Support Vector Machine Algorithms: Analysis and Applications

by

Tong Wen

B.S., Nankai University, June 1992

Submitted to the Department of Mathematics
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy
at the
MASSACHUSETTS INSTITUTE OF TECHNOLOGY
June 2002

Signature of Author:.....
Department of Mathematics
May 22, 2002

Certified by.....
Alan Edelman
Associate Professor of Applied Mathematics
Thesis Supervisor

Accepted by....
Chair, Applied Mathematics Committee

Accepted by.....
Tomasz S. Mrowka
Chair, Departmental Committee on Graduate Students

Support Vector Machine Algorithms: Analysis and Applications

by

Tong Wen

Submitted to the Department of Mathematics
on May 14, 2002, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Abstract

Support Vector Machines (SVMs) have attracted recent attention as a learning technique to attack classification problems. The goal of my thesis work is to improve computational algorithms as well as the mathematical understanding of SVMs, so that they can be easily applied to real problems.

SVMs solve classification problems by learning from training examples. From the geometry, it is easy to formulate the finding of SVM classifiers as a linearly constrained Quadratic Programming (QP) problem. However, in practice its dual problem is actually computed. An important property of the dual QP problem is that its solution is sparse. The training examples that determine the SVM classifier are known as support vectors (SVs).

Motivated by the geometric derivation of the primal QP problem, we investigate how the dual problem is related to the geometry of SVs. This investigation leads to a geometric interpretation of the scaling property of SVMs and an algorithm to further compress the SVs. A random model for the training examples connects the Hessian matrix of the dual QP problem to Wishart matrices. After deriving the distributions of the elements of the inverse Wishart matrix $W_n^{-1}(n, nI)$, we give a conjecture about the summation of the elements of $W_n^{-1}(n, nI)$. It becomes challenging to solve the dual QP problem when the training set is large. We develop a fast algorithm for solving this problem. Numerical experiments show that the MATLAB implementation of this projected Conjugate Gradient algorithm is competitive with benchmark C/C++ codes such as SVM^{light} and SvmFu. Furthermore, we apply SVMs to time series data. In this application, SVMs are used to predict the movement of the stock market. Our results show that using SVMs has the potential to outperform the solution based on the most widely used geometric Brownian motion model of stock prices.

Thesis Supervisor: Alan Edelman
Title: Associate Professor of Applied Mathematics

Acknowledgments

Writing a Ph.D. thesis in mathematics is definitely not a one-man job. I am forever indebted to all of those who have helped me along the way. It is because of them that the road seemed like a goal itself. I would like to dedicate my thesis to my grandfather and Professor Yao T. Li. Without them, I could not have been in such a great place known as MIT.

My thanks first go to my advisor Professor Alan Edelman for his guidance, insights, patience and encouragement. From him, I learned how research is supposed to be done. I found the joy of research by working with him.

I would like to thank Professors Hung Cheng, Gilbert Strang and Gian-Carlo Rota for showing me the beauty of mathematics. Especially I would like to thank Professor Cheng for his inspiration and encouragement. His stories and thrilling novels have made the study of mathematics more enjoyable.

I owe many thanks to my friends. They added other tunes to my life as a MIT graduate student. The weekly poker games among mathematicians, the outings, the karaoke parties, the CSSA activities ... were all unforgettable. I want to thank Dan Stefanica, Lizhao Zhang, David Gorsich and Ryan Rifkin for their valuable discussions and suggestions to my research. Particularly, thanks go to Dr. David Gorsich of U.S. Army TACOM Automotive Research Center, who provided generous financial support to this research project.

Finally, I would like to thank my parents for their love and spiritual support when I am far away from them. I am also truly grateful to my beloved Tonia for sharing good and bad times with me.

Preface

Support Vector Machines (SVMs) solve classification problems by learning from training examples. Although the fundamental idea of SVMs was introduced by Vapnik in 1979 [92], this subject started to attract increasing attention in the 90's. Since then, SVMs have evolved as a subfield of machine learning. My first knowledge of SVMs came from the tutorial [18] and the class 9.520 at M.I.T. (Professor Poggio, Fall 1999). But today, people can find a comprehensive introduction to this subject from more places such as [24]. Details of statistical learning theory can be found in [93] and [95]. Another good reference is [80] which outlines the development of SVMs from the perspectives of theory, implementation as well as application. If people want to know more about this field, the web site `www.kernel-machines.org` is a good place to go. Instead of focusing on the learning aspect of SVMs, our research interest is mathematical and computation oriented.

This research project was initiated to develop a MATLAB SVM training code for U.S. Army TACOM (Tank-Automotive Armaments Command). To compete with the already existing C/C++ codes , we wanted our implementation to be at least as fast as them, while preserving all the good features that a high-level language such as MATLAB can provide. With this goal in mind, we develop our algorithm (in Chapter 4) in terms of linear algebra operations, which makes it easy for us to leverage off previous work instead of hand-coding everything. Equivalent emphasis is also given in Chapter 4 to implementation issues such as how to make the code adaptive to both the computer memory hierarchy and the training set. The result is a highly portable and efficient MATLAB training code.

When we first looked at SVMs, we were immediately attracted by the underlying geometry of this subject. Our investigation of the geometry in Chapter 2 provides an interpretation of the scaling property of SVMs and an algorithm to further compress the support vectors (SVs) so as to achieve a sparser solution. A random model for the training examples leads us to Wishart matrices. Motivated by the separation of $n+1$ general points in $\mathbb{R}^n$, we look at the marginal distributions of the inverse Wishart matrix $W_n^{-1}(n, nI)$ in Chapter 3. As a new application of SVMs, in Chapter 5 we predict the movement of the stock market. Our results show that using SVMs has the potential to outperform the solution based on the most widely used geometric Brownian motion model of stock prices.

In our discussion, familiarity with numerical linear algebra and linearly constrained quadratic programming (QP) problems is assumed. As a matter of notation, we use bold typeface for vectors and normal typeface for vector and matrix components as well as for

scalars; matrices are indicated by capital letters. In the following, the frequently used notations are listed.

$\boldsymbol{x}_i$	the ith training point
y_i	the label of the ith training point
$\boldsymbol{\alpha}$	the vector of Lagrange multipliers
α_i	the ith Lagrange multiplier
$\boldsymbol{w}$	the vector giving the normal direction of a hyperplane
b	the bias term
n	the dimension of a training point
m	the number of training points
H	the Hessian matrix
X	a matrix whose columns are the training points
P	the matrix that spans the current search space
Q	the orthogonal complement of P
S_1	the set of SVs that are correctly separated
S_2	the set of SVs that are separated with errors
$h(\cdot)$	a decision rule (function)

Contents

1 Introduction to Support Vector Machines **11**

1.1 Classification 11
1.2 Learning From Examples 14
1.3 Support Vector Machines 16
1.4 A Regularization Formulation of Support Vector Machines 20
1.5 Applications 21

2 The Geometry of the Support Vectors **23**

2.1 Reduction to a Linear System 23
2.2 A Trigonometric Interpretation of α 27
2.3 The Simplex Decomposition Relation at Optimality 31
2.4 Compressing the SVs 34
2.5 Making the Non-separable Case Separable 37
2.6 Future Work 39

3 The Inverse Wishart Matrix $W_n^{-1}(n, nI)$ **41**

3.1 Observations from the Separation of $n+1$ General Points in R^n 41
3.2 The Marginal Distribution of $W_n^{-1}(n, nI)$ 42
3.3 Future Work 46

4 Solving the SVM Dual QP Problem Efficiently **47**

4.1 A Projected Conjugate Gradient Algorithm 47
4.1.1 Constructing a Subproblem 49
4.1.2 Solving the Subproblem by the Conjugate Gradient Method 50
4.1.3 The Optimality Conditions 52
4.1.4 The Algorithm 53
4.2 Speed Considerations 54
4.2.1 Being Adaptive to the Computer Memory Hierarchy 54
4.2.2 Setting k and l Adaptively 55

4.3 Numerical Experiments 56
4.4 Distributing Large Training Problems 58
4.5 Conclusion and Future work 61

5 Using SVMs to Predict the Stock Market **63**
5.1 Setting the Stage 63
5.2 The Geometric Brownian Motion Solution 65
5.3 The SVM Solution 68
5.4 Conclusion and Future Work 70

A List of Kernel Functions **73**

B List of Estimated Optimal Settings for SVM^{light} and SvmFu **75**

C Our MATLAB Codes **77**
C.1 FMSvm.m 77
C.2 FMSvm1.m 83
C.3 boundMEX.c 84
C.4 colOfA.m 85

Chapter 1

Introduction to Support Vector Machines

The goal of this introduction is to set the ground for our later discussion. For more exhaustive treatments of Support Vector Machines (SVMs), we refer readers to [18] [24] [80] [95]. SVMs solve classification problems by learning from training examples. Hence, in this introduction, we first give a brief review of the classification and learning-from-example problems.

1.1 Classification

The main theme of classification problems is to determine a *decision rule* that assigns class labels to the objects of interest. A (deterministic) decision rule is a function $h : \mathbb{X} \subseteq \mathbb{R}^n \rightarrow \mathbb{Y} \subseteq \mathbb{R}$, where $\boldsymbol{x} \in \mathbb{X}$ is a n-vector that describes an object and $\mathbb{Y}$ is the set of class labels (one label for each class). When $\mathbb{Y}$ contains two (a finite number of) labels, we have a *binary* (*multi-class*) *classification* problem. If $y \in \mathbb{Y}$ takes real values, the classification problem is referred to as *regression*. Our discussion will focus on the binary classification because it is the simplest and also the most fundamental case. From now on, $\mathbb{Y}$ is set to be $\{\pm 1\}$.

For non-trivial classification problems, it is often impossible to find a 100% accurate decision rule, due to the fact that either the underlying mechanism generating the input-output pair $(\boldsymbol{x}, y)$ can not be represented by a function or we simply do not have enough information to determine the right $y(\boldsymbol{x})$ if such a function exists. Usually it is satisfactory to have a solution $h^*(\boldsymbol{x})$ that minimizes a certain measure of error. In the following paragraph, a binary example is given to show how such a $h^*(\boldsymbol{x})$ can be derived using the Bayesian approach when $\boldsymbol{x}, y$ are random variables and the probability distribution $P(\boldsymbol{x}, y)$ is known.

Assume that with the same probability, $\boldsymbol{x} \in \mathbb{R}^n$ can be generated by either the probabilistic model $P(\boldsymbol{x} \mid y = 1)$ or $P(\boldsymbol{x} \mid y = -1)$, where ± 1 are the labels for these two models, and by assumption $\Pr[y = 1] = \Pr[y = -1] = \frac{1}{2}$. Given an observation of $\boldsymbol{x}$, the goal of this problem is to detect which model actually generated this observation, that is, to determine the value of y. This classification problem is interesting because y is not deterministic for each observation. The Bayesian decision rule $h^*(\boldsymbol{x})$ is obtained by minimizing the *Bayesian risk*, i.e., the expected number of the errors made by $h(\boldsymbol{x})$:

$$\begin{aligned} R[h] &= E[(\frac{1}{2}|y - h(\boldsymbol{x})|)] \\ &= E_{\boldsymbol{x}}\{E_y[\frac{|y - h(\boldsymbol{x})|}{2} \mid \boldsymbol{x}]\} \\ &= \int E_y[\frac{|y - h(\boldsymbol{x})|}{2} \mid \boldsymbol{x}]p(\boldsymbol{x})d\boldsymbol{x}. \end{aligned}$$

Here, $p(\boldsymbol{x})$ is the density function. It is easy to show that

$$p(\boldsymbol{x})d\boldsymbol{x} = \frac{1}{2}[dP(\boldsymbol{x} \mid y = 1) + dP(\boldsymbol{x} \mid y = -1)].$$

Since $p(\boldsymbol{x}) \geq 0$, the Bayesian risk R is minimized if $E_y[\frac{|y-h(\boldsymbol{x})|}{2} \mid \boldsymbol{x}]$ is minimized for each particular value of $\boldsymbol{x}$. It follows that $h^*(\boldsymbol{x})$ can be determined on a point by point basis. For each value of $\boldsymbol{x}$, if $h(\boldsymbol{x}) = 1(-1)$, then $E_y[\frac{|y-h(\boldsymbol{x})|}{2} \mid \boldsymbol{x}] = \Pr[y = -1(1) \mid \boldsymbol{x}]$. By Bayes' Rule,

$$\begin{aligned} \Pr[y = 1 \mid \boldsymbol{x}] &= \frac{p(\boldsymbol{x} \mid y = 1)\Pr[y = 1]}{p(\boldsymbol{x})} \\ &= \frac{p(\boldsymbol{x} \mid y = 1)}{p(\boldsymbol{x} \mid y = 1) + p(\boldsymbol{x} \mid y = -1)} \end{aligned}$$

and

$$\begin{aligned} \Pr[y = -1 \mid \boldsymbol{x}] &= \frac{p(\boldsymbol{x} \mid y = -1)\Pr[y = -1]}{p(\boldsymbol{x})} \\ &= \frac{p(\boldsymbol{x} \mid y = -1)}{p(\boldsymbol{x} \mid y = 1) + p(\boldsymbol{x} \mid y = -1)}. \end{aligned}$$

Therefore, to minimize $R[h]$, or equivalently to minimize $E_y[\frac{|y-h(\boldsymbol{x})|}{2} \mid \boldsymbol{x}]$, the optimal decision rule is

$$h^*(\boldsymbol{x}) = \underset{\hat{y} \in \{\pm 1\}}{\operatorname{argmax}}\, p(\boldsymbol{x} \mid y = \hat{y}). \tag{1.1.1}$$

The above Bayesian decision rule is also called the *maximum likelihood* (ML) decision rule.

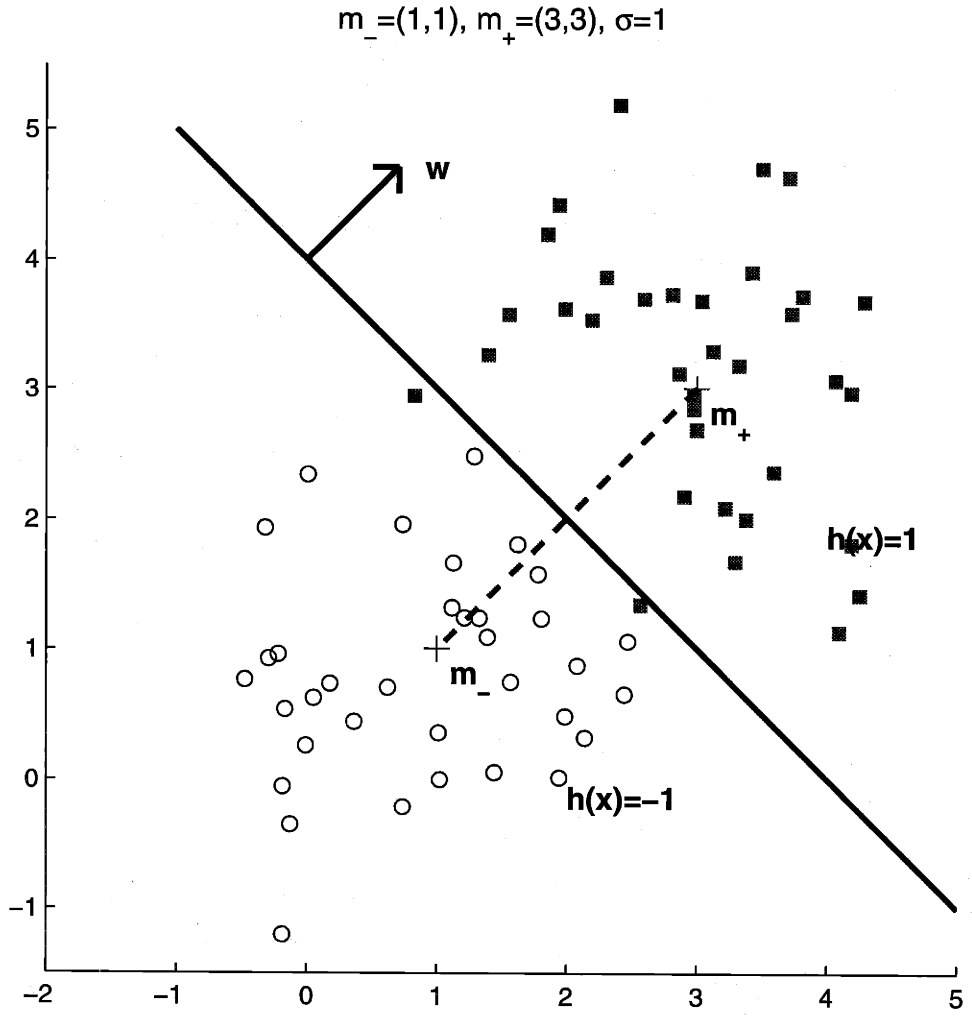


Figure 1.1: A two-dimensional binary Bayesian detection problem, where the means of the two normal distributions are $m_- = (1,1)$ and $m_+ = (3,3)$, and the covariance matrices are I. The circles and squares represent respectively the instances of the negative and positive classes. The solid line is the separating boundary.

Geometrically, each decision rule determines a boundary in $\mathbb{R}^n$ which separates the whole space into two sets of regions, each one corresponding to a class labeled by ± 1. For instance, if we further assume that the two probabilistic models are normal distributions, say $P(\boldsymbol{x} \mid y = 1) = N(\boldsymbol{x}; m_+, \sigma^2 I)$ and $P(\boldsymbol{x} \mid y = -1) = N(\boldsymbol{x}; m_-, \sigma^2 I)$, where I is the identity matrix , then the separating boundary is the hyperplane that perpendicularly bisects the line $\overline{m_- m_+}$ as indicated in Figure 1.1. In the region above this separating hyperplane, $h^*(\boldsymbol{x}) = 1$, while in the region below, $h^*(\boldsymbol{x}) = -1$. An oriented hyperplane in $\mathbb{R}^n$ such as the one in Figure 1.1 can be represented by the following equation:

$$\boldsymbol{w}^T \boldsymbol{x} + b = 0, \tag{1.1.2}$$

where $\boldsymbol{w} \in \mathbb{R}^n$ defines the orientation of this hyperplane and $b \in \mathbb{R}$ is the displacement term. It is easy to see that the ML decision rule (1.1.1) has another form:

$$h_{\mathrm{ML}}(\boldsymbol{x}) = \mathrm{sgn}(\boldsymbol{w}^T \boldsymbol{x} + b). \tag{1.1.3}$$

People have been interested in using hyperplanes to separate classes. As we will see, SVMs use two parallel hyperplanes to do this job, and the SVM decision rule also have the above form:

$$h_{\mathrm{SVM}}(\boldsymbol{x}) = \mathrm{sgn}(\boldsymbol{w}^T \boldsymbol{x} + b). \tag{1.1.4}$$

If the separating boundary can be represented by the equation

$$f_{\mathrm{SB}}(\boldsymbol{x}) = 0, \tag{1.1.5}$$

then the corresponding decision rule (binary) has a general form

$$h(\boldsymbol{x}) = \mathrm{sgn}(f_{\mathrm{SB}}(\boldsymbol{x})). \tag{1.1.6}$$

1.2 Learning From Examples

In practice, there are many cases where the probability distribution $P(\boldsymbol{x}, y)$ is not known, instead examples of the input-output relation $(\boldsymbol{x}, y)$ are available. Denote the set of these examples by $S = \{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_m, y_m)\}$. The procedure of obtaining a decision rule $h(\boldsymbol{x})$ based on S is referred to as *learning from examples*, and S is called *the training set.* Since $P(\boldsymbol{x}, y)$ is unknown, a decision rule can not be derived by minimizing the expected error such as $R[h] = E[(\frac{1}{2}|y - h(\boldsymbol{x})|)]$. Although the training error $R_S[h]$ on S is easy to compute, it is not a good idea to minimize it. Here,

$$R_S[h] = \frac{1}{m} \sum_{i=1}^{m} \frac{1}{2} |y_i - h(\boldsymbol{x_i})|.$$

The reason is that by choosing the separating boundary flexible enough, we can alway reduce $R_S[h]$ to zero, but this separating boundary may give totally uncorrelated prediction on unseen $\boldsymbol{x}$. That is, $f_{\mathrm{SB}}(\boldsymbol{x})$ or $h(\boldsymbol{x})$ may *overfit* the training set. Therefore, we need to control the flexibility of the candidate functions from which $f_{\mathrm{SB}}(\cdot)$ is chosen. The other extreme case is to use an inflexible function as the separating boundary. A decision rule giving a large $R_S[h]$ can not predict well either. The main theme of learning from examples is to determine the trade-off between the training accuracy and the predicting accuracy, or equivalently, the trade-off between the training accuracy and the flexibility of the candidate functions.

Statistical learning theory [93] [94] [95] provides a principle to solve this problem by looking at bounds (confidence intervals) on the risk $R[h]$. These bounds depend on both the training error and the flexibility of the candidate functions. In this theory, the famous concept for measuring the flexibility of a function class is *VC dimension* [103], which is defined as "the largest number l of points that can be separated in all possible ways using functions of the given class". For the set of oriented hyperplanes in $\mathbb{R}^n$, its VC dimension is $n+1$. The reason is that given $n+1$ point in $\mathbb{R}^n$ with one of them set to be the origin, if the position vectors of the other n points (except the origin) are linearly independent, then it is guaranteed that these $n+1$ points can be separated in all the possible (2^{n+1}) ways by at least one oriented hyperplane. Therefore, the VC dimension is $n+1$. As we will see later, this hyperplane is determined by a linear system.

As an example, a VC bound based on VC dimension is given as the following. If $l < m$ is the VC dimension of the class of candidate functions, then for all the functions $f(\cdot)$ in this class, with a probability of at least $1-\eta$, the following bound holds

$$R[h] \leq R_S[h] + \sqrt{\frac{l(\log \frac{2m}{l} + 1) - \log \frac{\eta}{4}}{m}}, \tag{1.2.1}$$

where $h(\boldsymbol{x}) = \mathrm{sgn}(f(\boldsymbol{x}))$ and m is the size of the training set. To be useful, $\frac{l}{m}$ must be small enough to make this bound tight. It implies that to obtain an accurate decision rule the size of the training set must be much larger than the VC dimension of the candidate functions. If there are two classes of candidate decision rules giving the same training error, then obviously the one with smaller l is preferred because the corresponding bound is smaller. Statistical learning theory suggests to obtain a suboptimal decision rule by minimizing risk bounds such as the above one. People use this principle to choose training parameters for SVMs.

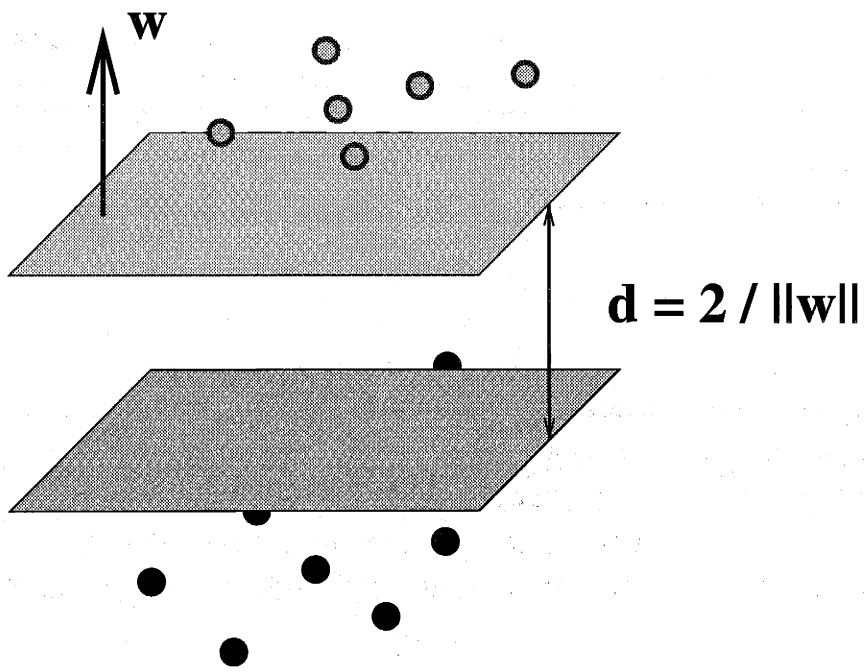


Figure 1.2: A pair of parallel hyperplanes separating examples from two classes labeled by ± 1, where point x_i along with its label y_i represents one training example (x_i, y_i). $\boldsymbol{w}$ gives the normal direction of these two hyperplanes, which points to the positive (gray) points. The positive and the negative (dark) points are linearly separable.

1.3 Support Vector Machines

SVMs solve classification problems by learning from the training examples

$$\{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_m, y_m)\}.$$

Geometrically, a decision rule corresponds to a separating boundary in $\mathbb{R}^n$. For example, we have seen that in the Bayesian detection problem, the boundary is an oriented hyperplane. Instead of one, SVMs use two parallel hyperplanes as the separating boundary, and these two hyperplanes are computed explicitly. To illustrate how to "learn" the SVM separating boundary from the training set S, the case where S is linearly separable is discussed first. Then the result is generalized for the non-separable case.

For a linearly separable training set as indicated in Figure 1.2, the *maximal-margin* SVM separating boundary is the pair of parallel hyperplanes that with the maximum gap separate the two sets of training points $\{\boldsymbol{x}_i \mid y_i = 1\}$ and $\{\boldsymbol{x}_i \mid y_i = -1\}$. Any pair of parallel

hyperplanes in $\mathbb{R}^n$ can be represented by the following equation:

$$\boldsymbol{w}^T\boldsymbol{x} + b = \pm 1, \tag{1.3.1}$$

where $\boldsymbol{w} \in \mathbb{R}^n$, $b \in \mathbb{R}$, and the gap between these two hyperplanes is $\frac{2}{\|\boldsymbol{w}\|_2}$. The orientation of the SVM separating hyperplanes is defined to point to the positive training points $\{\boldsymbol{x}_i \mid y_i = 1\}$. To be able to separate the positive and negative training points, these two hyperplanes must satisfy the following inequalities:

$$\begin{cases} \boldsymbol{w}^T\boldsymbol{x}_i + b \geq 1 & \text{if } y_i = 1 \\ \boldsymbol{w}^T\boldsymbol{x}_i + b \leq -1 & \text{if } y_i = -1 \end{cases}$$

or equivalently,

$$y_i(\boldsymbol{w}^T\boldsymbol{x}_i + b) \geq 1,$$

where $i = 1, \ldots, m$. It follows that the two maximal-margin SVM separating hyperplanes can be computed by solving the following linearly constrained quadratic programming problem:

$$\underset{w,b}{\text{minimize}} \quad \frac{1}{2}\|\boldsymbol{w}\|_2^2 \tag{1.3.2}$$

subject to

$$y_i(\boldsymbol{w}^T\boldsymbol{x}_i + b) \geq 1, \quad \text{for } i = 1, \ldots, m. \tag{1.3.3}$$

Since the training set is separable, the above problem has a unique solution. Hence, the maximal-margin SVM decision rule is

$$h_{\text{SVM}}(\boldsymbol{x}) = \text{sgn}(\boldsymbol{w}^T\boldsymbol{x} + b).$$

Note that given two (separable) sets in $\mathbb{R}^n$, the maximum gap that two parallel separating hyperplanes can achieve defines the distance between these two sets.

