of insufficient training samples, this estimated probability is not zero and hence the posterior probability may not be zero. We can give a more intuitive explanation about the equation (6.4). If there are no training samples, then the term \(\frac{a \cdot p + N_i^y}{a + N^y}\) becomes \(\frac{a \cdot p}{a}\) = \(p\). This makes sense since we do not have any training data, and then our only choice is the prior probability. On the other hand, we have lots of training samples, and then the term \(\frac{a \cdot p + N_i^y}{a + N^y}\) is dominated by \(\frac{N_i^y}{N^y}\). These many samples could represent the population and hence the prior probability does not much affect the probability estimate.

6.3 Improving Naive Bayes Classifier

Some of difficulties, however, arise when we apply the naive Bayes classifier. We will consider three such situations. The difficulties arise when the training data are not balanced, NB is trained with only one feature, and a test input is a vector of zeros. In this section, we describe how each of these problems occurs and we suggest how to resolve them.

6.3.1 Imbalanced Data

One of the problems originates when the training data are not balanced between the classes. The following example shows such a case. Suppose we are given five training input and its class label pairs, as shown in Table 6–1, and a test input \(x = (3, 2, 7)\). Obviously the test input is closest to the training instance \((3, 2, 7)\), and hence we expect the naive Bayes would classify the test instance as class \(-1\). We will show the intermediate steps of the naive Bayes classifier. First, we compute \(P(x_i|y)\). Table 6–2 shows the probability estimates where, for example, the first column and first row means \(P(w_1|+1)\). Prior probabilities for \(+1\) and \(-1\) are \(P(+1) = 4/5 = 0.8\)
Table 6–1: Imbalanced Training Data Example

<table>
<thead>
<tr>
<th>training input</th>
<th>label</th>
</tr>
</thead>
<tbody>
<tr>
<td>(7, 2, 5)</td>
<td>+1</td>
</tr>
<tr>
<td>(6, 2, 6)</td>
<td>+1</td>
</tr>
<tr>
<td>(7, 2, 5)</td>
<td>+1</td>
</tr>
<tr>
<td>(7, 2, 5)</td>
<td>+1</td>
</tr>
<tr>
<td>(3, 2, 7)</td>
<td>-1</td>
</tr>
</tbody>
</table>

Table 6–2: Probability Estimates

<table>
<thead>
<tr>
<th></th>
<th>(w_1)</th>
<th>(w_2)</th>
<th>(w_3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(y = +1)</td>
<td>(\frac{1+27}{3+56} = 0.475)</td>
<td>(\frac{1+8}{3+56} = 0.153)</td>
<td>(\frac{1+21}{3+66} = 0.373)</td>
</tr>
<tr>
<td>(y = -1)</td>
<td>(\frac{1+3}{3+12} = 0.267)</td>
<td>(\frac{1+2}{3+12} = 0.200)</td>
<td>(\frac{1+7}{3+12} = 0.533)</td>
</tr>
</tbody>
</table>

and \(P(-1) = 1/5 = 0.2\), respectively. The likelihoods of +1 and -1 with respect to the test instance \(x = (3, 2, 7)\) are:

\[
P((3, 2, 7)| +1) = (0.475)^3 \cdot (0.153)^2 \cdot (0.373)^7 = 0.0000025, \\
P((3, 2, 7)| -1) = (0.267)^3 \cdot (0.200)^2 \cdot (0.533)^7 = 0.0000093.
\]

Note that the likelihoods of \(Y = -1\) with respect to \((3, 2, 7)\), i.e., \(P((3, 2, 7)| -1)\) is almost four times greater than that of +1. But, the posterior probabilities of +1 and -1 are:

\[
P(+1|(3, 2, 7)) \propto P(+1) \cdot P((3, 2, 7)| +1) = 0.0000020, \\
P(-1|(3, 2, 7)) \propto P(-1) \cdot P((3, 2, 7)| -1) = 0.0000019.
\]

Thus, the naive Bayes classifies the test instance \((3, 2, 7)\) as class +1 while the test instance is the closest to the training instance \((3, 2, 7)\) (in fact, both are the same) which belongs to class -1. This problem comes from the imbalanced training data. The prior probabilities are far biased to the class +1. Despite an almost four times larger likelihood, it was not big enough to overcome the biased prior probability. This problem was also observed by other researchers’ work [76]. This problem may be resolved by assuming uniform prior probability, which means we use the Maximum Likelihood (ML) Bayes.
6.3.2 Training with Only One Feature

Consider we apply the naive Bayes classifier to predict a new test sample using only one feature in the training data. This may happen when we use the naive Bayes classifier with feature selection. Then the naive Bayes predicts the sample in an unreasonable way. We will illustrate this problem with an example. The following example shows such a case. Suppose we have found the most discriminating feature, $w_j$. With feature $w_j$, the data are perfectly separated. The feature value and its class label pairs with feature (word) $w_j$ are $(3, +1), (2, +1), (3, +1), (1, -1), (1, -1), (0, -1)$. The data are perfectly separated using feature $w_j$. Specifically, if the occurrences of feature $w_j$, i.e., $x_j$ in an instance are greater than or equal to 2, then the instances belong to $+1$, and otherwise $-1$. The problem occurs when we try to classify a new instance by the naive Bayes learning. Here we apply the naive Bayes with the Laplace correction [87]. Then,

$$P(w_j | +1) = \frac{1+8}{1+8} = 1$$
$$P(w_j | -1) = \frac{1+2}{1+2} = 1$$

Now for a test instance $x = (1)$,

$$P(+1|(1)) \propto \{P(w_j | +1)\}^1 \cdot P(+1) = 1^1 \cdot \frac{1}{2} = \frac{1}{2}$$
$$P(-1|(1)) \propto \{P(w_j | -1)\}^1 \cdot P(-1) = 1^1 \cdot \frac{1}{2} = \frac{1}{2}$$

This is not an expected result since the test instance $x = (1)$ is obviously similar to the training instances in class $-1$ than those in class $+1$. But the prediction based on naive Bayes is not what it should be. One way of resolving this problem is to compute an average of Euclidean distances between the test instance and training instances in class $+1$ ($-1$). Then the class giving the minimum average distance is the winner.
6.3.3 A Vector of Zeros Test Input

A similar difficulty arises when a test instance \( x \) is a vector of zeros. That is, \( (x = (0, 0 \cdots 0)) \). We consider the naive Bayes classifier, \( g(x) = \sum_i c_i \cdot x_i + b \).

\[
g(x) = \sum_i c_i \cdot x_i + b = \sum_i c_i \cdot 0 + b = b.
\]

We remind that \( b = \log \frac{P(+1)}{P(-1)} \). This means the prior probabilities decide the test instance. Specifically, if \( P(+1) > P(-1) \), we classify it as class +1; otherwise −1. This classification decision does not make sense since the probability of occurrence of a vector of zeros does not depend on the prior probabilities. We can resolve this problem in the same way as we did in training with only one feature, that is, we use the minimum Euclidean distance to decide the winner.

6.4 Text Datasets and Preprocessing

In this section, we will show how the text document can be represented as a dataset suitable for supervised learning. Text data mining [20] is a research domain involving many research areas, such as natural language processing, machine learning, information retrieval [79], and data mining. Text categorization and feature selection are two of the many text data mining problems. The text document categorization problem has been studied by many researchers [54, 55, 61, 64, 65, 66, 75, 77, 79, 95]. In this research, the main machine learning tools used are the naive Bayes method [67, 76] and SVM [80]. Nigam et al. [70] have proposed the use of maximum entropy classifiers for text classification problem. Yang et al. showed comparative research on feature selection in text classification [95]. The text classification problem in the SVM application has been extensively studied by Joachims [53]. For the document data to be available for a supervised learning such as the naive Bayes or SVM, the document is first converted into a vector space notation (also called “bag of words”). For a more compact dataset without any information loss, stop-words are removed. Such stop-words are “then,” “we,” “are,” and so forth. Again, different forms of the same word
Table 6–3: Keywords Used to Collect the Text Datasets

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Categ.</th>
<th>Keywords</th>
</tr>
</thead>
<tbody>
<tr>
<td>textAB</td>
<td>1</td>
<td>functional genomics OR gene expression</td>
</tr>
<tr>
<td></td>
<td>-1</td>
<td>structural genomics OR proteomics</td>
</tr>
<tr>
<td>textCD</td>
<td>1</td>
<td>HIV infection</td>
</tr>
<tr>
<td></td>
<td>-1</td>
<td>cancer tumor</td>
</tr>
<tr>
<td>textCL</td>
<td>1</td>
<td>acute chronic leukemia</td>
</tr>
<tr>
<td></td>
<td>-1</td>
<td>colon cancer</td>
</tr>
</tbody>
</table>

root are processed by the stemmer program. The stemmer program converts each word to its root form. We used a stop-words list which contains 571 such words, which is available from the Internet (http://www.unine.ch/Info/clef/). For the stemmer program, we used the Porter’s stemming algorithm. The Java version of the algorithm is publicly available (http://www.tartarus.org/~martin/PorterStemmer). The resulting matrix converted from the documents is very high-dimensional and sparse. We have collected three different text document datasets: textCL, textAB, and textCD. All three datasets were generated from PubMed (http://www.pubmed.org). By entering the keywords “acute chronic leukemia,” we collected 30 abstracts and by entering “colon cancer” 30 as well. This dataset is called textCL. For the textAB, two sets of keywords are “functional genomics OR gene expression” and “structural genomics OR proteomics.” For the textCD, the keywords are “HIV infection” and “cancer tumor.” Table 6–3 shows the keywords used to collect the textCL, textAB, and textCD.

