

## INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

**The quality of this reproduction is dependent upon the quality of the copy submitted.** Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

ProQuest Information and Learning  
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA  
800-521-0600

UMI<sup>®</sup>



UNIVERSITY OF OKLAHOMA

GRADUATE COLLEGE

**DECOMPOSITION TECHNIQUES FOR SUPPORT VECTOR  
MACHINES TRAINING AND APPLICATIONS**

A Dissertation

SUBMITTED TO THE GRADUATE FACULTY

in partial fulfillment of the requirements for the

degree of

Doctor of Philosophy

By

HUSEYIN INCE

Norman, Oklahoma

2002

UMI Number: 3070638



---

UMI Microform 3070638

Copyright 2003 by ProQuest Information and Learning Company.  
All rights reserved. This microform edition is protected against  
unauthorized copying under Title 17, United States Code.

---

ProQuest Information and Learning Company  
300 North Zeeb Road  
P.O. Box 1346  
Ann Arbor, MI 48106-1346

DECOMPOSITION TECHNIQUES FOR SUPPORT VECTOR  
MACHINES TRAINING AND APPLICATIONS

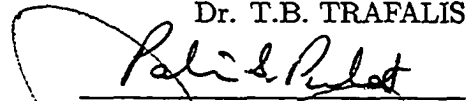
A Dissertation

APPROVED FOR THE  
SCHOOL OF INDUSTRIAL ENGINEERING

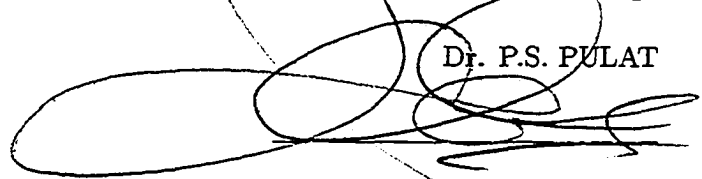
BY



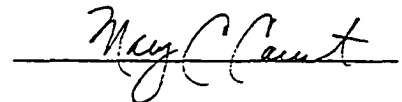
Dr. T.B. TRAFALIS



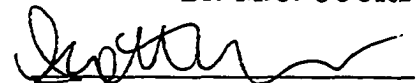
Dr. P.S. PULAT



Dr. J.Y. CHEUNG



Dr. M.C. COURT



Dr. S.A. MOSES

©Copyright by Huseyin Ince 2002

All Rights Reserved.

# Acknowledgements

I would like to thank the faculty and staff of the University of Oklahoma for having supported me throughout the course of my studies. In particular, I would like to express my gratitude to Dr. T.B. Trafalis, Dr. P.S. Pulat, Dr. J.Y. Cheung, Dr. S.A. Moses and Dr. M.C. Court who have provided careful guidance and advice while serving on my committee during my doctoral studies. Special thanks to Dr. T.B. Trafalis for having served as my advisor. His expertise and guidance were of great importance for this research. I would like to thank the Republic of Turkey Ministry of Education for giving me a chance to study in the United States. Finally, I would also like to acknowledge the support of the National Science Foundation under NSF Grant ECS-9978813 and ECS-0099378.

Dedicated to my parents, SERIFE INCE and RIFAT INCE



# Contents

<b>Acknowledgements</b>	<b>iv</b>
	<b>v</b>
<b>Abstract</b>	<b>xiv</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Overview . . . . .	1
1.2 Research Objectives . . . . .	4
1.3 Scope of the Dissertation . . . . .	5
<b>2 Mathematical Formulation of Support Vector Machine</b>	<b>6</b>
2.1 Empirical Risk Minimization . . . . .	6
2.2 Structural Risk Minimization . . . . .	8
2.3 Definitions . . . . .	9
2.4 SVM formulation by using $l_1$ , $l_2$ , and $L_\infty$ norms . . . . .	13
2.4.1 $l_2$ Formulation . . . . .	14
2.4.2 $l_1$ norm formulation . . . . .	22

2.4.3	$l_\infty$ norm formulation . . . . .	25
2.4.4	Kernelization of Linear Programming Formulation . . . . .	27
2.5	Support vector regression formulation by using $l_1, l_2$ norms . . . . .	28
2.5.1	$l_2$ norm formulation . . . . .	28
2.5.2	$l_1$ norm formulation . . . . .	34
<b>3</b>	<b>Benders' Decomposition Technique</b>	<b>37</b>
3.1	Introduction . . . . .	37
3.2	Benders' Decomposition Technique for Support Vector Machines . . . . .	38
<b>4</b>	<b>Training support vector machines with very large datasets</b>	<b>46</b>
4.1	Clustering and Bagging techniques for Support Vector Machines . . . . .	46
4.1.1	Support Clusters . . . . .	47
4.1.2	Sub-sampling and Bagging Methods . . . . .	50
4.1.3	Hierarchical Training Methods . . . . .	54
4.2	$\gamma$ -SVM Algorithm . . . . .	55
4.2.1	$\gamma$ -SVM in feature space . . . . .	56
<b>5</b>	<b>Applications and Computational Results</b>	<b>63</b>
5.1	Comparison of the methods with other decomposition techniques . . . . .	64
5.1.1	Classification case . . . . .	64
5.1.2	Regression Case . . . . .	68
5.2	Applications to Financial Markets . . . . .	70
5.2.1	Prediction of S&P 500 Daily Return Value . . . . .	70

5.2.2	Option Pricing Model . . . . .	74
<b>6</b>	<b>Summary, Conclusions and Recommendations</b>	<b>84</b>
6.1	Summary . . . . .	84
6.2	Recommendations for Future Research . . . . .	85
<b>A</b>	<b>Implementation of Benders Decomposition for Support Vector regres-</b>	
	<b>sion</b>	<b>94</b>
<b>B</b>	<b>C++ implementation of e-SVM algorithm</b>	<b>109</b>
<b>C</b>	<b>Bagging and Subsampling Codes in Matlab</b>	<b>116</b>

# List of Tables

2.1	.....	12
2.2	.....	12
3.1	.....	44
3.2	.....	45
4.1	.....	48
4.2	.....	62
5.1	.....	66
5.2	.....	67
5.3	.....	68
5.4	.....	69
5.5	.....	70
5.6	.....	70
5.7	.....	73
5.8	.....	73
5.9	.....	78

5.10 . . . . .	82
5.11 . . . . .	82
5.12 . . . . .	83

# List of Figures

2.1	$l_1, l_2, l_\infty$ norm distances . . . . .	11
2.2	Linear separability of AND function . . . . .	13
2.3	Linearly non-separable Case . . . . .	14
2.4	Separating hyper-plane and optimal separating hyper-plane. Both solid lines in (a) and (b) separate the two identical sets described by circles and triangles. But the solid line in (b) leaves the closest points (filled circles and triangles) at the maximum distance. The distance between dashed lines in (b) gives the maximum margin. . . . .	15
2.5	Mapping input space to feature space . . . . .	20
2.6	Maximum margin separation . . . . .	24
2.7	The soft margin lost setting corresponds for a linear SV Machine [54] . .	30
2.8	$\varepsilon$ -insensitive case . . . . .	31
4.1	The checkerboard data without clusters. . . . .	51

4.2	Clustering the checker-board data. Red color shows the whole data (486 points) and blue circles are the clusters centers (50 points). We can represent the whole data set by only 50 points. Those points are used for the SVM algorithm. . . . .	52
4.3	The checkerboard data set and 131 total clusters. The big circle shows the clusters. We choose only 20 % of data to represent the whole data set (650 points). . . . .	53
4.4	Support clusters which are found by using the SVM and original data that we extract by using those clusters. We use the clusters in figure 4.3. Support clusters are the clusters that we found by SVM. . . . .	54
4.5	Solution of proximal support vector machine classifier . . . . .	58
4.6	Identifying the closest points by using the algorithm we proposed. By exploring the sparsity of SVM theory, we do not need to whole dataset, we can use only filled data points shown in figure. . . . .	59
4.7	First figure(a) shows the decision surface of radial basis function kernel with $\sigma = 1$ . In the second figure depicts the decision surface with $\sigma = 3.2$ . . . . .	60
4.8	(a): In this figure, linear kernel function is used to train whole data set. (b): Linear kernel function is used with our algorithm. Decision surface are same for SVM and our algorithm . . . . .	61
5.1	CPU time vs. Number of Clusters SVM. . . . .	71
5.2	Correctly classified test data (percent) vs. number of clusters . . . . .	72
5.3	Estimated and actual value of $\sin x$ funtion (+ is actual, solid estimated). . . . .	74

5.4	MSE's for SVM Regression (MSE vs. RBF Parameter) . . . . .	75
5.5	MSE's for SVM Regression (MSE vs. RBF Parameter) . . . . .	76
5.6	MSE vs. Number of Hidden Units for MLP Network . . . . .	77
5.7	Sample path of the stock price . . . . .	80



# Abstract

The theory of the Support Vector Machine (SVM) algorithm is based on statistical learning theory and can be applied to pattern recognition and regression. Training of SVMs leads to either a quadratic programming (QP) problem, or a linear programming (LP) problem. This depends on the specific norm that is used when the distance between the two classes is computed.  $l_1$  or  $l_\infty$  norm distance leads to a large scale linear programming problem in the case where the sample size is very large. We propose to apply the Benders Decomposition technique to the resulting LP for the regression case. In addition to this, other decomposition techniques like support clusters and bagging methods have been developed. Also, a very efficient data preprocessing method for SVM which is called  $\gamma$ -SVM has been developed. This method reduces the size of the training set and preserves the important data points which are high candidates as support vectors for defining the decision function.

Comparisons with other decomposition methods such as SVMTorch and SVMFu reveal that  $\gamma$ -SVM, support clusters, subsampling and bagging methods are more efficient in terms of CPU time.  $\gamma$ -SVM method outperforms SVMFu and SVMTorch for generalization error. In the case of support clusters, subsampling and bagging methods,

generalization error is close or higher than SVMFu and SVMTorch. Some information is lost for speeding up the algorithm. SVM regression has been applied to an option pricing model and prediction of S&P 500 daily return. Comparisons with multilayer perceptron and radial basis function networks is also presented.

# Chapter 1

## Introduction

### 1.1 Overview

The amount of data gathered in recent years has been exponentially increasing largely due to new technologies such as internet related ones: To name a standard example, the Web has billions of “visible” pages and hundreds of billions connected through the “deep Web,” while the average company has hundreds of thousands of web pages and is expected to gather hundreds of terabytes of customer data within the next few years (see also [61] for more examples). However, until now a very small portion of this data has been analyzed using data mining techniques, and even when this was the case, very simple machine learning methods have been used. An important reason that advanced machine learning methods have not been used in many cases is simply that it is often practically impossible to train these methods with such amounts of data. It is therefore an important challenge for advanced learning machines to develop methods for training with very large datasets (i.e. often millions of data points).