Given a training set, it may not be separable by hyperplanes. To apply the maximal-margin SVM, where linear separability is assumed, a map $\phi(\cdot)$ is used to map the non-separable training points to a higher dimensional space $\mathbb{R}^{n'}$ $(n' > n)$, such that in $\mathbb{R}^{n'}$ the two sets $\{\phi(\boldsymbol{x}_i) \mid y_i = 1\}$ and $\{\phi(\boldsymbol{x}_i) \mid y_i = -1\}$ can be separated by hyperplanes. In $\mathbb{R}^{n'}$, the pair of optimal separating hyperplanes are computed in the same way as in Problem (1.3.2), with the only difference that $\boldsymbol{x}_i$ is replaced by $\phi(\boldsymbol{x}_i)$. Note that a linear separating boundary in $\mathbb{R}^{n'}$ corresponds to a nonlinear separating boundary in $\mathbb{R}^n$. The idea of using a map $\phi(\cdot)$ is the same as we mentioned before: if linear functions are not enough to separate the training set, more flexible functions are used. However, using a class of more flexible functions increases the risk of overfitting. Considering that a separating boundary with training errors on S may give an overall more accurate decision rule than those without errors, in practice people do not always want to fully separate the training set.

The "soft-margin" SVMs are used for non-separable training sets. To accommodate the separation errors on S, nonnegative slack variables (errors) ε_i are introduced in the separability condition (1.3.3):

$$y_i(\boldsymbol{w}^T\boldsymbol{x}_i + b) \geq 1 - \varepsilon_i.$$

Since $\sum_{i=1}^{m} \varepsilon_i$ gives the total training error on S, it is added to the objective function in Problem (1.3.2) to be minimized along with the square norm of $\boldsymbol{w}$. The generalized quadratic programming problem becomes

$$\underset{w,b,\varepsilon}{\text{minimize}} \quad \frac{1}{2}\|\boldsymbol{w}\|_2^2 + c\sum_i \varepsilon_i \tag{1.3.4}$$

subject to (for $i = 1, \ldots, m$)

$$y_i(\boldsymbol{w}^T\boldsymbol{x}_i + b) \geq 1 - \varepsilon_i \text{ and } \varepsilon_i \geq 0. \tag{1.3.5}$$

If the training set turns out to be separable, then all the slack variables (errors) are zeros, so the above problem becomes Problem (1.3.2). Note that the larger gap (the smaller $\|\boldsymbol{w}\|_2$) the SVM separating hyperplanes have, the better predicting accuracy it tends to give on unseen data; but at the same time, the goal to separate the training set correctly tends to reduce the gap. The coefficient c controls the trade-off between the size of the gap and how well S is separated, that is, the trade-off between the predicting ability of a SVM decision rule and the training error on S. Recall that the goal of learning is the overall accuracy on $\mathbb{X}$.

In practice, the dual problem of (1.3.4) is computed, because it has simpler constraints and it uses $\phi(\cdot)$ implicitly. The dual problem is derived from the corresponding primal problem using Lagrange multipliers. Details can be found in [18] [24]. The dual problem of (1.3.2) has the form:

$$\underset{\boldsymbol{\alpha}}{\text{maximize}} \quad \boldsymbol{\alpha}^T\mathbf{1} - \frac{1}{2}\boldsymbol{\alpha}^T H\boldsymbol{\alpha} \tag{1.3.6}$$

subject to

$$\boldsymbol{y}^T\boldsymbol{\alpha} = 0, \tag{1.3.7}$$

$$\boldsymbol{\alpha} \geq \mathbf{0}, \tag{1.3.8}$$

where $\boldsymbol{\alpha} = [\alpha_1, \ldots, \alpha_m]^T$, $\boldsymbol{y} = [y_1, \ldots, y_m]^T$. Surprisingly, it turns out that the dual problem of (1.3.4) is only a little bit more complicated:

$$\underset{\boldsymbol{\alpha}}{\text{maximize}} \quad \boldsymbol{\alpha}^T\mathbf{1} - \frac{1}{2}\boldsymbol{\alpha}^T H\boldsymbol{\alpha} \tag{1.3.9}$$

subject to

$$\boldsymbol{y}^T\boldsymbol{\alpha} = 0, \tag{1.3.10}$$

$$\mathbf{0} \leq \boldsymbol{\alpha} \leq \boldsymbol{c}, \tag{1.3.11}$$

where $\boldsymbol{c} = [c, \ldots, c]^T$. The Hessian matrix $H_{ij} = y_i(\phi(\boldsymbol{x}_i)^T\phi(\boldsymbol{x}_j))y_j$ is a $m \times m$ symmetric positive semi-definite matrix, where $\phi(\cdot)$ is the map. Note that the coefficient c in the primal problem (1.3.4) becomes the upper bound in the box constraint (1.3.11). It controls the flexibility of the two parallel separating hyperplanes. In practice, a number of choices of c are tried and the one with the best performance is chosen.

Since only inner products are involved in the dual problems, a positive definite kernel function $k(\cdot,\cdot)$ [2] [68] is used to replace $\phi(\cdot)$, where $k(\boldsymbol{x}_i, \boldsymbol{x}_j) = \phi(\boldsymbol{x}_i)^T\phi(\boldsymbol{x}_j)$ and $H_{ij} = y_i k(\boldsymbol{x}_i, \boldsymbol{x}_j)y_j$. The kernel function $k(\cdot,\cdot)$ is preferred due to the curse of dimensionality. Note that the eigen-functions of a positive definite kernel span a space of functions from which $f_{\mathrm{SB}}(\cdot)$ is chosen. The analogy in the discrete world is that the eigenvectors of a symmetric positive definite matrix span its column space. If two positive definite kernel functions $k_1(\cdot,\cdot)$ and $k_2(\cdot,\cdot)$ all make the training set separable or give the same R_S, then the one with smaller VC dimension (of the function space it defines) is preferred. A list of positive definite kernel functions frequently used by SVMs is available in the appendix. For details of how kernel functions are used in SVMs, we refer readers to [18] [24] [82] [80].

Each dual variable (Lagrange multiplier) α_i corresponds to one example $(\boldsymbol{x}_i, y_i)$ in the primal problem. To simplify our notation, we use $\boldsymbol{x}_i$ in place of $\phi(\boldsymbol{x}_i)$ in our remaining discussion. From the optimality conditions (the Kuhn-Tucker conditions [44]), it can be shown that at optimality, the primal variables $\boldsymbol{w}$, b and the dual variable α are connected by the following equations:

$$\boldsymbol{w} = \sum_{i=1}^{m} y_i\alpha_i\boldsymbol{x}_i, \tag{1.3.12}$$

and if $y_i(\boldsymbol{w}^T\boldsymbol{x}_i + b) = 1$ then

$$b = y_i - \boldsymbol{w}^T\boldsymbol{x}_i = y_i(1 - \boldsymbol{e}_i^T H\boldsymbol{\alpha}). \tag{1.3.13}$$

Here, $\boldsymbol{e}_i$ is the ith column of the identity matrix $I_{m\times m}$. From (1.3.12), it is easy to show that the SVM decision has the following form:

$$h_{\mathrm{SVM}}(\boldsymbol{x}) = \mathrm{sgn}(\sum_{i=1}^{m} y_i\alpha_i k(\boldsymbol{x}_i, \boldsymbol{x}) + b). \tag{1.3.14}$$

Also at optimality, the value of $\boldsymbol{\alpha}$ partitions the set $\{x_1, \ldots, x_m\}$ into three parts S_1, S_2 and S_3, and vice versa:

$$\begin{cases} 0 < \alpha_i < c & \longleftrightarrow \boldsymbol{x}_i \in S_1,\ S_1 = \{\boldsymbol{x}_i |\ y_i(\boldsymbol{w}^T\boldsymbol{x}_i + b) = 1\ (\varepsilon_i = 0)\}; \\ \alpha_i = c & \longleftrightarrow \boldsymbol{x}_i \in S_2,\ S_2 = \{x_i |\ y_i(\boldsymbol{w}^T\boldsymbol{x}_i + b) = 1 - \varepsilon_i \text{ and } \varepsilon_i > 0\}; \\ \alpha_i = 0 & \longleftrightarrow \boldsymbol{x}_i \in S_3,\ S_3 = \{\boldsymbol{x}_i |\ y_i(\boldsymbol{w}^T\boldsymbol{x}_i + b) > 1\ (\varepsilon_i = 0)\}. \end{cases}$$

$S_1 \cup S_2$ contain the support vectors (SVs). We can see that at optimality, the equality $y_i(w^T x_i + b) = 1 - \varepsilon_i$ holds only when x_i is a SV. Geometrically, S_1 contains the training points that are in the SVM separating hyperplanes, whereas S_3 and S_2 contain respectively the training points separated correctly and with errors by the SVM hyperplanes. An important property of SVMs is that $\boldsymbol{\alpha}$ is generally sparse [28]. The intuition explaining the sparseness is that in $\mathbb{R}^n$, to determine a hyperplane with $n+1$ unknowns, we only need $n+1$ equations (SVs), but usually we have $m > n$ training examples. This sparsity property enables us to develop a fast algorithm for solving the dual problem (1.3.9) in Chapter 4.

1.4 A Regularization Formulation of Support Vector Machines

From the Moore-Aronszajn theorem [105], we know that every positive definite kernel function $k(\cdot,\cdot)$ on $\mathbb{R}^n \times \mathbb{R}^n$ determines a unique Reproducing Kernel Hilbert Space (RKHS) $\mathcal{H}_k$ of real valued functions on $\mathbb{R}^n$ and vice versa. Denote $k(\boldsymbol{x}, \boldsymbol{z})$ by $k_{\boldsymbol{z}}(\boldsymbol{x})$. For fixed $\boldsymbol{s}, \boldsymbol{t} \in \mathbb{R}^n$, $k(\cdot,\cdot)$ has the following "reproducing" property:

$$< k_{\boldsymbol{s}}, k_{\boldsymbol{t}} >_k = k(\boldsymbol{s}, \boldsymbol{t}),$$

where $< \cdot, \cdot >_k$ is the inner product in $\mathcal{H}_k$. It can also be shown that all the finite linear combinations of $k_{\boldsymbol{z}_i}$ are in $\mathcal{H}_k$, where $\boldsymbol{z}_i \in \mathbb{R}^n$. For any $f \in \mathcal{H}_k$, we have

$$< f, k_{\boldsymbol{z}} >_k = f(\boldsymbol{z}).$$

To show that the SVM decision rule can be derived using the same methodology to derive the ML decision rule, a risk functional is defined for SVMs:

$$H_{\text{SVM}}[f] = \frac{1}{m}\sum_{i=1}^{m} u(1 - y_i f(\boldsymbol{x}_i))(1 - y_i f(\boldsymbol{x}_i)) + \frac{1}{2c}\|f\|_k^2. \tag{1.4.1}$$

Here, $u(\cdot)$ is the Heaviside function, $\|\cdot\|_k$ is the norm in $\mathcal{H}_k$ induced by the above inner product, and $\frac{1}{2c}\|f\|_k^2$ is the regularization term or stabilizer. The minimizer of $H_{\text{SVM}}[f]$

over the RKHS $\mathcal{H}_k$ has the following form:

$$f^*(\boldsymbol{x}) = \sum_{i=1}^{m} \beta_i k(\boldsymbol{x}_i, \boldsymbol{x}) + b.$$

It can be shown that minimizing $H_{\text{SVM}}[f]$ over $\mathcal{H}_k$ is equivalent to solving Problem (1.3.4), where equation $f^*(\boldsymbol{x}) = \pm 1$ defines the SVM separating hyperplanes and the decision rule is

$$h_{\text{SVM}}(\boldsymbol{x}) = \text{sgn}(f^*(\boldsymbol{x})).$$

Note that with this formulation, SVMs can be connected to other learning mechanisms such as regularization networks (RNs). Measuring the training error differently, the risk functional for RNs is

$$H_{\text{RN}}[f] = \frac{1}{m} \sum_{i=1}^{m} (y_i - f(\boldsymbol{x}_i))^2 + \frac{1}{2c} \|f\|_k^2. \tag{1.4.2}$$

Its minimizer over $\mathcal{H}_k$ has the same form: $f^*(\boldsymbol{x}) = \sum_{i=1}^{m} \alpha_i k(\boldsymbol{x}_i, \boldsymbol{x}) + b$. For more detailed discussion on this topic, please refer to [27] [29] [83] [85].

1.5 Applications

In addition to the well known case of recognizing handwritten digits, today, more applications can be found in the literature. For examples, SVMs have been used to categorize text [33], to detect human faces in a picture [60], to identify speakers [71] , and to process genomic and biostatistical data [14] [45]. SVMs have also been used to solve regression problems such as predicting times series [54] [55]. As you will see, we use SVMs to predict the movement of the stock market.

SVM^{light} [35], SvmFu [66] and SVM Torch [19] are three major implementations of SVMs. They are written in either C or C++. In Chapter 4, our MATLAB training code is compared against SVM^{light} and SvmFu.

Chapter 2

The Geometry of the Support Vectors

The unknowns α_i of the dual SVM quadratic programming (QP) problem (1.3.9) are Lagrange multipliers of the constraints in the primal QP problem (1.3.4). Motivated by the geometric derivation of the primal QP problem, in this chapter, we investigate how the non-trivial α_i ($0 < \alpha_i < c$) are related to the geometry of the SVs. The results of this investigation lead to a geometric interpretation of the scaling property of SVMs and a way to further compress the SVs so as to get a sparser solution. During the following discussion, it is assumed that the SVs are known. We will discuss how to solve the dual QP problem efficiently in Chapter 4. Again, for simplicity, we use $\boldsymbol{x}_i$ in place of $\phi(\boldsymbol{x}_i)$, where $(\boldsymbol{x}_i, y_i)$ are the training examples with $\boldsymbol{x}_i \in \mathbb{R}^n$ and $y_i \in \mathbb{R}$. An important fact we should keep in mind is that we generally do not know the map $\phi(\cdot)$, which is implicitly determined by the kernel function $k(\cdot, \cdot)$. As you will see, linear algebra is the tool for this investigation.

2.1 Reduction to a Linear System

From (1.3.12) and (1.3.13), we know that discarding non-SVs does not change the solution to the SVM QP problems. In other words, $\boldsymbol{w}$ and b or $\boldsymbol{\alpha}$ are determined only by the SVs. In this section, we show in details how the SVM separating hyperplanes are uniquely determined by the SVs through linear systems. There are totally four cases, which are summarized in Table 2.1. This property is used frequently in our later discussion.

Assuming that $S_1 \cup S_2$ is known, then for $\boldsymbol{x}_i \in S_1 \cup S_2$ we have that (1.3.5) becomes the equality (becomes *active*)

$$\boldsymbol{w}^T \boldsymbol{x}_i + b = y_i(1 - \varepsilon_i),$$

The primal QP problem	
X is in full rank	X is rank deficient
$X^T\boldsymbol{w} = \boldsymbol{v}$	$R^T\hat{\boldsymbol{w}} = \boldsymbol{v}$ $(X = QR)$
The dual QP problem	
S_2 is empty	S_2 is not empty
$\begin{bmatrix} H & \boldsymbol{y} \\ \boldsymbol{y}^T & 0 \end{bmatrix} \begin{bmatrix} \boldsymbol{\alpha} \\ b \end{bmatrix} = \begin{bmatrix} \mathbf{1} \\ 0 \end{bmatrix}$	$\begin{bmatrix} H_{11} & \boldsymbol{y}_1 \\ \boldsymbol{y}_1^T & 0 \end{bmatrix} \begin{bmatrix} \boldsymbol{\alpha}_1 \\ b \end{bmatrix} = \begin{bmatrix} \boldsymbol{r}_1 \\ -\boldsymbol{y}_2^T\boldsymbol{c} \end{bmatrix}$

Table 2.1: The four linear systems derived from the primal and dual QP problems.

where $\varepsilon_i = 0$ if $\boldsymbol{x}_i \in S_1$. Let $\hat{n}$ be the rank of the matrix whose columns are SVs. Since there are $n+1$ unknowns in $\boldsymbol{w}$ and b, it is easy to see that they can be determined by a linear system if $\hat{n} = n$ and $|S_1 \cup S_2| \geq n+1$. The linear system contains any $n+1$ equations listed above with the corresponding SVs satisfying the condition that $(\boldsymbol{x}_{i_1} - \boldsymbol{x}_{i_{n+1}}), \ldots, (\boldsymbol{x}_{i_n} - \boldsymbol{x}_{i_{n+1}})$ are linearly independent. Define $X = [(\boldsymbol{x}_{i_1} - \boldsymbol{x}_{i_{n+1}}), \ldots, (\boldsymbol{x}_{i_n} - \boldsymbol{x}_{i_{n+1}})]$. It follows that $\boldsymbol{w}$ is determined by the linear system

$$X^T\boldsymbol{w} = \boldsymbol{v}, \tag{2.1.1}$$

where the jth element of $\boldsymbol{v}$ is $v_j = [y_{i_j}(1 - \varepsilon_{i_j}) - y_{i_{n+1}}(1 - \varepsilon_{i_{n+1}})]$.

However, when $\hat{n} < n$, it is not that trivial to show that $\boldsymbol{w}$ can be uniquely determined by a linear system. Suppose there are $\hat{n}+1$ SVs satisfying the above independence condition. For this case, we still have a linear system like (2.1.1), but matrix X is thin instead of square. Let $X = Q_{n\times\hat{n}}R_{\hat{n}\times\hat{n}}$ be the compact QR decomposition [30] of X and $\hat{\boldsymbol{w}} = Q^T\boldsymbol{w}$. It follows that

$$\begin{aligned} X^T\boldsymbol{w} &= \boldsymbol{v} \\ R^T(Q^T\boldsymbol{w}) &= \boldsymbol{v} \\ R^T\hat{\boldsymbol{w}} &= \boldsymbol{v} \\ \hat{\boldsymbol{w}} &= R^{-T}\boldsymbol{v}. \end{aligned}$$

If P is the orthogonal complement of Q, then $\boldsymbol{w} = Q\hat{\boldsymbol{w}} + P\tilde{\boldsymbol{w}}$, where $\hat{\boldsymbol{w}} = R^{-T}\boldsymbol{v}$ and $\tilde{\boldsymbol{w}}$ is unknown so far. Since $\|\boldsymbol{w}\|_2^2 = \|\hat{\boldsymbol{w}}\|_2^2 + \|\tilde{\boldsymbol{w}}\|_2^2$, to minimize $\|\boldsymbol{w}\|_2$, $\tilde{\boldsymbol{w}}$ must be zero (remember that the first term of the primal objective function is $\frac{1}{2}\|\boldsymbol{w}\|_2^2$). Therefore, for this case w is still uniquely determined by a linear system:

$$R^T\hat{\boldsymbol{w}} = \boldsymbol{v}, \tag{2.1.2}$$

where $\boldsymbol{w} = Q\hat{\boldsymbol{w}}$. In other words, the pair of parallel hyperplanes determined by $\hat{\boldsymbol{w}}$ and b is in a space of lower dimension $\hat{n}$. It is easy to see that the above result is also true when $\hat{n} < n$ and $|S_1 \cup S_2| = \hat{n}$. Thus, if $\phi(\cdot)$ and ε_i are known, then the knowledge of which training points are the SVs uniquely determines the primal unknowns $\boldsymbol{w}$ and b through a linear system. For example, if a linear kernel is used ($\phi(\boldsymbol{x}) = \boldsymbol{x}$) and the training set is separable ($\varepsilon_i = 0$), then knowing S_1 enables us to solve $\boldsymbol{w}$ and b by inversing the corresponding linear system.

In practice, $\phi(\cdot)$ and ε_i are usually not known. Therefore, to determine the pair of SVM separating hyperplanes, we need to compute the dual QP problem. Since the Hessian matrix H is generally semi-definite, the solution to the dual problem is not guaranteed to be unique. This explains why the SVs may not be *compact*. We define the SVs to be compact if the SVs in S_1 satisfy the independence condition, that is, $(\boldsymbol{x}_1 - \boldsymbol{x}_{|S_1|}), \ldots, (\boldsymbol{x}_{|S_1|-1} - \boldsymbol{x}_{|S_1|})$ are linearly independent, where $S_1 = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{|S_1|}\}$. In this section, it is assumed that the SVs are compact. As we will see, this assumption enables us to solve the dual problem through a linear system. Later in this chapter, we will discuss how to compress a given set of SVs so as to make it compact.

Let us consider the dual QP problem based on the training set containing the SVs only. If the training set is separable, that is, $|S_2| = 0$, then the dual problem is

$$\underset{\boldsymbol{\alpha}}{\text{maximize}} \quad \boldsymbol{\alpha}^T\mathbf{1} - \frac{1}{2}\boldsymbol{\alpha}^T H\boldsymbol{\alpha} \tag{2.1.3}$$

subject to

$$\boldsymbol{y}^T\boldsymbol{\alpha} = 0. \tag{2.1.4}$$

The constraint $\boldsymbol{\alpha} \geq \mathbf{0}$ is removed because we know that it must be satisfied by the solution. Let λ be the Lagrange multiplier of the equality constraint above. The Lagrangian is

$$L(\boldsymbol{\alpha}, \lambda) = \frac{1}{2}\boldsymbol{\alpha}^T H\boldsymbol{\alpha} - \boldsymbol{\alpha}^T\mathbf{1} + \lambda(\boldsymbol{y}^T\boldsymbol{\alpha}).$$

Setting $\frac{\partial L}{\partial \boldsymbol{\alpha}} = 0$ and $\frac{\partial L}{\partial \lambda} = 0$ gives the following equations:

$$H\boldsymbol{\alpha} - \mathbf{1} + \lambda\boldsymbol{y} = 0 \tag{2.1.5}$$

$$\boldsymbol{y}^T\boldsymbol{\alpha} = 0. \tag{2.1.6}$$

If we substitute $\boldsymbol{w}$ with $\boldsymbol{w} = \sum_{\boldsymbol{x}_i \in S_1} y_i\alpha_i\boldsymbol{x}_i$ in the primal constraint

$$y_i(\boldsymbol{w}^T\boldsymbol{x}_i + b) = 1,$$

we get an equation similar to (2.1.5):

$$H\boldsymbol{\alpha} - \mathbf{1} + b\boldsymbol{y} = 0.$$

This implies that $\lambda = b$ at optimality. Therefore, $\boldsymbol{\alpha}$ and b are determined by the linear system

$$\begin{bmatrix} H & \boldsymbol{y} \\ \boldsymbol{y}^T & 0 \end{bmatrix} \begin{bmatrix} \boldsymbol{\alpha} \\ b \end{bmatrix} = \begin{bmatrix} \mathbf{1} \\ 0 \end{bmatrix}. \tag{2.1.7}$$

Let $\boldsymbol{\alpha} = \begin{bmatrix} \boldsymbol{\alpha}_1 \\ \boldsymbol{\alpha}_2 \end{bmatrix}$ such that $\boldsymbol{\alpha}_1$ and $\boldsymbol{\alpha}_2$ corresponds to the SVs in S_1 and S_2 respectively. If S_2 is not empty, then $\boldsymbol{\alpha}_2 = \boldsymbol{c}$. The dual problem becomes

$$\underset{\boldsymbol{\alpha}_1}{\text{maximize}} \quad [\boldsymbol{\alpha}_1^T, \boldsymbol{c}^T]\mathbf{1} - \frac{1}{2}[\boldsymbol{\alpha}_1^T, \boldsymbol{c}^T] \begin{bmatrix} H_{11} & H_{12} \\ H_{12} & H_{22} \end{bmatrix} \begin{bmatrix} \boldsymbol{\alpha}_1 \\ \boldsymbol{c} \end{bmatrix}$$

subject to

$$\boldsymbol{y}_1^T \boldsymbol{\alpha}_1 = -\boldsymbol{y}_2^T \boldsymbol{c},$$

which is equivalent to

$$\underset{\boldsymbol{\alpha}_1}{\text{maximize}} \quad \boldsymbol{\alpha}^T \boldsymbol{r}_1 - \frac{1}{2}\boldsymbol{\alpha}_1^T H_{11} \boldsymbol{\alpha}_1 \tag{2.1.8}$$

subject to

$$\boldsymbol{y}_1^T \boldsymbol{\alpha}_1 = -\boldsymbol{y}_2^T \boldsymbol{c}, \tag{2.1.9}$$

where $\boldsymbol{r}_1 = \mathbf{1} - H_{12}\boldsymbol{c}$. Using the Lagrange Multiplier method again, we obtain the following linear system:

$$\begin{bmatrix} H_{11} & \boldsymbol{y}_1 \\ \boldsymbol{y}_1^T & 0 \end{bmatrix} \begin{bmatrix} \boldsymbol{\alpha}_1 \\ b \end{bmatrix} = \begin{bmatrix} \boldsymbol{r}_1 \\ -\boldsymbol{y}_2^T \boldsymbol{c} \end{bmatrix}. \tag{2.1.10}$$

Combining the above results, we have the following lemma:

LEMMA 2.1.1 *If $\phi(\cdot)$ and ε_i are known or the set of SVs is compact, then the knowledge of who are the SVs reduces the SVM QP problems to linear systems as indicated in Table 2.1.*

2.2 A Trigonometric Interpretation of α

Motivated by the words

> *If the minimum problem has a geometric meaning, or it has a physical application, then so does the (dual) maximum problem.*

from Gilbert Strang [90, page 100], we are interested in knowing how the dual problem is related to the geometry. In this section, we give a trigonometric interpretation of the Lagrange multipliers associated with the SVs in S_1, that is, those α_i satisfying $0 < \alpha_i < c$. In Theorem 2.2.1, we show that these α_i depend on three angles and the area of a triangle. Again, it is assumed that the SVs are compact. Let $S_1 = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_l\}$. Without loss of generality, we assume that $\boldsymbol{x}_l$ is the origin, i.e., $\boldsymbol{x}_l = \boldsymbol{0}$. Since the SVs are compact, $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{l-1}$ are linearly independent.

At optimality, we know that the following two equations hold:

$$\boldsymbol{w} = \sum_{\boldsymbol{x}_i \in S_1} y_i \alpha_i \boldsymbol{x}_i + c \sum_{\boldsymbol{x}_j \in S_2} y_j \boldsymbol{x}_j$$

and

$$\sum_{\boldsymbol{x}_i \in S_1} y_i \alpha_i + c \sum_{\boldsymbol{x}_j \in S_2} y_j = 0.$$

Let $X = [y_1 \boldsymbol{x}_1, \ldots, y_{l-1} \boldsymbol{x}_{l-1}]$, $\boldsymbol{\beta} = [\alpha_1, \ldots, \alpha_{l-1}]^T$ and $\boldsymbol{v} = \boldsymbol{w} - c \sum_{\boldsymbol{x}_j \in S_2} y_j \boldsymbol{x}_j$. The above two equations become

$$X\boldsymbol{\beta} = \boldsymbol{v} \tag{2.2.1}$$

and

$$\alpha_l = -y_l \Big(\sum_{i=1}^{l-1} y_i \alpha_i + c \sum_{\boldsymbol{x}_j \in S_2} y_j \Big). \tag{2.2.2}$$

If $l - 1 = n$, then we have a square linear system; otherwise, X is a thin matrix. For the second case, we still can reduce it to a square linear system by projecting it onto the column space of X. Therefore, it is enough to only consider the first case.