After collecting the documents, each document was converted into a so-called “bag-of-words” representation. The conversion steps into a bag-of-words representation are shown as follows:

**Steps for bag-of-words representation of Text**

1. Identify all the words in the 60 documents and make a list of words
2. Remove **stop-words** such as “a,” “the,” “and,” “then,” and so forth from the list
3. Apply the **stemming** algorithm to each word in the list. The **stemming** leaves out the root form of the words. For example, “computer,” “computing,” “computation,” and “computes” all have the same comput root.

4. Sort the word list.

5. Build a matrix $M$: each column ($j$) is a word in the list, and each row ($i$) is a document. $M(i, j) =$ how many times the $j^{th}$ word occurs in the $i^{th}$ document.

6. Prune the words which appear less than three times in the documents. This results in a smaller $M$.

Step 6 in the bag-of-words representation steps was applied to the 20 Newsgroup datasets only, but it was not applied to other text datasets. Typically, after constructing the frequency count matrix $M$, we normalize the matrix row-wise (document-wise) so that each row of $M$ becomes a unit length. The rationale for this unit-length normalization is that each document may have a different number of words and this may negatively affect a learning algorithm [76]. This issue will be addressed later again.

Figure 6–1 shows the pictorial illustration of the bag-of-words representation model of text document datasets. After converting the set of abstracts into a bag-of-words representation, we have a $60 \times 1823$ matrix, a $60 \times 1976$ matrix, and a $60 \times 1844$ matrix from textCL, textAB, and textCD, respectively. And the class labels are +1 for the first 30 abstracts and -1 for the last 30 for all three datasets. Now we have datasets for supervised learning.

Also, we have used another set of text data called 20 Newsgroup data [54]. The [http://www-2.cs.cmu.edu/~TextLearning/datasets.html](http://www-2.cs.cmu.edu/~TextLearning/datasets.html) is the URL for the 20 Newsgroup datasets. This dataset contains 19,974 documents evenly distributed into 20 newsgroups. In our experiments, we have selected three newsgroup documents. They are rec.autos, sci.med, and politics.guns. Each category contains 100 documents, with a total of 300 documents. We randomly but evenly divided each
Simple Probability Estimation

\[
P('car' \mid +1) = \frac{\# 'car' \text{ in the bag } '+1'}{\text{the size of the bag } '+1'}
\]

Figure 6–1: Pictorial Illustration of the Bag-Of-Words Model.

Table 6–4: 20 Newsgroup Dataset Distribution

<table>
<thead>
<tr>
<th>Category</th>
<th>Training Set</th>
<th>Test Set</th>
</tr>
</thead>
<tbody>
<tr>
<td>rec.autos</td>
<td>50</td>
<td>50</td>
</tr>
<tr>
<td>sci.med</td>
<td>50</td>
<td>50</td>
</tr>
<tr>
<td>politics.guns</td>
<td>50</td>
<td>50</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>150</strong></td>
<td><strong>150</strong></td>
</tr>
</tbody>
</table>

category document into 50 training dataset and 50 test dataset so that we have 150 training documents and 150 test documents. Table 6–4 shows the distributions of the datasets. In our experiments, we did a one-vs-all binary classification, that is, when we train with the training set, labels for all rec.autos samples become +1 and labels for all other training samples become −1. The same procedure is done on a test set. This is one set of binary classification, and similarly for sci.med and politics.guns.
6.5 Feature Selection Approaches in Text Datasets

For text datasets, many feature selection methods have been developed. We show some of the popular ranking schemes: information gain, odds ratio, Fisher score, t-test, and correlation coefficient.

6.5.1 Information Gain

One of the popular feature selection approaches in the text document data is using information gain [54, 65, 75]. The idea is based on the information theory [24, 66]. The information gain for the word $w_k$ is the entropy difference between the class variable and the class variable conditioned on the presence or absence of the word $w_k$ [75]. The information gain for the word $w_k$ is computed as follows:

$$Info(w_k) = Entropy(Y) - Entropy(Y|w_k)$$

$$= - \sum_{y \in Y} P(y) \log(P(y)) + \sum_{w_k \in \{0,1\}} P(w_k) \sum_{y \in Y} P(y|w_k) \log(P(y|w_k))$$

$$= \sum_{y \in Y} \sum_{w_k \in \{0,1\}} P(y, w_k) \log \frac{P(y|w_k)}{P(y)},$$

where $P(+1)$ is the fraction of the documents in class +1 over the total number of documents, $P(w_k)$ is the fraction of the documents containing the word $w_k$ over the total number of documents, and $P(y, w_k)$ is the fraction of documents in the class $y$ that contain the word $w_k$ over the total number of documents. The higher the information gain is, the more discriminating the word (feature) is.

6.5.2 t-test

Some of statistics are useful to compute a discriminating score. One of them is the t-test[11]. Suppose we have $n_{+1}(n_{-1})$ samples in class +1(−1). Then the t score is computed as follows:

$$tScore(w_k) = \frac{(\bar{y}_{-1} - \bar{y}_{+1})}{s \sqrt{1/n_{+1} + 1/n_{-1}}}$$
where $\overline{y}_{-1}(\overline{y}_{+1})$ are the sample mean for the feature $w_k$ in class $+1(-1)$ samples and $s$ is the pooled sample standard deviation. The pooled variance $s^2$ can be computed as follows:

$$s^2 = \frac{S_{+1} + S_{-1}}{\nu_{+1} + \nu_{-1}}$$

where $S_{+1}$ ($S_{-1}$) are the sample variances for the feature $w_k$ in $+1(-1)$ samples and $\nu_{+1}$ ($\nu_{-1}$) is the degrees of freedom for class $+1(-1)$ samples. The higher the $t$ score, the more discriminating the word is.

### 6.5.3 Correlation Coefficient

Golub [42] used a feature ranking based on the correlation coefficient score $w_k$, which can be computed as:

$$Coef(w_k) = \frac{\mu_k(+1) - \mu_k(-1)}{\sigma_k(+1) + \sigma_k(-1)},$$

(6.5)

where $\mu_k(+1)(\mu_k(-1))$ and $\sigma_k(+1)(\sigma_k(-1))$ are the sample average and sample standard deviation for the feature $w_k$ in class $+1$ samples. Large positive values have a strong correlation with the class $+1$ and large negative values with the class $-1$. Golub used a decision function, $g(x)$ based on the correlation coefficients: [42]

$$g(x) = w \cdot (x - \mu) = \sum_{i=1}^{d} w_i \cdot x_i + \sum_{i=1}^{d} w_i \cdot (\mu_i(+) + \mu_i(-))/2,$$

where $w_i$ is defined as in (6.5). Computing the correlation coefficient score requires a simple sample mean and standard deviation.

### 6.5.4 Fisher Score

Given a two category dataset, ratio of the *between-class* to *within-class* scatter for a feature gives a ranking criterion. This kind of score is called the Fisher score (index). In our experiment, we use the following ranking score [19].

$$Fisher(w_k) = \frac{(m_{+1} - m_{-1})^2}{v_{+1} + v_{-1}},$$
where \( m+1(m-1) \) and \( v+1(v-1) \) are the mean and variance of the feature \( w_k \) in class \( +1(-1) \) samples.

### 6.5.5 Odds Ratio

The odds ratio score of a feature \( w_k \) can be computed as follows: [14, 15]

\[
\text{Odds}(w_k) = \log \frac{P(w_k|+1)(1 - P(w_k|-1))}{(1 - P(w_k|+1))P(w_k|-1)},
\]

where \( P(w_k|+1)(P(w_k|-1)) \) is the fraction of the documents containing the word \( w_k \) in class \( +1(-1) \) samples. Some of the characteristics of this score are that this score favors features occurring very often in class \( +1 \). One of the disadvantages of this score is that word \( w_k \) gets higher odds ratio if \( w_k \) occurs rarely in class \( +1 \) but never occurs in class \( -1 \) [14]. We note that the odds ratio ranking scheme does not use the absolute value of the odds ratio when ranking the score. As in the Forman’s study [34], we add one to any zero frequency count in the denominator computation to avoid division by zero.

### 6.5.6 NB Score

For text datasets, we define a ranking criterion on a word \( w_k \), based on probability estimates as follows:

\[
\text{NBScore}(w_k) = |P(w_k|+1) - P(w_k|-1)|,
\]  \hspace{1cm} (6.6)

where \( P(w_k|+1) \) is the probability that a randomly chosen word from a randomly drawn document in class \( +1 \) will be the word \( w_k \), and similarly for \( P(w_k|-1) \). We rank the word \( w_i \) higher than the word \( w_j \) if \( \text{NBScore}(w_i) > \text{NBScore}(w_j) \).