Much work has been done in the direction of speeding up (scaling up) some standard data mining methods. Provost and Kolluri [49] give a recent review of various approaches, mostly focusing on learning methods for finding rules and for training decision trees. The paper categorizes the approaches into three groups: designing fast algorithms, partitioning the data, and using relational representations. In the first group there are approaches such as building only “good but simple” learning methods such as one or two level only decision trees [50, 27], or for designing various algorithmic or programming optimizations [19, 41]. In the second group there are approaches like sampling (in various proposed ways) a small dataset from the initial large one and using only that for training [43], using small subsets of features [56], variations of online learning [35] such as Quinlan’s windowing method [51], or training many machines in parallel each using a small subset of the initial large dataset and then combining the results [12, 20] (see also the proceedings of the workshop on distributed data mining [32] and references therein). Finally, in the third group there are approaches which have more to do with forms of data storage and retrieval in databases (see for example [1, 2]).

The SVM algorithm developed by Vapnik [62] is based on statistical learning theory. In the classification case [14, 17, 44, 47], we try to find an optimal hyperplane that separates two classes. In order to find an optimal hyperplane, we need to minimize the norm of the vector  $w$ , which defines the separating hyperplane. This is equivalent to maximizing the margin between two classes.

At the same time, the recent development of a new family of learning machines, namely Support Vector Machines (SVMs) [14, 17, 62], whose training can be formulated as optimizing a quadratic programming (QP) problem with box constraints, has led to a

series of fast optimization methods for this type of QP problem [46, 44, 52].

Without any decomposition techniques, solving the SVM problem is very difficult if the number of training examples is large. Therefore, some decomposition techniques should be applied. We can divide decomposition techniques applied to SVM into groups. The first one is called matrix splitting. The following procedure is proposed by Vapnik [62] to solve the QP problem efficiently. First, the training data are divided into a number of portions. Each of the portions has a reasonable small number of data points; then we start solving the QP problem determined by the first portion of training data. There are two possibilities: either this portion of data cannot be separated by a hyperplane (in this case, the full set of training data as well cannot be separated [62]), or the optimal hyperplane for separating the first set of the training data is found and defined by the vector  $\Lambda_1$ . Specifically, among the coordinates of the vector  $\Lambda_1$ , some are equal to zero. Therefore some of the training data correspond to non-support vectors of this portion. After that, make a new set of training data containing the support vectors from the previous set and the vectors of the second portion that do not satisfy the separation constraints, where  $w$  (vector defining separation hyperplane) is determined by the previous solution,  $\Lambda_1$ . For this set a new QP problem is constructed and solved. Let  $\Lambda_2$  be the solution of this problem. We continue this process until a solution vector  $\Lambda_*$  covering all the portions of the training data is found or one finds that is impossible to separate the training data without error. The drawback of this procedure is that we do not know the SV set before solving the problem. Therefore, we choose the first portion of the data arbitrarily. This is called chunk that fits the memory. Another algorithm for making the solution procedure of the QP problem is the Advanced Working

Set Algorithm. The solution is to use only a subset of variables as a working set and optimize the problem with respect to them while freezing the other variables. For the pattern recognition case, this method is described in detail in [44]. First, the problem is extracted into two groups, the working set denoted by  $S_w$  and fixed set,  $S_f$ . Another technique is Sequential Minimum Optimization [46], which uses only two variables as a working set. An LP boosting algorithm which is a column generation technique and Bender's Decomposition technique can also be put on this group [33, 59].

In the second group of methods, we try to reduce the training data into a smaller group. This can be done by first clustering the data and then using the centroids of the resulting clusters as the representative of whole data [57]. In addition to the clustering method, some data points can be chosen randomly to represent the data [34]. Secondly, data can be divided into  $k$  batches randomly and each of the batches can be solved independently. Then all the SVs are combined and the SVM problem is solved again by using all SVs.

## 1.2 Research Objectives

The formulation of the support vector machine for classification and regression yields quadratic or linear programming problems. The size of the problem increases with the number of data points. Suppose we have a dataset with  $l$  points or observations. Then the matrix defining the QP problem has dimension  $l^2$ , where  $l$  could be as large as thousands or millions. Without any decomposition techniques, it is very hard to solve these problems. Decomposition algorithms divide the problem into two sets, the

active set and non active set. This idea has been used with all the techniques in QP formulation. For the LP formulation, a column generation technique has been proposed by Bennet [33]. Mangasarian has suggested a linear chunking algorithm. The SVM problem can also be formulated without any constraints. This gives an unconstrained QP problem which is easy to solve [37, 34].

In this study, we propose three algorithms for solving large scale SVM classification and regression problems. We apply the Benders decomposition technique to the regression problem. Also, we propose three pre-processing heuristic algorithms to identify the most important data points in the dataset for classification.

### 1.3 Scope of the Dissertation

Chapter 1 gives a brief introduction of SVM methods and decomposition techniques which have been applied to the SVM problem. Support vector machines for classification and regression with empirical and structural risk minimization is explained in chapter 2. Benders decomposition for SVMs is given in chapter 3. In chapter 4, support clusters, bagging method and data preprocessing algorithm will be discussed. Comparison of those method and an application to option pricing model is described in chapter 5. Conclusions and future work is discussed in chapter 6.

## Chapter 2

# Mathematical Formulation of Support Vector Machine

### 2.1 Empirical Risk Minimization

The task of learning from data can be formulated in the following way for a two class pattern classification problem. Suppose  $(x_1, y_1), (x_2, y_2), \dots, (x_l, y_l)$  are given such that  $x_i \in \mathbb{R}^d$ , and  $y_i \in \{-1, 1\}$  and are taken from an unknown distribution  $P(x, y)$ .

We are also given a set of decision functions

$$\{f_\lambda(x) : \lambda \in \Lambda\}, f_\lambda : \mathbb{R}^d \rightarrow \{-1, 1\}, \quad (2.1)$$

where  $\Lambda$  is a set of abstract parameters and  $f_\lambda(x), \{f_\lambda(x) : \lambda \in \Lambda\}$  are called *hypothesis* and *hypothesis space*, respectively.

We want to find a  $f_\lambda^*$  which provides the smallest possible value for the expected



risk which is defined as follows.

$$R(\lambda) = \int 1/2 |f_\lambda(x) - y| dP(x, y), \quad (2.2)$$

The problem is that the distribution function  $P(x, y)$  is unknown. Therefore, we can not compute the expected risk given in (2.2). On the other hand, we have examples  $(x_i, y_i)$  from the distribution  $P(x, y)$ . Hence, we compute a stochastic approximation of  $R(\lambda)$  which is called empirical risk,  $R_{emp}$ , and defined as follows:

$$R_{emp}(\lambda) = 1/2l \sum_{i=1}^l |f_\lambda(x_i) - y_i|. \quad (2.3)$$

As can be seen from above,  $R_{emp}(\lambda)$  is a fixed number given a particular choice of  $\lambda$  and training set  $\{x_i, y_i\}$ . The law of large numbers [62] guarantees that the empirical risk converges to the expected risk. So, one minimizes the empirical risk instead of the expected risk [14]. The idea behind minimizing the empirical risk is that if  $R_{emp}$  converges to the expected risk, then the minimum of  $R_{emp}$  may converge to the minimum of the expected risk. The quantity  $\frac{1}{2l} |f_\lambda(x) - y|$  is called loss. It can only take values 0 and 1. If  $f_\lambda(x)$  is equal to  $y$ , then the value of this quantity is 0, otherwise it is equal to 1.

The theory of uniform converge in probability provides bounds on the deviation of the empirical risk from the expected risk. A typical uniform VC ( Vapnik-Chernovenkis) bound, which holds with probability  $1 - \eta$ , has the following form. [13].

$$R(\lambda) \leq R_{emp}(\lambda) + \sqrt{(h(\ln 2l/h + 1) - \ln \eta/4)/l}, \quad (2.4)$$

where  $h$  is the VC dimension of  $f_\lambda$  and  $l$  is the number of examples.

We can make some observations from this inequality bound. As it can be seen from (2.4), the right hand side is an upper bound on the expected risk. It is independent of the distribution of the data,  $P(x, y)$ . The test data and training data are drawn independently according to some  $P(x, y)$ . It is not possible to compute the left-hand side of the bound in equation (2.4) [17]. Extensive study of the VC dimension can be found in [14, 17, 31].

## 2.2 Structural Risk Minimization

Structural risk minimization developed by Vapnik is an attempt to overcome the problem of choosing an appropriate VC dimension. One can see that a small value of the empirical risk does not necessarily imply a small value of expected risk. A different induction principle called structural risk minimization (SRM) was developed by Vapnik. The principle is based on the observation that in order to make the expected risk small, both sides of (2.4) should be small. Therefore, VC dimension and empirical risk should be minimized at the same time.

In order to do this, we need a nested structure of the hypothesis space such that  $H_1 \subset H_2 \subset H_3 \subset \dots, H_n \subset \dots$  with the property that  $h(n) \leq h(n+1)$ , where  $h(n)$  is the VC dimension of  $H(n)$  where  $n$  is the number of hypothesis spaces. Therefore we need to solve the following problem,

$$\min_{H_n} (R_{emp}(\lambda) + \sqrt{\frac{h(n)}{l}}). \quad (2.5)$$

The SRM principle is mathematically well founded. According to [54], it can be difficult to implement since the VC dimension of  $H_n$  may be difficult to compute and

even if one computes the VC dimension, it is not easy to solve the above problem. The SVM algorithm minimizes the bound on the VC dimension and the number of training errors at the same time.

## 2.3 Definitions

In this section, we will give the definitions of distance with respect to the  $L_p$  norm, margin and linear separability respectively. We will use these definitions for formulating the SVM problem. Our presentation follows Pedroso and Murata(1999) [45].

**Definition 1:** Given two parallel hyperplanes in  $R^n$ , the distance between these two hyperplanes is defined as

$$d_p(H_1, H_2) = \min_{x \in H_1, y \in H_2} \|x - y\|_p, \quad (2.6)$$

where  $w$  is a vector which is perpendicular the hyperplane.  $b_1, b_2$  are the biases.  $p$  is an arbitrary norm.  $H_1$  is described by  $w \cdot x + b_1 = 0$  and  $H_2$  is described by  $w \cdot x + b_2 = 0$  respectively.

Shifting both hyper-planes so that  $H_2$  passes through the origin, we obtain two hyper-planes separated by the same distance. Their evaluations are  $H'_1 : w \cdot x + b_1 - b_2 = 0$  and  $H'_2 : w \cdot x = 0$  respectively. If we choose  $y$  to be the origin, then the distance between these two hyper-planes is

$$d_p(H'_1, H'_2) = \min_{x \in H'_1} \|x\|_p \quad (2.7)$$

Hölder's inequality states that for conjugate norm of  $L_p$  and  $L_q$ , the following inequality holds:

$$\|x\|_p \cdot \|w\|_q \geq |x \cdot w| \quad (2.8)$$

where  $\frac{1}{p} + \frac{1}{q} = 1$ .