Given a nonsingular square linear system $X\boldsymbol{\beta} = \boldsymbol{v}$, Cramer's rule says that

$$\beta_i = \frac{\det X_i}{\det X}, \tag{2.2.3}$$

where $X_i = [y_1 \boldsymbol{x}_1, \ldots, y_{i-1} \boldsymbol{x}_{i-1}, \boldsymbol{v}, y_{i+1} \boldsymbol{x}_{i+1}, \ldots, y_n \boldsymbol{x}_n]$. Define vol $X_i = |\det X_i|$. At optimality, since $\alpha_i > 0$ for $i = 1, \ldots, n$, we have

$$\beta_i = \frac{\text{vol } X_i}{\text{vol } X}. \tag{2.2.4}$$

To further relate α_i to the geometry of the SVs, we need the following lemma:

LEMMA 2.2.1 *Let* $X^{(1)} = [\boldsymbol{x}_1, \ldots, \boldsymbol{x}_j]$ *and* $X^{(2)} = [\boldsymbol{x}_{j+1}, \ldots, \boldsymbol{x}_n]$ *be a partition of a general square matrix* $X = [\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n]$ *with* $j > n - j$. *Denote the compact QR decompositions of* $X^{(1)}$ *and* $X^{(2)}$ *by* $X^{(1)} = Q_1^{n\times j} R_1^{j\times j}$ *and* $X^{(2)} = Q_2^{n\times(n-j)} R_2^{(n-j)\times(n-j)}$ *respectively. After defining* vol $X^{(1)}$ *and* vol $X^{(2)}$ *by*

$$\text{vol } X^{(1)} = |\det R_1|$$

and

$$\text{vol } X^{(2)} = |\det R_2|,$$

we have the following decomposition of vol X*:*

$$\text{vol } X = \text{vol } X^{(1)} \text{vol } X^{(2)} \prod_{i=1}^{n-j} \sin\theta_i, \tag{2.2.5}$$

where θ_i $(0 < \theta_i \leq \frac{\pi}{2})$ *are the principal angles between the column spaces of* $X^{(1)}$ *and* $X^{(2)}$.

Proof. Let $P_1^{n\times(n-j)}$ and $P_2^{n\times j}$ be the orthogonal complements of Q_1 and Q_2. We have

$$[Q_1, P_1]^T X = \begin{bmatrix} R_1 & Q_1^T Q_2 R_2 \\ \mathbf{0} & P_1^T Q_2 R_2 \end{bmatrix}.$$

It follows that

$$\det X = \pm \det([Q_1, P_1]^T X) = \pm \det R_1 \det R_2 \det P_1^T Q_2.$$

Define $W = \begin{bmatrix} Q_1^T Q_2 \\ P_1^T Q_2 \end{bmatrix}$. It is easy to show that the columns of W are orthonormal. By the CS decomposition theorem (thin version) [30, page 77], we know that there exist orthogonal matrices $U_1 \in \mathrm{R}^{j\times j}$, $U_2 \in \mathrm{R}^{(n-j)\times(n-j)}$ and $V_1 \in \mathrm{R}^{(n-j)\times(n-j)}$ such that

$$\begin{bmatrix} U_1 & \mathbf{0} \\ \mathbf{0} & U_2 \end{bmatrix}^T \begin{bmatrix} Q_1^T Q_2 \\ P_1^T Q_2 \end{bmatrix} V_1 = \begin{bmatrix} C \\ S \end{bmatrix}, \tag{2.2.6}$$

where

$$C = \text{diag}(\cos\theta_1, \ldots, \cos\theta_{n-j}),$$

$$S = \text{diag}(\sin\theta_1, \ldots, \sin\theta_{n-j}),$$

and

$$0 < \theta_1 \leq \theta_2 \leq \cdots \leq \theta_{n-j} \leq \frac{\pi}{2}.$$

Note that θ_i are the principal angles [30, page 603] between the column spaces of $X^{(1)}$ and $X^{(2)}$. Thus, we have

$$\det P_1^T Q_2 = \pm \det U_2 P_1^T Q_2 V_1 = \pm \det S = \pm \prod_{i=1}^{n-j} \sin\theta_i.$$

It follows that

$$\det X = \pm \det R_1 \det R_2 \prod_{i=1}^{n-j} \sin\theta_i.$$

Therefore,

$$\text{vol } X = \text{vol } X^{(1)} \text{vol } X^{(2)} \prod_{i=1}^{n-j} \sin\theta_i.$$

■

For each β_i, let us permute the columns of X_i and X such that

$$X_i = [y_1\boldsymbol{x}_1, \ldots, y_{i-1}\boldsymbol{x}_{i-1}, y_{i+1}\boldsymbol{x}_{i+1}, \ldots, y_n\boldsymbol{x}_n, \boldsymbol{v}]$$

and

$$X = [y_1\boldsymbol{x}_1, \ldots, y_{i-1}\boldsymbol{x}_{i-1}, y_{i+1}\boldsymbol{x}_{i+1}, \ldots, y_n\boldsymbol{x}_n, y_i\boldsymbol{x}_i].$$

Since X_i and X are permuted in the same way, it is still true that

$$\beta_i = \frac{\det X_i}{\det X} = \frac{\text{vol } X_i}{\text{vol } X}.$$

Let $X_i^{(1)} = [y_1\boldsymbol{x}_1, \ldots, y_{i-1}\boldsymbol{x}_{i-1}, y_{i+1}\boldsymbol{x}_{i+1}, \ldots, y_n\boldsymbol{x}_n]$. Using the lemma above, we get

$$\begin{aligned} \beta_i = \frac{\text{vol } X_i^{(1)} \text{vol } \boldsymbol{v} \sin\theta_i}{\text{vol } X_i^{(1)} \text{vol } y_i\boldsymbol{x}_i \sin\psi_i} &= \frac{\text{vol } \boldsymbol{v} \sin\theta_i}{\text{vol } \boldsymbol{x}_i \sin\psi_i} \\ &= \frac{\|\boldsymbol{v}\|_2}{\|\boldsymbol{x}_i\|_2} \frac{\sin\theta_i}{\sin\psi_i}, \end{aligned} \tag{2.2.7}$$

where θ_i is the principle angle between the column space of $X_i^{(1)}$ and $\boldsymbol{v}$, and ψ_i is the principle angle between the column space of $X_i^{(1)}$ and $y_i\boldsymbol{x}_i$.

Define $\tilde{\boldsymbol{x}}_i$ (the dual vector of $X_i^{(1)}$) such that $X^T\tilde{\boldsymbol{x}}_i = \boldsymbol{e}_i$, where $\boldsymbol{e}_i$ is the ith column of the identity matrix $I_{n\times n}$. If θ_i is redefined as the angle between $\tilde{\boldsymbol{x}}_i$ and $\boldsymbol{v}$, and ψ_i as the angle between $\tilde{\boldsymbol{x}}_i$ and $y_i\boldsymbol{x}_i$, then the formula (2.2.7) becomes

$$\beta_i = \frac{\|\boldsymbol{v}\|_2}{\|\boldsymbol{x}_i\|_2} \frac{\cos\theta_i}{\cos\psi_i}, \tag{2.2.8}$$

where $0 \leq \theta_i, \psi_i < \frac{\pi}{2}$. It is not hard to verify the above formula. Multiplying both sides of Equation (2.2.1) with $\tilde{\boldsymbol{x}}_i^T$ gives us:

$$\begin{aligned} \tilde{\boldsymbol{x}}_i^T X\beta &= \tilde{\boldsymbol{x}}_i^T \boldsymbol{v} \\ \beta_i &= \tilde{\boldsymbol{x}}_i^T \boldsymbol{v} \\ &= \|\tilde{\boldsymbol{x}}_i\|_2 \|\boldsymbol{v}\|_2 \cos\theta_i. \end{aligned}$$

From the definition of $\tilde{\boldsymbol{x}}_i$, we know that

$$\|\tilde{\boldsymbol{x}}_i\|_2 = \frac{1}{\cos\psi_i \|\boldsymbol{x}_i\|_2},$$

where $\cos\psi_i > 0$. Therefore, it is true that $\beta_i = \frac{\|\boldsymbol{v}\|_2}{\|\boldsymbol{x}_i\|_2} \frac{\cos\theta_i}{\cos\psi_i}$. Note that $\cos\theta_i > 0$ because $\beta_i > 0$.

Let us define $\boldsymbol{h} = \frac{2}{\|\boldsymbol{v}\|_2^2}\boldsymbol{v}$. If the training set is separable, then $\boldsymbol{v} = \boldsymbol{w}$, and $\|\boldsymbol{h}\|_2$ is the size of the margin between the two SVM separating hyperplanes. From the fact that $\boldsymbol{v} = \frac{2}{\|\boldsymbol{h}\|_2^2}\boldsymbol{h}$, it follows that

$$\begin{aligned} \beta_i &= \frac{2}{\|\boldsymbol{h}\|^2} \frac{\|\boldsymbol{h}\|}{\|\boldsymbol{x}_i\|} \frac{\cos\theta_i}{\cos\psi_i} \\ &= \frac{1}{\frac{1}{2}\|\boldsymbol{h}\|\|x_i\|} \frac{\cos\theta_i}{\cos\psi_i}. \end{aligned} \tag{2.2.9}$$

Let φ_i be the angle between $\boldsymbol{h}$ and $\boldsymbol{x}_i$ so that

$$\text{Area}(\triangle \boldsymbol{x}_i \boldsymbol{o} \boldsymbol{h}) = \frac{1}{2}\|\boldsymbol{h}\|\|\boldsymbol{x}_i\| \sin\varphi_i. \tag{2.2.10}$$

Thus, we have a formula for β_i $(i = 1, \ldots, n)$ in terms of three angles and the area of a triangle:

$$\beta_i = \frac{1}{\text{Area}(\triangle \boldsymbol{x}_i \boldsymbol{o} \boldsymbol{h})} \frac{\sin\varphi_i \cos\theta_i}{\cos\psi_i}. \tag{2.2.11}$$

We conclude the above discussion by the following theorem (square version).

THEOREM 2.2.1 *Suppose that $\alpha_1, \ldots, \alpha_{n+1}$ are the Lagrange multipliers associated with the SVs in $S_1 = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{n+1}\}$, where $\boldsymbol{x}_i \in \mathbb{R}^n$. Let $\boldsymbol{v} = \boldsymbol{w} - c\sum_{\boldsymbol{x}_j \in S_2} y_i(\boldsymbol{x}_j - \boldsymbol{x}_{n+1})$, $\boldsymbol{h} = \frac{2}{\|\boldsymbol{v}\|_2^2}\boldsymbol{v}$ and $\hat{\boldsymbol{x}}_i = \boldsymbol{x}_i - \boldsymbol{x}_{n+1}$. Define $\tilde{\boldsymbol{x}}_i$ such that $[y_1\hat{\boldsymbol{x}}_1, \ldots, y_n\hat{\boldsymbol{x}}_n]^T\tilde{\boldsymbol{x}}_i = \boldsymbol{e}_i$. For $i = 1, \ldots, n$, we have*

$$\alpha_i = \frac{1}{\text{Area}(\triangle \hat{\boldsymbol{x}}_i \boldsymbol{o} \boldsymbol{h})} \frac{\sin\varphi_i \cos\theta_i}{\cos\psi_i},$$

where θ_i is the angle between $\tilde{\boldsymbol{x}}_i$ and $\boldsymbol{h}$, ψ_i the angle between $\tilde{\boldsymbol{x}}_i$ and $y_i\hat{\boldsymbol{x}}_i$, and φ_i the angle between $\hat{\boldsymbol{x}}_i$ and $\boldsymbol{h}$. Note that $\sin\varphi_i, \cos\theta_i, \cos\psi_i > 0$ and

$$\alpha_{n+1} = -y_{n+1}\Big(\sum_{i=1}^{n} y_i\alpha_i + c\sum_{\boldsymbol{x}_j \in S_2} y_j\Big).$$

If X is a thin matrix, then we have the following equation to solve:

$$(Q^T X)\boldsymbol{\beta} = Q^T \boldsymbol{v}, \tag{2.2.12}$$

where Q defines an orthogonal basis for the column space of X. Since (2.2.12) is still a square linear system, this theorem also holds when X is thin. Although there exists a simpler proof of this theorem, the above version is more insightful, which connects all these interesting linear algebra concepts together.

The above theorem clearly explains the scaling property of SVMs. When the geometry is inflated (deflated) by a factor t, the term $\frac{\sin\varphi_i \cos\theta_i}{\cos\psi_i}$ does not change, while the term $\frac{1}{\text{Area}(\triangle \hat{\boldsymbol{x}}_i \boldsymbol{o} \boldsymbol{h})}$ decreases (increases) by a factor t^2. Hence, β_i also decreases (increases) by a factor t^2 correspondingly. Note that to make the SVM decision rule $h_{\text{SVM}}(\boldsymbol{x})$ invariant to the inflation (deflation) of the geometry, the coefficient c must also be decreased (increased) by a factor t^2. This property is important when the dual problem is computed, where the training points $\boldsymbol{x}_i$ are usually scaled to prevent α_i from being too small or too large. For example, if an interior point method is used, then very small α_i is not preferred from the numerics point of view.

COROLLARY 2.2.1 *When the geometry is inflated (deflated) by a factor t, in order to make the SVM decision rule invariant to the inflation (deflation), all the Lagrange multipliers must be decreased (increased) by a factor t^2.*

2.3 The Simplex Decomposition Relation at Optimality

When the training set are separable, S_2 is empty. Suppose that there are still $n+1$ SVs in S_1. From the fact that the primal objective function is equal to the dual objective function at optimality, it follows that

$$\|\boldsymbol{w}\|_2^2 = \sum_{i=1}^{n+1} \alpha_i. \tag{2.3.1}$$

Without loss of generality, let us assume that $\boldsymbol{x}_{n+1} = 0$ and $y_{n+1} = -1$. When $y_{n+1} = -1$, the constraint $\sum_{i=1}^{n+1} y_i \alpha_i = 0$ is equivalent to $\alpha_{n+1} = \sum_{y_i=1} \alpha_i - \sum_{y_j=-1} \alpha_j$. Plugging it into Equation (2.3.1) gives us

$$\begin{aligned} \|\boldsymbol{w}\|_2^2 = \sum_{i=1}^{n+1} \alpha_i = 2 \sum_{y_i=1} \alpha_i = 2 \sum_{y_i=-1} \alpha_i \\ = \frac{4}{\|\boldsymbol{h}\|_2^2}. \end{aligned} \tag{2.3.2}$$

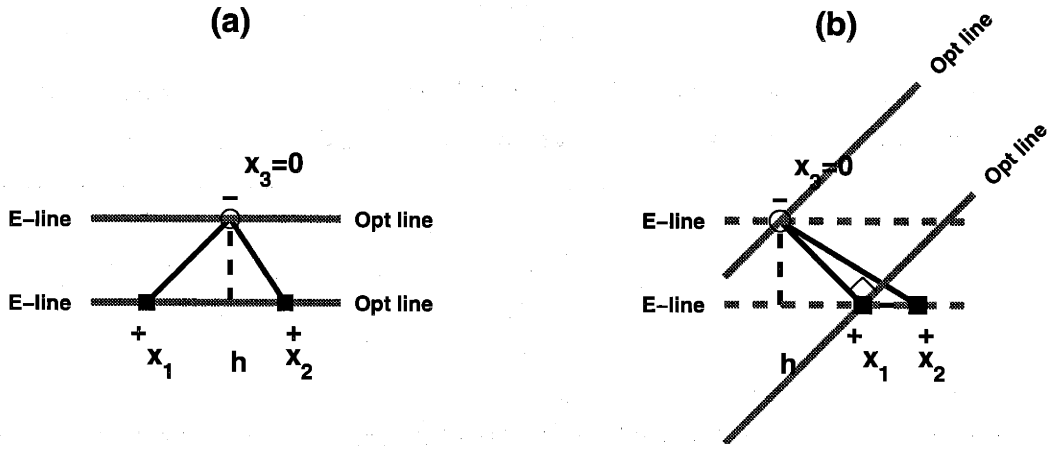


Figure 2.1: Two SVM separation problems in $\mathbb{R}^2$. The horizontal lines labeled by "E-line" are the pair of parallel separating lines determined by the equalities $y_i(\boldsymbol{w}^T\boldsymbol{x}_i + b) = 1$ for $i = 1, 2, 3$. For both cases, the solid lines labeled by "opt line" are the optimal separator. Squares and circles are used to indicate the positive and negative points respectively.

By further substituting α_i with

$$\alpha_i = \frac{2}{\|\boldsymbol{h}\|_2^2} \frac{\text{vol } X_i}{\text{vol } X},$$

where $\boldsymbol{v}$ has been replaced by $\frac{2}{\|\boldsymbol{h}\|_2^2}\boldsymbol{h}$, we derive the following volume decomposition relation

$$\text{vol } X = \sum_{y_i=1} \text{vol } X_i.$$

Note that the value of y_{n+1} does not mater. If $y_{n+1} = 1$, then

$$\text{vol } X = \sum_{y_i=-1} \text{vol } X_i.$$

Hence, the general volume decomposition relation is

$$\text{vol } X = \sum_{y_i=-y_{n+1}} \text{vol } X_i. \tag{2.3.3}$$

At optimality, the volume decomposition relation must hold. It can be shown that the opposite direction is also true. When the volume decomposition holds, $\boldsymbol{h}$ must lie in the cone generated by $y_1\boldsymbol{x}_1, \ldots, y_n\boldsymbol{x}_n$. It follows that $\alpha_i \geq 0$ for $i = 1, \ldots, n$. Therefore, the solution must be optimal.

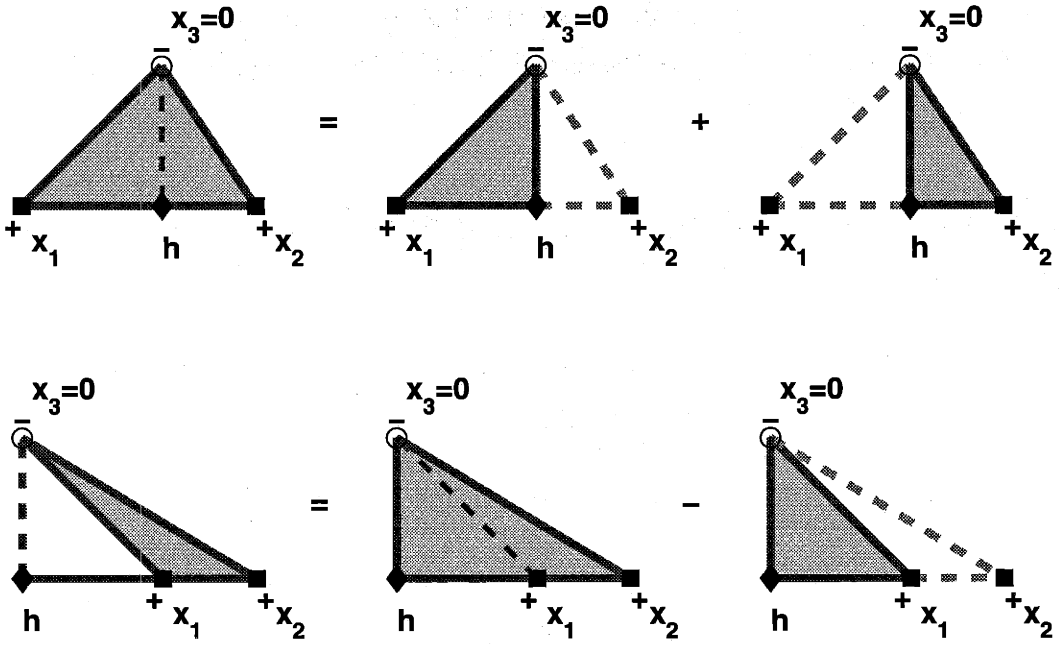


Figure 2.2: The simplex decompositions for the acute and obtuse triangle cases shown in Figure 2.1.

If $\boldsymbol{x}_{n+1}$ is the only negative (positive) point, then (2.3.3) can be replaced by the following simplex decomposition relation:

$$\text{simplex } X = \sum_{y_i = y_{n+1}} \text{simplex } X_i. \tag{2.3.4}$$

In Figure 2.1, there are two examples of the SVM separation problem. We use *E-lines* here to indicate the pair of parallel separating lines determined by the equalities $y_i(\boldsymbol{w}^T\boldsymbol{x}_i + b) = 1$ for $i = 1, 2, 3$. The letter E represents "equality". Using the simplex decomposition relation, we can see that the pair of E-lines for the acute triangle case are optimal, while they are not optimal for the obtuse triangle case. The simplex decompositions for these two cases are illustrated in Figure 2.2. For the acute triangle case, α_i can be either expressed in terms

of areas:

$$\alpha_1 = \frac{2}{\|h\|_2^2} \frac{\text{Area}(\triangle x_2 x_3 h)}{\text{Area}(\triangle x_1 x_3 h)}$$

$$\alpha_2 = \frac{2}{\|h\|_2^2} \frac{\text{Area}(\triangle x_1 x_3 h)}{\text{Area}(\triangle x_2 x_3 h)}$$

$$\alpha_3 = \frac{2}{\|h\|_2^2} \frac{\text{Area}(\triangle x_1 x_2 h)}{\text{Area}(\triangle x_1 x_3 h)}$$

$$= \frac{2}{\|h\|^2},$$

or in terms of angles:

$$\begin{aligned}
\alpha_1 &= \frac{\sin \angle x_1 x_3 h}{\text{Area}(\triangle x_1 x_3 h)} \times \frac{\cos \angle x_3 x_2 x_1}{\sin \angle x_1 x_3 x_2} \\
&= \frac{2}{h^2} \times \frac{\cos \angle x_1 x_3 h \cos \angle x_3 x_2 x_1}{\sin \angle x_1 x_3 x_2} \\
&= \frac{2}{h^2} \times \frac{\cos \angle x_1 x_3 h \sin \angle x_2 x_3 h}{\sin \angle x_1 x_3 x_2} \\
\alpha_2 &= \frac{\sin \angle x_2 x_3 h}{\text{Area}(\triangle x_2 x_3 h)} \times \frac{\cos \angle x_3 x_1 x_2}{\sin \angle x_1 x_3 x_2} \\
&= \frac{2}{h^2} \times \frac{\cos \angle x_2 x_3 h \cos \angle x_3 x_1 x_2}{\sin \angle x_1 x_3 x_2} \\
&= \frac{2}{h^2} \times \frac{\sin \angle x_3 x_1 x_3 h \cos \angle x_2 x_3 h}{\sin \angle x_1 x_3 x_2} \\
\alpha_3 &= \alpha_1 + \alpha_2 = \frac{2}{h^2}.
\end{aligned}$$

Figure 2.3 shows an example of a three-dimensional simplex decomposition. Note that the simplex decomposition relation is equivalent to that $\boldsymbol{h}$ is in the cone generated by $y_i \boldsymbol{x}_i$ for $i = 1, \ldots, n$.

2.4 Compressing the SVs

In the previous discussion, it is assumed that the SVs are compact. We have seen that this assumption enables us to reduce the dual QP problem to a linear system. However, S_1 is not guaranteed to be compact due to the rank deficiency of the Hessian matrix H. In this section, we discuss how to compress S_1 to make it compact. The difficulty is that generally $\phi(\cdot)$ is not known. Hence, we need find an algorithm that only involves the inner products $k(\boldsymbol{x}_i, \boldsymbol{x}_j)$, where $\boldsymbol{x}_i, \boldsymbol{x}_j \in S_1 \cup S_2$. Again, we use $\boldsymbol{x}_i$ in place of $\phi(\boldsymbol{x}_i)$ for the simplicity of notation.

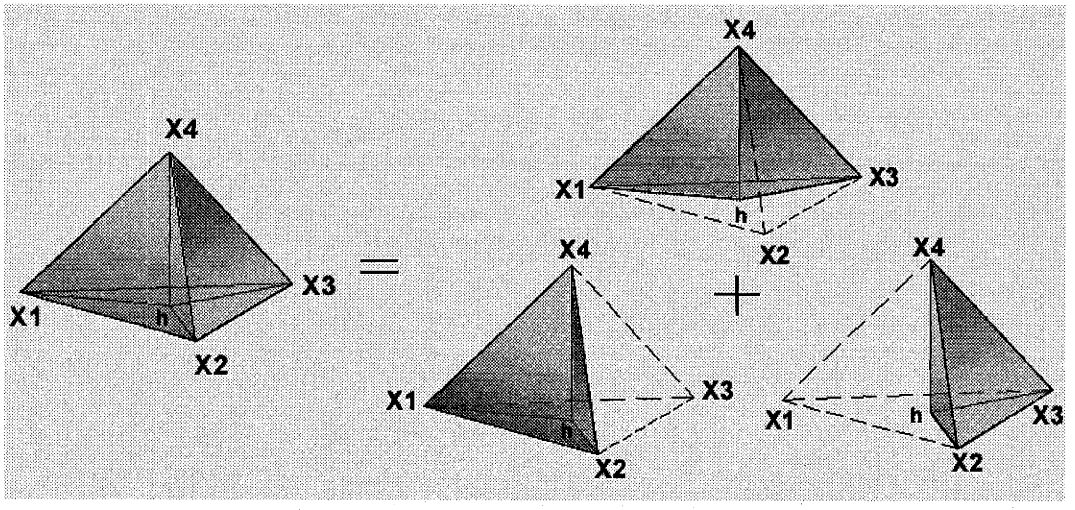


Figure 2.3: A three-dimensional simplex decomposition.

Let $S_1 = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_l\}$, $X = [\boldsymbol{x}_1, \ldots, \boldsymbol{x}_l]$ and $\hat{n} = \text{rank}(X^T X)$. If $\hat{n} < l - 1$, then S_1 is not compact. To further compress S_1, we need to find a subset $\hat{S}_1 = \{\boldsymbol{x}_{i_1}, \ldots, \boldsymbol{x}_{i_{\hat{n}+1}}\}$ of S_1 such that $y_{i_j}(\boldsymbol{x}_{i_j} - \boldsymbol{x}_{i_{\hat{n}+1}})$ $(j = 1, \ldots, \hat{n})$ are linearly independent, and $\boldsymbol{w} - c\sum_{j \in S_2} y_j \boldsymbol{x}_j$ can be expressed as a linear combination of $y_{i_j}\boldsymbol{x}_{i_j}$ with nonnegative coefficients β_j. That is,

$$\begin{aligned} \boldsymbol{v} = \boldsymbol{w} - c\sum_{j \in S_2} y_j \boldsymbol{x}_j &= \sum_{j=1}^{l} y_j \alpha_j \boldsymbol{x}_j \\ &= \sum_{j=1}^{\hat{n}+1} \beta_j y_{i_j} \boldsymbol{x}_{i_j}, \end{aligned}$$

where $\beta_j \geq 0$. Geometrically, we want to know whether $\boldsymbol{v}$ is in the cone generated by $y_{i_j}\boldsymbol{x}_{i_j}$ $(j = 1, \ldots, \hat{n} + 1)$. Since the equality

$$\sum_{j=1}^{\hat{n}+1} \beta_j y_{i_j} = 0$$

must hold at optimality, $\boldsymbol{\beta} = \{\beta_1, \ldots, \beta_{\hat{n}+1}\}$ is determined by the following two linear equations:

$$\boldsymbol{v} = \sum_{j=1}^{\hat{n}} \beta_j y_{i_j} (\boldsymbol{x}_{i_j} - \boldsymbol{x}_{i_{\hat{n}+1}})$$

and

$$\beta_{\hat{n}+1} = -y_{i_{\hat{n}+1}} \sum_{j=1}^{\hat{n}} \beta_j y_{i_j}.$$

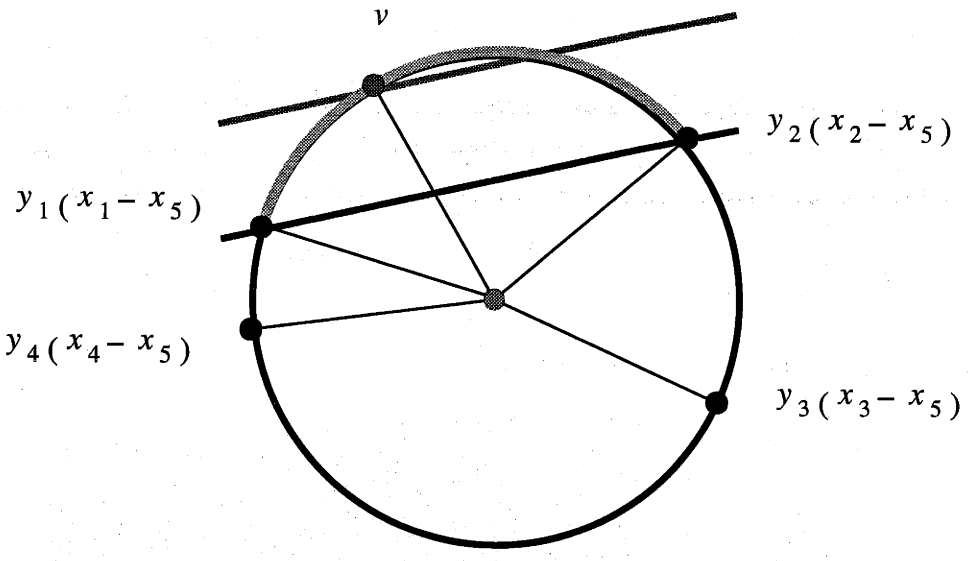


Figure 2.4: A two-dimensional example of how to determine $\hat{S}_1$. Assume that $\boldsymbol{v}$ and $y_i(\boldsymbol{x}_i - \boldsymbol{x}_5)$ for $i = 1, \ldots, 4$ are of unit length, and that $\boldsymbol{x}_5$ has been chosen at the first step. This algorithm picks up $y_1(\boldsymbol{x}_1 - \boldsymbol{x}_5)$ and $y_2(\boldsymbol{x}_2 - \boldsymbol{x}_5)$ as the positive SVs for the new SVM separation problem, where $\boldsymbol{v}$ is the only negative point. Hence, $\hat{S}_1 = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{x}_5\}$.