### 6.5.7 Note on Feature Selection

For any linear discriminant classifier, we can devise a feature ranking scheme, that is, the weight vector in the classifier function is the feature ranking criterion. This issue will be addressed in detail in Chapter 7. For some classifiers, we can still get a better feature ranking by applying a recursive feature selection. But we
cannot apply the recursive feature elimination universally to any linear classifier. One such case is that applying recursive elimination technique does not give any better or different feature ranking. For example, a correlation coefficient classifier, information gain, odds ratio, Fisher score, and $t$-test ranking implicitly assume the orthogonality between features. Hence, applying RFE gives the same ranking as the one without applying RFE. This issue will be revisited in Chapter 7.

### 6.6 Class Dependent Term Weighting Naive Bayes

The naive Bayes classifier has been successful despite its crude class conditional independence assumption. Obviously, most real datasets violate this assumption. Due to the naive assumption, the naive Bayes often leads to the poor posterior probability. Webb et al. [93] and Bennett [8] studied to get the better posteriors to accommodate the violation of the assumption. The feature independence assumption related to the naive Bayes has been studied in conjunction with the naive Bayes classification performance [28, 78]. Many researchers, however, tried to relax this crude assumption in the hope of getting better classification accuracy. But it seems that there is no greater advance in this direction. Friedman et al. [37] compared the Bayesian classifier with the Bayesian network which supposedly less violates the independence assumption, and found the latter did not give significant improvement. More survey on this issue can be found in Domingos et al.’s study [27].

Despite the naive assumption of the naive Bayes, its success has not been well explained or understood until recently. Domingos and Pazzani [27, 28], and Friedman [36] have investigated this issue recently. Their findings are essentially the distinction between probability estimation and classification performance. Their claims are even stronger to say that detecting and relaxing this assumption is not necessarily the best way to improve performance. Keeping this in mind, we have developed the class-dependent-term-weighting NB (CDTW-NB) approach.
Table 6–5: Illustrative Data for the Class Dependent Term Weighing

<table>
<thead>
<tr>
<th>Class</th>
<th>Sample</th>
<th>( w_j )</th>
<th>( w_k )</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>( D_1 )</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>+</td>
<td>( D_2 )</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>+</td>
<td>( D_3 )</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>+</td>
<td>( D_4 )</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>+</td>
<td>( D_5 )</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>+</td>
<td>( D_6 )</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>-</td>
<td>( D_7 )</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>-</td>
<td>( D_8 )</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>-</td>
<td>( D_9 )</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>-</td>
<td>( D_{10} )</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>-</td>
<td>( D_{11} )</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>-</td>
<td>( D_{12} )</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

6.6.1 Motivation for Class Dependent Term Weighing

As Friedman [36] and Domingos et al. [27] have observed, to get a better classification performance, the class dependent term weighting is not intended to get a better or more accurate probability estimate, which may come from by relaxing the independence assumption. Rather, it is intended to get a better classification accuracy by scaling the data class discriminative through a word occurrence probability. Below is an example to illustrating the motivation for the CDTW. From Table 6–5, we can estimate the probabilities as follows:

\[
\begin{align*}
P(w_j|+1) &= \frac{6 + 1}{F + D} \\
P(w_k|+1) &= \frac{6 + 1}{F + D} \\
P(w_j|-1) &= \frac{0 + 1}{F + D} \\
P(w_k|-1) &= \frac{0 + 1}{F + D}
\end{align*}
\]

where \( F = \# \text{features} \) and \( D = 6 (\# \text{documents in } +1(\text{or } -1)) \). The word \( w_j \) appears only in one document (\( D_4 \)) in class +1 while the word \( w_k \) appears in all the documents in the class +1. But probabilities, \( P(w_j|+1) \) and \( P(w_k|+1) \) are the same. We remind that \( P(w_j|+1) \) is the probability that a randomly chosen word from a randomly drawn document in class +1 will be the word \( w_j \). This does not make sense since the word \( w_k \) appears in all the documents in class +1 and \( w_j \) appears only in one document.
in the same class and hence we could have more chance to see the word \( w_k \) from a randomly drawn document in class +1 than to see the word \( w_j \). This is a weakness of the naive Bayes classifier. We try to fix this weakness by giving more weight on such words as \( w_k \). From Table 6–5, it is reasonable to think the word \( w_k \) could be a more representative word for the class +1, while the word \( w_j \) may not be such a case. Giving more term weighting on words appearing in more documents in the same class helps those words as class discriminating features. Obviously, we want to rank the word \( w_k \) higher (more important) than the word \( w_j \). But the term weighting is done differently dependent on the class. For each word \( w_i \), and each sample \( x \in \text{class } +1 \), CDTW is done as follows:

\[
x_i = x_i \times \frac{(n_{i+} + C \times N^+)}{N^+}, \tag{6.7}
\]

where \( x_i \) is the frequency count of the word \( w_i \) in \( x \), \( N^+ \) is the number of documents in class +1, \( n_{i+} \) is the number of documents in class +1 containing the word \( w_i \) and \( C > 0 \) is a scale constant. We do this term weighting for documents in class −1 in similar way. Let us see how this class dependent term weighting ranks the word \( w_k \) higher than the word \( w_j \). From Table 6–5, the word \( w_k \) appears in all the documents in class +1 while it does not in −1. Now we apply the term weighting as described in (6.7). For the word \( w_j \), scaling is done as follows:

\[
x_j = x_j \times \frac{(6 + C \times 6)}{6} = x_j \cdot (C + 1), \quad \text{for class } +1 \text{ samples},
\]

\[
x_j = x_j \times \frac{(0 + C \times 6)}{6} = x_j \cdot C, \quad \text{for class } -1 \text{ samples}.
\]

For the word \( w_k \), scaling is done as follows:

\[
x_k = x_k \times \frac{(1 + C \cdot 6)}{6}, \quad \text{for class } +1 \text{ samples},
\]

\[
x_k = x_k \times \frac{(0 + C \cdot 6)}{6} = x_k \cdot C, \quad \text{for class } -1 \text{ samples}.
\]

Let us now see how the ranking scores changed before and after the term weighting. Before scaling the frequency counts, we get the following probability estimates for
the data shown in Table 6–5:

\[ P(w_j|+1) = \frac{6 + 1}{V^+ + F} \quad P(w_j|-1) = \frac{1}{V^- + F} \]
\[ P(w_k|+1) = \frac{6 + 1}{V^+ + F} \quad P(w_k|-1) = \frac{1}{V^- + F} \]

where \( V^+(V^-) \) is the total number of words in class +1(-1). Then we compute the \( NBscores \) for the words \( w_j \), and \( w_k \) as defined in (6.6) as follows:

\[ NBscore(w_j) = |P(w_j|+1) - P(w_j|-1)| = \left| \frac{6 + 1}{V^+ + F} - \frac{1}{V^- + F} \right|, \]
\[ NBscore(w_k) = |P(w_k|+1) - P(w_k|-1)| = \left| \frac{6 + 1}{V^+ + F} - \frac{1}{V^- + F} \right|. \]

That is, both \( w_j \) and \( w_k \) have the same score. After scaling the frequency counts in Table 6–5, we get the following probability estimates:

\[ P(w_j|+1) = \frac{6C + 2}{U^+ + F} \quad P(w_j|-1) = \frac{1}{U^- + F} \]
\[ P(w_k|+1) = \frac{6C + 7}{U^+ + F} \quad P(w_k|-1) = \frac{1}{U^- + F} \]

where \( U^+(U^-) \) is the scaled total number of words in class +1(-1). Then \( NBscores \) for the words \( w_j \), and \( w_k \):

\[ NBscore(w_j) = \left| \frac{6C + 2}{U^+ + F} - \frac{1}{U^- + F} \right| \]
\[ = \left| \frac{(6C + 2)(U^- + F) - (U^+ + F)}{(U^+ + F)(U^- + F)} \right| \]
\[ = \frac{K_N}{K_D}, \]
\[ \text{NBScore}(w_k) = |P(w_k| + 1) - P(w_k| - 1)| \]
\[ = | \frac{6C + 7}{U^+ + F} - \frac{1}{U^- + F} | \]
\[ = | \frac{(6C + 7)(U^- + F) - (U^+ + F)}{(U^+ + F)(U^- + F)} | \]
\[ = \frac{5(U^- + F) + K_N}{K_D}, \]

where \( K_N = (6C + 2)(U^- + F) - (U^+ + F) \) and \( K_D = (U^+ + F)(U^- + F) \). Hence,
\[
\frac{|K_N|}{K_D} < \frac{|5(U^- + F) + K_N|}{K_D} \quad \text{(since } \frac{5(U^- + F)}{K_D} > 0) \]
\[ \Rightarrow \text{NBScore}(w_j) < \text{NBScore}(w_k). \]

This makes the word \( w_k \) higher ranked than the word \( w_j \). If a word appears with similar frequencies in both class, then this term weighting effect cancels out. So this term weighting scheme does not harm from such words. We formalize the discussion we have done so far in Theorem 6.6.1.

**Theorem 6.6.1** The CDTW (Class Dependent Term Weighting) NB gives the word \( w_k \) higher ranking than the word \( w_j \), while the original NB gives both the same ranking, where occurrences for the word \( w_j \) in +1 and −1 class are \( t_j^+ = [1, 1, \ldots, 1], t_j^- = [0, 0, \ldots, 0] \), respectively and occurrences for the word \( w_k \) in +1 and −1 class are \( t_k^+ = [m, 0, 0, \ldots, 0], t_k^- = [0, 0, \ldots, 0] \), respectively and \( m \) is \#samples \( \in \) class +1, assuming the number of samples in both classes are balanced.