We will use the conjugate norm for defining the distance between two hyper-planes. Since the equation of  $H_1$  is  $w \cdot x + b_1 - b_2$ , we have that  $|w \cdot x| = |b_1 - b_2|$ . Therefore,  $\min_{x \in H_1} \|x\|_p \|w\|_q = |b_1 - b_2|$ . Using this formula, we can write the distance between the two hyper-planes as follows:

$$d_p(H_1, H_2) = \min_{x \in H_1} \|x\|_p = \frac{|b_1 - b_2|}{\|w\|_q}, \quad (2.9)$$

where  $q$  is the conjugate norm of  $p$ .

Consider two arbitrary points in  $\mathbb{R}^2$ .  $l_2$  norm distance is the traditional euclidean distance from one point to another. Distance can be measured by moving horizontally and vertically. The  $l_1$  norm is the summation of horizontal and vertical movements as in figure 2.1. The  $l_\infty$  norm distance is the biggest of the horizontal and vertical distances. As it can be seen from figure 2.1, for any two points  $x, y$  the relationship between distances in the three norms is as follows

$$\|x - y\|_\infty \leq \|x - y\|_2 \leq \|x - y\|_1. \quad (2.10)$$

Assume that we are given a set  $S$  of points  $x_i \in \mathbb{R}^n$  with each  $x_i$  belongs to either of two classes defined by  $y_i \in \{-1, 1\}$ . The objective is to find a hyperplane that divides  $S$  leaving all the points of the same class on the same side while maximizing the minimum distance between either of the two classes and the hyperplane [42].

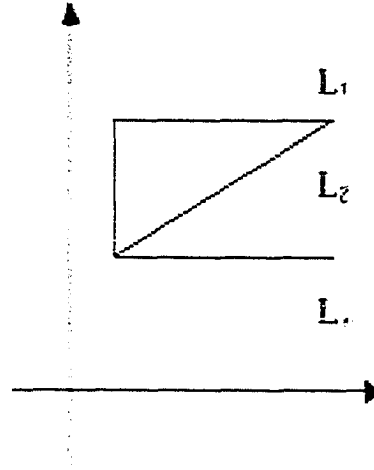


Figure 2.1:  $l_1, l_2, l_\infty$  norm distances

**Definition 2:** The set  $S$  is linearly separable if there exist  $w \in \mathbb{R}^n$  and  $b \in \mathbb{R}$  such that

$$\begin{aligned} w \cdot x_i + b &\geq 1, \text{ if } y_i = 1 \\ w \cdot x_i + b &\leq -1, \text{ if } y_i = -1. \end{aligned} \quad (2.11)$$

Let us explain this by using the AND function with its truth table given in table 2.1.

Figure 2.2 shows the input space. As can be seen from figure 2.2, we can find a hyperplane that separates class 1 and class -1 respectively. In the nonlinear separable case as in the case of the XOR problem (see table 2.2 and figure 2.3), there is no hyperplane that separates class 1 and class -1 linearly.

If  $w$  and  $b$  are scaled by the same quantity, then the decision surface given by the above equation is unchanged. In order to remove this redundancy and to make the

Table 2.1: AND function truth table

$x_1$	$x_2$	$y$
1	1	1
1	-1	-1
-1	1	-1
-1	-1	-1

Table 2.2: XOR function truth table

$x_1$	$x_2$	$y$
1	1	-1
1	-1	1
-1	1	1
-1	-1	-1

decision surface corresponding to one unique pair  $(w, b)$ , the following constraint is imposed:

$$\min_{i=1,2,\dots,l} |w \cdot x_i + b| = 1. \quad (2.12)$$

Using the above equation, we can obtain the canonical representation which has VC dimension  $n + 1$ . In the case of linear separability, the separating hyperplane satisfies the following:

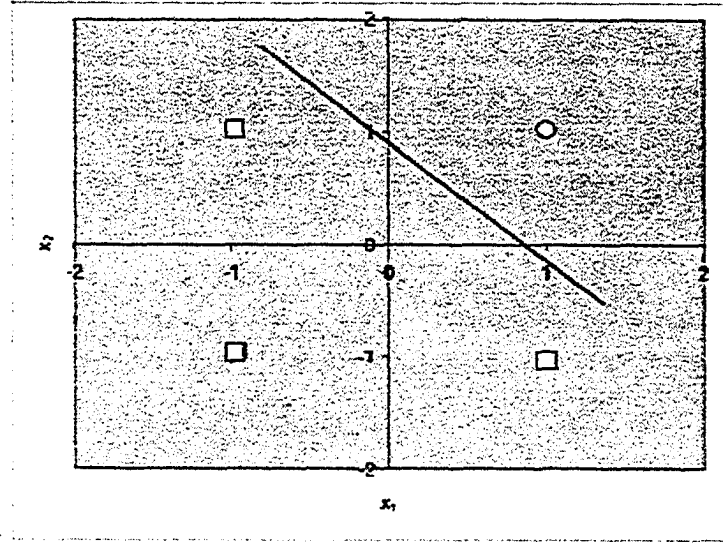


Figure 2.2: Linear separability of AND function

$$y_i \cdot (w \cdot x_i + b) \geq 1, i = 1, 2, \dots, l. \quad (2.13)$$

The distance from a point  $x$  to the hyperplane associated to the pair  $(w, b)$  is

$$d(x; w, b) = \frac{|w \cdot x + b|}{\|w\|} \quad (2.14)$$

According to the normalization that we made in 2.12, the distance between the canonical hyperplane and  $x$  is  $\frac{1}{\|w\|}$ .

## 2.4 SVM formulation by using $l_1$ , $l_2$ , and $L_\infty$ norms

In this section, Support vector machines for classification formulation will be explained for different norms. The problem is either a QP problem or an LP problem depending

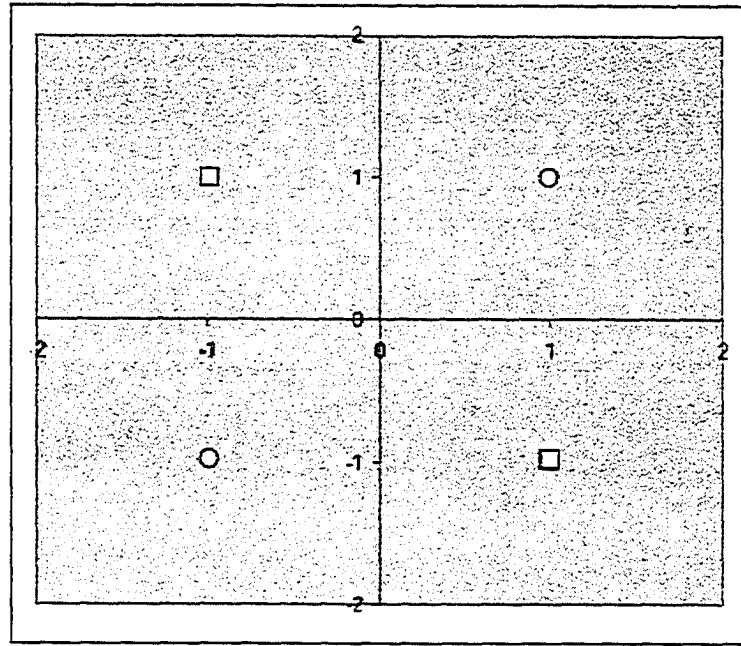


Figure 2.3: Linearly non-separable Case

on the norm we choose.

### 2.4.1 $l_2$ Formulation

#### 2.4.1.1 Linearly Separable Case

The goal of the SVM is to find, among all the hyperplanes that correctly classify the data the one with the minimum norm, or minimum  $\|w\|^2$ . As it can be seen from 2.4, minimizing  $\|w\|^2$  is equivalent to finding a separating hyperplane for which the distance between two classes is maximized.

In order to construct the maximum margin, or optimal separating hyperplane, we need to correctly classify the vectors  $x_i$  of the training set into two different classes, where  $y_i \in \{-1, 1\}$  by solving the following problem



$$\min_{w,b} \phi(w) = \frac{1}{2} \|w\|^2$$

subject to (2.15)

$$y_i \cdot (w \cdot x_i + b) \geq 1$$

$$i = 1, 2, \dots, l, \quad y_i = \pm 1.$$

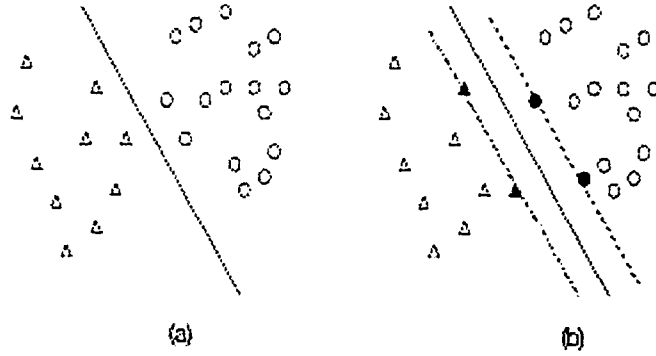


Figure 2.4: Separating hyper-plane and optimal separating hyper-plane. Both solid lines in (a) and (b) separate the two identical sets described by circles and triangles. But the solid line in (b) leaves the closest points (filled circles and triangles) at the maximum distance. The distance between dashed lines in (b) gives the maximum margin.

We need to construct a lagrangian function of (2.15). There are two advantages of doing this [14]. The constraints of (2.15) will be replaced by constraints on the lagrangian multipliers themselves, which will be much easier to solve. In addition to this, the training data will only appear in the form of dot products between vectors. This will allow us to generalize the procedure to the nonlinear case by using the concept

of the kernel function [62].

If we denote by  $\Lambda = (\lambda_1, \lambda_2, \dots, \lambda_l)$  the  $l$  nonnegative lagrange multipliers associated with the constraints, the solution to the problem is equivalent to determining the saddle point of the lagrangian function

$$L(w, b, \Lambda) = \frac{1}{2} \|w\|^2 - \sum_{i=1}^l \lambda_i \cdot [y_i \cdot (w \cdot x_i + b) - 1]. \quad (2.16)$$

By using saddle point optimality conditions ( first derivatives with respect to  $w$  and  $b$ , should be equal to zero at optimal point) we have

$$\begin{aligned} \frac{\partial L(w, b, \Lambda)}{\partial w} &= w - \sum_{i=1}^l \lambda_i \cdot y_i \cdot x_i = 0 \\ \frac{\partial L(w, b, \Lambda)}{\partial b} &= \sum_{i=1}^l \lambda_i \cdot y_i = 0. \end{aligned} \quad (2.17)$$

So, at the optimal point,

$$w^* = \sum_{i=1}^l \lambda_i^* \cdot y_i \cdot x_i. \quad (2.18)$$

Substituting (2.18) in the lagrangian function, we obtain the following expression

$$\begin{aligned} \max F(\Lambda) &= \sum_{i=1}^l \lambda_i - \frac{1}{2} \sum_{i=1}^l \sum_{j=1}^l \lambda_i \cdot \lambda_j \cdot y_i \cdot y_j \cdot x_i \cdot x_j \\ &\quad \text{subject to} \\ &\quad \sum_{i=1}^l \lambda_i \cdot y_i = 0 \\ &\quad \lambda_i \geq 0, i = 1, 2, \dots, l. \end{aligned} \quad (2.19)$$