If $\boldsymbol{\beta} \geq \mathbf{0}$, then $\hat{S}_1 \cup S_2$ is a compact set of SVs and $\boldsymbol{\beta}$ gives the new values of the Lagrange multipliers corresponding to $\hat{S}_1$. Unfortunately, we can not invert the above linear system to see if β_j are all nonnegative. To find an algorithm only involving inner products, we need to look at the dual problem.

The property of SVMs says that if the points in $S_1 - \hat{S}_1$ are redundant, then the QP problem (2.1.8) based on $\hat{S}_1 \cup S_2$ has nonnegative solution. Since the QP problem (2.1.8) is equivalent to the linear system (2.1.10), inverting this linear system gives us $\boldsymbol{\beta}$. Note that if $|S_2| = 0$ we need to solve (2.1.7). Although it is easy to verify whether $\boldsymbol{\beta} \geq \mathbf{0}$ for a given $\hat{S}_1$, the problem is how to find such a $\hat{S}_1$. We can always find a desirable $\hat{S}_1$ by trying all the possible combinations. But this approach is not practical. Let us consider the worst case where $C(l, \hat{n}+1)$ linear systems of size $(\hat{n}+2) \times (\hat{n}+2)$ have to be solved. Although $\hat{n}$ and l are generally small comparing with the size of the training set, $C(l, \hat{n}+1)$ can be very large. When $C(l, \hat{n}+1)$ is large, this approach is impractical.

To determine $\hat{S}_1$, our idea is that we first normalize $\boldsymbol{v}$ and $y_i(\boldsymbol{x}_i - \boldsymbol{x}_{i_{\hat{n}+1}})$ for $i = 1, \ldots, l$ except $i_{\hat{n}+1}$, so that they have the same length. Here, $\boldsymbol{x}_{i_{\hat{n}+1}}$ can be any point in S_1. Since every vector has the same length now, the angles between them give a measure of how these

vectors are close to each other. From the geometry, $\boldsymbol{v}$ and these $l-1$ points $y_i(\boldsymbol{x}_i - \boldsymbol{x}_{i_{\hat{n}+1}})$ must be linearly separable. Motivated by the fact that the size of the gap between the two SVM separating hyperplanes defines the distance between the positive and negative training points if they are separable. We then use a maximal-margin SVM to pick up $\hat{n}$ points $y_{i_j}(\boldsymbol{x}_{i_j} - \boldsymbol{x}_{i_{\hat{n}+1}})$ that are close to $\boldsymbol{v}$. The simplex decomposition relation guarantees that $\boldsymbol{v}$ is in the cone generated by these $\hat{n}$ selected points.

In details, this method consists of three steps. We first choose $\boldsymbol{x}_{i_{\hat{n}+1}}$ arbitrarily. Then we use the algorithm introduced in the following paragraph to pick up $\hat{n}$ points $\boldsymbol{x}_{i_1}, \ldots, \boldsymbol{x}_{i_{\hat{n}}}$, such that $\boldsymbol{v}$ is in the cone generated by $y_{i_j}(\boldsymbol{x}_{i_j} - \boldsymbol{x}_{i_{\hat{n}+1}})$ for $j = 1, \ldots, \hat{n}$. Finally, let $\hat{S}_1 = \{\boldsymbol{x}_{i_1}, \ldots, \boldsymbol{x}_{i_{\hat{n}+1}}\}$. We compute $\boldsymbol{\beta}$ by inverting the linear system (2.1.10) or (2.1.7) based on $\hat{S}_1 \cup S_2$. If $\beta_{\hat{n}+1} \geq 0$, then $\hat{S}_1 \cup S_2$ is a compact set of SVs; otherwise, we restart this procedure with a new $\boldsymbol{x}_{i_{\hat{n}+1}}$. You can see that at most we need to try l times.

Suppose that $\boldsymbol{x}_{i_{\hat{n}+1}} = \boldsymbol{x}_l$. To determine $\hat{S}_1$, let us normalize $\boldsymbol{v}$ and $(\boldsymbol{x}_1 - \boldsymbol{x}_l), \ldots, (\boldsymbol{x}_{l-1} - \boldsymbol{x}_l)$ so that they are on the unit sphere. If we construct a new training set by marking $\boldsymbol{v}$ (normalized) as the only negative point and $y_1(\boldsymbol{x}_1 - \boldsymbol{x}_l), \ldots, y_{l-1}(\boldsymbol{x}_{l-1} - \boldsymbol{x}_l)$ (normalized) as the positive points, then from the geometry we know that this training set must be linearly separable. By the simplex decomposition relation, it is guaranteed that $\boldsymbol{v}$ must lie in the cone generated by the positive SVs of this new SVM separation problem. Therefore, these positive SVs are the ones we are looking for. As indicated by Figure 2.4, this algorithm generally pick up $\hat{n}$ normalized $y_i(\boldsymbol{x}_i - \boldsymbol{x}_l)$ near the normalized $\boldsymbol{v}$. The resulting $\hat{n}+1$ points gives us $\hat{S}_1$. Since l is generally small, it is cheap to the solve the corresponding QP problem, which only involves inner products.

The worst case for our approach is to solve l SVM separation problems (based on a training set of size l) and l $(\hat{n}+2) \times (\hat{n}+2)$ linear systems. Given that l is generally small, it is not expensive to compute these problems. Therefore, computationally our approach is practical. Although it is not hard to see that this algorithm works for the two and three dimensional cases, we still owe readers a general proof for all the dimensions.

2.5 Making the Non-separable Case Separable

If the training set can not be separated by the pair of hyperplanes determined by $(\boldsymbol{w}, b)$, then S_2 is not empty. To make the points in S_2 separable, we can project them onto the corresponding hyperplanes, that is, the positive points onto the positive one and the negative points onto the negative one. In this section, we show that the current hyperplanes

determined by $(\boldsymbol{w}, b)$ still optimally separate the new training points $S_1 \cup \tilde{S}_2$, where $\tilde{S}_2$ is the projected version of S_2.

Let $\boldsymbol{x}_i + \Delta\boldsymbol{x}_i$ be the projected version of $\boldsymbol{x}_i \in S_2$. From the equations

$$y_i(\boldsymbol{w}^T\boldsymbol{x}_i + b) = 1 - \varepsilon_i$$

and

$$y_i[\boldsymbol{w}^T(\boldsymbol{x}_i + \Delta\boldsymbol{x}_i) + b] = 1,$$

we know that $\Delta\boldsymbol{x}_i = \frac{y_i\varepsilon_i}{\|\boldsymbol{w}\|_2^2}\boldsymbol{w} = y_i\delta_i\boldsymbol{w}$. Let $\tilde{\alpha}_i$ be the Lagrange multipliers associated with the points in $S_1 \cup \tilde{S}_2$. To preserve $\boldsymbol{w}$, $\tilde{\alpha}_i$ need to satisfy the following equation:

$$\boldsymbol{w} = \sum_{i=1}^{l} \alpha_i y_i \boldsymbol{x}_i + c \sum_{j=l+1}^{m} y_j \boldsymbol{x}_j = \sum_{i=1}^{l} \tilde{\alpha}_i y_i \boldsymbol{x}_i + \sum_{j=l+1}^{m} \tilde{\alpha}_j y_j (\boldsymbol{x}_j + \Delta\boldsymbol{x}_j), \tag{2.5.1}$$

where $m = |S_1 \cup \tilde{S}_2|$. Denote $[y_1\boldsymbol{x}, \ldots, y_l\boldsymbol{x}_l, y_{l+1}\boldsymbol{x}_{l+1}, \ldots, y_m\boldsymbol{x}_m]$ by X. The above equation becomes

$$X \begin{bmatrix} \alpha_1 \\ \vdots \\ \alpha_l \\ c \\ \vdots \\ c \end{bmatrix} = (X + [0, \ldots, 0, \delta_{l+1}\boldsymbol{w}, \ldots, \delta_m\boldsymbol{w}]) \begin{bmatrix} \tilde{\alpha}_1 \\ \vdots \\ \tilde{\alpha}_l \\ \tilde{\alpha}_{l+1} \\ \vdots \\ \tilde{\alpha}_m \end{bmatrix}, \tag{2.5.2}$$

equivalently

$$X \begin{bmatrix} \alpha_1 \\ \vdots \\ \alpha_l \\ c \\ \vdots \\ c \end{bmatrix} = (X + X \begin{bmatrix} \alpha_1 \\ \vdots \\ \alpha_l \\ c \\ \vdots \\ c \end{bmatrix} [0, \ldots, 0, \delta_{l+1}, \ldots, \delta_m]) \begin{bmatrix} \tilde{\alpha}_1 \\ \vdots \\ \tilde{\alpha}_l \\ \tilde{\alpha}_{l+1} \\ \vdots \\ \tilde{\alpha}_m \end{bmatrix}. \tag{2.5.3}$$

If X is a square nonsingular matrix, then we have

$$\begin{bmatrix} \alpha_1 \\ \vdots \\ \alpha_l \\ c \\ \vdots \\ c \end{bmatrix} = (I + \begin{bmatrix} \alpha_1 \\ \vdots \\ \alpha_l \\ c \\ \vdots \\ c \end{bmatrix} [0, \ldots, 0, \delta_{l+1}, \ldots, \delta_m]) \begin{bmatrix} \tilde{\alpha}_1 \\ \vdots \\ \tilde{\alpha}_l \\ \tilde{\alpha}_{l+1} \\ \vdots \\ \tilde{\alpha}_m \end{bmatrix}. \tag{2.5.4}$$

The good thing about this equation is that the unknowns of $\tilde{\alpha}_{l+1}, \ldots, \tilde{\alpha}_m$ are independent of the unknowns $\tilde{\alpha}_1, \ldots, \tilde{\alpha}_l$:

$$\begin{bmatrix} c \\ \vdots \\ c \end{bmatrix} = (I + \begin{bmatrix} c \\ \vdots \\ c \end{bmatrix} [\delta_{l+1}, \ldots, \delta_m]) \begin{bmatrix} \tilde{\alpha}_{l+1} \\ \vdots \\ \tilde{\alpha}_m \end{bmatrix}. \tag{2.5.5}$$

It follows that for $i = l+1, \ldots, m$

$$\tilde{\alpha}_i = \frac{c}{1 + c\sum_{i=l+1}^{m} \delta_i},$$

and for $i = 1, \ldots, l$

$$\tilde{\alpha}_i = \frac{\alpha_i}{1 + c\sum_{i=l+1}^{m} \delta_i},$$

where $\delta_i = \frac{\varepsilon_i}{\|w\|_2^2}$. It is easy to verify that the above solution still satisfies Equation (2.5.1) when X is an arbitrary matrix. Since $\tilde{\alpha}_i > 0$, the pair of hyperplanes determined by $(\boldsymbol{w}, b)$ is still optimal for the new training set. We conclude this discussion by the following theorem:

THEOREM 2.5.1 *Let $S_1 \cup S_2$ ($|S_2| > 0$) be the SVs and $(\boldsymbol{w}, b)$ the solution to the primal SVM QP problem. If $S_2 = \{\boldsymbol{x}_{l+1}, \ldots, \boldsymbol{x}_m\}$ is replaced by $\tilde{S}_2 = \{\boldsymbol{x}_{l+1} + \Delta\boldsymbol{x}_{l+1}, \ldots, \boldsymbol{x}_m + \Delta\boldsymbol{x}_m\}$, where*

$$\Delta\boldsymbol{x}_i = \frac{y_i \varepsilon_i}{\|\boldsymbol{w}\|_2^2} \boldsymbol{w},$$

then the new training points $S_1 \cup \tilde{S}_2$ are separable by the original SVM separating hyperplanes and this pair of hyperplanes are still optimal, that is, they separate $S_1 \cup \tilde{S}_2$ with the maximum gap.

2.6 Future Work

The results derived in this chapter are useful for us to understand SVMs better and to compute them efficiently. It is also interesting to know if these results could be used to develop bonds on the performance of SVMs.

Chapter 3

The Inverse Wishart Matrix $W_n^{-1}(n, nI)$

When $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_m$ are independent Gaussian random vectors, the Hessian matrix H ($H_{ij} = y_i \boldsymbol{x}_i^T \boldsymbol{x}_j y_j$) is the Wishart matrix $W_m(n)$ in statistics [53, page 82]. In this chapter, we show how the separation of $n+1$ random points in $\mathbb{R}^n$ leads us to the marginal distributions of the inverse Wishart matrix $W_n^{-1}(n, nI_{n\times n})$.

3.1 Observations from the Separation of $n+1$ General Points in R^n

Given $n+1$ points $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{n+1}$ at general positions in $\mathbb{R}^n$, a fact is that no mater how we label them, the positive and negative points can always be separated by the two parallel hyperplanes determined by the following equations:

$$y_i(\boldsymbol{w}^T \boldsymbol{x}_i + b) = 1 \quad i = 1, \ldots, n+1.$$

Remember that these two hyperplanes are the E-hyperplanes in Chapter 2. Since there are $n+1$ non-degenerated linear equations for $n+1$ unknowns, the E-hyperplanes must exist. From this fact, it follows that the VC dimension of the set of oriented hyperplanes in $\mathbb{R}^n$ is $n+1$.

Again, let $X = [(\boldsymbol{x}_1 - \boldsymbol{x}_{n+1}), \ldots, (\boldsymbol{x}_1 - \boldsymbol{x}_{n+1})]$, which is nonsingular due to the assumption that $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{n+1}$ are at general positions. Suppose these $n+1$ points are the SVs. From Chapter 2, we know that the corresponding Lagrange multipliers are

$$\boldsymbol{\alpha}(1:n) = \text{diag}(\boldsymbol{y}) H^{-1} \begin{bmatrix} \mathbf{2} \\ \mathbf{0} \end{bmatrix} \tag{3.1.1}$$

and

$$\alpha_{n+1} = \sum_{i=1}^{n} y_i \alpha_i, \tag{3.1.2}$$

where $\boldsymbol{x}_{n+1}$ is labeled by $y_{n+1} = -1$. The hypothesis that these $n+1$ points are the SVs is true, or equivalently that the E-hyperplanes are optimal is true, if and only if $\boldsymbol{\alpha} \geq \mathbf{0}$. From (3.1.1), it is obvious that the signs of α_i depend on the elements of H^{-1}.

If $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{n+1}$ are random vectors, say the elements of $\boldsymbol{x}_i$ have the same Gaussian distribution $N(0, \frac{1}{\sqrt{2}})$ and they are independent, then in statistics, H^{-1} is known as the inverse Wishart matrix $W_n^{-1}(n, nI)$. Here, nI is the covariance matrix. In the following discussion, we use a short-hand notation W^{-1} for this inverse Wishart matrix. Since permuting the columns of X does not affect the distribution of W^{-1}, we conclude that the Lagrange multipliers associated with the positive points must have identical distributions and so do the Lagrange multipliers associated with the negative points. Without loss of generality, let us assume that the first l points are labeled positive, where $l < n$. To learn the distributions of α_i, it is enough to consider only two Lagrange multipliers α_1 and α_{l+1}, which are denoted by α^+ and α^- respectively.

Figure 3.1 plots the histograms of α^+ and α^- with different l. From the histograms at $l = 1$, we can see that the diagonal elements of W^{-1} are always positive and the off-diagonal elements have a symmetric distribution. As l increases, more off-diagonal elements are added in, so the histograms of α^+ and α^- tend to spread out, i.e., α^+ and α^- tend to have larger variance. The histogram of α^+ also becomes more symmetric as l increases. This observation is consistent with the Monte Carlo approximations to $\Pr(\alpha_i \geq 0)$ for $i = 1, \ldots, n$ as shown in Figure 3.2, where $\Pr(\alpha^+ \geq 0)$ monotonically decreases as l increases and $\Pr(\alpha^- \geq 0) = 0.5$ for every l. The right plot in Figure 3.2 indicates that $\Pr(\alpha^+ \geq 0)$ and $\Pr(\alpha^- \geq 0)$ do not depend on n. These observations motivate us to examine how the elements of W^{-1} are distributed.

3.2 The Marginal Distribution of $W_n^{-1}(n, nI)$

Although the joint density function of W^{-1} is known [53, page 113], we have not at this time chosen to verify the above observations by using this density function. To shed light on the distributions of α_i, we examine the distributions of the elements of W^{-1} with a linear algebra approach.

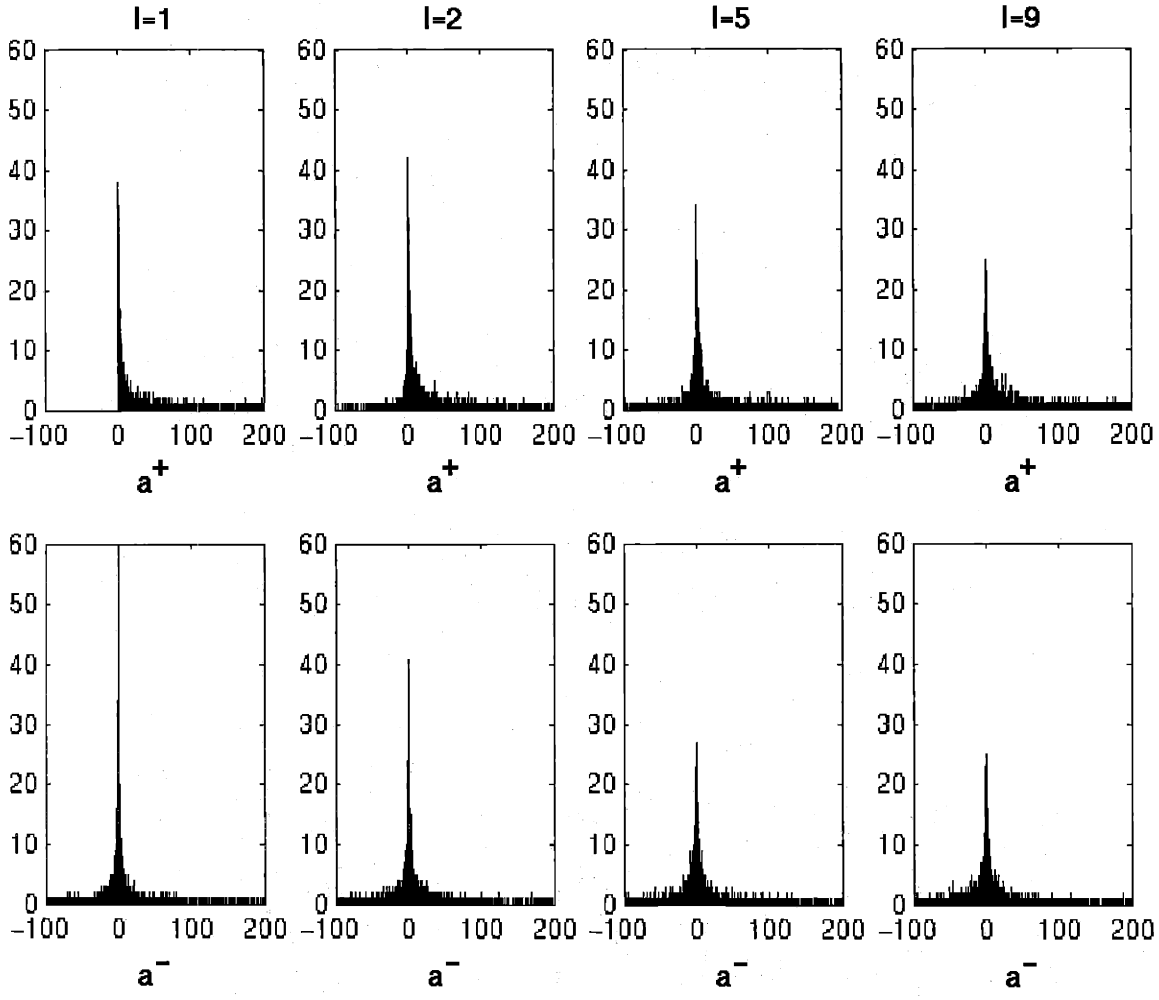


Figure 3.1: Histograms of α^+ and α^- with different l. The size of each experiment is 8000 and $n = 12$.

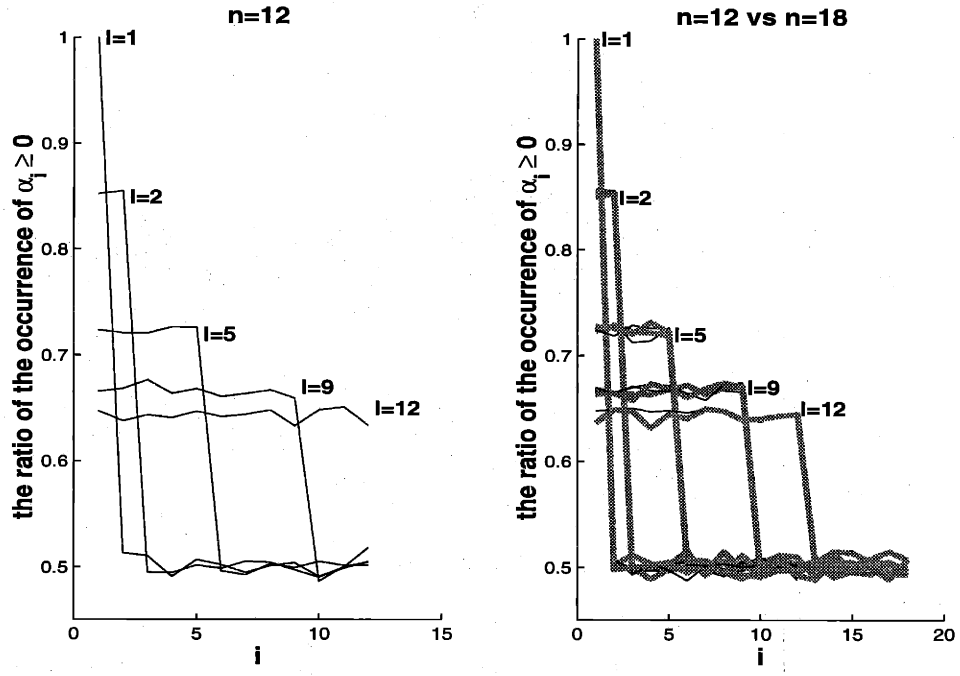


Figure 3.2: Each curve plots the ratio of the occurrence of $\alpha_i \geq 0$ for $i = 1, \ldots, n$ at l during 8000 experiments . In the right plot, two cases with $n = 12$ and $n = 18$ are plotted against each other indicating that $\Pr(\alpha_i \geq 0)$ is independent of n.

Let $\frac{1}{n}W = T^T T$ be the Cholesky factorization of $\frac{1}{n}W$, where T is an upper-triangular matrix with positive diagonal elements. It is known that the elements t_{ij} $(1 \leq i \leq j \leq n)$ of T are all independent, and that t_{ii}^2 $(i = 1, \ldots, n)$ are $\mathcal{X}_{n-i+1}^2$ random variables and t_{ij} are $N(0,1)$ random variables $(1 \leq i < j \leq n)$ (Theorem 3.2.14 [53, page 99]). Here, $\mathcal{X}_{n-i+1}^2$ represents the chi-square distribution with $n - i + 1$ degrees of freedom [36]. Again by symmetry, the diagonal elements of W^{-1} must have identical distributions and so do the off-diagonal elements. Therefore, examining one diagonal and one off-diagonal element is enough to derive the marginal distributions for all the elements of W^{-1}. From the following equation

$$W^{-1} = \frac{1}{n} T^{-1} T^{-T},$$

we have

$$W_{nn}^{-1} = \frac{1}{n \times t_{nn}^2}, \tag{3.2.1}$$

and

$$W_{n(n-1)}^{-1} = \frac{-t_{n-1n}}{n \times t_{nn}^2 \times t_{n-1n-1}}. \tag{3.2.2}$$

It follows that the following theorem is true:

THEOREM 3.2.1 *Let X be a $n \times n$ matrix whose elements are independent random variables with the standard normal distribution $N(0,1)$. Define $W = X^T X$. The diagonal elements of W^{-1} have the same distribution as*

$$\frac{1}{n \times r_1}, \tag{3.2.3}$$

and the off-diagonal elements have the same distribution as

$$\frac{r_3}{n \times r_1 \times \sqrt{r_2}} = \frac{r_4}{\sqrt{2} \times n \times r_1}. \tag{3.2.4}$$

Here, r_1, r_2 and r_3 are independent random variables with the distributions $\mathcal{X}_1^2$, $\mathcal{X}_2^2$ and $N(0,1)$ respectively, while $r_4 = \frac{\sqrt{2} \times r_3}{\sqrt{r_2}}$ has the t-distribution with 2 degrees of freedom (t_2) [36].

This theorem should explain the observations in Figures 3.1 and 3.2. Let us check several simple cases as examples. At $l = 1$, it is obvious that $\Pr(\alpha^+ \geq 0) = \Pr(r_1 \geq 0) = 1$, and since the p.d.f. of a t-distribution random variable is symmetric, $\Pr(\alpha^- \geq 0) = \Pr(r_4 \geq 0) = 0.5$. At $l = 2$,

$$\Pr(\alpha^+ \geq 0) = \Pr(1 - \frac{r_4}{\sqrt{2}} \geq 0) = \Pr(r_4 \leq \sqrt{2}).$$

We know that the p.d.f of a t_2 random variable is

$$f(r) = \frac{\sqrt{2}}{4(1+\frac{r^2}{2})^{\frac{3}{2}}},$$

where $-\infty < r < \infty$ [36, page 600]. Integrating it from $-\infty$ to $\sqrt{2}$ gives us

$$\Pr(\alpha^+ \geq 0) = \frac{\sqrt{2}}{4} + \frac{1}{2} \approx 0.8536 < 1,$$

which matches the plot in Figure 3.2. In the above theorem, n only appears in the scaling terms. Therefore, n does not affect $\Pr(\alpha^+ \geq 0)$ and $\Pr(\alpha^- \geq 0)$. As we have seen, this theorem is consistent with our observations.