**Proof:** Probability estimates based on the original NB:
\[
P(w_j| + 1) = \frac{m + 1}{V^+ + F} \quad P(w_j| - 1) = \frac{1}{V^- + F} \]
\[
P(w_k| + 1) = \frac{m + 1}{V^+ + F} \quad P(w_k| - 1) = \frac{1}{V^- + F} \]
where \( V^+(V^-) \) is the total number of words in class +1(-1) and \( F \) is the number of features. By the original ranking scheme, ranking scores for \( w_j \) and \( w_k \):

\[
NBScore(w_j) = |P(w_j|+1) - P(w_j|-1)| = \left| \frac{m+1}{V^++F} - \frac{1}{V^-+F} \right|,
\]

\[
NBScore(w_k) = |P(w_k|+1) - P(w_k|-1)| = \left| \frac{m+1}{V^++F} - \frac{1}{V^-+F} \right|.
\]

This proves that the original ranking scheme gives the same ranking for both \( w_j \) and \( w_k \). Now by the CDTW, probability estimates:

\[
P(w_j|+1) = \frac{m+1}{U^++F} \cdot \frac{1+Cm}{m},
\]

\[
P(w_j|-1) = \frac{1}{U^-+F} \cdot \frac{Cm}{m},
\]

\[
P(w_k|+1) = \frac{m+1}{U^++F} \cdot \frac{m+Cm}{m},
\]

\[
P(w_k|-1) = \frac{1}{U^-+F} \cdot \frac{Cm}{m},
\]

where \( U^+(U^-) \) is the scaled total number of words in class +1(-1) and \( F \) is the number of features. Ranking scores for \( w_j, w_k \):

\[
NBScore(w_j) = |P(w_j|+1) - P(w_j|-1)|
\]

\[
= \left| \frac{m+1}{U^++F} \cdot \frac{1+Cm}{m} - \frac{1}{U^++F} \cdot \frac{Cm}{m} \right|
\]

\[
= \left| \frac{m+1}{U^++F} \cdot \frac{1+Cm}{m} - \frac{C}{U^-+F} \right|
\]

\[
= \left| \frac{(m+1)(1+Cm)(U^-+F) - Cm(U^++F)}{m(U^++F)(U^-+F)} \right|
\]

\[
= \left| \frac{(m+1)(U^-+F) + L_N}{L_D} \right|,
\]

\[
NBScore(w_k) = |P(w_k|+1) - P(w_k|-1)|
\]

\[
= \left| \frac{m+1}{U^++F} \cdot \frac{m+Cm}{m} - \frac{1}{U^++F} \cdot \frac{Cm}{m} \right|
\]

\[
= \left| \frac{m+1}{U^++F} \cdot \frac{m+Cm}{m} - \frac{C}{U^-+F} \right|
\]

\[
= \left| \frac{m(m+1)(U^-+F) + L_N}{L_D} \right|,
\]
where $L_N = Cm(m+1)(U^- + F) - Cm(U^+ + F)$ and $L_D = m(U^+ + F)(U^- + F)$.

$$\frac{(m+1)(U^- + F)}{L_D} < \frac{m(m+1)(U^- + F)}{L_D}$$

$$\Rightarrow |\frac{(m+1)(U^- + F) + L_N}{L_D}| < |\frac{m(m+1)(U^- + F) + L_N}{L_D}|$$

$$\Rightarrow \text{NBScore}(w_j) < \text{NBScore}(w_k).$$

This completes the proof. ■

Although we did not prove this for a general case, the idea is that we want to give a word more weight as long as it appears in more documents, and the CDTW achieves this goal.

6.6.2 CDTW-NB-RFE Algorithm

We have showed that the MAP naive Bayes is a linear classifier. This means that $\mathbf{w}$ is the direction of the decision hyperplane, $g(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x} + b$ and $b$ is a bias. As we have shown in SVM-RFE, the absolute value of the component of $\mathbf{w}$ can be an importance of that component (feature) in terms of classification. Hence, this can be a feature selection criterion. Also, we apply the recursive feature elimination mechanism to rank the features. This gives the feature ranking using the naive Bayes approach. The formal algorithm for the CDTW-NB-RFE is presented as follows:

CDTW-NB-RFE Algorithm

Input: $X, Y$: input samples and labels

Each row of $X$: sample

Each column of $X$: term (feature)

Output: rank

Initialization: rank = empty list, $s = (1, 2, \cdots, j, \cdots, \text{dim})$

1. Unit length normalization:

   For each sample vector $\mathbf{x} \in X : \mathbf{x} = \mathbf{x}/\|\mathbf{x}\|

2. Class Dependent Term Weighting:

   For each term vector $\mathbf{t} \in X : \mathbf{t} = \frac{\mathbf{t}}{\sum_{n=1}^{N_c} \frac{CN_c}{N_c}}$.
\( n_c \): number of non-zero components in \( t \) belonging to Class \( c \)

\( N_c \): number of samples in class \( c \)

\( C \): a scale constant

3. Recursive Feature Elimination

```markdown
while \( s \) is not empty do
    Construct new samples, \( \text{newX} \) whose features contain only \( s \) from \( X \)
    Compute the ranking score \( c_i \) from \( (\text{newX}, Y) \), for \( i = 1 \cdots |s| \)
    \( c_i = \text{NBScore}(w_i) \)
    Find the feature \( j \) such that \( j = \arg\min_i |c_i| \)
    Update \( s \) by removing the feature \( j \) from \( s \)
    Update rank by adding the feature \( j \) on top of the rank
end while
```

We want to make a couple of comments on the unit length normalization. The purpose of the unit length normalization is to make each document equal length since all documents may have different length. Two benefits are the byproduct of the normalization. One is it improves the classification accuracy. Other researchers also observed the similar result by doing the unit length normalization on text data [76]. The other is we does not need to scale very small probability estimates. When we apply the naive Bayes without any data normalization probability estimates can be extremely small. We exponentiate the small numbers by occurrence counts to make it even smaller and the product of thousands of these small numbers easily make the posterior probability zero. Hence, no clear classification decision can be made. This can be prevented by a scaling. But, by doing the unit length normalization, we may avoid this scaling needs.
Table 6–6: Top 10 Words Selected from the TextAB Dataset

<table>
<thead>
<tr>
<th>Ranking</th>
<th>t-test</th>
<th>SVM-RFE</th>
<th>CDTW NB-RFE</th>
<th>Info. Gain</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>express</td>
<td>express</td>
<td>promot</td>
<td>proteom</td>
</tr>
<tr>
<td>2</td>
<td>proteom</td>
<td>protein</td>
<td>proteom</td>
<td>gene</td>
</tr>
<tr>
<td>3</td>
<td>genom</td>
<td>genom</td>
<td>transcript</td>
<td>express</td>
</tr>
<tr>
<td>4</td>
<td>transcript</td>
<td>proteom</td>
<td>induc</td>
<td>transcript</td>
</tr>
<tr>
<td>5</td>
<td>recent</td>
<td>gene</td>
<td>genom</td>
<td>promot</td>
</tr>
<tr>
<td>6</td>
<td>promot</td>
<td>induc</td>
<td>mrna</td>
<td>genom</td>
</tr>
<tr>
<td>7</td>
<td>protein</td>
<td>diseases</td>
<td>diseases</td>
<td>mediat</td>
</tr>
<tr>
<td>8</td>
<td>gene</td>
<td>mass</td>
<td>inhibit</td>
<td>inhibit</td>
</tr>
<tr>
<td>9</td>
<td>mediat</td>
<td>promot</td>
<td>structur</td>
<td>significantly</td>
</tr>
<tr>
<td>10</td>
<td>induc</td>
<td>transcript</td>
<td>mass</td>
<td>diseas</td>
</tr>
</tbody>
</table>

6.7 Experimental Results

6.7.1 Feature Ranking Works

As a preliminary experiment, we have applied four feature ranking schemes, $t$ test, SVM-RFE, naive Bayes-RFE, and information gain, to see how well each feature ranking algorithm works. We have used all 60 instances in the text document datasets to rank the features. Table 6–6 shows the top 10 selected words from the TextAB dataset. As we can see in this table, some high-ranked words are not from the keywords used, but they could be meaningful when we look at the data. They could be more discriminating words. We cannot tell how good a ranking algorithm is only by looking at whether the top-ranked features are from the keywords or not. These tables serve only to show that all the mentioned ranking algorithms work correctly in a sense by ranking some important words high. But these tables should not serve as a judging tool for which algorithm is better. Considering the keywords which were used to retrieve the 60 abstract documents, the four ranking schemes gave a similar top words list. Specifically, “proteom” and “genom” are ranked the top six selected words list in all four ranking algorithms. Table 6–7 shows the top 10 selected words from the TextCD dataset. The keywords used for retrieving the TextCD dataset were “hiv infection” and “cancer tumor.” Three of the four ranking schemes rank at least
Table 6–7: Top 10 Words Selected from the TextCD Dataset