The Karush-Kuhn-Tucker (KKT) optimality conditions play a crucial role in determining the optimal value of the bias ( $b$ ) and  $w$  respectively. By using the KKT conditions for (2.15) , which are shown below, we can compute  $w$  and  $b$ .

$$\begin{aligned}
\frac{\partial L(w, b, \Lambda)}{\partial w} &= w - \sum_{i=1}^l \lambda_i \cdot y_i \cdot x_i = 0 \\
\frac{\partial L(w, b, \Lambda)}{\partial b} &= - \sum_{i=1}^l \lambda_i \cdot y_i = 0 \\
1 - y_i \cdot (w \cdot x_i + b) &\leq 0 \\
\lambda_i &\geq 0 \\
\lambda_i^* \cdot [y_i \cdot (w^* \cdot x_i + b^*) - 1] &= 0.
\end{aligned} \tag{2.20}$$

Specifically, the complementary condition gives  $b$ . At the optimal solution, if  $\lambda_i > 0$ , then the second part of the complementary slackness equation should be zero. Therefore,

$$\begin{aligned}
b^* &= y_i - w^* \cdot x_i \\
w^* &= \sum_{i=1}^l \lambda_i^* \cdot y_i \cdot x_i
\end{aligned} \tag{2.21}$$

Complementary conditions imply that  $\lambda^* > 0$  when the constraints in (2.20) are active. Therefore, the vectors for which  $\lambda^* > 0$  are called support vectors. So, the decision function becomes,

$$f(x) = \text{sign}\left(\sum_{i=1}^l \lambda_i^* \cdot y_i (x \cdot x_i) + b^*\right). \tag{2.22}$$

#### 2.4.1.2 Linearly non-separable case (Soft Margin optimal hyperplane)

Next, we consider the case of linearly non-separable data. In this case, the constraint  $y_i \cdot (w \cdot x_i + b) \geq 1$  is not satisfied for some points. In order to overcome this difficulty, one introduces a new set of variables,  $\{\xi\}_{i=1}^l$ , that measure the amount of violation of

the constraints. Then we try to minimize  $\|w\|^2$ , paying a penalty proportional to the amount of constraint violation. Specifically, we solve the following problem:

$$\begin{aligned}
\min_{w, b, \xi} \phi(w, \xi) &= \frac{1}{2} \|w\|^2 + C \left( \sum_{i=1}^l \xi_i \right)^k \\
&\text{subject to} \\
y_i \cdot (w \cdot x_i + b) &\geq 1 - \xi_i \\
\xi_i &\geq 0 \\
i &= 1, 2, \dots, l,
\end{aligned} \tag{2.23}$$

where parameters  $C$  and  $k$  are non-negative and determined before the training phase. Minimizing the first term in (2.23) corresponds to minimizing the VC dimension in equation (2.4). Minimizing the second term controls the empirical risk (minimizes the misclassification). In order to solve problem 2.23, we need to construct the lagrangian function,

$$\begin{aligned}
L(w, b, \xi, \Lambda, \Gamma) &= \frac{1}{2} \|w\|^2 - \sum_{i=1}^l \lambda_i \cdot [y_i \cdot (w \cdot x_i + b) - 1 + \xi_i] \\
&\quad - \sum_{i=1}^l \gamma_i \cdot \xi_i + C \cdot \left( \sum_{i=1}^l \xi_i \right)^k,
\end{aligned} \tag{2.24}$$

where the non-negative multipliers  $\Lambda = (\lambda_1, \lambda_2, \dots, \lambda_l)$  and  $\Gamma = (\gamma_1, \gamma_2, \dots, \gamma_l)$  are associated with the constraints in 2.23. The solution is determined by the saddle point optimality conditions that are shown below:

$$\frac{\partial L}{\partial w} = w - \sum_{i=1}^l \lambda_i \cdot y_i \cdot x_i = 0$$

$$\frac{\partial L}{\partial b} = - \sum_{i=1}^l \lambda_i \cdot y_i = 0 \quad (2.25)$$

$$\frac{\partial L}{\partial \xi_i} = \left\{ \begin{array}{l} k \cdot C \left( \sum_{i=1}^l \xi_i \right)^{k-1} - \lambda_i - \gamma_i = 0 \quad k > 1 \\ C - \lambda_i - \gamma_i = 0, \quad k = 1 \end{array} \right\}.$$

At the optimal solution, we obtain the optimal values of  $w^*$  and  $b^*$ . Specifically,

$$w^* = \sum_{i=1}^l \lambda_i^* \cdot y_i \cdot x_i. \quad (2.26)$$

By substituting the above equations into the lagrangian function, we obtain the dual problem as shown below:

$$\begin{aligned} \max \quad & \sum_{i=1}^l \lambda_i - \frac{1}{2} \sum_{i=1}^l \sum_{j=1}^l \lambda_i \cdot \lambda_j \cdot y_i \cdot y_j \cdot x_i \cdot x_j \\ \text{subject to} \quad & \sum_{i=1}^l \lambda_i \cdot y_i = 0 \\ & \lambda_i \leq C, i = 1, 2, \dots, l \\ & \lambda_i \geq 0, i = 1, 2, \dots, l. \end{aligned} \quad (2.27)$$

In the linearly non-separable case, one needs to solve this problem. This is a quadratic programming problem.

We will explain which techniques are available and which are the best selections. But before we do this, we will continue to explain SVMs with non-linear decision surfaces.

### 2.4.1.3 Non-linear case

If the decision surface is non-linear or more complex, we need to map the input variable  $x$  into a higher dimensional feature space. This is proposed by Vapnik [62]. Specifically,

$$x \rightarrow \phi(x) = (a_1\phi_1(x), a_2\phi_2(x), \dots, a_n\phi_n(x), \dots), \quad (2.28)$$

where  $\{a_n\}_{n=1}^{\infty}$  are some real numbers and  $\phi_n(x)$  are some real functions (basis functions).

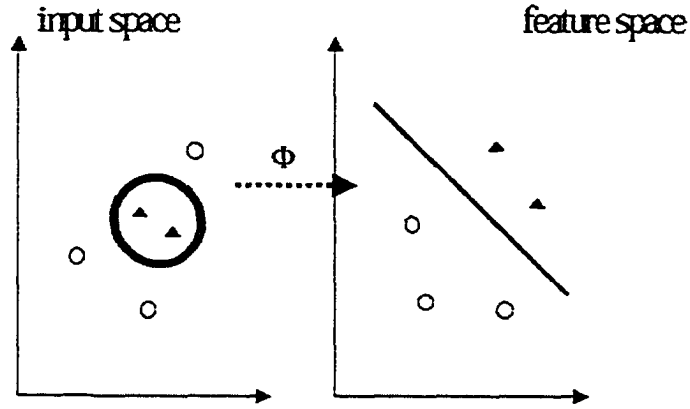


Figure 2.5: Mapping input space to feature space

If we map the input space to a feature space, whose dimension is infinity, then it is highly probable that we will obtain a linear surface that separates the data into two groups in the feature space. Then, the discriminant function is defined as follows

$$f(x) = \text{sign}(\phi(x) \cdot w^* + b^*) \Rightarrow f(x) = \text{sign}\left(\sum_{i=1}^l \lambda^* \cdot y_i \cdot \phi(x) \cdot \phi(x_i) + b^*\right) \quad (2.29)$$

As it can be seen from the above equation, we need to compute dot products of the form  $\phi(x) \cdot \phi(y)$ . It is convenient to replace these dot products through a kernel function denoted by  $K$ . Specifically,

$$K(x, y) = \phi(x) \cdot \phi(y) = \sum_{n=1}^{\infty} a_n^2 \cdot \phi(x) \cdot \phi(y). \quad (2.30)$$

Using this, the solution of SVM has the following form:

$$f(x) = \text{sign}\left(\sum_{i=1}^l \lambda^* \cdot y_i \cdot K(x, x_i) + b^*\right). \quad (2.31)$$

If we replace the dot product of the inputs with the kernel function in the linearly non separable case, we obtain a similar problem like the one we have in the linear case. Specifically,

$$\begin{aligned} \max \sum_{i=1}^l \lambda_i - \frac{1}{2} \sum_{i=1}^l \sum_{j=1}^l \lambda_i \cdot \lambda_j \cdot y_i \cdot y_j \cdot K(x_i, x_j) \\ \text{subject to} \\ \sum_{i=1}^l \lambda_i \cdot y_i = 0 \\ 0 \leq \lambda_i \leq C, \quad i = 1, 2, \dots, l. \end{aligned} \quad (2.32)$$

Therefore we obtain a quadratic programming problem. The difference between the linear case and the non-linear case is that in the case of non-linear separability, we try to find a separating hyperplane in the feature space. The next step is to solve this problem with an efficient and robust algorithm.

### 2.4.2 $l_1$ norm formulation

If we use the  $l_1$  norm to compute the distance between the training points and separating hyperplane, the problem can be formulated as a linear programming problem.

If we impose  $w \cdot x_i + b = \pm 1$ , then we have two hyperplanes,  $H^+ : w \cdot x_i + b = 1$  and  $H^- : w \cdot x_i + b = -1$ . The distance between them by using the  $l_1$  norm (see section 2.3) is given by,

$$d_1(H^+, H^-) = \frac{|(b+1) - (b-1)|}{\|w\|_\infty} = \frac{2}{\max_j |w_j|}, \quad j = 1, 2, \dots, d. \quad (2.33)$$

Therefore, we want to solve the following problem:

$$\min_{w, b} \max_j |w_j|. \quad (2.34)$$

We can convert this to a linear programming problem by introducing a new auxiliary variable  $a$  and adding  $a \geq w_j$  and  $a \geq -w_j$ . Then we obtain the following LP problem:

$$\begin{aligned} & \min_{w, b, a} a \\ & \text{subject to} \\ & y_i \cdot (w \cdot x_i + b) \geq 1, i = 1, 2, \dots, l \\ & a \geq w_j \\ & a \geq -w_j, \quad j = 1, 2, \dots, d. \end{aligned} \quad (2.35)$$

The support vectors are the training data points for which  $y_i \cdot (w \cdot x_i + b) \geq 1$  is active. We have explained the linear separable case. If the constraints are not satisfied,



we can use the soft margin constraints and include a penalty in the objective function. By doing that, we penalize the objective function. Soft margin formulation is shown below:

$$\begin{aligned}
& \min_{w,b,a,\xi} a + C \cdot \sum_{i=1}^l \xi_i \\
& \text{subject to} \\
& y_i \cdot (w \cdot x_i + b) + \xi_i \geq 1, \quad i = 1, 2, \dots, l \\
& a \geq w_j \\
& a \geq -w_j, \quad j = 1, 2, \dots, d,
\end{aligned} \tag{2.36}$$

where  $C$  is a positive number and is chosen by the user. So far, linearly separable and linearly non-separable cases have been explained. In real-life problems, we have the nonlinear case. In order to solve real-life problems, we need to map the input space to feature space by using a kernel function.