We end this chapter with the following conjecture:

CONJECTURE 3.2.1 *Let $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{n+1}$ be $n+1$ independent random vectors in $\mathbb{R}^n$, where the elements of $\boldsymbol{x}_j$ are i.i.d. Gaussian random variables with the distribution $N(0, \frac{1}{\sqrt{2}})$. For the E-hyperplanes separating the positive points $\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_l\}$ and the negative points $\{\boldsymbol{x}_{l+1}, \ldots, \boldsymbol{x}_{n+1}\}$ $(l < n)$, $\Pr(\alpha^+ \geq 0)$ decreases monotonically as the number of positive points l increases and it does not depend on n, where $\alpha^+ = \alpha_1$ and $\alpha^- = \alpha_{l+1}$. At the same time, $1 \geq \Pr(\alpha^+ \geq 0) > \Pr(\alpha^- \geq 0) = 0.5$.*

From W^{-1} point of view, the above conjecture can be restated as

CONJECTURE 3.2.2 *For the inverse Wishart matrix W^{-1} $(W_n^{-1}(n, nI))$, let us define*

$$\alpha^+ = \sum_{i=1}^{l} W_{1i}^{-1}$$

and

$$\alpha^- = \sum_{i=1}^{l} W_{(l+1)i}^{-1}.$$

$\Pr(\alpha^+ \geq 0)$ decreases monotonically as l increases and it does not depend on n. At the same time, $1 \geq \Pr(\alpha^+ \geq 0) > \Pr(\alpha^- \geq 0) = 0.5$.

3.3 Future Work

In this chapter, we investigate the probabilities for each α_i to be nonnegative. Our final goal is to compute $\Pr(\boldsymbol{\alpha} \geq \mathbf{0})$, which is much more challenging and therefore interesting. We leave it as our future work.

Chapter 4

Solving the SVM Dual QP Problem Efficiently

The more examples SVMs learn from, the better performance they tend to have. However, when the training set $S = \{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_m, y_m)\}$ is large, it becomes expensive to store and access the $m \times m$ Hessian matrix H. Hence, it is hard for the standard QP solvers that take H explicitly as an input to solve the dual QP problem. For instance, when $m = 10,000$, it needs almost 1 gigabyte to store H. Many modern computers may not even have that much memory. Another concern is that when $m > n$, H is rank deficient, which may bring numerical difficulties for certain solvers where the positive definiteness of H is assumed. In this section, we describe a fast, memory efficient and stable algorithm for solving the SVM dual QP problem. Numerical experiments show that the MATLAB implementation of this algorithm is competitive with benchmark C/C++ codes such as SVM^{light} and SvmFu.

4.1 A Projected Conjugate Gradient Algorithm

An important property of the dual QP problem is that its solution is sparse. Ignoring the training examples $(\boldsymbol{x}_i, y_i)$ with $\alpha_i = 0$ does not change the solution. For example, in Figure 4.1, solving the right-hand-side separation problem based on the SVs is equivalent to solving the left-hand-side one, but it is much cheaper. Knowing which training points are the SVs in advance enables us to solve a much smaller problem. One of the main themes of this chapter is to exploit the sparsity property so as to solve the dual QP problem fast and with less memory usage. Recall that the dual problem is

$$\underset{\boldsymbol{\alpha}}{\text{maximize}} \quad F(\boldsymbol{\alpha}) \equiv \boldsymbol{\alpha}^T \mathbf{1} - \frac{1}{2}\boldsymbol{\alpha}^T H \boldsymbol{\alpha}$$

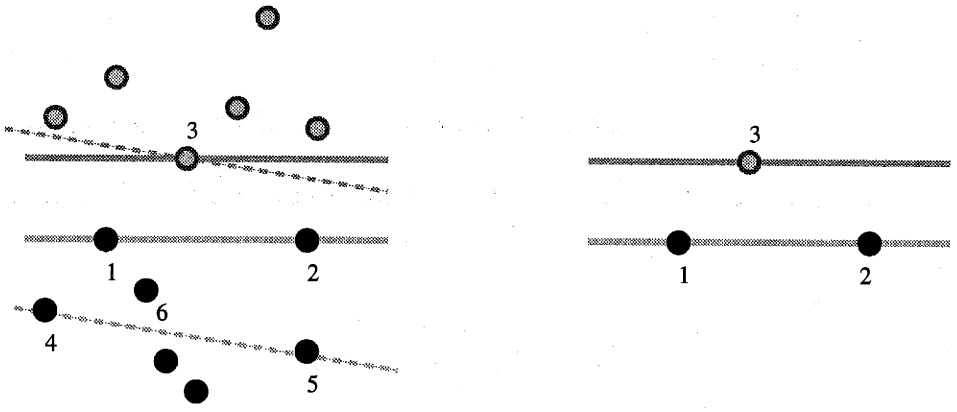


Figure 4.1: An example of a two-dimensional SVM separation problem. The optimal solution is the pair of lines that separate the positive (gray) and negative (dark) points with the largest gap. The SVs $\{\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{x}_3\}$ and the optimal solution are indicated in the right-hand-side figure. The two dotted lines are the SVM solution to separate points $\boldsymbol{x}_3$, $\boldsymbol{x}_4$ and $\boldsymbol{x}_5$.

subject to

$$\boldsymbol{y}^T\boldsymbol{\alpha} = 0,$$
$$\mathbf{0} \leq \boldsymbol{\alpha} \leq \boldsymbol{c}.$$

To illustrate a better way to compute the optimal separating hyperplanes, as an example, let us solve the SVM separation problem indicated in Figure 4.1 in the following way. Initially, set $\boldsymbol{\alpha} = 0$ and we randomly choose a small set, say $G = \{\boldsymbol{x}_3, \boldsymbol{x}_4, \boldsymbol{x}_5\}$ as our guess of the SVs. Then we solve the 3×3 separation problem based on the working set G, and we use its solution (indicated by the doted lines) to test the $\boldsymbol{x}_i$ that are not in G. For any $\boldsymbol{x}_i \in \overline{G}$ separated correctly by the current solution, it is still considered to be a non-SV; otherwise, we add it to G. For this problem, $\{\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{x}_6\}$ are added to G. Finally, solving the updated 6×6 separation problem gives us the optimal solution with the SVs $\{\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{x}_3\}$. Since solving these two smaller subproblems is cheaper than solving the original separation problem as a whole, this idea works. By controlling how many points to add to the working set G at each time, we can control the size of each subproblem. For instance, we can add only one point to G to formulate the second subproblem. In summary, the procedure is

1. Initially, set $\alpha = 0$ and a guess G of the SVs is chosen.

2. The subproblem based on the working set G is constructed and solved.

3. If the solution from Step 2 separates all the points in $\overline{G}$ correctly, **stop**; otherwise, a certain number of points in $\overline{G}$ with the largest separation errors are added to G, and at the same time, the points in G that are not SVs for the current solution can be dropped to $\overline{G}$, then go to step 2.

When $\boldsymbol{\alpha}$ is sparse, solving the sequence of smaller subproblems determined by the above procedure is much cheaper than solving the dual QP problem directly.

As indicated by Figure 4.1, this strategy can be easily appreciated from the primal problem point of view. Our goal is to show how to use this strategy to solve the dual QP problem efficiently, where the training examples may not be separable. In this section, issues such as how to construct, update, and solve the subproblems are addressed. As you will see, our algorithm can be easily implemented in a high-level programming language such as MATLAB. Issues related to the performance improvement will be discussed separately.

Other methods can be found in [34] [40] [61]. Some ideas used here are from the Constrained Conjugate Gradient Algorithm described in [40], while [34] and [61] describe respectively the ideas underlying SVM^{light} and SvmFu. To our knowledge, SVM^{light} is the most widely used training code.

4.1.1 Constructing a Subproblem

First, let us consider in general how to project a linearly constrained QP problem onto an affine space. Suppose a new constraint is added to the dual QP problem requiring that the solution must lie in a subspace, say the column space of a matrix $P \in \mathbb{R}^{n \times p}$, where $p \leq n$. Under this restriction, $\boldsymbol{\alpha}$ can be expressed as $\boldsymbol{\alpha} = P\hat{\boldsymbol{\alpha}}$, where $\hat{\boldsymbol{\alpha}} \in \mathbb{R}^p$. More generally, $\boldsymbol{\alpha}$ can be written as $\boldsymbol{\alpha} = \boldsymbol{\alpha}_0 + P\hat{\boldsymbol{\alpha}}$, where $\boldsymbol{\alpha}_0$ is the initial feasible value of $\boldsymbol{\alpha}$. For this case, $\boldsymbol{\alpha}$ is restricted in the affine space determined by $\boldsymbol{\alpha}_0$ and P. By substituting $\boldsymbol{\alpha}$ with $\boldsymbol{\alpha}_0 + P\hat{\boldsymbol{\alpha}}$ in the dual QP problem, we have

$$\underset{\hat{\boldsymbol{\alpha}}}{\text{maximize}} \quad \hat{F}(\hat{\boldsymbol{\alpha}}) \equiv \hat{\boldsymbol{\alpha}}^T P^T \boldsymbol{r}_0 - \frac{1}{2}\hat{\boldsymbol{\alpha}}^T P^T H P \hat{\boldsymbol{\alpha}} \tag{4.1.1}$$

subject to

$$y^T P\hat{\boldsymbol{\alpha}} = 0, \tag{4.1.2}$$

$$\mathbf{0} \leq \boldsymbol{\alpha}_0 + P\hat{\boldsymbol{\alpha}} \leq \boldsymbol{c}, \tag{4.1.3}$$

where $\boldsymbol{r}_0$ is the gradient of $F(\boldsymbol{\alpha})$ at $\boldsymbol{\alpha}_0$. Note that $y^T\boldsymbol{\alpha}_0 = 0$ because $\boldsymbol{\alpha}_0$ is feasible. So far, P is still a general matrix. It will be specified later either directly or indirectly by its orthogonal complement Q, where $P^T Q = 0$.

Consider $\boldsymbol{\alpha}_0$ as an intermediate solution of the dual QP problem and $P\hat{\boldsymbol{\alpha}}$ as an update. To compute an update, we choose $P = E_p$, where the columns of E_p are all the $\boldsymbol{e}_i$ satisfying $0 < \boldsymbol{e}_i^T\boldsymbol{\alpha}_0 < c$ (non-active constraints). If $\boldsymbol{e}_i^T\boldsymbol{\alpha}_0 = 0$ or $\boldsymbol{e}_i^T\boldsymbol{\alpha}_0 = c$ (active constraints), then respectively $\boldsymbol{e}_i$ or $-\boldsymbol{e}_i$ becomes one column of Q. Note that P and Q are simply the matrix representations of the indices of the points in G and $\overline{G}$. We extend the definition of $\overline{G}$ to incorporate the points $\boldsymbol{x}_i$, where $\boldsymbol{e}_i^T\boldsymbol{\alpha}_0 = c$. That is, if $\boldsymbol{x}_i \in \overline{G}$ then $\boldsymbol{\alpha}_0(i)$ can be either 0 or c. Define $\hat{H} = P^THP$, $\hat{\boldsymbol{r}}_0 = P^T\boldsymbol{r}_0$, $\hat{\boldsymbol{\alpha}}_0 = P^T\boldsymbol{\alpha}_0$, and so on so forth. With the above choice of P, $\hat{\boldsymbol{\alpha}}$ is determined by Problem (4.1.1), which is the projected dual QP problem onto the affine space determined by $\boldsymbol{\alpha}_0$ and E_p:

$$\underset{\hat{\boldsymbol{\alpha}}}{\text{maximize}} \quad \hat{F}(\hat{\boldsymbol{\alpha}}) \equiv \hat{\boldsymbol{\alpha}}^T\hat{\boldsymbol{r}}_0 - \frac{1}{2}\hat{\boldsymbol{\alpha}}^T\hat{H}\hat{\boldsymbol{\alpha}} \tag{4.1.4}$$

subject to

$$\hat{y}^T\hat{\boldsymbol{\alpha}} = 0, \tag{4.1.5}$$

$$\mathbf{0} \le \hat{\boldsymbol{\alpha}}_0 + \hat{\boldsymbol{\alpha}} \le \boldsymbol{c}. \tag{4.1.6}$$

Note that $\mathbf{0}$ and $\boldsymbol{c}$ are of size p. Since $P\hat{\boldsymbol{\alpha}}$ is orthogonal to the column space of Q, it does not affect any active constraints at $\boldsymbol{\alpha}_0$. In other words, the update determined by the above QP problem only improves the part of $\boldsymbol{\alpha}_0$ whose values are strictly between 0 and c. When p is small, it is cheap to solve the above problem.

To deal with the equality constraint (4.1.5), Problem (4.1.4) is projected onto the hyperplane determined by this constraint. We choose $Q = \hat{y}$ and $P = I - \frac{\hat{y}\hat{y}^T}{p}$ for this projection. It is easy to see that the column spaces of P and Q are orthogonal complements to each other, and $I - P$ is the orthogonal projector onto the column space of Q. The resulting subproblem has a simpler form:

$$\underset{\hat{\boldsymbol{\alpha}}}{\text{maximize}} \quad \hat{F}(\hat{\boldsymbol{\alpha}}) \equiv \hat{\boldsymbol{\alpha}}^TP^T\hat{\boldsymbol{r}}_0 - \frac{1}{2}\hat{\boldsymbol{\alpha}}^TP^T\hat{H}P\hat{\boldsymbol{\alpha}} \tag{4.1.7}$$

subject to

$$\mathbf{0} \le \hat{\boldsymbol{\alpha}}_0 + P\hat{\boldsymbol{\alpha}} \le \boldsymbol{c}, \tag{4.1.8}$$

where $P = I - \frac{\hat{y}\hat{y}^T}{p}$. Remember that in Problem (4.1.4) $P = E_p$ and $E_p\hat{\boldsymbol{\alpha}}$ gives the update.

4.1.2 Solving the Subproblem by the Conjugate Gradient Method

If the box constraint (4.1.8) turns out to be loose, i.e., not active, then the above problem is equivalent to a linear system

$$P^T\hat{H}P\hat{\boldsymbol{\alpha}} = P^T\hat{\boldsymbol{r}}_0,$$

or equivalently

$$P\hat{H}P\hat{\boldsymbol{\alpha}} = P\hat{\boldsymbol{r}}_0 \text{ (P is symmetric).} \tag{4.1.9}$$

Since $P\hat{H}P$ must be symmetric positive definite under the above assumption, this linear system could be solved numerically by the Conjugate Gradient (CG) method [30] [91]:

$$\begin{aligned}
&\hat{\boldsymbol{\alpha}}^{(0)} = 0, \quad \boldsymbol{\gamma}^{(0)} = P\hat{\boldsymbol{r}}_0, \quad \boldsymbol{\rho}^{(0)} = \boldsymbol{\gamma}^{(0)} \\
&\lambda = \frac{\boldsymbol{\gamma}^{(l-1)^T}\boldsymbol{\gamma}^{(l-1)}}{\boldsymbol{\rho}^{(l-1)^T}P\hat{H}P\boldsymbol{\rho}^{(l-1)}} \\
&\hat{\boldsymbol{\alpha}}^{(l)} = \hat{\boldsymbol{\alpha}}^{(l-1)} + \lambda\boldsymbol{\rho}^{(l-1)} \\
&\boldsymbol{\gamma}^{(l)} = \boldsymbol{\gamma}^{(l-1)} - \lambda P\hat{H}P\boldsymbol{\rho}^{(l-1)} \\
&\mu = \frac{\boldsymbol{\gamma}^{(l)^T}\boldsymbol{\gamma}^{(l)}}{\boldsymbol{\gamma}^{(l-1)^T}\boldsymbol{\gamma}^{(l-1)}} \\
&\boldsymbol{\rho}^{(l)} = \boldsymbol{\gamma}^{(l)} + \mu\boldsymbol{\rho}^{(l-1)}.
\end{aligned}$$

From the fact $P^2 = P$, it follows that the above CG iterations can be simplified as

$$\begin{aligned}
&\hat{\boldsymbol{\alpha}}^{(0)} = 0, \quad \boldsymbol{\gamma}^{(0)} = P\hat{\boldsymbol{r}}_0, \quad \boldsymbol{\rho}^{(0)} = \boldsymbol{\gamma}^{(0)} \\
&\lambda = \frac{\boldsymbol{\gamma}^{(l-1)^T}\boldsymbol{\gamma}^{(l-1)}}{\boldsymbol{\rho}^{(l-1)^T}\hat{H}\boldsymbol{\rho}^{(l-1)}} \\
&\hat{\boldsymbol{\alpha}}^{(l)} = \hat{\boldsymbol{\alpha}}^{(l-1)} + \lambda\boldsymbol{\rho}^{(l-1)} \\
&\boldsymbol{\gamma}^{(l)} = \boldsymbol{\gamma}^{(l-1)} - \lambda P(\hat{H}\boldsymbol{\rho}^{(l-1)}) \\
&\mu = \frac{\boldsymbol{\gamma}^{(l)^T}\boldsymbol{\gamma}^{(l)}}{\boldsymbol{\gamma}^{(l-1)^T}\boldsymbol{\gamma}^{(l-1)}} \\
&\boldsymbol{\rho}^{(l)} = \boldsymbol{\gamma}^{(l)} + \mu\boldsymbol{\rho}^{(l-1)}.
\end{aligned}$$

Note that $\hat{H}$ is a $p \times p$ matrix. Since $P = I - \frac{\hat{y}\hat{y}^T}{p}$, the multiplication of P with a vector is cheap. It only involves two Level-1 (vector-vector) operations. The most expensive operation in each iteration is the Level-2 (matrix-vector) operation $\hat{H}\boldsymbol{\rho}^{(l-1)}$. When p is small, it is cheap to implement the above operations and the storage requirement (one matrix and four vectors) is also small.

However, to solve Problem (4.1.7) with the above algorithm, the box constraint (4.1.8) has to be considered. If $0 \leq (\hat{\boldsymbol{\alpha}_0})_i + (P\hat{\boldsymbol{\alpha}})_i \leq c$ becomes active during the Conjugate Gradient iterations, then its global index representation $\boldsymbol{e}_{l_i}$ is added to Q, and at the same time, some other active constraints can be relaxed by moving their index representations from Q to P, if doing so improves the objective function. After P and Q is updated, the subproblem (4.1.7) is reformulated correspondingly and solved again. In the following

section, the stopping criteria and the criteria for which active constraints should be relaxed are given. An important fact worth to point out is that $\hat{H}$ is not guaranteed to be in full rank. Therefore, we need to control the steps of the CG iterations before the solution blows up.

4.1.3 The Optimality Conditions

Note that Problem (4.1.7) is the compact version of Problem (4.1.1) with

$$\begin{aligned} Q &= [\pm \boldsymbol{e}_{i_1}, \pm \boldsymbol{e}_{i_2}, \ldots, \pm \boldsymbol{e}_{i_q}, \boldsymbol{y}] \\ &= [E_q, \boldsymbol{y}] \end{aligned}$$

and

$$P = I - Q(Q^T Q)^{-1} Q^T,$$

where E_q is the orthogonal complement of E_p, and the column space of P is the orthogonal complement of the column space of Q. In other words, the previous two projections are incorporated in the current definition of P. Let us define

$$\boldsymbol{d} = P\boldsymbol{r}$$

and

$$\boldsymbol{\beta} = -[I_{q\times q}, \mathbf{0}](Q^T Q)^{-1} Q^T \boldsymbol{r},$$

where $\boldsymbol{r} = \mathbf{1} - H\boldsymbol{\alpha}$ and P is a $n \times n$ matrix. The optimality conditions [1] [8] tell that $\boldsymbol{\alpha}$ is optimal iff

$$\boldsymbol{d} = 0 \text{ and } \boldsymbol{\beta} \geq 0. \tag{4.1.10}$$

Note that $\boldsymbol{d}$ is the projected gradient onto the search space spanned by P. If $\boldsymbol{d} = \mathbf{0}$, then we have reached a stationary point. The condition $\boldsymbol{\beta} \geq 0$ implies that relaxing the active constraints represented by Q does not improve the objective function nearby. It follows that if (4.1.10) is true then $\boldsymbol{\alpha}$ is a local optimum, thus a global optimum due to the convex property of a linearly constrained QP problem.

Since $(Q^T Q)^{-1}$ has a simple closed form:

$$\begin{bmatrix} I + \frac{E_q^T \boldsymbol{y}\boldsymbol{y}^T E_q}{p} & \frac{-E_q^T \boldsymbol{y}}{p} \\ \frac{-\boldsymbol{y}^T E_q}{p} & \frac{1}{p} \end{bmatrix},$$

it is easy to compute d and $\boldsymbol{\beta}$. For instance,

$$\boldsymbol{\beta} = E_q^T \boldsymbol{r} + \frac{E_q^T \boldsymbol{y}((E_q^T \boldsymbol{y}) E_q^T \boldsymbol{r} - \boldsymbol{y}^T \boldsymbol{r})}{p}, \tag{4.1.11}$$

where E_q is an indexing operator. Each time when active constraints are to be relaxed, we start with the one that has the most negative β_i.

4.1.4 The Algorithm

As a summary of the previous discussion, the algorithm is outlined briefly as the following:

$\boldsymbol{\alpha} = 0$;
initialize E_p;
while $\boldsymbol{d} \neq 0$ or $\boldsymbol{\beta} < 0$
  at most k steps of the CG iterations for Problem (4.1.7);
  update $\boldsymbol{\alpha}$ and $\boldsymbol{r}$:
  $\boldsymbol{\alpha} = \boldsymbol{\alpha} + E_p\hat{\boldsymbol{\alpha}}$;
  $\boldsymbol{r} = \boldsymbol{r} - HE_p\hat{\boldsymbol{\alpha}}$;
  compute $\boldsymbol{\beta}$;
  relax at most l active constraints with the most negative β_i;
  update E_p;
  compute the columns of H corresponding to the relaxed constraints;
  update $\hat{H}$;
end

During each iteration of this algorithm, only p columns of H (HE_p) are used. Thus, there is no need to precompute H so that a lot of flops (floating point operations) and memory can be saved. In our implementation of this algorithm, two matrices $X = [\boldsymbol{x}_1, \ldots, \boldsymbol{x}_m]$ and HE_p are stored in memory. Since n and p are generally much smaller than m, most modern computers can meet the memory requirement for problems of size $m = O(10,000)$. Later in this chapter, we will give an alternative algorithm which consumes less memory by using part of HE_p, but taking more steps to converge.

Note that the ith column of H is computed only when $\alpha_i = 0$ or $\alpha_i = c$ is relaxed. It turns out that computing the columns of H is the most expensive operation in this algorithm. When l is large, it can be considered as a Level-3 (matrix-matrix multiplication) operation. Although Level-1 and Level-2 operations are cheaper in terms of flops, Level-3 operations can be implemented more efficiently by exploiting the computer memory hierarchy. More discussions will be given on the implementation of these operations in the next section.

The setting of parameters k and l affects the performance. We will discuss more on how to choose k and l in the next section. In our implementation, k and l are determined adaptively based on each training set. To initialize G, we can randomly pick a small number of points from the training set. Alternatively we can construct a pair of hyperplanes as the

initial solution using the ML separating hyperplane introduced in Chapter 1. Points in or near the two initial hyperplanes are chosen for G. The ML separating hyperplane is easy to compute. Let m_+ and m_- be the means of the positive and negative points. The ML hyperplane is the one perpendicular to the line $\overline{m_- m_+}$ and passing through its midpoint. In our implementation, the two hyperplanes are chosen to be the ones parallel to the ML separating hyperplane and passing through the points $\frac{m_- + 3m_+}{4}$ and $\frac{3m_- + m_+}{4}$ respectively. Our observation is that when the training set is easy to separate, the second method is better; otherwise, we should use the first method to initialize G.

Note that the residual or the gradient $\boldsymbol{e}_i^T \boldsymbol{r} = 1 - y_i \boldsymbol{w}^T \boldsymbol{x}_i$ tells how well the current solution separates $\boldsymbol{x}_i$. When (4.1.10) is true, the optimality is achieved. In summary, this algorithm successfully exploits the sparsity property of SVMs. When the final solution is sparse, it solves the dual QP problem with much less flops and memory consumption. For details of our MATLAB implementation, please refer to the downloadable codes [106].

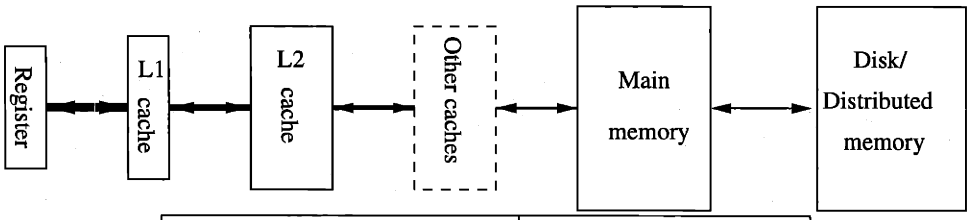
4.2 Speed Considerations

We have seen that the sparsity property of SVMs enables us to avoid unnecessary memory consumption and flops. To achieve better performance, in this section, we discuss how to make the implementation of this algorithm adaptive to both the computer memory hierarchy and the training set.

4.2.1 Being Adaptive to the Computer Memory Hierarchy

When large data sets are involved in computations, the number of flops is no longer an accurate indicator of the running time. The cost of accessing data must also be taken into account. In Figure 4.2, a model of a modern computer memory hierarchy is given. Loading data from main memory to cache and writing data from cache back to main memory are expensive operations. To reduce the cost of accessing data, computations should be arranged so that more data can be found in cache at the first time when needed. For instance, on computers with the memory hierarchy described in Figure 4.2, Level-3 operations can be implemented more efficiently than Level-2 and Level-1 operations. In general, fast codes are developed by explicitly using the information of each configuration of the memory hierarchy model.

To achieve a portable performance, we want our implementation to be adaptive to each machine's memory hierarchy. Since we have described our projected Conjugate Gradient algorithm in terms of basic linear algebra (BLA) operations, it is easy for us to leverage



level	*clock cycles*
register	1
L1 cache	$2-3$
L2 cache	$6-12$
near main memory	$6-100$
far main memory	$100-250$
distributed main memory	$O(100)$
message-passing	$O(1000)-O(10000)$

Figure 4.2: A model of the modern computer memory hierarchy. The statistics in the table above are cited from [25].

off previous work. To be adaptive, we use ATLAS BLAS [108] [109] [110] to compute BLA operations. ATLAS BLAS is a software package for computing BLA operations, which is automatically tuned for different machines. MATLAB uses ATLAS BLAS starting from its version 6. Thus, by using MATLAB, the implementation of this algorithm is free from the low-level issues such as how to implement matrix-matrix multiplications efficiently on a specific machine. Hence, its performance is portable.

4.2.2 Setting k and l Adaptively

As mentioned in the previous section, the running time of this algorithm depends on the setting of k and l. In this section, we discuss how to set these two parameters adaptively for each training problem.

The convergence rate of our algorithm is related to the location of the spectrum of H and the value of c. The good news about H is that its leading eigenvalues lie well separated. The number of these leading eigenvalues is a good indicator of the size of S_1. While the value of c affects the size of S_2. The smaller c is, the larger S_2. For the extreme case where the training set is separable and c is set very large (such that the constraints $\alpha_i \leq c$ never become active), the spectrum of H is an important factor determining the running time. Based on this observation, we set k and l adaptively by using the information of the spectrum of H. From the convergence property of the CG method, we know that the residual of a

symmetric positive definite linear system with j well-separated leading eigenvalues tends to be reduced quickly for the first j iterations by the CG method. According to this property, we estimate the number of the leading eigenvalues by $\hat{k}$, the number of iterations for the CG method to reduce the residual norm to a scale of 0.01.

k is used to control the steps of the CG iterations, because there is no need to compute Problem (4.1.7) exactly when it is not the final subproblem. The other reason to not solve Problem (4.1.7) exactly is due to the rank deficiency of H. The CG method converges slowly or may not even converge for ill-conditioned matrices. For example, if $\boldsymbol{x}_i \in \mathbb{R}^2$ then $\hat{H}$ has two nonzero eigenvalues in general. It makes sense to solve the subproblems with only two steps of the CG iterations before the solution blows up. To deal with the rank deficiency, people usually add a small positive value to the diagonals of $\hat{H}$. In our algorithm, k is set to be $\hat{k}$ obtained from the initial subproblem.