<table>
<thead>
<tr>
<th>Ranking</th>
<th>t-test</th>
<th>SVM-RFE</th>
<th>CDTW NB-RFE</th>
<th>Info. Gain</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>hiv</td>
<td>hiv</td>
<td>cancer</td>
<td>hiv</td>
</tr>
<tr>
<td>2</td>
<td>immunodefici</td>
<td>infect</td>
<td>tumor</td>
<td>cancer</td>
</tr>
<tr>
<td>3</td>
<td>infect</td>
<td>viru</td>
<td>hiv</td>
<td>infect</td>
</tr>
<tr>
<td>4</td>
<td>viru</td>
<td>tat</td>
<td>mutat</td>
<td>immunodefici</td>
</tr>
<tr>
<td>5</td>
<td>cancer</td>
<td>specif</td>
<td>tumour</td>
<td>viru</td>
</tr>
<tr>
<td>6</td>
<td>tumor</td>
<td>gag</td>
<td>viru</td>
<td>tumor</td>
</tr>
<tr>
<td>7</td>
<td>antiretrovir</td>
<td>cd4</td>
<td>allel</td>
<td>antiretrovir</td>
</tr>
<tr>
<td>8</td>
<td>therapi</td>
<td>immunodefici</td>
<td>tat</td>
<td>progress</td>
</tr>
<tr>
<td>9</td>
<td>replic</td>
<td>region</td>
<td>gastric</td>
<td>replic</td>
</tr>
<tr>
<td>10</td>
<td>progress</td>
<td>cancer</td>
<td>immunodefici</td>
<td>vaccin</td>
</tr>
</tbody>
</table>

Table 6–8: Top 10 Words Selected from the TextCL Dataset

<table>
<thead>
<tr>
<th>Ranking</th>
<th>t-test</th>
<th>SVM-RFE</th>
<th>CDTW NB-RFE</th>
<th>Info. Gain</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>chronic</td>
<td>cancer</td>
<td>colon</td>
<td>colon</td>
</tr>
<tr>
<td>2</td>
<td>colon</td>
<td>colon</td>
<td>leukemia</td>
<td>chronic</td>
</tr>
<tr>
<td>3</td>
<td>leukemia</td>
<td>leukemia</td>
<td>chronic</td>
<td>leukemia</td>
</tr>
<tr>
<td>4</td>
<td>cancer</td>
<td>chronic</td>
<td>cancer</td>
<td>cancer</td>
</tr>
<tr>
<td>5</td>
<td>acut</td>
<td>acut</td>
<td>acut</td>
<td>acut</td>
</tr>
<tr>
<td>6</td>
<td>marrow</td>
<td>patient</td>
<td>express</td>
<td>bone</td>
</tr>
<tr>
<td>7</td>
<td>bone</td>
<td>express</td>
<td>transplant</td>
<td>marrow</td>
</tr>
<tr>
<td>8</td>
<td>express</td>
<td>hematolog</td>
<td>colorect</td>
<td>express</td>
</tr>
<tr>
<td>9</td>
<td>patient</td>
<td>malign</td>
<td>marrow</td>
<td>patient</td>
</tr>
<tr>
<td>10</td>
<td>diseas</td>
<td>bone</td>
<td>bone</td>
<td>colorect</td>
</tr>
</tbody>
</table>

three keywords in the top 10 list. The word “hiv” is especially ranked in the top three from all four top 10 lists.

Table 6–8 shows the top 10 selected words from the TextCL dataset. Words used to retrieve the TextCL dataset are “acute chronic leukemia” and “colon cancer.” All these five words are among top five words from all four rankings. Actually, the TextCL dataset is the easiest one among the three text document datasets in terms of linear separability. This easiness is shown by the ranked lists. In this sense, the easiness is in order of the TextCL, TextCD, and TextAB, the latter the more difficult. All three tables show the validity of each ranking algorithm since top-ranked words similarly appear in the top 10 words list.
Table 6–9: CDTW-NB-RFE vs. Original NB-RFE

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Original NB (1) %</th>
<th>CDTW-NB (2) %</th>
<th>(1)-(2)/(1) %</th>
</tr>
</thead>
<tbody>
<tr>
<td>rec.autos</td>
<td>Max. 97.33</td>
<td>98.00</td>
<td>-0.69</td>
</tr>
<tr>
<td></td>
<td>Average 95.77</td>
<td>96.26</td>
<td>-0.51</td>
</tr>
<tr>
<td>sci.med</td>
<td>Max. 95.33</td>
<td>97.33</td>
<td>-2.10</td>
</tr>
<tr>
<td></td>
<td>Average 92.60</td>
<td>94.67</td>
<td>-2.24</td>
</tr>
<tr>
<td>politics.guns</td>
<td>Max. 98.67</td>
<td>98.00</td>
<td>0.68</td>
</tr>
<tr>
<td></td>
<td>Average 93.23</td>
<td>94.19</td>
<td>-1.03</td>
</tr>
</tbody>
</table>

Average Accuracy Gain of CDTW-NB-RFE over Orig. NB-RFE 1.26

6.7.2 Experimental Results on the Newsgroup Datasets

We have applied the algorithm to the 20 newsgroup datasets. Table 6–9 shows the experimental results. In the experiment, we did a one-vs-all, that is, when we did the rec.autos vs. all, we labeled the rec.autos samples as +1 and the rest as −1, and similarly for the sci.med and politics.guns. We have computed the accuracies for top 1, 2, · · ·, 1000, 1100,1200, · · ·, 3000, 3023 features. Table 6–9 is the average of the accuracies. For the original NB, each sample was unit length normalized. Then we applied the recursive feature elimination using the original NB. Using this ranking, the original NB was used to get the test accuracies for the previously mentioned best features. For CDTW-NB, each sample was unit length normalized. Then CDTW-NB was used to get ranking using RFE. Then with this ranking, CDTW-NB was used to get the accuracies with the above mentioned best features. Table 6–9 shows that CDTW-NB-RFE is better than the original naive Bayes by about 1.26% on average. We have also used the five ranking schemes, information gain, odds ratio, Fisher score, t-test and correlation coefficient, and then applied the CDTW-NB. We ranked the features using the ranking scheme, information gain, and we computed the accuracies for top 1, 2, · · ·, 1000, 1100, 1200, · · ·, 3000, 3023 features using the original NB and CDTW-NB, and similarly for other ranking schemes. Table 6–10 shows CDTW-NB gives better accuracy results for almost all the ranking schemes than the original NB. The average accuracy gain is about 0.644% over the original
Table 6–10: Original NB vs. CDTW-NB

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Ranking Scheme</th>
<th>Orig. NB (1) %</th>
<th>CDTW-NB (2) %</th>
<th>(2)-(1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>rec.autos</td>
<td>Information Gain</td>
<td>95.45</td>
<td>95.47</td>
<td>0.02</td>
</tr>
<tr>
<td></td>
<td>Odds Ratio</td>
<td>83.36</td>
<td>85.12</td>
<td>1.76</td>
</tr>
<tr>
<td></td>
<td>Fisher Score</td>
<td><strong>95.86</strong></td>
<td>95.70</td>
<td>-0.16</td>
</tr>
<tr>
<td></td>
<td>t-test</td>
<td>94.58</td>
<td>94.96</td>
<td>0.38</td>
</tr>
<tr>
<td></td>
<td>Corr. Coef.</td>
<td>95.52</td>
<td>95.57</td>
<td>0.05</td>
</tr>
<tr>
<td></td>
<td>NB-RFE</td>
<td>95.77</td>
<td><strong>96.26</strong></td>
<td>0.49</td>
</tr>
<tr>
<td>sci.med</td>
<td>Information Gain</td>
<td>94.03</td>
<td>93.86</td>
<td>-0.17</td>
</tr>
<tr>
<td></td>
<td>Odds Ratio</td>
<td>79.11</td>
<td><strong>81.23</strong></td>
<td>2.12</td>
</tr>
<tr>
<td></td>
<td>Fisher Score</td>
<td>93.56</td>
<td>93.99</td>
<td>0.43</td>
</tr>
<tr>
<td></td>
<td>t-test</td>
<td>92.38</td>
<td>92.56</td>
<td>0.18</td>
</tr>
<tr>
<td></td>
<td>Corr. Coef.</td>
<td>94.09</td>
<td>94.29</td>
<td>0.20</td>
</tr>
<tr>
<td></td>
<td>NB-RFE</td>
<td>92.60</td>
<td><strong>94.67</strong></td>
<td>2.07</td>
</tr>
<tr>
<td>politics.guns</td>
<td>Information Gain</td>
<td>93.30</td>
<td>92.83</td>
<td>-0.47</td>
</tr>
<tr>
<td></td>
<td>Odds Ratio</td>
<td>87.44</td>
<td><strong>89.23</strong></td>
<td>1.79</td>
</tr>
<tr>
<td></td>
<td>Fisher Score</td>
<td>93.11</td>
<td>93.82</td>
<td>0.71</td>
</tr>
<tr>
<td></td>
<td>t-test</td>
<td>90.93</td>
<td><strong>91.85</strong></td>
<td>0.92</td>
</tr>
<tr>
<td></td>
<td>Corr. Coef.</td>
<td>93.20</td>
<td>93.42</td>
<td>0.22</td>
</tr>
<tr>
<td></td>
<td>NB-RFE</td>
<td>93.14</td>
<td><strong>94.19</strong></td>
<td>1.05</td>
</tr>
</tbody>
</table>