Consider the following AND function that is given in table 2.1. Let us look at the graph of this function again. We want to maximize the distance between these classes by using the  $l_1$  norm. Since we have the linearly separable case, we use equation (2.35) to formulate the problem.

The following problem has to be solved to find the optimal hyperplane which is also called decision or discriminant function:

$$\begin{aligned}
& \min a \\
& \text{subject to}
\end{aligned} \tag{2.37}$$

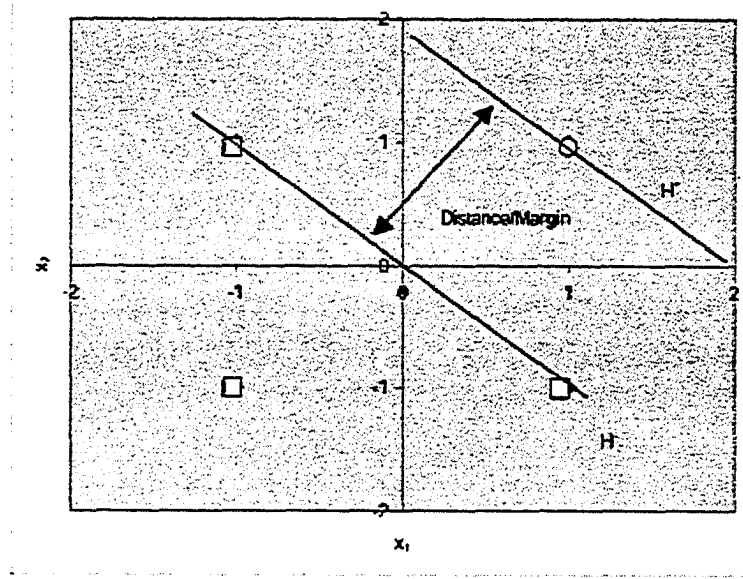


Figure 2.6: Maximum margin separation

$$w_1 + w_2 + b \geq 1$$

$$-w_1 + w_2 - b \geq 1$$

$$w_1 - w_2 - b \geq 1$$

$$-w_1 - w_2 - b \geq 1$$

$$w_1 + a \geq 0$$

$$-w_1 + a \geq 0$$

$$w_2 + a \geq 0$$

$$-w_2 + a \geq 0$$

$$w_1, w_2, b, \text{ any sign and } a \geq 0$$

### 2.4.3 $l_\infty$ norm formulation

If  $l_\infty$  norm is used to determine the distance between the training points and the separating hyperplane, the problem can also be formulated as a linear programming problem. As for the  $l_\infty$  formulation, we impose that  $w \cdot x_i + b = \pm 1$  for support vectors. Then, the distance between the two hyper-planes for each class is,

$$d_1(H^+, H^-) = \frac{|(1 - b) - (-1 - b)|}{|w|_1} = \frac{2}{\sum_{j=1}^d |w_j|}. \quad (2.38)$$

We want to maximize this value, which is equivalent to,

$$\min_{w,b} \sum_{j=1}^d |w_j|. \quad (2.39)$$

Equation 2.39 can be transformed to a linear programming problem by adding auxiliary variables  $a_j$  for each  $w_j$ , and constraints  $a_j \geq w_j$  and  $a_j \geq -w_j \ \forall j = 1, 2, \dots, d$ . Then, the problem becomes,

$$\begin{aligned} & \min_{w,b,a} \sum_{j=1}^d a_j \\ & \text{subject to} \\ & y_i \cdot (w \cdot x_i + b) \geq 1, \quad i = 1, 2, \dots, l \\ & a_j \geq w_j, \quad j = 1, 2, \dots, d \\ & a_j \geq -w_j, \quad j = 1, 2, \dots, d \\ & b \in \mathbb{R}, w \in \mathbb{R}^d \text{ and } a \geq 0. \end{aligned} \quad (2.40)$$

In the problem 2.40, we assume that each class is linearly separable. If this is not the case, then we use soft margin formulation as we did in  $l_1$  and  $l_2$  case, respectively.

$$\min_{w, b, a} \sum_{j=1}^d a_j + C \cdot \left( \sum_{i=1}^l \xi_i \right)$$

subject to

$$y_i \cdot (w \cdot x_i + b) + \xi_i \geq 1, \quad i = 1, 2, \dots, l$$

$$a_j \geq w_j, \quad j = 1, 2, \dots, d \quad (2.41)$$

$$a_j \geq -w_j, \quad j = 1, 2, \dots, d$$

$$b \in \mathbb{R}, w \in \mathbb{R}^d, \xi \geq 0 \text{ and } a \geq 0$$

In the  $l_\infty$  case, we have a similar problem like the one we had in the  $l_1$  case. In the next section, we will discuss the  $\varepsilon$ -insensitive support vector regression formulation by using  $l_1$  and  $l_2$  norm, respectively. The training problem for the AND function can be formulated as follows:

$$\min a_1 + a_2$$

$$\text{subject to} \quad (2.42)$$

$$w_1 + w_2 + b \geq 1$$

$$-w_1 + w_2 - b \geq 1$$

$$w_1 - w_2 - b \geq 1$$

$$-w_1 - w_2 - b \geq 1$$

$$w_1 + a_1 \geq 0$$

$$\begin{aligned}
-w_1 + a_1 &\geq 0 \\
w_2 + a_2 &\geq 0 \\
-w_2 + a_2 &\geq 0
\end{aligned}
\tag{2.43}$$

where  $w_1, w_2, b$  are unrestricted in sign and  $a_1, a_2 \geq 0$ .

#### 2.4.4 Kernelization of Linear Programming Formulation

Instead of using the  $l_2$  norm, if we use the infinity norm for measuring the margin which is the distance between the supporting hyperplanes for each class, we have the following formulation:

$$\begin{aligned}
\min_{w, b, \xi} \|w\|_1 + C \sum_{i=1}^l \xi_i \\
\text{subject to}
\end{aligned}
\tag{2.44}$$

$$y_i(x_i \cdot w + b) + \xi_i \geq 1$$

$$z_i \geq 0 \quad i = 1, \dots, l, \quad \text{and } y_i = \pm 1.$$

To solve a nonlinear discriminants problem, we need to map this problem into the feature space. This can be accomplished through the use of the kernel function. From the original SVM formulation, we have come up with the decision function

$f(x) = \text{sign}(\sum_i y_i \cdot \lambda_i \cdot K(x_i, x_j) + b)$ . By using this, we obtain the following formulation of equation (2.44).

$$\min_{\lambda, b, \xi} \|\lambda\|_1 + C \sum_{i=1}^l \xi_i \quad (2.45)$$

subject to

$$y_i \left( \sum_{j=1}^l y_j \cdot \lambda_j \cdot K(x_i, x_j) + b \right) + \xi_i \geq 1$$

$$z_i, \lambda_i \geq 0 \quad i = 1, \dots, l$$

By minimizing  $\|\lambda\|_1 = \sum_{i=1}^l \lambda_i$  we obtain a solution which is sparse [6].

## 2.5 Support vector regression formulation by using $l_1$ , $l_2$ norms

### 2.5.1 $l_2$ norm formulation

We follow the same structures that we use in the SVM case for classification. First, we will look at the linearly separable case, then soft margin support vector regression will be explained and finally, the non-linear case will be given.

The  $\varepsilon$ -insensitive support vector regression will be explained. In the  $\varepsilon$ -insensitive support vector regression, our goal is to find a function  $f(x)$  that has  $\varepsilon$  deviation from the actually obtained target  $y_i$  for all training data and at the same time is as flat as possible. Suppose  $f(x)$  takes the following form:

$$f(x) = w \cdot x + b, \quad w, x \in R^d, b \in R. \quad (2.46)$$

If we have a  $w$  with small norm, then we can say that  $f$  is flat. One way to make  $f$  flat is to minimize  $\|w\|^2$  (euclidean norm) [54]. Specifically,

$$\begin{aligned} & \min \frac{1}{2} \|w\|^2 \\ & \text{subject to} \end{aligned} \tag{2.47}$$

$$y_i - w \cdot x_i - b \leq \varepsilon$$

$$w \cdot x_i + b - y_i \leq \varepsilon$$

$$i = 1, 2, \dots, l.$$

$$(2.48)$$

One has to solve this problem in order to obtain an  $\varepsilon$ -insensitive SV regression solution. Usually, we need to allow for some errors. We introduce slack variables  $\xi_i, \xi_i^*$  to cope with this situation. This case is called soft margin formulation. Specifically,

$$\begin{aligned} & \min \frac{1}{2} \|w\|^2 + C \sum_{i=1}^l (\xi_i + \xi_i^*) \\ & \text{subject to} \end{aligned} \tag{2.49}$$

$$y_i - w \cdot x_i - b \leq \varepsilon + \xi_i$$

$$w \cdot x_i + b - y_i \leq \varepsilon + \xi_i^*$$

$$\xi_i, \xi_i^* \geq 0, : C > 0$$

$$i = 1, 2, \dots, l,$$

$$(2.50)$$

where  $C$  determines the trade-off between the flatness of  $f(x)$  and the amount up to which deviations larger than  $\varepsilon$  are tolerated. This is called loss function  $|\xi|_\varepsilon$  and is

described by,

$$|\xi| = \begin{cases} 0 & \text{if } |\xi| \leq \varepsilon \\ |\xi| - \varepsilon & \text{if } |\xi| > \varepsilon \end{cases}. \quad (2.51)$$

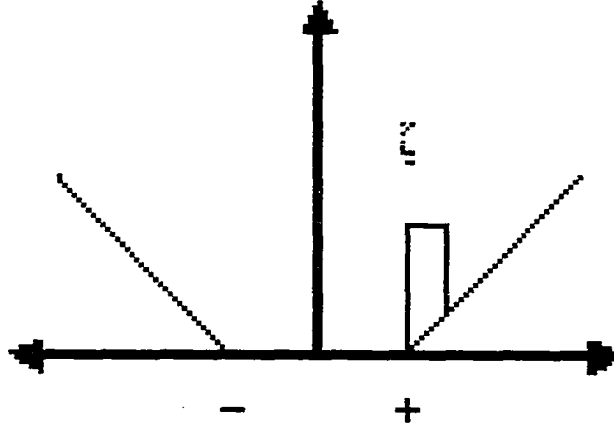


Figure 2.7: The soft margin lost setting corresponds for a linear SV Machine [54]

Figure 2.8 illustrates the case where some of the data are inside the  $\varepsilon$ -insensitive zone and some outside.

Let us look at the lagrangian function of the above problem. Obtaining the lagrangian function will help us to formulate the dual problem, which will give us quadratic programming problem formulation.