When l active constraints are relaxed, $[\boldsymbol{x}_{i_1}, \ldots, \boldsymbol{x}_{i_l}]^T X$ is computed to generate the corresponding columns of H (H is not precomputed). Considering that $[\boldsymbol{x}_{i_1}, \ldots, \boldsymbol{x}_{i_l}]^T X$ is a Level-3 operation when l is large, we tend to relax all the constraints with negative β_i (because Level-3 operations can be implement efficiently). However, with this strategy we increase the probability that some relaxed constraints become active again, which slows down the convergence. Our observation is that when the number of the SVs is small, it is better off to set l small. Since the number of the leading eigenvalues of H provides information about the number of the SVs, in our algorithm l is also set to be $\hat{k}$.

4.3 Numerical Experiments

Our goal is to minimize overhead so that the high performance and the nice features provided by a high-level language such as MATLAB can be achieved simultaneously. For example, to further improve the performance of our MATLAB implementation, we replace a frequently used slow MATLAB function with a much faster C function. The MATLAB implementation of this algorithm is named "FMSvm", where "FM" stands for "Fast MATLAB". In this section, FMSvm [106] is compared with two bench mark C and C++ codes, SVM^{light} [35] and SvmFu [66] respectively.

The training functions of FMSvm, SVM^{light} and SvmFu are compared based on two training sets, one is sparse and the other one is dense. The sparse training set Digit17 contains 13007 samples of hand-written digits 1 and 7, while the dense training set Face is a face detection training set containing 31022 examples. Digit17 is easy to separate and it is balanced (6742 positive examples vs. 6265 negative examples). On the opposite, Face is

Training sets: Digit17-6000 and Digit17							
l(1)	l(0.1)	p2(0.1)	p2(0.001)	r10(10)	r10(1)	r3(1)	r3(0.1)
Training sets: Face-6000, Face-13901 and Face							
l(4)	l(1)	l(0.1)	p2(1)	p2(0.001)	r10(10)	r3(10)	r3(1)

Table 4.1: Choices of kernels and c. Here, "l" stands for the linear kernel, "p2" stands for the polynomial kernel with degree 2 and "rx" stands for the radial basis function kernel with $\sigma = x$. The number in the parenthesis is the value of c. The definitions of these kernel functions are listed in the appendix.

not balanced (2901 positive examples vs. 28121 negative examples) and harder to separate. The training points in Digit17 and Face are of dimensions 784 and 361 respectively. The two training set are downloadable at [106].

Both the SVM^{light} and SvmFu training functions have parameters whose setting affects their running time[1]. Bad choices can make the convergence very slow. Unfortunately the optimal setting of these parameters are not known a priori. If they are not set, the default setting is taken. The FMSvm training function also has two parameters k and l, but users are not required to set them. Instead it tries to estimate the optimal setting by itself based on each particular training problem. In this section we compare FMSvm with SVM^{light} and SvmFu using both default and estimated optimal settings.

From Digit17 and Face, we create five training sets: Digit17-6000, Digit17, Face-6000, Face-13902 and Face, with sizes 6000, 13007, 6000, 13902 and 31022 respectively. The two smallest training sets are used to obtain an estimate of the optimal settings for SVM^{light} and SvmFu. The estimated optimal setting for each training function is the setting among ten trials which gives the best performance. Then FMSvm is compared against SVM^{light} and SvmFu with the estimated optimal settings on larger training sets Digit17, Face-13902 and Face. Table 4.1 lists the kernels and the values of c used for each training set.

In Figure 4.3, we can see that for the easy-to-separate training set Digit17-6000, there is no big difference in performance between the FMSvm training function and the other two with default settings. But for Face-6000 's case l(4), FMSvm does much better than both SVM^{light} and SvmFu. The timing results of FMSvm against SVM^{light} and SvmFu plotted

[1]The convergence time is measured by cpu seconds. For SVM^{light} and SvmFu, the time spent in loading the input data file is excluded. For instance, SvmFu needs around 24 cpu seconds to load the training set Face, while SVM^{light} needs around 115 cpu seconds. All the timing results in this paper are obtained on the sever `newton.mit.edu`, which is a linux machine with 2 1.2 *GHZ* Athlon MP processors and 2 *GB* memory.

in the right column of Figure 4.3 are for the case where the estimated optimal settings are used for all of them. We can see that in average, FMSvm might even converge faster than its two counterparts. The estimated optimal settings obtained here for SVM^{light} and SvmFu are used for later comparisons in Figure 4.4. These estimated optimal settings are listed in the appendix.

Since the optimal parameter settings for all the three training functions are not known a priori, bad settings can cause very slow convergence. FMSvm tries to avoid this situation by being adaptive to each training set. Shown in Figure 4.4 are the timing results of FMSvm against SVM^{light} and SvmFu, where the estimated optimal settings are used for SVM^{light} and SvmFu. We can see that FMSvm matches the estimated best performance of SVM^{light} and SvmFu on the two training sets.

If we measure the difference between a setting A and the estimated optimal setting O by the ratio of relative slowdown:

$$s = \frac{\text{time}(A) - \text{time}(O)}{\text{time}(O)},$$

then in Figure 4.3, for each training function we get 16 such ratios that measure the difference between the setting used in the left column and the estimated optimal setting used in the right column. The mean and standard deviation of these ratios are respectively $(0.16, 0.14)$, $(0.24, 0.43)$ and $(0.76, 0.60)$ for FMSvm, SVM^{light} and SvmFu. We can see that the setting used by FMSvm is closer to its estimated optimal setting.

In summary, FMSvm's idea of being adaptive to each training set works, which helps preventing the slow convergence caused by bad parameter settings. Based on training sets Digit17 and Face, we can see that FMSvm matches the performance of SVM^{light} and SvmFu.

4.4 Distributing Large Training Problems

The basic memory requirement for our algorithm is to store the training example matrix X and the computed part of the Hessian matrix HE_p. Instead of $E_p^T HE_p$, HE_p is stored because we need to update the residual $r = r_0 - HE_p\hat{\alpha}$. To control memory consumption, we can update part of the residual so that only the corresponding rows of HE_p are needed. From the primal problem point of view, this is equivalent to testing part of the training points in $\overline{G}$ using the current solution. This approach can be easily extended to distribute large training problems. The idea is to use the master to compute all the subproblems, where only $E_p^T HE_p$ is needed, and let slaves to update the residual in parallel. Updating the residual is expensive because the corresponding part of H has to be computed.

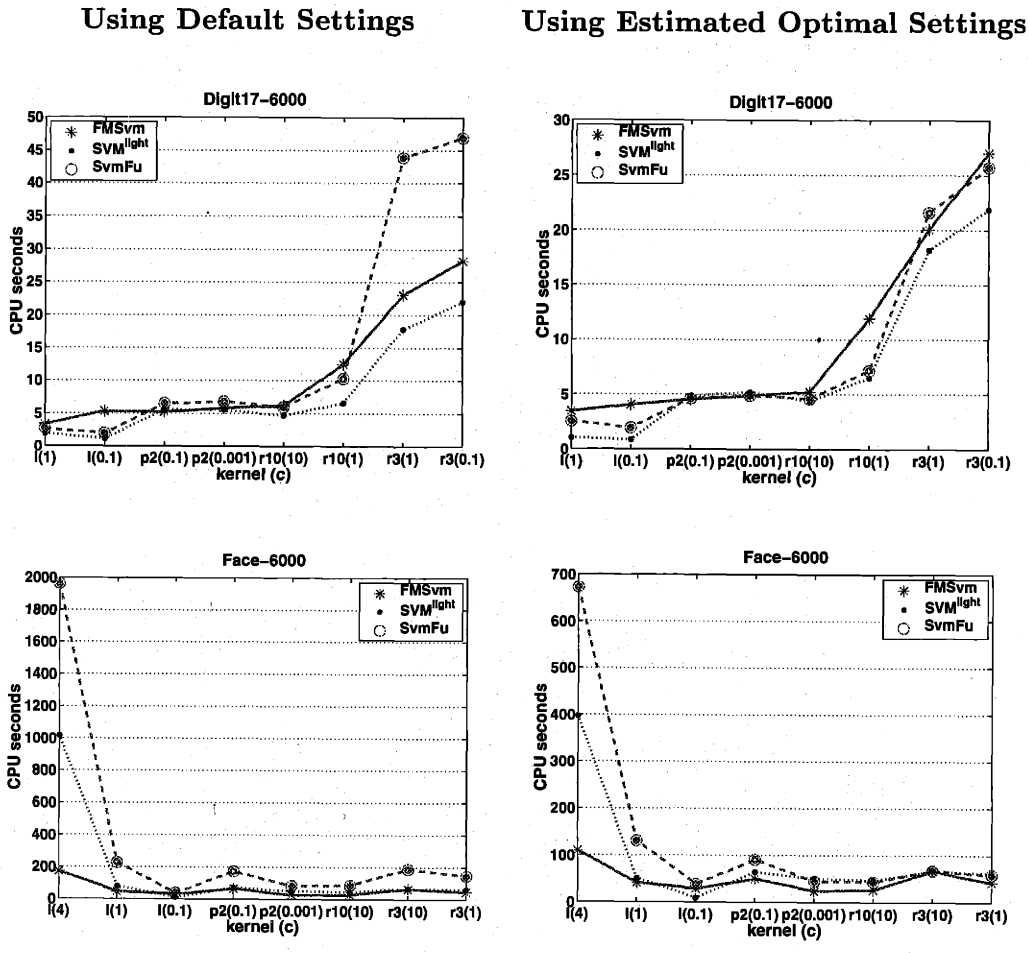


Figure 4.3: Timing results on the two training subsets of size 6000. Plots in the left column are the timing results of FMSvm against SVM^{light} and SvmFu with default parameter settings. For problems of size larger than 2000, the default setting for SvmFu is "(h, c)" = $(2000, 0)$ and the default setting for SVM^{light} is "(q, n, m)" = $(10, 10, 40)$. Note that in SVM^{light} and SvmFu, h, c and q, n, m have different meanings than ours. However, for training set Face-6000 we use $(20, 10, 40)$ as SVM^{light}'s default setting, following the hint to choose $n < q$ to prevent zig-zagging, where $(20, 10, 40)$ is the estimated optimal setting for Digit17-6000. Plots in the right column are the timing results of FMSvm against SVM^{light} and SvmFu, where the estimated optimal settings are used for all of them. With default settings, for the easy-to-separate training set Digit17-6000, there is no big difference in performance between the FMSvm training function and the other two; but for Face-6000 's case l(4), FMSvm does much better than both SVM^{light} and SvmFu.

Comparison of FMSvm with SVM^{light} and SvmFu with the estimated optimal parameter settings

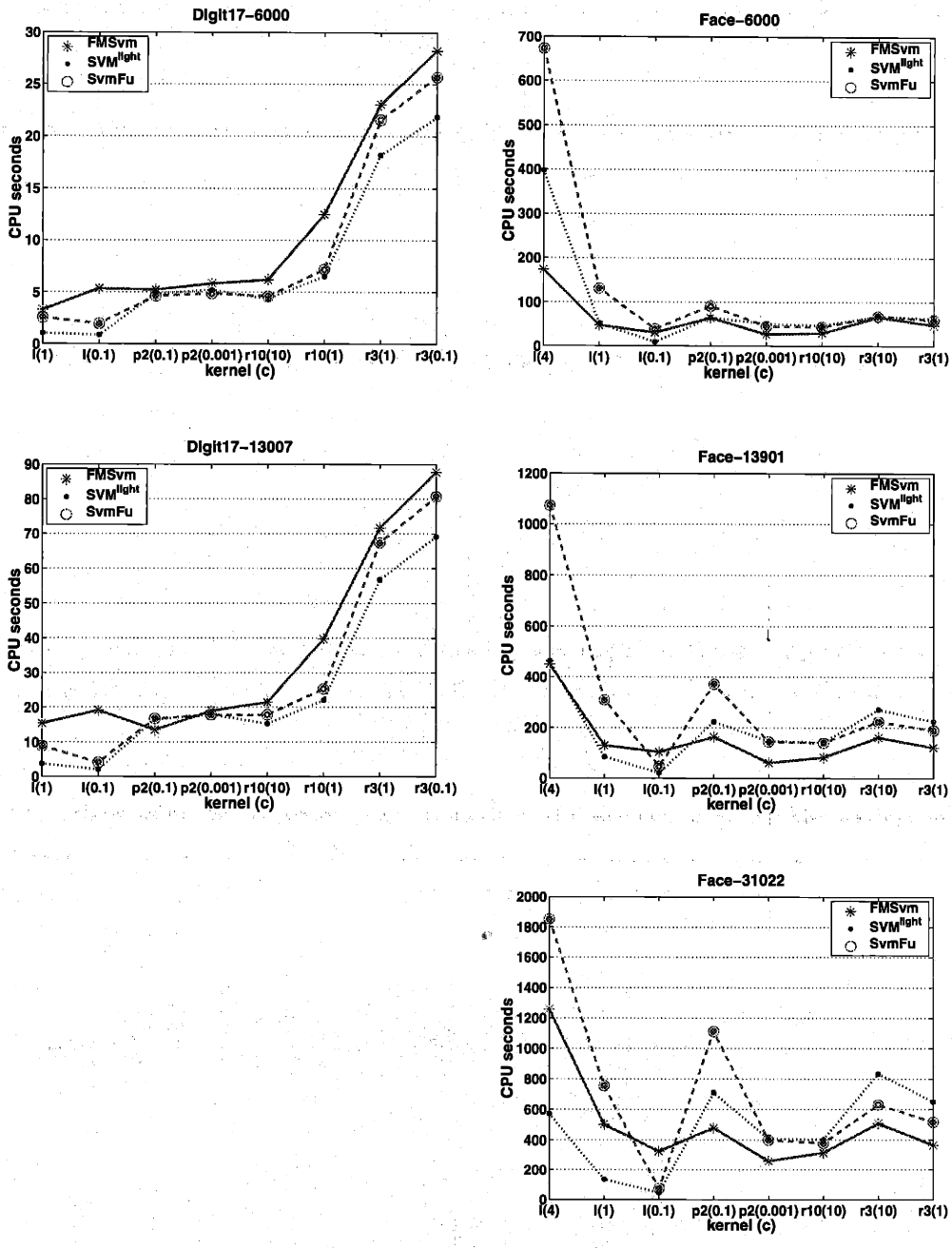


Figure 4.4: FMSvm VS. SVM^{light} and SvmFu. Plots in the left column are the timing results of FMSvm against SVM^{light} and SvmFu on training sets Digit17-6000 and Digit17, while those in the right column are the timing results of FMSvm against SVM^{light} and SvmFu on training sets Face-6000, Face-13901 and Face. The estimated optimal parameter settings are used for the training functions of SVM^{light} and SvmFu. For those parameters that seem to depend on the size of the training set, their values are adjusted proportionally as the size increases. All the settings are listed in the appendix. We can see that FMSvm (without using the estimated optimal parameter settings) matches the estimated best performance of SVM^{light} and SvmFu.

To make our MATLAB code support distributed computations, we use an extended MATLAB environment called MATLAB*P [39], where an almost transparent interface to distributed matrices and the operations on them is provided. MATLAB*P consists of two parts, a MATLAB front end and a server living in a supercomputer. On the server, distributed matrices and the operations on them are stored and executed. Unlike MATLAB's transparent support to sparse matrices, small modification is required to make ordinary MATLAB codes to support MATLAB*P's distributed matrices. But once MATLAB*P knows that X is a distributed matrix, all the operations involving X are automatically executed by the server. Since our algorithm is described in terms of basic linear algebra operations, we expect that it should be easy to extend FMSvm to support MATLAB*P's distributed matrices in addition to MATLAB's dense and sparse data formats.

4.5 Conclusion and Future work

In summary, our MATLAB training function has the following features:

- easy to use;
- efficient;
- highly portable;
- can be easily extended to support distributed matrices with MATLAB*P, in addition to dense and sparse matrices.

We still have potential to improve the performance of our implementation by making the spectrum of the submatrices of H well-behaved. Preconditioning is the method. We will add this feature in the next version of FMSvm.

The FMSvm training function was originally written with the assumption that the training example matrix X was dense. It works fine with MATLAB sparse matrices, but we want to make sure that it is also optimized for the sparse input. Currently, we are extending FMSvm to support MATLAB*P distributed matrices in order to solve larger problems. We plan to release the next version of FMSvm in the summer of 2002. In our future work, we want to make our algorithm more adaptive and we want to better understand its convergence behavior.

Chapter 5

Using SVMs to Predict the Stock Market

The two examples in Chapter 4 involve classifying images. In this chapter, we apply SVMs to time series data. Here, SVMs are used to predict the movement of the Dow Jones Index. The solution using SVMs is compared against the one based on the geometric Brownian motion model of stock price behavior, which is the foundation of the Nobel Prize winning Black-Sholes formula for pricing stock options [6] [51]. Our results show that using SVMs has the potential to predict the movement better than using this model.

5.1 Setting the Stage

In the geometric Brownian motion model, it is assumed that stock prices follow a stochastic process known as geometric Brownian motion. It follows that in this model stock prices have the Markov property, that is, only the current price of a stock is relevant for predicting its future price. This property is consistent with the weak form of the market-efficiency hypothesis [69] [70], which states that the present price of a stock reflects all the information contained in its past prices. Therefore, there is no need to interpret the history of stock prices in order to achieve more predicting power. Since there is very little evidence against this hypothesis, we can believe it is true. However, it does not imply that other publicly available information such as trading volumes and interest rates will not help either. By contrast, SVMs predict the future by learning from experience, where the known information (experience) is incorporated in the training set. The theme of this chapter is to investigate how well SVMs can learn from time series data such as stock prices. Our results indicate that using SVMs has the potential to outperform the solution based on the geometric Brownian motion model. This investigation is also interesting for people who want to test the semi-strong form of market efficiency, which states that the present price

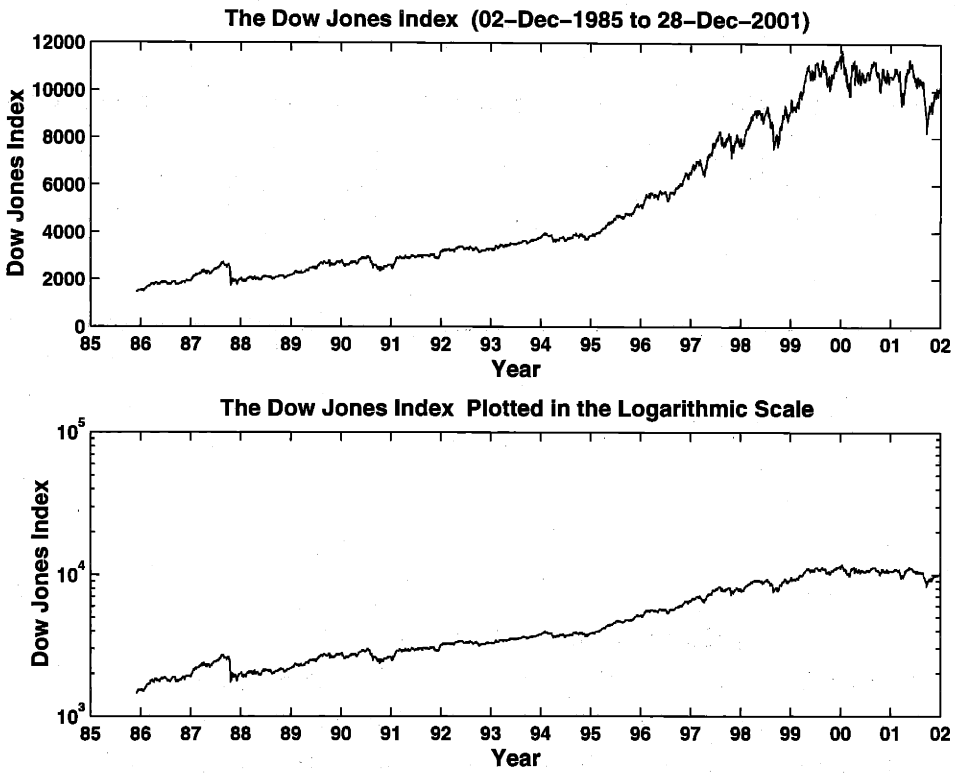


Figure 5.1: The Dow Jones Index from December 2, 1985 to December 28, 2001. Note that the top plot is in regular scale, while the bottom plot is in the logarithmic scale.

of a stock reflects all the publicly available information. If this hypothesis is true, then we should not be able to make above-average returns using SVMs.

In the following discussion, the Dow is used as an abbreviation for the Dow Jones Index. We want to predict on Thursday, after the market is closed, whether the closing index of the Dow on Friday will be higher than its opening index on the previous Monday. We choose this model problem because it is easy to solve and to compare the two solutions, one based on the geometric Brownian motion model and the other one using SVMs. However, the simplicity of this model problem does not affect the generality of the points we try to make. In Figure 5.1, the closing indexes of the Dow from December 2, 1985 to December 28, 2001 are plotted. To reveal the volatility better, these indexes are also plotted in the logarithmic scale. The record of the Dow can be downloaded from the Yahoo Finance website.

5.2 The Geometric Brownian Motion Solution

Let S be the stock price at time t. The geometric Brownian motion model consists of the following equation

$$dS = \mu S dt + \sigma S dz, \tag{5.2.1}$$

which can also be written as

$$\frac{dS}{S} = \mu dt + \sigma dz. \tag{5.2.2}$$

Here, μ is the expected continuously compounded return per unit of time and σ is the volatility of the stock price per unit of time; the Wiener process dz [38, page 220] [113] has a drift rate of zero and a variance rate of 1. By Itô's lemma [38, page 229], it can be shown that

$$d \ln S = (\mu - \frac{\sigma^2}{2})dt + \sigma dz. \tag{5.2.3}$$

It follows that if S_T is the stock price at a future time T and S_0 is the stock price at time 0, then $\ln \frac{S_T}{S_0}$ has the normal distribution with mean $(\mu - \frac{\sigma^2}{2})T$ and standard deviation $\sigma\sqrt{T}$:

$$\ln \frac{S_T}{S_0} \sim N((\mu - \frac{\sigma^2}{2})T, \sigma^2 T).$$

The discrete-time version of (5.2.2) is

$$\frac{\Delta S}{S} = \mu \Delta t + \sigma \epsilon \sqrt{\Delta t}. \tag{5.2.4}$$

The variable ΔS is the change of S in a small interval of time Δt and ϵ is a standard normal random variable. Note that $\frac{\Delta S}{S}$ is the return over Δt. Here, $\mu \Delta t$ is the expected value of this return and $\sigma \epsilon \sqrt{\Delta t}$ is the noise part. Equation (5.2.4) indicates that $\frac{\Delta S}{S}$ is a normal random variable with mean $\mu \Delta t$ and standard deviation $\sigma\sqrt{\Delta t}$:

$$\frac{\Delta S}{S} \sim N(\mu \Delta t, \sigma^2 \Delta t).$$

For our problem, we need to know the daily return of the Dow on Friday. Since Δt is small (one day), we use the discrete-time model (5.2.4) to determine this return. Let S_{Th} and S_{Fr} be the closing prices on Thursday and Friday. The daily return on Friday is

$$R = \frac{S_{\text{Fr}} - S_{\text{Th}}}{S_{\text{Th}}}.$$

From (5.2.4), we know that

$$R = \mu + \sigma \epsilon, \tag{5.2.5}$$

where μ is the expected daily return and σ is the daily volatility. It follows that

$$S_{\text{Fr}} = (1 + R) S_{\text{Th}} = (1 + \mu + \sigma \epsilon) S_{\text{Th}}.$$

Finally, we want to know whether or not S_{Fr} is higher than S_{Mo}, where S_{Mo} is the opening price on Monday, that is,

$$S_{\text{Fr}} \underset{<}{\overset{>}{}} S_{\text{Mo}}.$$

This can be expressed equivalently as

$$\epsilon \underset{<}{\overset{>}{}} \frac{S_{\text{Mo}}/S_{\text{Th}} - 1 - \mu}{\sigma} = \xi. \tag{5.2.6}$$

The decision rule $h_{\text{GBM}}(\cdot)$ is defined as follows: the output is 1 if it predicts a higher price on Friday; the output is -1 otherwise. Since ϵ has a symmetric density function, i.e., the bell-shaped normal density function,

$$\begin{cases} \Pr(\epsilon > \xi) > \Pr(\epsilon < \xi) \text{ if } \xi < 0 \\ \Pr(\epsilon > \xi) < \Pr(\epsilon < \xi) \text{ if } \xi > 0. \end{cases}$$

Therefore, the decision rule is

$$h_{\text{GBM}}(\xi) = \text{sgn}(-\xi). \tag{5.2.7}$$

This is the same decision rule for predicting the outcome of tossing a biased coin. Knowing which way the coin is biased is enough for us to make the right decision.

Although the bias term $-\xi$ depends on two parameters μ and σ, its sign only depends on μ. It follows that the decision rule $h_{\text{GBM}}(\xi)$ is independent of σ. This is good news because estimating σ is not easy, due to the fact that σ changes over time. Usually, the implied volatility model, the stochastic volatility model [37] or the GARCH model [7] is used to estimate σ. For details, we refer readers to [38]. Our situation is exactly the opposite of the case when people use the Black-Sholes formula to price stock options, where σ is critically important and μ turns out to be irrelevant in general.

We use historical daily returns to estimate μ. Since we also assume that μ is time dependent, we do not average all the historical data. Instead, we only use the past daily returns back to a certain date from the latest Thursday. Let the window length be l weeks. In Figure 5.2, the estimated daily returns with different l for each trading Friday during the period from December 7, 1990 to December 28, 2001 are plotted (Trading weeks in which Monday is a holiday are not considered). We apply the decision rule (5.2.7) to Fridays during the above period. At $l = 300$, we get the best performance: 96 errors out of 511 trials. In other words, the accuracy of h_{GBM} on this test set is 81.21%. From Table 5.1, we can see that when we further increase l, the performance stays the same. The reason is that compared with the term $S_{\text{Mo}}/S_{\text{Th}} - 1$ (the return for the first four days) in (5.2.6), μ (the return on Friday) is insignificant. When l is increased, the estimation of μ becomes smoother and converging to a constant as indicated in Figure 5.2. Therefore, after l is larger than a certain value, it no longer affects the decision rule.

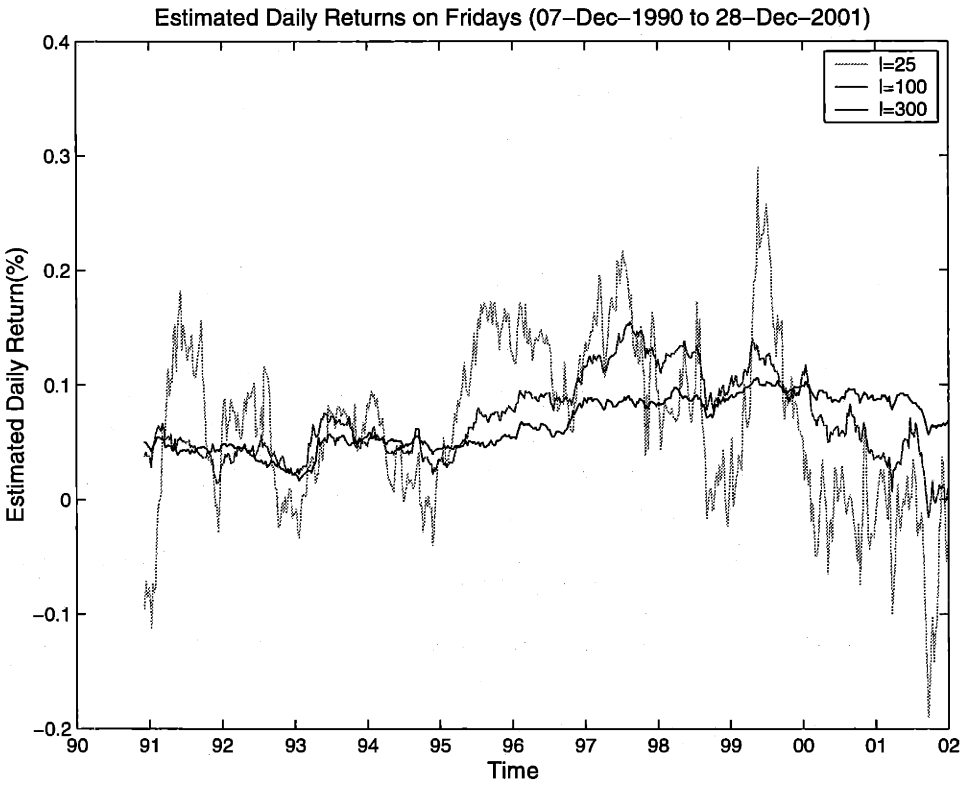


Figure 5.2: Estimated daily returns on Fridays between December 7, 1990 to December 28, 2001. The estimation becomes smoother as l increases, where l is the window length.