Average  **0.644**

NB. Table 6–11 is a subset of Table 6–10. The accuracy results are the average for the best 1, 2, 3, · · ·, 1000, 1100, 1200, · · ·, 3000, 3023 features using only CDTW-NB for all the ranking schemes. The results show that the CDTW-NB-RFE is the best for all three datasets over all the five ranking schemes. So far, we have shown that the CDTW-NB-RFE works better than other feature ranking algorithms. We now turn to the feature selection issue on text data. Figure 6–2 shows the accuracies for

Table 6–11: Accuracy Results for Ranking Schemes

<table>
<thead>
<tr>
<th>Ranking Scheme</th>
<th>rec.autos (%)</th>
<th>sci.med (%)</th>
<th>politics.guns (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Information Gain</td>
<td>95.47</td>
<td>93.86</td>
<td>92.83</td>
</tr>
<tr>
<td>Odds Ratio</td>
<td>85.12</td>
<td>81.23</td>
<td>89.23</td>
</tr>
<tr>
<td>Fisher Score</td>
<td>95.70</td>
<td>93.99</td>
<td>93.82</td>
</tr>
<tr>
<td>t-test</td>
<td>94.96</td>
<td>92.56</td>
<td>91.85</td>
</tr>
<tr>
<td>Corr. Coef.</td>
<td>95.57</td>
<td>94.29</td>
<td>93.42</td>
</tr>
<tr>
<td>CDTW-NB-RFE</td>
<td><strong>96.26</strong></td>
<td><strong>94.67</strong></td>
<td><strong>94.19</strong></td>
</tr>
</tbody>
</table>
Figure 6–2: Test Accuracy with Best-ranked Features on 20 Newsgroup Datasets.

the best 10, 20, …, 1000 features in a solid line and that of full features (3023) in a dotted horizontal line. From Figure 6–2, we can see that the similar accuracy as that of full features was achieved with the best 300 features on rec.autos, 400 on sci.med, and 650 on politics.guns datasets. With this number of features, all have achieved better than or the same accuracy as using the full features. This shows that the feature selection works on text data. This findings are consistent with other researchers’ results [19, 20, 67, 60, 65, 95].

6.8 Naive Bayes for Continuous Features

So far, we have applied the naive Bayes to text document data whose feature values are the occurrences of words and they are discrete. When features are continuous variables, an easy way of applying the naive Bayes is by assuming Gaussian distribution for each feature. Real world datasets, however, may not follow this assumption. A different approach is a discretization of the continuous feature. Dougherty et al.
showed that discretization of a continuous variable can approximate the distribution for the variable and this can help to overcome the Gaussian assumption used for continuous variables with the naive Bayes. In this section, we showed a simple discretization method for the naive Bayes method, and in the discretized data, we showed how we can do feature ranking.

A ten-bin discretization works well compared to more complicated discretization schemes, such as fuzzy learning discretization, entropy minimization discretization, and lazy discretization [29, 50, 96, 97]. Although more complicated discretization schemes give better performance than the simple ten-bin discretization, its gain is minor while its computation is more complicated [29]. Considering this, we show the naive Bayes with a ten-bin discretization.

In the $k$-bin discretization scheme, each feature is divided into $k$ bins. A simple but effective $k$-bin discretization technique is equal width discretization (EWD). The EWD divides the line between $v_{min}$ and $v_{max}$ into $k$ bins. Each interval then has width $w = (v_{max} - v_{min})/k$, and $k - 1$ boundary points are at $v_{min} + w, v_{min} + 2w, \ldots, v_{min} + (k - 1)w$. This method applies to each feature separately. Applying the discretization to the original input features makes the features into nominal values between $1, 2, \ldots, k$.

The naive Bayes classifier now for the discretized features is straightforward. For a discretized test input $\mathbf{x}$,

$$P(y|\mathbf{x}) = \frac{P(\mathbf{x}|y)P(y)}{P(\mathbf{x})} \propto P(x_1 \cdots x_d|y)P(y)$$

$$\propto P(y) \prod_i P(x_i|y)$$

$$\propto P(y) \prod_i \frac{N_{yi} + mp}{N_y + m}.$$
where \( N_y^i \) is the number of samples in class \( y \) satisfying \( X_i = x_i \), \( p \) is \( P(X_i = x_i) \), and \( m \) is the equal sample size parameter. Then the naive Bayes classifier assigns the test instance \( x \) as \( c \) where \( c = \text{argmax}_y P(x|y)P(y) \). We have applied the ten-bin discretization and the naive Bayes to the MNIST and sonar datasets. We remind that the MNIST dataset, however, does not have continuous features, but discrete values in \([0,255]\). The reason for applying discretization is that these data have already 256 discrete values which could result in too many bins (256), and probability estimates for each bin may be unreliable for a small number of counts for each bin. The ten-bin discretization can give a better probability estimate for each occurrence count than using the given discrete values (256 bins). The experimental setting is the same as in Chapter 5, that is, compute the ranking first, and compute accuracies for top \( 1, 2, 3, \ldots, 249, 250, 300, \ldots, 750, 784 \). For the ranking scheme, we do not apply the same ranking, as we did in text document datasets, since the discretization makes the naive Bayes a piecewise linear boundary. Instead, we apply an entropy-based ranking scheme here. An entropy-based ranking scheme is explained below.

**Entropy-based Ranking Algorithm**

**input**: \( X, Y \) : Input samples and labels

**output**: \( \text{rank} \)

1. Compute the entropy on \( k^{th} \) bin and \( j^{th} \) feature, \( E_{jk}^j \), for \( j = 1 \cdots \text{dim} \), \( k = 1 \cdots 10 \)

\[
E_{jk}^j = - \frac{N_j^+}{N_j^+ + N_j^-} \log \frac{N_j^+}{N_j^+ + N_j^-} - \frac{N_j^-}{N_j^+ + N_j^-} \log \frac{N_j^-}{N_j^+ + N_j^-}
\]

2. Compute the weighted average of \( E_{jk}^j \, w_j \), for \( j = 1 \cdots \text{dim} \)

\[
w_j = \sum_{k=1}^{10} E_{jk}^j \cdot \frac{N_j^+ + N_j^-}{N} \]

\( N \) : total number of samples in \( X \)

3. \text{rank} \( w_j \) for \( j = 1 \cdots \text{dim} \) such that the smaller \( w_j \) is ranked the higher
Table 6–12: Comparison between FLD, SVM, and Discretized NB

<table>
<thead>
<tr>
<th>Classifier</th>
<th>MNIST (%)</th>
<th>Sonar (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>FLD (Simple)</td>
<td>90.51</td>
<td>67.08</td>
</tr>
<tr>
<td>FLD (SV-RFE)</td>
<td>84.36</td>
<td>68.39</td>
</tr>
<tr>
<td>SVM (SV-RFE)</td>
<td>96.69</td>
<td>76.20</td>
</tr>
<tr>
<td>Disc. Naive Bayes</td>
<td>93.34</td>
<td>70.91</td>
</tr>
</tbody>
</table>

The sonar dataset has purely continuous feature values in \([0.0, 1.0]\). In this case, ten-bin discretization is natural. An experimental setting is the same as in Chapter 5. That is, we compute accuracies for top 1, 2, \ldots, 60. The ranking scheme is based on entropy. The accuracies for both datasets were already known for SVM and FLD from the previous chapter. This can give a comparative idea between those and the naive Bayes.

Table 6–12 shows the results for applying a ten-bin discretization and the naive Bayes to the MNIST and sonar datasets. It shows that the discretized naive Bayes is no better than SVM, but it is always better than FLD. Although the naive Bayes is based on the “naive” assumption, and ten-bin discretization does not elegantly approximate the data, its performance is still competitive.

While two microarray datasets have purely continuous features and may be a good choice for applying discretization, we did not show any experimental results here since they have too small number of samples (31 and 38 samples), and it is unlikely to get reliable probability estimates from them.

6.9 Discussion

We have proposed a new ranking scheme based on NB. We applied it to text datasets and showed that this scheme outperformed other popular ranking schemes. For a text dataset, it seems that the naive Bayes classifier is the only choice although SVM had been applied to text data and it was reported to be successful [55]. There are several reasons. One reason would be time and space complexity. The SVM is not scalable. The NB is a linear time and space complexity algorithm in terms of
number of input samples. On the other hand, the best known linear SVM has time complexity in the order of $n^{1.8}$, where $n$ is the number of samples [18].

Another issue is simplicity of implementation and easiness of usage. The SVM needs to solve a quadratic programming problem. Besides the implementation issue, SVM needs to tune parameters to get a best performance. Tuning SVM parameters is not a trivial job since the performance can widely vary at different set of parameter values. The NB is very simple and basically there are no parameters to tune.