We will construct the dual problem. The reason is that solving the primal problem is difficult due to many variables. If we use the dual problem, we can get rid of some of these variables and the size of the problem becomes smaller. The lagrangian is as follows:

$$L = \frac{1}{2} \|w\|^2 + C \cdot \sum_{i=1}^l (\xi_i + \xi_i^*) - \sum_{i=1}^l \lambda_i \cdot (\varepsilon + \xi_i - y_i + w \cdot x_i + b) \quad (2.52)$$



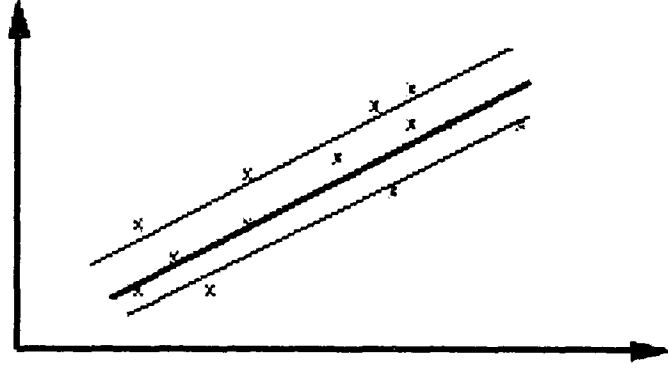


Figure 2.8:  $\varepsilon$ -insensitive case

$$\begin{aligned}
 & - \sum_{i=1}^l \lambda_i^* \cdot (\varepsilon + \xi_i^* + y_i - w \cdot x_i - b) - \sum_{i=1}^l (\eta_i \cdot \xi_i + \eta_i^* \cdot \xi_i^*) \\
 & \lambda_i, \lambda_i^*, \eta_i, \eta_i^* \geq 0, \quad i = 1, 2, \dots, l
 \end{aligned}$$

At the optimal solution, the first derivative of lagrangian function with respect to  $w, b, \xi_i, \xi_i^*$  should be zero,

$$\begin{aligned}
 \frac{\partial L}{\partial w} &= w - \sum_{i=1}^l (\lambda_i - \lambda_i^*) \cdot x_i = 0 \\
 \frac{\partial L}{\partial b} &= \sum_{i=1}^l (\lambda_i^* - \lambda_i) = 0 \\
 \frac{\partial L}{\partial \xi_i} &= C - \lambda_i - \eta_i = 0 \\
 \frac{\partial L}{\partial \xi_i^*} &= C - \lambda_i^* - \eta_i^* = 0.
 \end{aligned} \tag{2.53}$$

We obtain the dual problem by substituting (2.53) into (2.52). Then the dual problem is as follows

$$\begin{aligned}
& \max -\frac{1}{2} \sum_{i=1}^l \sum_{j=1}^l (\lambda_i - \lambda_i^*) \cdot (\lambda_j - \lambda_j^*) \cdot x_i \cdot x_j \\
& \quad -\varepsilon \cdot \sum_{i=1}^l (\lambda_i + \lambda_i^*) + \sum_{i=1}^l y_i \cdot (\lambda_i - \lambda_i^*) \\
& \quad \text{subject to} \\
& \quad \sum_{i=1}^l (\lambda_i - \lambda_i^*) = 0 \\
& \quad \lambda_i, \lambda_i^* \in (0, C)
\end{aligned} \tag{2.54}$$

After finding the optimal solution of this problem, we can get the optimal  $w$  and  $b$  by using the KKT conditions of the primal problem. Then,

$$\begin{aligned}
w^* &= \sum_{i=1}^l (\lambda_i - \lambda_i^*) \cdot x_i, \text{ and} \\
f(x) &= \sum_{i=1}^l (\lambda_i - \lambda_i^*) \cdot x_i \cdot x + b^*.
\end{aligned} \tag{2.55}$$

We compute the optimal value of  $b$  from the KKT optimality conditions. The complementary conditions of the problem are shown below:

$$\begin{aligned}
\lambda_i \cdot (\varepsilon + \xi_i - y_i + w \cdot x_i + b) &= 0 \\
\lambda_i^* \cdot (\varepsilon + \xi_i^* + y_i - w \cdot x_i - b) &= 0 \\
(C - \lambda_i) \cdot \xi_i &= 0 \\
(C - \lambda_i^*) \cdot \xi_i^* &= 0.
\end{aligned} \tag{2.56}$$

One can make some useful conclusion from the above equation [54]. First of all, only samples  $(x_i, y_i)$  with corresponding  $\lambda_i = C$  lie outside the  $\varepsilon$ -insensitive tube around  $f$ .

The set of dual variables can never be non-zero at the same time, i.e.  $\lambda_i \cdot \lambda_i^* = 0$ . If  $\lambda_i$  is non-zero, then  $\lambda_i^*$  is zero, and vice versa. Finally, if  $\lambda_i \in (0, C)$ , then the corresponding  $\xi_i$  is zero. Therefore,  $b$  can be computed by using

$$b^* = y_i - w^* \cdot x_i - \varepsilon \text{ for } \lambda_i \in (0, C) \quad (2.57)$$

$$b^* = y_i - w^* \cdot x_i + \varepsilon \text{ for } \lambda_i^* \in (0, C).$$

Let us look at the non-linear case briefly. First of all, we need to map the input space into the feature space and try to find a hyperplane in the feature space. We can accomplish that by using the kernel function trick as we did in the classification case. After this process, we use the standard support vector machine algorithm in the feature space. Specifically,

$$k(x, y) = \phi(x) \cdot \phi(y), \quad (2.58)$$

where  $k$  is a kernel function.

Since, in equation (2.58) the dot product of inputs is used, we need to perform this operation in the feature space by using kernel functions. If we can find a function that performs this operation, then we obtain a similar quadratic programming problem in the feature space as in (2.54) by replacing the dot product of  $x_i, x_j$  by  $k(x_i, x_j)$ . Therefore, problem (2.54) becomes

$$\begin{aligned} \max -\frac{1}{2} \sum_{i=1}^l \sum_{j=1}^l (\lambda_i - \lambda_i^*) \cdot (\lambda_j - \lambda_j^*) \cdot K(x_i, x_j) - \\ \varepsilon \cdot \sum_{i=1}^l (\lambda_i + \lambda_i^*) + \sum_{i=1}^l y_i \cdot (\lambda_i - \lambda_i^*) \\ \text{subject to} \end{aligned} \quad (2.59)$$

$$\sum_{i=1}^l (\lambda_i - \lambda_i^*) = 0$$

$$\lambda_i, \lambda_i^* \in (0, C).$$

At the optimal solution, we obtain

$$w^* = \sum_{i=1}^l (\lambda_i - \lambda_i^*) \cdot k(x_i, x_i), \text{ and} \quad (2.60)$$

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

The main difference between the linear and non-linear case is that in the non-linear case  $w$  is not given explicitly anymore. On the other hand, by using dot products, it is defined uniquely[54]. Also, we work in the feature space not in the input space.

### 2.5.2 $l_1$ norm formulation

The problem that arises in SVM regression case can be formulated as follows:

$$\min \sum_{i=1}^l (\alpha_i + \alpha_i^*) + C \cdot \sum_{i=1}^l (\xi_i + \xi_i^*) \quad (2.61)$$

subject to

$$y_i - \sum_{j=1}^l (\alpha_j - \alpha_j^*) \cdot k(x_j, x_i) - b \leq \varepsilon + \xi_i$$

$$\sum_{j=1}^l (\alpha_j - \alpha_j^*) \cdot k(x_j, x_i) + b - y_i \leq \varepsilon + \xi_i^*$$

$$\alpha_i, \alpha_i^*, \xi_i, \xi_i^* \geq 0.$$

By rearranging the known and unknown variables and the constraints in (2.61), we obtain

$$\min \sum_{i=1}^l (\alpha_i + \alpha_i^*) + C \cdot \sum_{i=1}^l (\xi_i + \xi_i^*) \quad (2.62)$$

subject to

$$\begin{aligned} \sum_{j=1}^l (\alpha_j - \alpha_j^*) \cdot k(x_j, x_i) + b - \xi_i^* &\leq \varepsilon + y_i \\ -\sum_{j=1}^l (\alpha_j - \alpha_j^*) \cdot k(x_j, x_i) - b - \xi_i &\leq \varepsilon - y_i \\ \alpha_i, \alpha_i^*, \xi_i, \xi_i^* &\geq 0. \end{aligned}$$

We solve problem (2.61) or problem (2.62) with respect to  $\alpha, \alpha^*, \xi, \xi^*$  and  $b$ .  $C$  is a tuning parameter. Generally, this is very large problem since we have  $2l$  constraints and  $4l+1$  variables where  $l$  is the number of observations. When the number of observations gets very large, we cannot solve the problem by the simplex method. In order to see the structure of the problem, we formulate the problem in matrix-vector notation by defining the following variables.

$$\alpha = [\alpha_i, \alpha_i^*, b]^T, i = 1, 2, \dots, l,$$

$$\Xi = [\xi, \xi^*]^T,$$

$$y = [y_1, y_2, \dots, y_l]^T, \quad (2.63)$$

$$f^T = [1, 1, \dots, 1, 0] \in \mathbb{R}^{1 \times (l+1)},$$

$$K = (k(x_i, x_j)) \text{ kernel matrix},$$

Then, the following problem is obtained:

$$\min f^T \cdot \alpha + C \cdot e^T \cdot \Xi \quad (2.64)$$

subject to

$$F \cdot \alpha + D \cdot \Xi \leq d$$

$$\Xi \geq 0, \text{ and } \alpha \text{ unrestricted}$$

where,

$$F = \begin{bmatrix} K & -K & 1 \\ -K & K & -1 \end{bmatrix},$$

*D is a diagonal matrix,*

$$d = [\varepsilon + y_1, \dots, \varepsilon + y_i, \varepsilon - y_1, \dots, \varepsilon - y_l]$$

Problem (2.64) cannot be solved efficiently without a decomposition approach since the problem is large scale in the case of a large sample. In quadratic programming formulation, several decomposition approaches are proposed and they are very efficient in terms of speed [38, 54, 62]. For the linear case, linear chunking or successive over relaxation can be used[11]. In the next chapter, we develop a decomposition for  $l_1$  regression based on Benders decomposition algorithm.

## Chapter 3

# Benders' Decomposition Technique

### 3.1 Introduction

Many large scale linear programming problems exhibit a block diagonal or l-shaped structure that makes to apply decomposition techniques such as Benders' Decomposition [5] or its dual Dantzig-Wolfe decomposition [18, 4]. Decomposition techniques have been applied to mixed-integer, stochastic programming , bilinear matrix inequalities, vehicle routing and facility location problems [40]. These problems have special structures so that they can be divided into sub problems which are easy to solve. Support vector machines for classification and regression problems lead to a linear programming problem which can be solved by using Benders decomposition technique.

### 3.2 Benders' Decomposition Technique for Support Vector Machines

Benders' decomposition will be explained and applied to problem (2.64). The SVM regression problem can be formulated as an LP problem by using the  $l_1$  norm. Problem (2.64) has a block diagonal structure. Therefore, we can use a decomposition technique such as Bender's or Dantzing-Wolfe Decomposition. We are interested in applying Bender's decomposition technique, which is the dual of the Dantzing-Wolfe decomposition technique. Next we discuss the Benders decomposition algorithm. Our exposition follows Minoux(1986) [40].