The window length l (weeks)	25	50	100	200	300	400	500
Errors (out of 511 trials)	101	100	97	98	96	96	96

Table 5.1: The number of Fridays on which the decision rule $h_{\mathrm{GBM}}(\cdot)$ gives wrong prediction. Different settings of l are tried. It turns out that the best performance we can get is 96 errors out of 511 trials. The time period of the test data is from December 7, 1990 to December 28, 2001. The trading weeks where Monday is a holiday are not included. When $l > 500$, the performance will be the same.

5.3 The SVM Solution

Unlike $h_{\mathrm{GBM}}(\cdot)$, the SVM decision rule

$$h_{\mathrm{SVM}}(\boldsymbol{x}) = \mathrm{sgn}(\boldsymbol{w}^T\boldsymbol{x} + b) \tag{5.3.1}$$

depends on the known (past) information, which is incorporated in $\boldsymbol{w}$ and b. As a predicting function, $h_{\mathrm{SVM}}(\cdot)$ relates what has happened in the past to the future movement of the Dow.

To apply SVMs, the first step is to construct the training set. As our first try, we use the record of the Dow from Monday to Thursday to build the training vectors (points) for each week. If the closing index on Friday is higher than the opening index on Monday in that week, we set the corresponding y-label to be 1; otherwise, we set it to be -1. There are five entries for each trading day in the Dow record: the opening index, the daily high index, the daily low index, the closing index and the volume. Therefore, our training vectors are of dimension 20. In the following example of a training vector, the first five elements are from the Monday record, while the last five elements are from the Thursday record:

$$\boldsymbol{x} = [341.3\ 343.6\ 339.4\ 340.9\ 23300\ \ldots\ 340.4\ 343.0\ 339.2\ 341.1\ 30000]^T.$$

When entries from different categories are assembled in the training vectors, we need to make sure that they are of the same scale if they are to be weighted equally. Note that in the above example, for instance, the opening index on Monday 341.3 and the trading volume on Monday 23300 are not of the same scale. In our training vectors, the real indexes and volumes are replaced by their relative values with respect to the Monday opening index and the Monday volume respectively. As an example, the scaled version $\boldsymbol{x}_s$ of the above example is listed in the following:

$$\boldsymbol{x}_s = [1\ 1.0067\ 0.9944\ 0.9988\ 1\ \ldots\ 0.9974\ 1.0050\ 0.9938\ 0.9994\ 1.2876]^T.$$

We use the record from July 5, 1954 to November 30,1990 to construct the training set, and the record from December 7, 1990 to December 28, 2001 to construct the test set. There are totally 1704 training vectors, and the test set contains 511 vectors. In general, learning from more examples tends to give better predicting performance. We use the record of the Dow starting from July 5, 1954 simply because before that date the record of the federal funds rate, which is used to build our next training set, is not available.

Following the principle of structural risk minimization [18] [24] [93], different kernels and choices of c are used to train SVMs, and their performance on the test set are compared. The linear kernel, the polynomial (poly) kernel and the radial base function (rbf) kernel are the three kernels we have tried:

poly(4)							
c	0.2	0.5	0.8	1	1.2	1.4	1.6
Errors	106	99	96	96	96	98	97
rbf(1)							
c	20	40	60	80	100	120	140
Errors	109	101	101	98	96	96	99

Table 5.2: The performance of poly(4) and rbf(1) on the test set.

$$\begin{aligned} &\text{linear:} && k(\boldsymbol{s},\boldsymbol{t}) = \boldsymbol{s}^T\boldsymbol{t}; \\ &\text{poly}(d): && k(\boldsymbol{s},\boldsymbol{t}) = (\boldsymbol{s}^T\boldsymbol{t}+1)^d; \\ &\text{rbf}(\sigma): && k(\boldsymbol{s},\boldsymbol{t}) = e^{-\|\boldsymbol{s}-\boldsymbol{t}\|_2^2/(2\sigma^2)}, \end{aligned}$$

where $\boldsymbol{s}$ and $\boldsymbol{t}$ are vectors. The settings of the parameters of these kernels are listed in the following table:

linear	poly(3)	poly(4)	poly(5)	rbf(0.2)
rbf(0.4)	rbf(0.6)	rbf(0.8)	rbf(1)	

The best performance achieved is 96 errors out of 511 trials, which matches the record of $h_{\text{GBM}}(\cdot)$. Some of these results are listed in Table 5.2. We can see that at the beginning the predicting accuracy gets better as c increases; however, when the overfitting effect takes over, increasing c makes the performance even worse.

Although the above result is not better than the result of $h_{\text{GBM}}(\cdot)$, we can at least see that SVMs can learn well from time series data, which can come from different categories. Our intuition is that if there is other information related to the movement of the Dow, then adding it to the training set might improve the performance. Considering that interest rates are important economic indicators, we add the daily federal funds rates [1] to the training set. We choose this rate simply because it is a daily rate. Again, we use the relative values with respect to the Monday rate in the training vectors, instead of real rates. If r_1, r_2, r_3 and r_4 are the federal funds rates for Monday through Thursday, then the four relative values are $1, \frac{r_2}{r_1}$, $\frac{r_3}{r_1}$ and $\frac{r_4}{r_1}$ respectively. With these four values added, our new training vectors are of dimension 24. With this training set, the best performance we have achieved so far is 89 errors on the same test set using the poly(3) kernel. The result is listed in Table 5.3.

[1] The federal funds rate is the cost of borrowing immediately available funds, primarily for one day. The effective rate is a weighted average of the reported rates at which different amounts of the day's trading through New York brokers occurs. This rate is provided by the Federal Reserve website.

poly(2)								
c	1	10	20	30	40	100	200	500
Errors	111	105	106	103	107	105	104	102
poly(3)								
c	0.05	1	1.5	2.5	10	20	30	35
Errors	103	102	96	96	94	93	89	89

Table 5.3: The performance of poly(2) and poly(3) using the new training set.

It seems that training SVMs with the new training set is harder than using the old one. Our observation is that the training time increases dramatically when c becomes larger. When $c = 30$, it takes around 2.6 hours (on the same machine used in the last chapter, i.e., `newton.mit.edu`) to train the SVM with the poly(3) kernel, comparing with the scale of seconds when using the old training set. In Table 5.3, we stop at $c = 40$ for poly(3) because the training time is longer than 12 hours. A prediction this slow would be useless for us. This highlights the reason why a fast training code is important. The distribution of the errors of $h_{\text{GBM}}(\cdot)$ and $h_{\text{SVM}}(\cdot)$ (using poly(3) and $c = 30$) on the test set are plotted in Figure 5.3.

5.4 Conclusion and Future Work

The above results have shown that SVMs can learn well from time series data. The better performance achieved by SVMs indicates that using SVMs has the potential to outperform the solution based on the most widely used geometric Brownian motion model for applications such as the above problem. Designing a training set needs domain knowledge. We want to improve our training set to achieve better predicting accuracy. We are also interested in building trading strategies using SVMs to see if we can earn above-average returns.

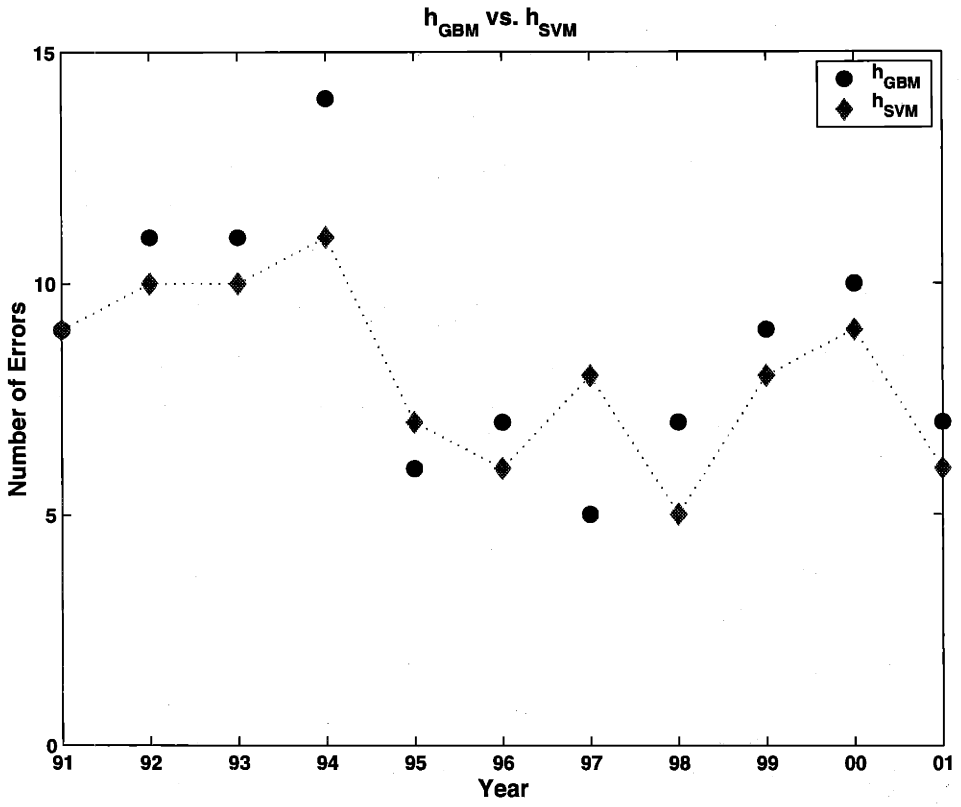


Figure 5.3: The comparison of the two solutions $h_{\mathrm{GBM}}(\cdot)$ and $h_{\mathrm{SVM}}(\cdot)$. The SVM decision rule is trained with poly(3) and $c = 30$.

Appendix A

List of Kernel Functions

The three kernels used in Chapter 4:

1. The linear kernel:

$$k(\boldsymbol{x}, \boldsymbol{y}) = \boldsymbol{x}^T \boldsymbol{y},$$

   where $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^n$.

2. The polynomial kernel with degree d:

$$k(\boldsymbol{x}, \boldsymbol{y}) = (\boldsymbol{x}^T \boldsymbol{y} + 1)^d,$$

   where $\boldsymbol{x}, y \in \mathbb{R}^n$ and d is a positive integer.

3. The radial basis function kernel:

$$k(\boldsymbol{x}, \boldsymbol{y}) = e^{-||\boldsymbol{x}-\boldsymbol{y}||_2^2/(2\sigma^2)},$$

   where $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^n$ and $\sigma \in \mathbb{R}$.

Other positive definite kernels:

- The sigmoid kernel:

$$k(\boldsymbol{x}, \boldsymbol{y}) = \tanh(a\boldsymbol{x}^T \boldsymbol{y} - b),$$

  where $\boldsymbol{x}, y \in \mathbb{R}^n$ and $a, b \in \mathbb{R}$.

- The "erbf" kernel:

$$k(\boldsymbol{x}, \boldsymbol{y}) = e^{-||\boldsymbol{x}-\boldsymbol{y}||_2/(2\sigma^2)},$$

  where $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^n$ and $\sigma \in \mathbb{R}$.

Appendix B

List of Estimated Optimal Settings for SVM^{light} and SvmFu

- Table B.1: The Estimated Optimal Settings for Training Sets Digit17-6000 and Digit17.
- Table B.2: The Estimated Optimal Settings for Training Sets Face-6000, Face-13901 and Face.

Kernels	*Digit17*-6000		*Digit17*	
(c)	SVM^{light}3.5	SvmFu3.0	SVM^{light}3.5	SvmFu3.0
l(1)	(20, 10, 40)	(2000, 100)	(20, 10, 80)	(2000, 200)
l(0.1)	(20, 10, 40)	(2000, 0)	(20, 10, 80)	(2000, 0)
p2(0.1)	(20, 10, 40)	(6000, 0)	(20, 10, 80)	(13007, 0)
p2(0.001)	(20, 10, 40)	(6000, 0)	(20, 10, 80)	(13007, 0)
r10(10)	(20, 10, 40)	(6000, 0)	(20, 10, 80)	(13007, 0)
r10(1)	(20, 10, 40)	(6000, 0)	(20, 10, 80)	(13007, 0)
r3(1)	(20, 10, 40)	(6000, 0)	(20, 10, 80)	(13007, 0)
r3(0.1)	(20, 10, 40)	(6000, 0)	(20, 10, 80)	(13007, 0)

Table B.1: The Estimated Optimal Settings for Training Sets Digit17-6000 and Digit17.

Kernels	*Face*-6000		*Face*-13901		*Face*	
(c)	SVM^{light}3.5	SvmFu3.0	SVM^{light}3.5	SvmFu3.0	SVM^{light}3.5	SvmFu3.0
l(4)	(60, 20, 40)	(6000, 0)	(60, 20, 80)	(13901, 0)	(60,20,200)	(31022,0)
l(1)	(60, 20, 40)	(6000, 0)	(60, 20, 80)	(13901, 0)	(60,20,200)	(31022,0)
l(0.1)	(40, 10, 40)	(2000, 40)	(40, 10, 80)	(2000, 80)	(40,10,200)	(2000,200)
p2(1)	(40, 10, 40)	(6000, 0)	(40, 10, 80)	(13901, 0)	(40,10,200)	(31022,0)
p2(0.001)	(60, 20, 40)	(6000, 0)	(60, 20, 80)	(13901, 0)	(60,20,200)	(31022,0)
r10(10)	(20, 10, 40)	(6000, 0)	(20, 10, 80)	(13901, 0)	(20,10,200)	(31022,0)
r3(10)	(20, 10, 40)	(6000, 0)	(20, 10, 80)	(13901, 0)	(20,10,200)	(31022,0)
r3(1)	(20, 10, 40)	(6000, 0)	(20, 10, 80)	(13901, 0)	(20,10,200)	(31022,0)

Table B.2: The Estimated Optimal Settings for Training Sets Face-6000, Face-13901 and Face.

Appendix C

Our MATLAB Codes

1. FMSvm.m: the training function which chooses the initial guess of G randomly.
2. FMSvm1.m: the training function which chooses the initial guess of G by the ML hyperplane. Since it is the same as FMSvm.m except the initialization of G. We only list the code for this part here.
3. boundMEX.c: the C subroutine of the training function.
4. colOfA.m: the subroutine of the training function that computes the columns of H.

C.1 FMSvm.m

```
%
% FMSvm solves SVM QP problems without chunking:
%       min f(x)=1/2*x'*A*x-1'x
%       s.t. Y'x=0 and 0<= x <=C.
% Usage:
%   [x,bias,steps,SPsizes,CGsteps,testR]=FMSvm(ker,p1,p2,C,accuracy)
%   or (if you want to play with parameters: firstdrops,maxdrops and maxsteps)
%   [x,bias,steps,SPsizes,CGsteps,testR]=FMSvm(ker,p1,p2,C,accuracy,
%                                              firstdrops,maxdrops,maxsteps)
%   Input parameters:
%       X(global) - training points stored in the format: Dimension
%                   by numberOfPoints.
%       Y(global) - the sign vector (a column vector where Y(i)=-/+ 1).
%       ker,p1,p2 - kernel parameters.
%       C - An integer giving the upperbound of the unknown x
%   Values for ker: 'linear'  -
%                   'poly'    - p1 is degree of polynomial
%                   'rbf'     - p1 is width of rbfs (sigma)
%                   'sigmoid' - p1 is scale, p2 is offset
```

```
%                  'spline'  -                              not available
%                  'bspline' - p1 is degree of bspline      not available
%                  'fourier' - p1 is degree                 not available
%                  'erfb'    - p1 is width of rbfs (sigma)
%   Outputs:
%         x - solution.
%         b - bias.
%         steps - steps of the outer loop (number of the subproblems solved).
%         CGsteps - steps of the Conjugate Gradient iteration for each
%                   subproblems.
%         SPsizes - sizes of each subproblems.
%         testR - An interface to an user-defined output.
%   Author: Tong Wen, MIT, June 2001

function [x,bias,steps,SPsizes,CGsteps,testR]=FMSvm(ker,p1,p2,C,accuracy,
                                                    firstdrops,maxdrops,maxsteps)

if ~(nargin == 8 | nargin==5) % check correct number of arguments
   help FMSvm
   return
end

%-------------Initialization-------------------
global X Y
if isempty(X),fprintf('X is undefined!\n');help FMSvm;return;end
if isempty(Y),fprintf('Y is undefined!\n');help FMSvm;return;end

global y numOfCalls normOfX
y=Y;numOfCalls=0;normOfX=[];n=length(Y);
[Dim,j]=size(X);
if n~=j,fprintf('please check the format of X or Y! \n');return;end
[i,j]=size(Y);
if n~=i,fprintf('Y must be a column vector! \n');return;end
testR={};steps=0;CGsteps=[];SPsizes=[];e=ones(n,1);

%-----------------Initialize firstdrops and maxsteps------------------
if nargin==5
    switch lower(ker)
    case 'linear'
        firstdrops=floor(n*0.01); maxstepsUB=Dim;
    case 'poly'
        firstdrops=floor(n*0.02);maxstepsUB=n;
    case 'rbf'
        if p1>=1,
           firstdrops=floor(n*0.02);maxstepsUB=n;
        elseif p1>0.3
```

```
            firstdrops=floor(n*0.025);maxstepsUB=n;
         else
            firstdrops= floor(n*0.03);maxstepsUB=n;
         end
    otherwise
         firstdrops=20;
    end
 firstdrops=max(min(firstdrops,80),20);
         maxsteps=min(firstdrops+1,maxstepsUB);
        %20<=firstdrops<=80, here 20 and 80 are determined based on experience.
end

accuracy0=sqrt(eps);
x=zeros(n,1); % make initial guess to be zero
r=e;
q=n; Ia(1:n,1)=[1:n]'; Ia(1:n,2)=1;% Initialize Ia and q. q is the size of Ia
Ey=y;                              % Ia keeps the indices of active constraints
nq=0;Ina=[];A=[]; % Initialize Ina and nq. nq is the size of Ina
Ehaty=[];         % Ina keeps the indices of non-active constraints

%------First drop: pick up initial sub training set------
postI=Ia(Y==1,1);temp=floor(firstdrops/2);
m1=min(length(postI),temp);
negI=Ia(Y==-1,1);temp=firstdrops-m1;
m2=min(length(negI),temp);
Ina=[postI(1:m1);negI(1:m2)];nq=m1+m2;
Ia(Ina,1)=0;Ia=Ia(Ia(:,1)>0,:);q=length(Ia(:,1));
%if q+nq~=n,fprintf('error in first-drop! \n');return;end
Ey=y(Ia(:,1));Ehaty=y(Ina); % Ia(:,1) and Ina are complementary here.
A(1:nq,:)=colOfA(ker,p1,p2,Ina);
%--------------Outer loop-----------------
stop=0;accuracy1=0.01;
drops=1;
if accuracy>accuracy1,accuracy=accuracy1;end
while ~stop
 steps=steps+1;
 InerLoopSteps=0;
 if drops>0
  if q~=n % compute the projection of r
    Ehatr=r(Ina);
    d=Ehatr-1/nq*Ehaty'*Ehatr*Ehaty;
  else
   d=[];
  end
 end
```

```
normOfd=norm(d);temp1=normOfd^2;
p=d;Ehatx=x(Ina);
EhatxO=Ehatx;InaO=Ina;nqO=nq;
addedI=[];
AA=A(1:nq,Ina);
if normOfd<accuracy1
  accuracy2=accuracy;maxsteps1=maxsteps;
else
  accuracy2=accuracy1;maxsteps1=min(max(nq,maxsteps),maxstepsUB );
end
SPsizes(steps)=nq;

%--------------Inner loop---------------
% Conjugate Gradient Iteration
while normOfd>accuracy2 & InerLoopSteps<maxsteps1
      IaChanged=0;
      EhatAp=p'*AA; % EhatAp is a row vector here!
      alpha=temp1/(EhatAp*p);
      result=boundMEX(Ehatx,p,C);
      l=result(1);lI=result(2);lS=result(3);L=result(4);LI=result(5); LS=result(6);
      if alpha<l
        alpha=l;
        realIndex=Ina(lI);
        q=q+1;Ia(q,1)=realIndex;Ia(q,2)=lS;
        Ey(q,1)=y(realIndex)*lS;
        IaChanged=1;index=lI;
      elseif alpha>L
        alpha=L;
        realIndex=Ina(LI);
        q=q+1;Ia(q,1)=realIndex;Ia(q,2)=LS;
        Ey(q,1)=y(realIndex)*LS;
        IaChanged=1;index=LI;
      end
      Ehatx=Ehatx+alpha*p;
      PAp=EhatAp'-Ehaty*(EhatAp*Ehaty)/nq;
      d=d-alpha*PAp;
      if IaChanged
        x(Ina)=Ehatx; % update x
        % update Ina
        addedI=[addedI index];
        Ina(index)=Ina(nq);%A(index,:)=A(nq,:);
        AA(index,:)=AA(nq,:);AA(:,index)=AA(:,nq);
        Ehaty(index)=Ehaty(nq);d(index)=d(nq);
        nq=nq-1;
        Ehaty=Ehaty(1:nq);AA=AA(1:nq,1:nq);
        Ina=Ina(1:nq);d=d(1:nq);
```

```
        % restart CG iteration
        if q==n
          d=[];
        else
          d=d-1/nq*Ehaty'*d*Ehaty;
        end
        normOfd=norm(d);temp1=normOfd^2;
        p=d;Ehatx=x(Ina);
      else
        temp2=d'*d;
        beta=temp2/temp1;temp1=temp2;
        p=d+beta*p;
        normOfd=sqrt(temp1);
      end
      InerLoopSteps=InerLoopSteps+1;
end
CGsteps(steps)=InerLoopSteps;
if steps==1
 % Adaptively set maxdrops and maxsteps
 if  nargin==5
      switch lower(ker)
        case 'linear'
             mindrops=1;
        case 'poly'
             mindrops=3;
        case 'rbf'
             mindrops=5;
        otherwise
             mindrops=5;
      end
     maxdrops=max(nq,mindrops);maxsteps=min(maxdrops+1,maxstepsUB);
     testR{1}='maxdrops';testR{2}=maxdrops;
 end
end
% update x
x(Ina)=Ehatx;
changeOfx=x(InaO)-EhatxO;
% update residue
if nqO==size(A,1)
    r=r-(changeOfx'*A)';
else
    AA=A(1:nqO,:);
    r=r-(changeOfx'*AA)';
end
m=length(addedI);
% update A
```

```
 if m>0
   for i=1:m
    A(addedI(i),:)=A(nq0,:);
    nq0=nq0-1;
   end
 end
 % compute lambda
 if q~=0
  Er=r(Ia(1:q,1)).*Ia(1:q,2);
  lambda=(Ey'*Er-y'*r)/nq;
  Lambda=-lambda*Ey-Er;
 else
  Lambda=[];
 end

 % drop maxdrop active constraints
 %Dsteps=[Dsteps length(find(Lambda<-normOfd))];
 nlI=find(Lambda<-accuracy0);
 drops=length(nlI);
 if drops
    nq0=nq;
    if drops>maxdrops,drops=maxdrops;end
    [sortedLambda,sortednlI]=sort(Lambda(nlI));
    dropIndexOfIa=nlI(sortednlI(1:drops));
    Ina=[Ina; Ia(dropIndexOfIa,1)];nq=nq+drops;
    Ehaty=Y(Ina);
    Ia(dropIndexOfIa,1)=0;
    temp=(Ia(:,1)>0);
    Ia=Ia(temp,:);Ey=Ey(temp);
    q=q-drops;
    A(nq0+1:nq,:)=colOfA(ker,p1,p2,Ina(nq0+1:nq));
 elseif normOfd<accuracy
    stop=1;
 end
end

 % compute bias
 [mv,mi]=max(x(Ina));
 if mv>accuracy0 & mv<C-accuracy0
     bias=Y(Ina(mi))*(1-A(mi,:)*x);
 else
     bias=0;fprintf('no valid bias is computed!\n');
 end

clear global y numOfCalls normOfX
return
```

C.2 FMSvm1.m

```
%
% FMSvm1 solves SVM QP problems without chunking:
%        min f(x)=1/2*x'*A*x-1'x
%        s.t. Y'x=0 and 0<= x <=C.
% Usage:
%   [x,bias,steps,SPsizes,CGsteps,testR]=FMSvm(ker,p1,p2,C,accuracy)
%   or (if you want to play with parameters: firstdrops,maxdrops and maxsteps)
%   [x,bias,steps,SPsizes,CGsteps,testR]=FMSvm1(ker,p1,p2,C,accuracy,
%                                              firstdrops,maxdrops,maxsteps)
%   Input parameters:
%         X(global) - training points stored in the format: Dimension by numberOfPoints.
%         Y(global) - the sign vector (a column vector where Y(i)=-/+ 1).
%         ker,p1,p2 - kernel parameters.
%         C - An integer giving the upperbound of the unknown x
%   Values for ker: 'linear'  -
%                   'poly'    - p1 is degree of polynomial
%                   'rbf'     - p1 is width of rbfs (sigma)
%                   'sigmoid' - p1 is scale, p2 is offset
%                   'spline'  -                              not available
%                   'bspline' - p1 is degree of bspline      not available
%                   'fourier' - p1 is degree                 not available
%                   'erfb'    - p1 is width of rbfs (sigma)
%   Outputs:
%         x - solution.
%         b - bias.
%         steps - steps of the outer loop (number of the subproblems solved).
%         CGsteps - steps of the Conjugate Gradient iteration for each subproblems.
%         SPsizes - sizes of each subproblems.
%         testR - An interface to an user-defined output.
%   Note: Different from FMSvm, FMSvm1 use the Bayesian hyperplane to
%         initialize thefirst subproblem.
%   Author: Tong Wen, MIT, June 2001

function [x,bias,steps,SPsizes,CGsteps,testR]=FMSvm1(ker,p1,p2,C,accuracy,
                                                     firstdrops,maxdrops,maxsteps)

 . . . % the same as FMSvm.m

%------First drop: pick up initial sub training set------
 %---compute the Beyesian separaring hyperplane----------
 postI=Ia(Y==1,1);negI=Ia(Y==-1,1);
 if (length(postI)==0 | length(negI)==0)
  fprintf('The training set only contains examples from one class!\n');
  return;
```

```
 end
 smPost=mean(X(:,postI),2);smNeg=mean(X(:,negI),2);
 w0=smPost-smNeg;gap=norm(w0);w0=w0/gap;
 sm=(smPost+smNeg)/2; b0=w0'*sm;
 f0=w0'*X-b0;[f0,f0I]=sort(f0);
 f0postI=find(f0>0);temp=floor(firstdrops/2);
 m1=min(length(f0postI),temp);
 m2=min(firstdrops-m1,n-length(f0postI));

Ina=[f0I((f0postI(1)-m2):(f0postI(1)-1))';f0I(f0postI(1:m1))'];nq=m1+m2;
Ia(Ina,1)=0;Ia=Ia(Ia(:,1)>0,:);q=length(Ia(:,1));
Ey=y(Ia(:,1));Ehaty=y(Ina); % Ia(:,1) and Ina are complementary here.
A(1:nq,:)=colOfA(ker,p1,p2,Ina);


%--------------Outer loop-----------------

 . . . % The same as FMSvm.m

return;
```