The NB is a linear classifier while SVM can do both linear and nonlinear classification. This may not be a big problem for NB in text data. As long as it is concerned with text datasets, linear discriminants are good [18]. The SVM may give a better performance in a large amount of data. But if it takes that much time and space to get a result, it may not be a choice for practitioners.
CHAPTER 7
FEATURE SELECTION AND LINEAR DISCRIMINANTS

In this chapter, we showed a generalization of feature selection using linear discriminants. The idea is that the weight vector of any linear discriminant can serve as a ranking criterion. Then we showed how different linear discriminants draw different decision boundaries for various two-dimensional datasets. The datasets vary in terms of covariances of samples and the number of samples in each class. We then presented a comparative study on ranking schemes, by classification accuracy of ranking schemes on SVM and on the naive Bayes classifier. So far, we have determined how good a ranking method is using a classification accuracy by applying the ranking on a classifier. In this chapter, we show that those top-ranked features are really more discriminating features than those ranked low by visualizing the top-ranked features in several different ways. We also tried to give an idea of how many features should be selected. Finally, we showed some of applications of the feature selection, particularly, an application of the feature selection in text document datasets.

7.1 Feature Selection with Linear Discriminants

We have shown that the linear SVM decision function is \( g(x) = \sum_{i=1}^{l} \alpha_i y_i (x_i \cdot x) + b \). Then, \( w \cdot x + b = 0 \) is the decision hyperplane, where \( w \) is the direction of the decision hyperplane and \( b \) is the bias for the hyperplane. We made use of the weight (normal) vector of the decision hyperplane as a ranking criterion whether we do a simple ranking scheme or RFE. This idea of using a weight vector of the linear decision function as a ranking criterion can be generalized. For a linear classifier \( g(x) = w \cdot x + b \), where \( w \) is the direction of the classifier and \( b \) is the bias, the component of the \( |w| \) is the magnitude of importance of the feature (word). For SVM \( g(x) = w \cdot x + b = \sum_{i=1}^{l} \alpha_i y_i x_i + b \). Constructing the decision hyperplane and
ranking the features based on it requires solving a quadratic programming problem. 
We have also shown that the naive Bayes is a linear classifier. In the naive Bayes, classification is the sign of \( \log P(+1|x)/P(-1|x) \) and \( i^{th} \) weight of the linear classifier is \( \log P(w_i|+1)/P(w_i|-1) \). Perceptron trains a linear classifier, too. Then the linear classifier assigns \( \text{sign}(\mathbf{w} \cdot \mathbf{x}) \) for a test instance \( \mathbf{x} \). Again, for the same reason as SVM and the naive Bayes, \( \mathbf{w} \) is the ranking criterion. We also showed that Fisher’s linear discriminant can be used to give a ranking criterion.

Conversely, a ranking criterion score can be a weight of a linear classifier with a suitably chosen bias term. The correlation coefficient is an example of that. The decision function based on the correlation coefficients now is:

\[
g(\mathbf{x}) = \mathbf{w} \cdot (\mathbf{x} - \mu) = \sum_{i=1}^{d} w_i \cdot x_i + \sum_{i=1}^{d} w_i \cdot (\mu_i(+) + \mu_i(-))/2,
\]

where \( w_i \) is defined as in (6.5), and \( \mu_i(+)(\mu_i(-)) \) is the sample average for \( i^{th} \) feature for all the instances of class +1(-1). The last term is the bias term in the decision function.

In summary, for any linear discriminant classifiers, we can devise a feature ranking approach, that is, the weight vector in the classifier function is the feature ranking criterion. For some classifiers, we can still get a better feature ranking by applying the recursive feature elimination. But we cannot apply the recursive feature elimination universally to any linear classifier. One such case is that applying the recursive elimination technique does not give any better or different feature ranking. For example, the correlation coefficient classifier, Fisher score, information gain, odds ratio, and \( t \)-test ranking implicitly assume the orthogonality between features. Hence, applying RFE gives the same ranking as the one without applying RFE. Another case in which we cannot apply the RFE is that applying RFE is not computationally feasible. For example, Fisher’s linear discriminant requires a matrix inverse computation, and computing a matrix inverse a thousand times could not be an efficient method. Here
the order of the matrix is the dimension of a training sample. It also may not be feasible to apply RFE on SVM when the order of the data size is more than several thousands and so is the dimension of each sample.

7.2 Linear Discriminants and Their Decision Hyperplanes

It would be interesting to see how different linear classifiers draw a separating hyperplane to some two-dimensional datasets. Since each linear classifier has a different weight computation, it would be reflected in the separating decision plane. Here we consider the Support Vector Machines (SVM), naive Bayes classifier (NBC), correlation coefficient classifier (CCC), and Fisher’s linear classifier (FLD). The given data are in two dimensions for our visualization purpose. We have considered three different kinds of datasets. First, we have generated 200 ‘*’ points and 200 ‘+’ points using multivariate random number generator on Matlab. The ‘*’ points were generated from \( \mu = (6, 12), \Sigma = \begin{pmatrix} 3 & 1.5 \\ 1.5 & 3 \end{pmatrix} \), where \( \mu \) and \( \Sigma \) are mean vector and covariance matrix, respectively. And ‘+’ points were generated from \( \mu = (12, 6) \) and the same covariance matrix as the one used in ‘*’ points. Figure 7–1 shows decision hyperplanes for SVM, NBC, CCC, and FLD. Two classes have equal prior probability. Points in two different classes are linearly separable. Three circled points are support vectors and SVM finds the maximum margin hyperplane from these points. The other three discriminants show similar hyperplanes. In the second dataset, we generated 50 ‘*’ points using multivariate normal distribution with \( \mu = (6, 12), \Sigma = \begin{pmatrix} 1 & 0.5 \\ 0.5 & 1 \end{pmatrix} \), and 200 ‘+’ points using multivariate normal distribution with \( \mu = (12, 6), \Sigma = \begin{pmatrix} 3 & 1.5 \\ 1.5 & 3 \end{pmatrix} \). This dataset is imbalanced, that is, two different class points have different prior probabilities (\( P(‘*’) = 1/5 \) and \( P(‘+’) = 4/5 \)). Figure 7–2 shows plots for different classifiers. The circled three points are support vectors, and SVM finds the maximum margin decision hyperplane between these three
points regardless of the prior probabilities while the other three hyperplanes show a similar behavior and are leaning toward the class with higher prior probability. Figure 7–2 shows an obvious difference between SVM and the other three classifiers. The SVM considers only the points lying on the border line while the other three consider the average case between all the points when they find the classifier.

In the third dataset, we generated 200 '*' points using multivariate normal distribution with $\mu = (6, 12), \Sigma = \begin{pmatrix} 1 & .3 \\ .3 & 1 \end{pmatrix}$, and 200 '+' points using multivariate normal distribution with $\mu = (12, 6), \Sigma = \begin{pmatrix} 4 & 2 \\ 2 & 4 \end{pmatrix}$. Two different classes have very different distributions. The '*' points are densely concentrated while '+' points are widely scattered. Figure 7–3 shows the decision lines for different classifiers. Again, as in Figure 7–3, SVM finds the maximum margin classifier between four support vectors regardless of the data distribution. But the other three classifiers are much closer to the widely scattered class. As we have seen in the three different figures,
Figure 7–2: Linear Discriminants-Separable, Imbalanced Data.

Figure 7–3: Linear Discriminants-Inseparable, Balanced, Diff. Var. Data.
all the classifiers but SVM find somehow an average case classifier while SVM finds a border line classifier. This is a unique characteristic of SVM.

### 7.3 Visualization of Selected Features

So far we have stated how well a ranking scheme works by showing accuracy for those top-ranked features. Here, we inspect the top-ranked features visually and see whether they have more discriminating power than those ranked low. First, we use the MNIST dataset. Figure 7–4 shows images having only top-ranked features. The images in the figure were done by keeping only top-ranked features and set zero for all other features. In the figure, the first two rows are digits 2 and 3 images with only best 50 features, the next two with 100, the next two with 200, the next two with 300, and the last two rows with full 784 features. From the figure, it may be difficult to classify it with only the best 50 features with human eyes, but it is not difficult to see that the top 50 features contain enough features for correct classification, and SVM already achieves almost the highest accuracy with that number of features. As we go down the figure, it is obvious that human eyes can do correct classification with 200 or 300 features. The best 200 features would be enough, if 50 is not.

Figure 7–5 shows the leukemia dataset with the best two features which were ranked using training data by SV-RFE (SVM). It shows that even two features out of 7129 give a reasonably good separating boundary, although they are not perfectly separable. The x-axis is the best feature and the y-axis is the second best feature.

Figure 7–6 shows the test accuracy on the MNIST dataset with best-ranked features. It shows the accuracy of the best 80 features with a solid line and that of full 784 features with a dotted horizontal line. From this figure, we can say that the best 50 to 80 features are enough for the highest classification accuracy. And this figure positively supports the claims we have shown in Figure 7–4.

For the other three datasets, a similar conclusion can be drawn. Figure 7–7 shows accuracies for the best 50 features in a solid line and that of full features in
Figure 7–4: Images with the Best-ranked Features.

Figure 7–5: Leukemia Data Plot with the Best Two Features.
a dotted horizontal line. From the figure, we can see that the colon cancer dataset achieved its best accuracy at about the best 35 features, and so forth.