Consider the following problem:

$$\begin{aligned} \min c^t \cdot \Xi + f^T \cdot \lambda \\ \text{subject to } D \cdot \Xi + F \cdot \lambda \leq d \\ \Xi \geq 0, \lambda \in Y \subset R^m, \end{aligned} \tag{3.1}$$

where

$$D = \begin{bmatrix} D_1 & 0 & \dots & 0 \\ 0 & D_2 & \dots & 0 \\ \dots & \dots & & \dots \\ 0 & 0 & \dots & D_k \end{bmatrix}, \tag{3.2}$$

and

$$\Xi = [\Xi_1, \Xi_2, \dots, \Xi_l], \tag{3.3}$$

$$c = [C, C, \dots, C], \tag{3.4}$$



$$\lambda = [\lambda_1, \lambda_2, \dots, \lambda_l, \lambda_1^*, \lambda_2^*, \dots, \lambda_l^*], \quad (3.5)$$

$$f = [1, 1, \dots, 1, 0], \quad (3.6)$$

where sizes of  $c$  and  $f$  are equal to  $2l$  and  $2l + 1$ , respectively.

Since  $D$  is a block diagonal matrix, we can decompose problem (3.1) into  $k$  parts.

Then (3.1) can be written as follows

$$\begin{aligned} \min \sum_{k=1}^K c_k^T \cdot \Xi_k + f^T \cdot \lambda \\ \text{subject to} \\ D_1 \cdot \Xi_1 + F_1 \cdot \lambda \leq d_1 \\ D_2 \cdot \Xi_2 + F_2 \cdot \lambda \leq d_2 \\ \vdots \\ D_K \cdot \Xi_K + F_K \cdot \lambda \leq d_K \\ \Xi_1, \Xi_2, \dots, \Xi_K \geq 0, \lambda \in Y \end{aligned} \quad (3.7)$$

Once the variable  $\lambda$  is given, and the following systems  $D_k \cdot \Xi_k \leq d_k - F_k \cdot \lambda$  have a solution, the solution of the problem restricted to variable  $\Xi$  is decomposed into  $K$  independent linear programming problems. The following problem must be solved at each iteration for fixed  $\lambda$ :

$$\min c^T \cdot \Xi \quad (3.8)$$

subject to

$$D \cdot \Xi \leq d - F \cdot \lambda$$

$$\Xi \geq 0$$

It is necessary that problem (3.8) has a non-empty feasible region. In order to satisfy this, we use Farkas and Minkowski lemma [40]. For every constraint  $i$  of problem (3.8), there is a dual variable  $u_i$ . Denote by  $u$  the row vector of dual variables. Then, according to this lemma, problem (3.8) has a solution  $\Xi \geq 0$  if and only if  $u \cdot (d - F \cdot \lambda) \leq 0$  for all  $u$  for which  $u \cdot D \leq 0$  holds. The set  $\{u/u \cdot D \leq 0\}$  is polyhedral having a finite number of generators which we denote by  $u^1, u^2, \dots, u^p$ . The necessary and sufficient condition of the Farkas and Mikowski lemma is equivalent to the following system of inequalities:

$$\begin{aligned} u^1 \cdot (d - F \cdot \lambda) &\leq 0 \\ u^2 \cdot (d - F \cdot \lambda) &\leq 0 \\ &\vdots \\ u^p \cdot (d - F \cdot \lambda) &\leq 0. \end{aligned} \tag{3.9}$$

If the system of inequalities in (3.9) has no solution in  $\lambda$ , then there is no  $\lambda \in \mathfrak{R}^m$  such that (3.8) has a solution. This implies that problem (3.1) itself has no solution. Because of this, we assume that the problem (3.8) has a solution. Problem (3.1) can be written as

$$\min_{\lambda \in \mathfrak{R}} \{f^T \cdot \lambda + \min_{\Xi} \{c^T \cdot \Xi / D \cdot \xi \leq d - F \cdot \lambda, \xi \geq 0\}\}, \tag{3.10}$$

where  $\mathfrak{R}$  is the set of vectors  $\lambda \in Y$  which satisfy (3.9).

For a given  $\lambda$ , the dual of (3.8) is

$$\max v \cdot (d - F \cdot \lambda)$$

subject to

$$v \cdot D \leq c \quad (3.11)$$

$$v \leq 0,$$

where  $v$  is a row vector.

Define  $V = \{v/v \cdot D \leq c\}$ . As it can be seen from (3.11),  $V$  does not depend on  $\lambda$ . The  $(u^1, u^2, \dots, u^p)$  as defined in (3.9) are extreme rays of the polytope  $V$ . If  $V$  is empty then we can conclude that either (3.8) has no solution or is unbounded. In our case, (3.8) always has a solution because of the structure of the problem. Since in general, we assume that  $\lambda \in \mathfrak{R}$ , this implies that (3.8) has a solution. Then problem (3.8) has an unbounded solution. Therefore, problem (3.1) is unbounded. By using the duality theorem [40], (3.1) becomes:

$$\min_{\lambda \in \mathfrak{R}} \{f^T \cdot \lambda + \max_u \{u \cdot (d - F \cdot \lambda) / u \cdot D \leq c\}\}. \quad (3.12)$$

Assuming  $V$  is not empty, define  $(v^1, v^2, \dots, v^q)$  as vertices of  $V$ . The number of vertices are finitely many. Then (3.12) can be written as:

$$\min_{\lambda \in \mathfrak{R}} \{f^T \cdot \lambda + \max_{i=1,2,\dots,q} \{v^i \cdot (d - F \cdot \lambda)\}\}. \quad (3.13)$$

Therefore, equation (3.13) is equivalent to

$$\begin{aligned} & \min_{(z,\lambda)} z \\ & \text{subject to} \\ & z \geq f^T \cdot \lambda + v^1 \cdot (d - F \cdot \lambda) \\ & z \geq f^T \cdot \lambda + v^2 \cdot (d - F \cdot \lambda) \\ & \dots\dots\dots \end{aligned} \quad (3.14)$$

$$z \geq f^T \cdot \lambda + v^q \cdot (d - F \cdot \lambda), \quad (3.15)$$

where  $\lambda \in \mathfrak{R}$  and  $z$  is unrestricted.

The problem (3.13) is called the master program. Since, (3.8) or (3.11) in our case always has an optimal solution, we do not need to add the system of inequalities in (3.7) to (3.14). In the general case, we have to do so. The number of constraints in (3.14) is equal to the number of vertices, which means that the number of constraints can be enormous. On the other hand, the number of active constraints is equal to  $m+1$  where  $m$  is the dimension of the  $\lambda$  vector. One can solve (3.14) explicitly if the problem size is small. Otherwise, we have to find an efficient way to solve this problem.

Suppose that at some stage only some constraints of (3.14) are known explicitly. We can construct another problem called restricted master problem by using subsets of inequalities selected from the constraints of (3.14). Specifically,

$$\begin{aligned} & \min_{(z, \lambda)} z \\ & \text{subject to} \quad (3.16) \\ & z \geq f^T \cdot \lambda + v^j \cdot (d - F \cdot \lambda), (\forall j \in J) \end{aligned}$$

where  $\lambda \in \mathfrak{R}$ ,  $z$  unrestricted and  $J \subset 1, 2, \dots, q$ .

This problem can be solved by using the simplex or other LP methods for  $Y = \mathfrak{R}^m$  (like interior point methods). In general, it is easy to solve the dual of (3.16) which gives nice structures [40].

Next, we explain the algorithm explicitly. Let  $(\bar{\lambda}, \bar{z})$  be an optimal solution of (3.16). If (3.16) does not have a solution, then this is also true for problem (3.1). Therefore,

we stop the computations. Then  $\bar{z}$  is a lower bound of  $z^*$  which is an optimal value of the master problem and of problem (3.1). We need to check if  $(\bar{\lambda}, \bar{z})$  satisfy all the constraints of (3.14). Then we solve problem (3.11) for  $\lambda = \bar{\lambda}$ . We assume that (3.11) has a solution for  $\lambda \in \mathcal{R}$ . Actually, (3.11) has an optimal solution for all  $\lambda$ . Then there are two cases we should consider.

**Case 1:**

We have a finite value optimal solution for the sub problem in (3.11). Since this is an LP, the optimal solution is reached at a vertex of  $V$ . In addition to this, we have

$$\bar{z} \geq f^T \cdot \bar{\lambda} + \bar{v} \cdot (d - F \cdot \bar{\lambda}).$$

This is also true for all vertices of  $V$ . So, we can write

$$\bar{z} \geq f^T \cdot \bar{\lambda} + v^j \cdot (d - F \cdot \bar{\lambda}), j = 1, 2, \dots, q.$$

It follows that the current solution of (3.16) is an optimal solution of (3.14) and it is an optimal solution of (3.1). Then, the algorithm terminates.

**Case 2:**

The optimum of (3.11) is bounded and we have

$$\bar{z} < f^T \cdot \bar{\lambda} + \bar{v} \cdot (d - F \cdot \bar{\lambda})$$

So, we can say that  $(\bar{z} \geq f^T \cdot \bar{\lambda} + \bar{v} \cdot (d - F \cdot \bar{\lambda}))$  is not satisfied by the current solution of (3.16). The following constraint must be added to (3.16) to form a new restricted programme.

$$z \geq f^T \cdot \lambda + \bar{v} \cdot (d - F \cdot \lambda)$$

---

**Initialize:**

Let  $(\bar{\lambda}, \bar{z})$  be an optimal solution of (3.16)

Solve the sub problem (3.11)

**Main Step:**

While  $z \geq f^T \cdot \lambda + \bar{v} \cdot (d - F \cdot \lambda)$  is not satisfied

-Add to the restricted program constraints not satisfied by  
the current solution

-Solve the restricted program with  $z \geq f^T \cdot \lambda + \bar{v} \cdot (d - F \cdot \lambda)$

-Solve the sub problem

End

---

Table 3.1: Benders Decomposition Algorithm

The algorithm is given in table 3.2.

Problem (3.11), which is called Benders sub problem, has a very nice structure in the case of SVR formulation, because constraints matrix  $D$  is a diagonal, and its diagonal element is -1. We can find an optimal solution for (3.11) by using the procedure described in table 3.2. In addition to this, subproblem (3.11) is bounded and always has a solution.

Note that the elements of vector  $c$  are the trade off values in SVR. So, we don't have to store vector  $c$  explicitly. We have explained the Benders Decomposition technique for support vector regression case(SVR). In SVR case, solving the sub problem is not difficult comparing with other applications of this technique. On the other hand, we

---

Given  $y$

For  $i = 1 : 2l$

    If  $(d - F \cdot \lambda)_i < 0$

$v_i = -c_i$

    Else  $v_i = 0$

    End If

Next

---

Table 3.2: Procedure to solve subproblem in SVR case

need to find an efficient way to solve the master program. Since at each step we have two matrix vector multiplications efficient matrix multiplication techniques can be applied.

Two problems have to be solved each iteration. Either the simplex algorithm or an interior point method can be applied to solve these problems. Benders' sub problems generates a cut at each iteration for the master problem. If the solution of the subproblem is not optimal, the cut in Benders decomposition is called inexact cut. For more information about inexact cut algorithm can be found in [63]. This would improve the performance of the algorithm. As we mentioned before, the subproblem has very nice structure and properties. There is no need to use any optimization algorithm at all. For solving master problem, primal-dual algorithm or analytical center method seem to be good selection [39, 26]. Bender decomposition algorithm is implemented in CPLEX [30].