C.3 boundMEX.c

```
#include <math.h>
#include "mex.h"

void mexFunction( int nlhs, mxArray *plhs[],
  int nrhs, const mxArray*prhs[] )

{
    double *result;
    double *x,*d,*C;
    unsigned int m,i;
    double temp1,temp2,temp3;
    double l=-1000000;
    double u=1000000;
    double  lIndex,uIndex, uSign,lSign;

    /* Check for proper number of arguments */
    if (nrhs != 3) {
mexErrMsgTxt("Three input arguments required.");
    } else if (nlhs > 1) {
mexErrMsgTxt("Too many output arguments.");
    }

    /* get the length of the input vector */
```

```
    m = mxGetM(prhs[0]);

    /* Create a matrix for the return argument */
    plhs[0] = mxCreateDoubleMatrix(6,1,mxREAL);

    /* Assign pointers to the various parameters */
    result = mxGetPr(plhs[0]);

    x = mxGetPr(prhs[0]);
    d = mxGetPr(prhs[1]);
    C = mxGetPr(prhs[2]);

    /* Do the actual computations here*/
    for (i=0;i<m;i++){
      if (d[i]>0){
        temp1=-x[i]/d[i];temp2=C[0]/d[i];temp3=temp2+temp1;
        if (temp1>l) {l=temp1;lIndex=i+1;lSign=1;}
        if (temp3<u) {u=temp3;uIndex=i+1;uSign=-1;}
      }
      else if (d[i]<0){
        temp1=-x[i]/d[i];temp2=C[0]/d[i];temp3=temp2+temp1;
        if (temp3>l) {l=temp3;lIndex=i+1;lSign=-1;}
        if (temp1<u) {u=temp1;uIndex=i+1;uSign=1;}
      }
    }
    result[0]=l;result[1]=lIndex;result[2]=lSign;
    result[3]=u;result[4]=uIndex;result[5]=uSign;
    return;

}
```

C.4 colOfA.m

```
%  Compute columns of the SVM Hessian matrix
%
%  Usage: x = colOfA(ker,p1,p2,colIndex)
%  Called by myKCGSVM classifier
%* X is stored in the format: Dimension by numberOfPoints.
%
%  Parameters: ker - kernel type
%
%  Values for ker: 'linear'  -
%                  'poly'    - p1 is degree of polynomial
%                  'rbf'     - p1 is width of rbfs (sigma)
%                  'sigmoid' - p1 is scale, p2 is offset
%                  'spline'  -                          not available
```

```
%                 'bspline' - p1 is degree of bspline     not available
%                 'fourier' - p1 is degree                not available
%                 'erfb'    - p1 is width of rbfs (sigma)
%
%
%  Auther: Tong Wen, May 2001 (M.I.T.)

function x=colOfA(ker,p1,p2,colIndex)
% X is stored in the format: Dimension by numberOfPoints.
global X y numOfCalls normOfX
% numOfCalls must be initialized as zero
if (nargin ~= 4) % check correct number of arguments
   help colOfA
   return
end

[m,n]=size(X);

x=X(:,colIndex)'*X;

switch lower(ker)
case 'linear'
   x=(y(colIndex)*y').*x;
case 'poly'
   x=(x+1).^p1;
   x=(y(colIndex)*y').*x;
case 'sigmoid'
   x = tanh(p1*x/m + p2);
   x=(y(colIndex)*y').*x;
case 'rbf'
   if numOfCalls==0
      for i=1:n, normOfX(i)=X(:,i)'*X(:,i);end
   end
   k=length(colIndex);
   s=ones(k,1)*normOfX;
   S=normOfX(colIndex)'*ones(1,n);
   x = exp(-(s-2*x+S)/(2*p1^2));
   x=(y(colIndex)*y').*x;
case 'erbf'
   if numOfCalls==0
      for i=1:n, normOfX(i)=X(:,i)'*X(:,i);end
   end
   k=length(colIndex);
   s=ones(k,1)*normOfX;
   S=normOfX(colIndex)'*ones(1,n);
   x = exp(-sqrt(s-2*x+S)/(2*p1^2));
```

```
   x=(y(colIndex)*y').*x;
otherwise
   x=(y(colIndex)*y').*x;
end
numOfCalls=numOfCalls+1;
return;
```

Bibliography

[1] Mordecai Ariel. *Nonlinear Programming: Analysis and Methods.* Prentice-Hall, N.J., 1976.

[2] N. Aronszajn. Theory of Reproducing Kernels. *Transactions of the Mathematical Society*, 68:337-404, 1950.

[3] K. P. Bennett. A Support Vector Machine Approach to Decision Trees. *Proceedings of IJCNN'98*, 2396-2401, Anchorage, Alaska, 1997.

[4] K. P. Bennett, D. H. Wu and L. Auslender. On Support Vector Decision Trees for Database Marketing. R.P.I. Math Report NO. 98-100, Rensselaer Polytechnic Institute, Troy, NY, 1998.

[5] D. Bertsekas. *Nonlinear Programming.* Athena Scientific, Belmont, MA, 1995.

[6] F. Black, M. Scholes. The Pricing of Options and Corporate Liabilities. *Journal of Political Economy*, 81 (May-June 1973), 637-59.

[7] T. Bollerslev. Generalized Autoregressive Conditional Heteroscedasticity. *Journal of Econometrics*, 31 (1986), 307-27.

[8] John C. G. Boot. *Quadratic Programming.* Rand McNally & Company, Chicago, 1964.

[9] B. E. Boser, I. M. Guyon and V. Vapnik. A Training Algorithm for Optimal Margin Classifiers. *Proceedings of the 5th Annual ACM Workshop on Computational Learning Theory*, 144-152, Pittsburgh, PA, 1992. ACM Press.

[10] L. Bottou, C. Cortes, J. Denker, H. Drucher, I. Guyon, L. Jackel, Y. LeGun, U. Müller, E. Säckinger, P. Simard and V. Vapnik. Comparison of Classifier Methods: a Case Study in Handwritten Digit Recognition. *Proceedings of the 12th International Conference on Pattern Recognition and Neural Networks, Jerusalem*, 77-87. IEEE Computer Society Press, 1994.

[11] P. S. Bradley, O. L. Mangasarian. Feature Selection via Concave Minimization and Support Vector Machines. Technical Report Mathematical Programming Technical Report 98-03, University of Wisconsin-Madison, 1998.

[12] P. S. Bradley, O. L. Mangasarian. Massive Data Discrimination via Support Vector Machines. Technical Report Mathematical Programming Technical Report 98-05, University of Wisconsin-Madison, 1998.

[13] E. J. Bredenstainer, K. P. Bennett. Multicategory Classification by Support Vector Machines. *Computational Optimization and Applications*, 1998.

[14] M. Brown, W. Grundy, D. Lin, N. Cristianini, C. Sugnet, T. Furey, M. Ares Jr., and D. Haussler. Knowledge-based Analysis of Microarray Gene Expression Data By Using Support Vector Machines. *Proceedings of the National Academy of Science*, 97(1):262-267.

[15] A. Buhot, M. Gordon. Robut Learning and Generalization with Support Vector Machines. *Jounal of Physics A*, 34, no. 21, 4377–4388, 2001.

[16] Christopher J.C. Burges. Simplified Support Vector Decision Rules. *Proceedings, 13th Intl. Conf. on Machine Learning*, 71-77, San Mateo, CA, 1996. Morgan Kaufmann.

[17] C. Burges, B. Schölkopf. Improving the Accuracy and Speed of Support Vector Learning Machines. *Advances in Neural Information Processing Systems 9*, edited by M. Mozer, M. Jordan, M. Kearns and T. Petsche, MIT Press, Cambridge, MA, 1997.

[18] Christopher J. C. Burges. A Tutorial on Support Vector Machines for Pattern Recognition. *Knowledge Discovery and Data Mining*, 2(2), 1998.

[19] Ronan Collobert. *SVM Torch II.* `http://www.idiap.ch/learning/SVMTorch.html`.

[20] Ronan Collobert and Samy Bengio. SVMTorch: Support Vector Machines for Large-Scale Regression Problems. *Journal of Machine Learning Research* 1, no. 2, 143-160, 2001.

[21] C. Cortes, V. Vapnik. Support Vector Networks. *Machine Learning*, 20:273-297, 1995.

[22] A. Crampton, J. Mason, D. Turner. Approximating Semi-Structured Data with Different Errors Using Support Vector Machine Regression. *Mathematical methods for curves and surfaces* (Oslo, 2000), 63–72. Innovations in Applied Mathematics, Vanderbilt Univ. Press, Nashville, TN, 2001.

[23] N. Cristianini, J. Shawe-Taylor and P. Sybacek. Bayesian Classifiers are Large Margin Hyperplanes in a Hilbert Space. *Machine Learning: Proceedings of the 15th International Conference*, San Francisco, CA, 1998. Morgan Kaufmann.

[24] Nello Cristianini, John Shawe-Taylor. *An Introduction to Support Vector Machines and Other Kernel-based Learning Methods.* Cambridge University Press, 2000.

[25] Craig C. Douglas, Gundolf Haase, Jonathan Hu, Markus Kowarschik, Ulrich Rüde and Christian Weiss. Portable Memory Hierarchy Techniques For PDE Solvers: Part I. *SIAM NEWS*, Volume 33, Number 5, June 2000.

[26] H. Drucker, C. Burges, L. Kaufman, A. smola and V. Vapnik. Support Vector Regression Machines. *Advances in Neural Information Processing Systems 9*, edited by M. Mozer, M. Jordan, M. Kearns and T. Petsche, MIT Press, Cambridge, MA, 1997.

[27] Theodoros Evgeniou, Massimiliano Pontil, Tomaso Poggio. A Unified Framework for Regularization Networks and Support Vector Machines, CBCL Paper, No. 171, MIT, 1999.

[28] F. Girosi. An Equivalence Between Sparse Approximation and Support Vector Machines. *Neural Computation*, 10(6):1455-1480, 1998.

[29] F. Girosi, M. Jones and T. Poggio. Regularization Theory and Neural Networks Architectures. *Neural Computation*, 7(2):219-269, 1995.

[30] Gene H. Golub, Charles F. Van Loan. *Matrix Computation.* The Johns Hopkins University Press, Baltimore and London, 1996.

[31] Y. Guo, P. Bartlett, J. Shawe-Taylor, R. Williamson. Covering Numbers for Support Vector Machines. *IEEE Transactions on Information Theory*, 48, no. 1, 230-250, 2002.

[32] I. Guyon, B. Boser and V. Vapnik. Automatic Capacity Tuning of Very Large VC-Dimension Classifiers. *Advances in Neural Information Processing Systems*, 5:147-155, edited by S. Hanson, J. Cowan and C. Giles. Morgan Kaufmann, San Mateo, CA, 1993.

[33] Thorsten Joachims. Text Categorization with Support Vector Machines. *European Conference on Machine Learning*, 1998.

[34] Thorsten Joachims. Making Large-Scale Support Vector Machine Learning Practical. *Advances in Kernel Methods: Support Vector Learning*, edited by Bernhard Schölkopf, Christopher J.C. Burges and Alexander J. Smola, The MIT Press, Cambridge, 1998.

[35] Thorsten Joachims. SVM^{light} 3.5. `http://svmlight.joachims.org`.

[36] Robert V. Hogg, Elliot A. Tanis. *Probability and Statistical Inference.* Prentice-Hall, Inc, New Jersey, 1997.

[37] J.C. Hull, A. White. The Pricing of Options on Assets with Stochastic Volatility. *Journal of Finance*, 42 (1987), 281-300.

[38] John C. Hull. *Options, Futures, and Other Derivatives.* Prentice-Hall, Inc, New Jersey, 1999.

[39] Parry Husbands. Interactive Supercomputing. Ph.D. Thesis, MIT, 1999.

[40] Linda Kaufman. Solving the Quadratic Programming Problem Arising in Support Vector Classification. *Advances in Kernel Methods: Support Vector Learning*, edited by Bernhard Schölkopf, Christopher J.C. Burges and Alexander J. Smola, The MIT Press, Cambridge, 1998.

[41] M. Kirby, L. Sirovich. Application of the Karhunen-Loève Procedure for the Characterization of Human Faces. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(1):103-108, 1990.

[42] U. Krebel. The Impact of the Learning-Set Size in Handwritten Digit Recognition. *Artificial Neural Networks – ICANN'91*, edited by T. Kohonen, 1685-1689, Amsterdam, 1991. North-Holland.

[43] U. Krebel. Polynomial Classifiers and Support Vector Machines. *Artificial Neural Networks – ICANN'97*, edited by W. Gerstner, 397-402, Berlin, 1997. Springer Lecture Notes in Computer Science, Vol. 1327.

[44] H. W. Kuhn, A. W. Tucker. Nonlinear Programming. *Proceedings of the Second Berkekey Symposium on Mathematical Statistics and Probability*, edited by J. Neyman, 481-92, 1951.

[45] Y. Lee, L. Mangasarian, and W. Wolberg. Breast Cancer Survial and Chemotherapy: A Support Vector Machine Analysis. *Discrete Mathematical Problems with Medical Applications* (New Brunswik, NJ, 1999), 1-10. DIMACS Series in Discrete Mathematics and Theoretical Computer Science, American Mathematical Society, Providence, RI, 2000.

[46] Y. Lee, L. Mangasarian. SSVM: A Smooth Support Vector Machine for Classification. *Computational Optimization and Applications*, 20, no. 1, 5-22, 2001.

[47] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard and L. Jackel. Backpropagation Applied to Handwritten Zip Code Recognition. *Neural Computation*, 1:541-551, 1989.

[48] Y. LeCun, L. Jackel, L. Bottou, C. Cortes, J. Denker, H. Drucker, I. Guyon, U. Müller, E. Säckinger, P. Simard and V. Vapnik. Comparison of Learning Algorithms for Handwritten Digit Recognition. *Proceedings ICANN'95 – International Conference on Artificial Neural Networks*, edited by F. Fogelman-Soulié and P. Gallinari. Volume II, 53-60, Nanterre, France, 1995.

[49] L. Mangasarian, D. Musicant. Data Discrimination via Nonlinear Generalized Support Vector Machines. *Complementarity: Applications, Algorithms and Extensions* (Madison, WI, 1999), 233–251. Applied Optimization, 50, Kluwer Acad. Publ., Dordrecht, 2001.

[50] L. Mangasarian, D. Musicant. Lagrangian Support Vector Machines. *Journal of Machine Learning Research* 1, no. 3, 161–177, 2001.

[51] R.C. Merton. Theory of Rational Option Pricing. *Bell Journal of Economics and Management Science*, 4 (Spring 1973), 141-183.

[52] R.C. Merton. *Continuous-Time Finance.* Blackwell Publishers Inc, Malden, MA, 1990.

[53] Robb J. Muirhead. *Aspects of Multivariate Statistical Theory.* John Wiley & Sons, Inc, 1982.

[54] S. Mukherjee, E. Osuna, F. Girosi. Nonlinear Prediction of Chaotic Time Series Using a Support Vector Machine. *Neural Networks for Signal Processing VII – Proceeding of the 1997 IEEE Workshop*, edited by J. Principe, L. Gile, N. Morgan and E. Wilson, 511-520, New York, 1997. IEEE.

[55] K, Müller, A. Smola, G. Rätch, B. Schölkopf, J. Kohlmorgen and V. Vapnik. Predicting Time Series with Support Vector Machines. *Artificial Neural Networks – ICANN'97*, edited by W. Gerstner, 397-402, Berlin, 1997. Springer Lecture Notes in Computer Science, Vol. 1327.

[56] R. Neal. *Bayesian Learning in Neural Networks.* Springer Verlag, 1996.

[57] N. J. Nilsson. *Learning machines: Foundations of Trainable Pattern Classifying Systems.* McGraw-Hill, 1965.

[58] E. Osuna, R. Freund and F. Girosi. An Improved Training Algorithm for Support Vector Machines. *Neural Networks for Signal Processing VII – Proceeding of the 1997 IEEE Workshop*, edited by J. Principe, L. Gile, N. Morgan and E. Wilson, 511-520, New York, 1997. IEEE.

[59] E. Osuna, R. Freund and F. Girosi. Support Vector Machines: Training and Applications. AI Memo 1602, M.I.T., 1997.

[60] E. Osuna, R. Freund and F. Girosi. Training Support Vector Machines: An Application to Face Detection. *Proceedings, Computer Vision and Pattern Recognition'97*, 130-136, 1997.

[61] John C. Platt. Fast Training of Support Vector Machines Using Sequential Minimal Optimization. *Advances in Kernel Methods: Support Vector Learning*, edited by Bernhard Schölkopf, Christopher J.C. Burges and Alexander J. Smola, The MIT Press, Cambridge, 1998.

[62] T. Poggio and F. Girosi. Networks for Approximation and Learning. *Proceedings of the IEEE*, 78(9), 1990.

[63] T. Poggio and F. Girosi. Regularization Algorithms for Learning That Are Equivalent to Multilayer Networks. *Science*, 247:978-982, 1990.

[64] M. Pontil and A. Verri. Properties of Support Vector Machines. *Neural Computation*, 10:955-974, 1997.

[65] M. Pontil, S. Mukherjee, F. Girosi. On the Noise Model of Support Vector Machines Regression. *Algorithmic learning theory* (Sydney, 2000), 316–324. Lecture Notes in Computer Science, 1968, Springer, Berlin, 2000.

[66] Ryan Rifkin. *SvmFu* 3.0. `http://fpn.mit.edu/SvmFu/`.

[67] R. Rifkin, M. Pontil, A, Verri. A Note on Support Vector Machine Degeneracy. *Algorithmic Learning Theory* (Tokyo, 1999), 252–263. Lecture Notes in Computer Science, 1720, Springer, Berlin, 1999.

[68] S. Saitoh. *Theory of Reproducing Kernels and its Applications.* Longman Scientific & Technical, Harlow, England, 1998.

[69] P.A. Samuelson. Proof That Properly Anticipated Prices Fluctuate Randomly. *Industrial Management Review*, 6 (Spring), 41-9.

[70] P.A. Samuelson. Proof That Properly Discounted Present Values of Assets Vibrate Randomly. *Bell Journal of Economics and Management Science*, 4 (Autumn), 369-74.

[71] M. Schmidt and H. Gish. Speaker Identification via Support Vector Classifiers. *Proceedings of ICASSP'96*, 105-108, Atlanta, GA, 1996.

[72] B. Schölkopf, V. Vapnik. Extracting Support Data for a Given Task. *Proceedings, First International Conference on Knowledge Discovery and Data Mining.* AAAI Press, Menlo Park, CA, 1995.

[73] B. Schölkopf. *Support Vector Learning.* R. Oldenbourg Verlag, Munich, 1997.

[74] B. Schölkopf, A. Smola and R. Williamson. Support Vector Regression with Automatic Accuracy Control. *Proceedings of the 8th International Conference on Artificial Neural Networks,* edited by L. Niklasson, M. Bodén and T. Ziemke. Perspectives in Neural Computing, Berlin, 1998.

[75] B. Schölkopf, C. Burges and V. Vapnik. Incorporating Invariances in Support Vector Learning Machines. *Artificial Neural Networks – ICANN'96*, edited by C. von der Malsburg, W. won Seelen, J. C¿ Vorbrüggen and B. Sendhoff, 47-52, Berlin, 1996. Springer Lecture Notes in Computer Science, Vol. 1112.

[76] B. Schölkopf, P. Knirsch, A. Smola and C. Burges. Fast Approximation of Support Vector Kernel Expansions, and an Interpretation of Clustering as Approximation in Feature Spaces. *20th DAGM Symposium Mustererkennung*, Lecture Notes in Computer Science. Springer, Berlin, 1998.

[77] B. Schölkopf, P. Simard, A. Smola and V. Vapnik. Prior Knowledge in Support Vector Kernels. *Advances in Neural Information Processing Systems 10*, edited by M. Jordan, M. Kearns and S. Solla, MIT Press, Cambridge, MA, 1998.

[78] B. Schölkopf, A. Smola and K. Müler. Nonlinear Component Analysis as a Kernel Eigenvalue Problem. *Neural Computation*, 10:1299-1319, 1998.

[79] B. Schölkopf, A. Smola, K. Müler, C. Burges and V. Vapnik. Support Vector Methods in Learning and Feature Extraction. *Proceedings of the Ninth Australian Conference on Neural Networks*, edited by T. Downs, M. Frean and M. Gallagher, University of Queensland, Brisbane, Australia, 1998.

[80] Edited by Bernhard Schölkopf, Christopher J.C. Burges and Alexander J. Smola. *Advances in Kernel Methods: Support Vector Learning.* The MIT Press, Cambridge, 1998.

[81] B. Schölkopf, K. Sung, K. C. Burges, F. Girosi, P. Niyogi, T. Poggio and V. Vapnik. Comparing Support Vector Machines with Gaussian Kernels to Radial Basis Function Classifiers. *IEEE Transactions on Signal Processing*, 45:2758-2765, 1997.

[82] Bernhard Schölkopf, Sebastian Mika, Chris J.C. Bergurges. Input Space Vs. Feature Space in Kernel-based Methods. *IEEE Transactions on Neural Networks*, 1999.

[83] A. Smola, B. Schölkopf. From Regularization Operators to Support Vector Kernels. *Advances in Neural Information Processing Systems 10*, edited by M. Jordan, M. Kearns and S. Solla, MIT Press, Cambridge, MA, 1998.

[84] A. Smola, B. Schölkopf. On a Kernel Based Method for Pattern Recognition, Regression, Approximation and Operator Inversion. *Algorithmica*, 1998.

[85] A. Smola, B. Schölkopf and K. Müler. The Connection Between Regularization Operators and Support Vector Kernels. *Neural Networks*, 11:637-49, 1998.

[86] A. Smola, B. Schölkopf and K. Müler. Convex Cost Functions for Support Vector Regression. *Proceedings of the 8th International Conference on Artificial Neural Networks,* edited by L. Niklasson, M. Bodén and T. Ziemke. Perspectives in Neural Computing, Berlin, 1998.

[87] A. Smola, B. Schölkopf and K. Müler. General Cost Functions for Support Vector Regression. *Proceedings of the Ninth Australian Conference on Neural Networks*, edited by T. Downs, M. Frean and M. Gallagher, University of Queensland, Brisbane, Australia, 1998.

[88] Ingo Steinwart. On the Influence of the Kernel on the Consistency of Support Vector Machines. *Journal of Machine Learning Research* 2, no. 1, 67-93, 2002.

[89] M. Stitson, A. Gammerman, V. Vapnik, V. Vovk, C. Watkins and J. Weston. Support Vector Regression with ANOVA Decomposition Kernels. Technical Report CSD-TR-97-22, Royal Holloway, University of London, 1997.

[90] Gilbert Strang. *Introduction to Applied Mathematics.* Wellesley-Cambridge Press, Wellesley, MA, 1986.

[91] Lloyd N. Trefethen, David Bau, III. *Numerical Linear Algebra.* SIAM, Philadelphia, 1997.

[92] Vladimir N. Vapnik. *Estimation of Dependences Based on Empirical Data (in Russian).* Nauka, Moscow, 1979. (English translation: Spring Verlag, New York, 1982).

[93] Vladimir N. Vapnik. *The Nature of Statistical Learning Theory.* Springer Verlag, New York, 1995.

[94] Vladimir N. Vapnik. Structure of Statistical Learning Theory. *Computational and Probabilistic Reasoning,* Chapter 1, edited by A. Gammerman. Wiley, Chichester, 1996.

[95] Vladimir N. Vapnik. *Statistical Learning Theory.* John Wiley & Sons, Inc, 1998.

[96] V. Vapnik, A. Chervonenkis. A Note on One Class of Perceptrons. *Automation and Remote Control,* 25, 1964.

[97] V. Vapnik, A. Chervonenkis. Uniform Convergence of Frequencies of Occurrence of Events to Their Probabilities. *Dokl. Akad. Nauk SSSR*, 181:915-918, 1968.

[98] V. Vapnik, A. Chervonenkis. On the Uniform Convergence of Relative Frequencies of Events to Their Probabilities. *Theory of Probability and its Applications*, 16(2):264-280, 1971.

[99] V. Vapnik, A. Chervonenkis. *Theory of Pattern Recognition (in Russian)*. Nauka, Moscow, 1974. (German Translation: W. Wapnik, A. Tscherwonenkis, *Theorie der Zeichenerkennung*. Akademie-Verlag, Berlin, 1979.)

[100] V. Vapnik, A. Chervonenkis. Necessary and Sufficient Conditions for the Uniform Convergence of Means to Their Expectations. *Theory of Probability and its Applications*, 26(3):532-553, 1981.

[101] V. Vapnik, S. Golowich and A. Smola. Support Vector Method for Function Approximation, Regression Estimation, and Signal Processing. *Advances in Neural Information Processing Systems 9*, edited by M. Mozer, M. Jordan and T. Petsche, 281-287. MIT Press, Cambridge, MA, 1997.

[102] V. Vapnik, A. Lerner. Pattern Recognition Using Generalized Portrait Method. *Automation and Remote Control*, 24, 1963.

[103] V. Vapnik, E. Levin and Y. Le Cun. Measuring the VC-dimension of a Learning Machine. *Neural Computation*, 6(5):851-876, 1994.

[104] G. Wahba. Multivariate Function and Operator Estimation, Based on Smoothing Splines and Reproducing Kernels. *Nonlinear Modeling and Forecasting, SFI Studies in the Sciences of Complexity, Proc. Vol XI*, edited by M. Casdagli, S. Eubank, 95-112. Addison-Wesley, 1992.

[105] G. Wahba. Support Vector Machines, Reproducing Kernel Hilbert Spaces, and Randomized GACV. *Advances in Kernel Methods: Support Vector Learning*, edited by Bernhard Schölkopf, Christopher J.C. Burges and Alexander J. Smola, The MIT Press, Cambridge, 1998.

[106] Tong Wen. *FMSvm* 1.0. `http://math.mit.edu/~tonywen/FMSvm/`.

[107] J. Weston and C. Watkins. Multi-Class Support Vector Machines. Technical Report CSD-TR-98-04, Department of Computer Science, Royal Holloway, University of London, UK, 1998.

[108] R. Whaley and J. Dongarra. Automatically Tuned Linear Algebra Software (ATLAS). *SC '89 Proceedings* (Electronic Publication). IEEE Publication.

[109] R. Clint Whaley, Antoine Petitet, and Jack Dongarra. Automated Empirical Optimizations of Software and the ATLAS Project. *Parallel Computing*, Volume 27, Numbers 1-2, pp 3-25, 2001.

[110] R.Clint Whaley, Antoine Petitet, Jack J. Dongarra. *Automated Empirical Optimization of Software and the ATLAS Project.* `http://math-atlas.sourceforge.net/`.

[111] R. C. Williamson, A. J. Smola and B. Schölkopf. Generalization Performance of Regularization Networks and Support Vector Machines via Entropy numbers of compact operators. Technical Report CSD-TR-98-04, Department of Computer Science, Royal Holloway, University of London, UK, 1998.

[112] R. C. Williamson, A. J. Smola and B. Schölkopf.Entropy Numbers, Operators and Support Vector Kernels. *Computational Learning Theory* (Nordkirchen, 1999), 285–299. Lecture Notes in Computer Science, 1572, Springer, Berlin, 1999.

[113] Alan S. Willsky, Gregory W. Wornell, Jeffrey H. Shapiro. *Stochastic Processes, Detection and Estimation.* 6.432 Course Notes, MIT, Fall 1998.