We have seen that from the performance plot, the best accuracy has been achieved at about the best 70 features on MNIST, 30 on leukemia, 35 on colon and 20 on sonar datasets. With this number of features, all have achieved better than or the same accuracy as using the full features. The ratio of this number of features to the full number of features are about 9%, 0.4%, 1.8%, and 33% for the four datasets. This only says that feature selection works, but it does not answer what number of features gives the best classification accuracy, that is, how many features we should choose. This issue seems to be ongoing research. Many of the feature selection algorithms first rank all the features and select a subset of top-ranked features from them by a threshold. Golub et al. [42] selected 25 top-ranked features from highly +1 class correlated features and another 25 from −1 class correlated features. Guyon et al. [44] selected the number of genes which corresponds to the minimum number of support vectors. Lee et al. [63] selected genes with frequencies higher than 2.5%. The 25 or 2.5 seems to be a subjective judgment. Recently, Fu et al. [39] suggested

Figure 7–6: Test Accuracy with Best-ranked Features on MNIST Data
Figure 7–7: Test Accuracy with Best-ranked Features on Three Datasets.

the $M$-fold cross-validation feature selection. The idea is to randomly divide the
given training data into $M$ (10) disjoint subsets, and then pick a certain number of
features by running a learning algorithm (SVM) using just one subset. Collect all the
selected features from 10 runnings and then pick genes according to frequency counts.
Compared to other feature selection algorithms, this feature selection algorithm uses
a more automatic threshold. All the above selection methods have been applied to
microarray datasets.

7.4 Compact Data Representation and Feature Selection

Now the linear SVM is used to get accuracy results for five different ranking
schemes: $t$-test, correlation coefficient, Fisher score, Guyon-RFE (SVM), and SV-
RFE (SVM). Datasets are leukemia, colon cancer, MNIST, sonar data. All datasets
but sonar are unit length normalized before conducting any experiments. Experimental
settings are exactly the same as we saw in Chapter 5. Table 7–1 shows that
Table 7–1: Performance of Various Ranking Schemes on Linear SVM (%)

<table>
<thead>
<tr>
<th>Ranking Schemes</th>
<th>Leukemia</th>
<th>Colon Cancer</th>
<th>MNIST</th>
<th>Sonar</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t$-test</td>
<td>83.33</td>
<td>90.74</td>
<td>96.06</td>
<td>75.61</td>
</tr>
<tr>
<td>Corr. Coef.</td>
<td>88.46</td>
<td>90.18</td>
<td>96.03</td>
<td>75.45</td>
</tr>
<tr>
<td>Fisher Score</td>
<td>85.75</td>
<td>88.97</td>
<td>96.06</td>
<td>75.59</td>
</tr>
<tr>
<td>Guyon RFE (SVM)</td>
<td>94.12</td>
<td>91.20</td>
<td>96.73</td>
<td>76.17</td>
</tr>
<tr>
<td>SV-RFE (SVM)</td>
<td>94.57</td>
<td>91.38</td>
<td>96.69</td>
<td>76.20</td>
</tr>
</tbody>
</table>

In all four datasets but MNIST, SV-RFE (SVM) gives the best accuracy. In the MNIST dataset, Guyon-RFE (SVM) gives slightly better accuracy than SV-RFE, but their differences are negligibly minor (0.04). Two SVM-RFE based rankings, Guyon RFE and SV-RFE are significantly better than the rest three ranking schemes in the leukemia dataset.

In the previous section, we have shown that the best classification accuracy can be achieved with a small fraction of features of the original features. In Chapter 5, we have shown that the better classification accuracy can be achieved with a small fraction of samples (SVs). The SV-based feature selection goes through these two steps: identifying the support vectors, selecting the discriminating features. Identifying the support vectors leads to the reduction in the number of the data samples, while selecting the discriminating features leads to the reduction in the size of each sample. Hence, the SV-based feature selection allows a compact representation of the data. Table 7–2 shows the reduction on SVM-RFE (SVM) for the classification purpose. In the table, we have used the number of support vector data on SVM from Chapter 5 and the number of selected feature data from the previous section. In the table, Reduction Rate refers to the reduction rate over the original data size and it was computed as \[
\frac{[(1) \cdot (2) - (3) \cdot (4)]}{[(1) \cdot (2)]}
\] where (1) is the number of samples in the original dataset, and so forth. The table shows that we can reduce the original data size by 98% in three datasets and by about 80% in one dataset. This implies that about 2% of the original data size suffices for the best classification. Although
Table 7–2: Compact Data Representation on SV-RFE (SVM)

<table>
<thead>
<tr>
<th>Datasets</th>
<th>Original Data</th>
<th>Reduced Data</th>
<th>Reduction Rate (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td># Samples</td>
<td># Features</td>
<td># Samples</td>
</tr>
<tr>
<td>Leukemia</td>
<td>38</td>
<td>7129</td>
<td>26</td>
</tr>
<tr>
<td>Colon Cancer</td>
<td>31</td>
<td>2000</td>
<td>29</td>
</tr>
<tr>
<td>MNIST</td>
<td>600</td>
<td>784</td>
<td>87</td>
</tr>
<tr>
<td>Sonar</td>
<td>104</td>
<td>60</td>
<td>63</td>
</tr>
</tbody>
</table>

we did not show any direct usefulness of this reduction, such a large data reduction as 98% could mean a significant implication and be critically utilized in some other research domains as well as machine learning.

7.5 Applications of Feature Selection

In this section, we show how feature selection can be effectively used. Now we inspected top 10 words ranked by CDTW-NB-RFE on the 20 Newsgroup datasets. The top 10 ranked words are

rec.autos vs. others :

- gun, car, doctor, studi, medic, radar, atf, auto, waco, brake.

sci.med vs. others :

- doctor, car, msg, patient, scienc, gun, infect, symptom, yeast, pain.

politics.guns vs. others :

- gun, car, atf, waco, doctor, govern, effect, fbi, weapon, back.

Chakrabarti et al. [19] mentioned a possible usefulness of the top-ranked features. One possible application scenario is that this feature selection method can provide descriptors for clusters from an unsupervised learning. Suppose we have found three clusters from unsupervised learning applied to a set of documents. We apply this binary feature selection method by treating cluster 1 documents as one class and the balance of documents as the other to get a ranked list of features. This process repeats for cluster 2 documents, and so forth. From each binary feature selection, pick the top
\( k \) words (say 10) and decide if these words are from this cluster by inspecting which cluster has more these words. Then collect those words from this cluster. This word list can be descriptors for this cluster. The results on the 20 Newsgroup datasets are as follows:

- **cluster 1**: car, radar, auto, brake.
- **cluster 2**: doctor, msg, patient, scienc, infect, symptom, yeast, pain.
- **cluster 3**: gun, atf, waco, govern, fbi, weapon.

Now car, radar, auto and brake can be descriptors for cluster 1, and similarly for clusters 2 and 3. These words are clearly salient keywords well representing corresponding clusters.
CHAPTER 8
CONCLUSIONS

We have presented SV-based feature selection and CDTW-NB-RFE. When SV-based feature selection is applied to SVM, it cuts down the famous SVM-RFE training time by the ratio of the number of support vectors while keeping the same classification accuracy or sometimes better. In this sense, our algorithm makes SVM-RFE more practical. On the other hand, ranking and discriminant function sought by our algorithm on FLD give a better performance on FLD at the expense of a little extra training time. We have shown the effectiveness of the SV-based feature selection on three different data domains.

The naive Bayes classifier has been extensively used on text categorization problems. We have shown CDTW, a new feature scaling scheme on text data and NB-RFE, a new feature selection algorithm using the naive Bayes. The CDTW gave a better performance on many popular feature ranking schemes when it was used with the naive Bayes on text data. By combining CDTW and NB-RFE, we have shown CDTW-NB-RFE outperformed the most popular feature ranking schemes on text data. We also have pointed out that NB has some systemic problems often arising when feature selection is applied to NB. We have shown how to resolve each of these problems.

While we have shown the effectiveness of SV-based feature selection on two linear discriminants, this algorithm has potential to be better on other discriminants. One issue is whether the SV-based feature selection can be generalized to any linear discriminant, or even further any discriminant including a nonlinear one. Another issue is how many support vectors are appropriate before applying feature ranking,
such as RFE, that is, when to stop removing the correct and well-classified points on FLD and what $C$ we use to get initial support vectors on SVM.

We also have shown that SV-RFE leads to a significant data reduction such as 98% for classification purpose. It would be exploratory to see what implication this reduction can have, and how this reduction could be critically utilized in other research domains as well as machine learning.
REFERENCES


[82] B. Scholkopf, Support Vector Learning, Ph.D Dissertation, Published by R. Oldenbourg Verlag, Munich, Germany, 1997.


BIOGRAPHICAL SKETCH

Eun Seog Youn was born April 18, 1968, in Hwasoon, Korea. He received his Bachelor of Science degree in industrial engineering from Hanyang University in Seoul, Korea, and his Master of Science degree in industrial engineering from the University of Wisconsin-Madison. His specialization was operations research, which included mathematical programming and the probability modeling.

He entered the graduate program in computer and information science and engineering at the University of Florida in Gainesville in August 1999. He received his Master of Science degree under the guidance of Dr. Li Min Fu. His master’s thesis research focused on feature selection in support vector machines. He is currently a Ph.D. candidate under the supervision of Dr. Li Min Fu. His Ph.D. research includes microarray analysis and text data mining using support vector machines and other linear discriminant analysis.