## Chapter 4

# Training support vector machines with very large datasets

We begin by discussing three approaches to training SVM with very large datasets, and then we discuss how these can be seen as special cases of a general approach to fast training that is in the spirit of hierarchical training.

### 4.1 Clustering and Bagging techniques for Support Vector Machines

In this section three methods will be discussed which are clustering, sub-sampling and bagging methods. k-means clustering algorithm will be discussed briefly and shown how to apply to the classification.



#### 4.1.1 Support Clusters

Given a finite sampling of points from a space  $X$ ,  $\{x^i\}_{i=1}^M$ , the target of clustering is to group the data into sets of "similar" points [10]. In another way, clustering is the task of grouping the objects together into meaningful subclasses. Clustering algorithms can be classified into two categories: partitional and hierarchical. Partitional clustering algorithms such as k-means, k median, obtain clusters of objects by selecting cluster representatives and assigning each object to the cluster with its representative closest to the object. On the other hand, hierarchical clustering algorithms produce a nested sequence of clusters. We will apply the k-means algorithm to our problem. The reason is that we need a fast and efficient algorithm to find the clusters of large datasets.

The k-means algorithm tries to find a partition of data which maximally separates the clusters through the minimization of a cost function. Let us give an explicit definition of the clustering problem that we deal with. Given  $l$  points  $\{x^1, x^2, \dots, x^l\}$  in the  $n$  dimensional real space  $R^n$ , and a fixed integer  $k$  of clusters, determine  $k$  centers in  $R^n$ ,  $\{c^1, c^2, \dots, c^k\}$ , such that sum of the distances of each point to a nearest cluster center is minimized. The clustering problem can be expressed as the following optimization problem

$$\min_{c^1, c^2, \dots, c^k} \sum_{i=1}^l \min_{l=1, 2, \dots, k} \|x^i - c^l\|, \quad (4.1)$$

where  $\|\cdot\|$  is some arbitrary norm on  $R^n$ .

If we consider problem (4.1) with the 2-norm squared, the iterative algorithm is the k-means approach to clustering [10]. The underlying optimization problem for the

k-means algorithm is

$$\begin{aligned} & \min_{c, t} \sum_{l=1}^k \sum_{i=1}^l t_{il} \cdot (0.5 \cdot \|x^i - c^l\|_2^2) \\ & \text{subject to } \sum_{l=1}^l t_{il} = 1 \quad t_{il} \geq 0 \quad i = 1, \dots, l, \quad l = 1, \dots, k, \end{aligned} \quad (4.2)$$

where  $t_{il}$  are selection variables.

---

Given  $k$  cluster centers  $c^{1,j}, c^{2,j}, \dots, c^{k,j}$  at iteration  $j$ , compute  $c^{1,j+1}, c^{2,j+1}, \dots, c^{k,j+1}$

by the following two steps:

1. *Cluster assignment* : For each  $i = 1, \dots, l$  assign  $x^i$  to cluster  $l(i)$  such that  $c^{l(i),j}$  is nearest to  $c^i$  in the two norm.

2. *Cluster Center Update*: For  $l = 1, \dots, k$  set  $c^{l,j+1}$  to be the mean of all  $x^i$  assigned to  $c^{l,j}$ .

Stop when  $c^{l,j} = c^{l,j+1}, l = 1, \dots, k$ .

---

Table 4.1: Prototype of k-mean algorithm

As a first step, the size of the problem (2.32) can be substantially reduced with the following preprocessing: instead of using all the example points to train a machine (a “global” SVM), first we can group the points of each class separately into a number of clusters and then use one representative point from each cluster, such as the centroid of each cluster (we could also use a point “difficult” to classify if we can find one), and train first an SVM using only these representative points. In a recent paper, a similar

idea is proposed [34]. By doing this, we reduce the size of the problem substantially because the number of clusters can be for example 5 % or 10 % or less of the whole data. In this case, we spend some time to find the cluster centers but we reduce the time spent to train the SVM.

Using the characteristics of SVM discussed above, namely that the final solution depends only on the support vectors, once an SVM is trained using only the centroids of the clusters we can then reconstruct the initial example data corresponding to the centroids that are support vectors in this first iteration (call them support clusters). We can then either train one SVM using all these initial example data if they are not too many, or repeat an iteration of clustering and training an SVM with the centroids only, this time though by clustering only the initial example data corresponding to the support clusters of the first iteration. Furthermore, instead of clustering this subset of the initial examples using the same support clusters found before, we can increase the number of clusters used (that is, use more clusters than the number of support clusters of the first iteration). We can repeat this process of clustering, training, using only the support clusters found, until a condition is satisfied: this can be for example either that the number of initial examples within the current support clusters is small enough to train one SVM using all these initial examples, or that the set of initial examples included within the current support clusters does not change significantly from the previous iteration.

The justification of a clustering based approach (each iteration in the algorithm described above) can be explained by using the checker-board data shown in figure 4.1 [34]. First we cluster the data of each of the two classes (separately) for example using

the k-means algorithm [7]. As it can be seen from 4.2, we can represent the class 1 examples with 23 centroids, and if we have those points, we do not need the whole data. The same argument holds for class -1. As it can be seen from figure 4.2, we can represent the whole data set (1000 points) with 100 cluster centers. We can then train an SVM using only those 100 points. This will lead to a set of support vectors - support clusters - from which we can then reconstruct the initial example points to train another SVM (or repeat the clustering and then training iteration). Since the size of the new problem is smaller than the initial one using all the example data, the computation time is less than the conventional SVM algorithm. Moreover the advantage of this procedure is that we do not lose any information like in the case of reduced support vector machines [34], because the procedure we used does not choose the data points randomly. In a sense the proposed method uses a combination of clustering and SVM as a way to choose the important data points for the next iteration. Of course on the other hand, we spend more time to find the cluster centers. If some fast clustering algorithms are used, the total time to find the optimal hyperplanes can be reduced. We show experiments of total computational time and predictive performance of the methods in the next chapter.

#### 4.1.2 Sub-sampling and Bagging Methods

Training many learning machines each using a small random sub-sample of the initial example data and then combining the machines through voting has been suggested in the past as a way for fast training with very large datasets (i.e. [20]). For the case of SVM, however, one can turn this approach “naturally” into a hierarchical one by using again the characteristic of SVM that only the support vectors are important for the final

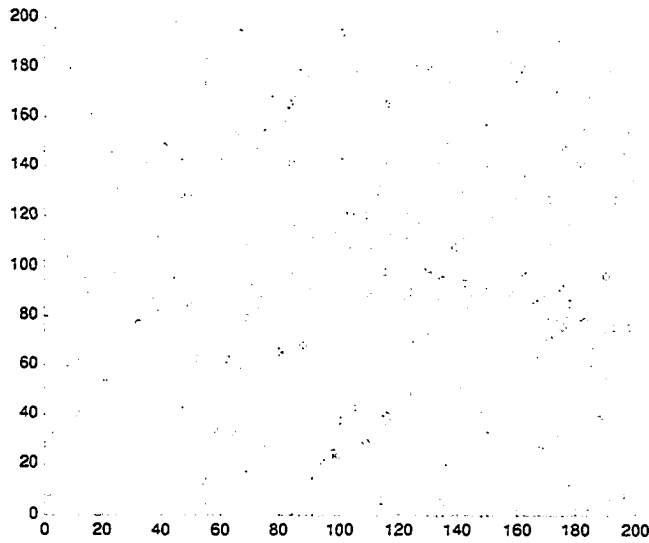


Figure 4.1: The checkerboard data without clusters.

solution. Consider the following method: first generate a number  $N$  of small random sub-samples (say 10% or less of the  $\ell$  example data) of the whole initial training set, and then train an SVM with each of the sub-samples separately (which can be done in parallel). This is the same process as that for bagging learning machines [12]. In bagging type methods [20, 12] the  $N$  resulting machines are combined through a voting scheme. In this case, however, we can consider only the support vectors of each of the  $N$  machines and then repeat the same process. Specifically, generate a new set of sub-samples of the support vectors of the first iteration, train an SVM with each of the new sub-samples, and continue until the following condition is satisfied: either the union of the support vectors after the iteration is small enough in size in which case train one SVM using all the support vectors of all the machines, or, if not, the set of all support vectors of all  $N$  machines does not change significantly from the previous iteration in which case the  $N$  machines can be combined through a voting scheme like in bagging.

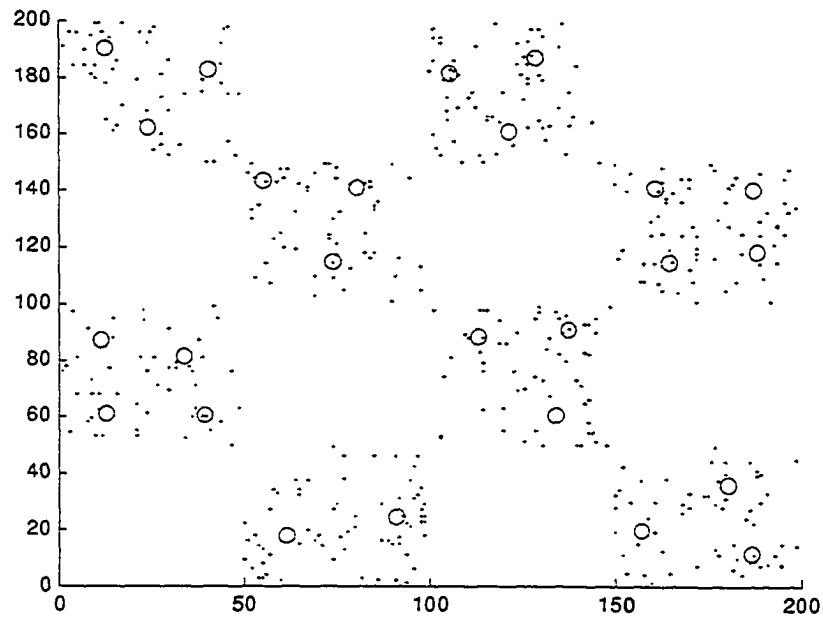


Figure 4.2: Clustering the checker-board data. Red color shows the whole data (486 points) and blue circles are the clusters centers (50 points). We can represent the whole data set by only 50 points. Those points are used for the SVM algorithm.

Notice that at each iteration each of the SVMs can be trained in parallel, and that at each iteration we “disregard” the initial example points that are not support vectors in any of the SVMs. Furthermore, each of the SVM is trained using a small set of data. Hence it can be fast. We have run this approach - only for one iteration however - for a number of datasets.

The bagging methods can be described by ensembles of kernel machines for support vector machines. Given kernel machines (decision functions)  $f_1(x), f_2(x), \dots, f_N(x)$  (e.g. each  $f_n$  is found by using different training data or different kernels), we combine them into the following

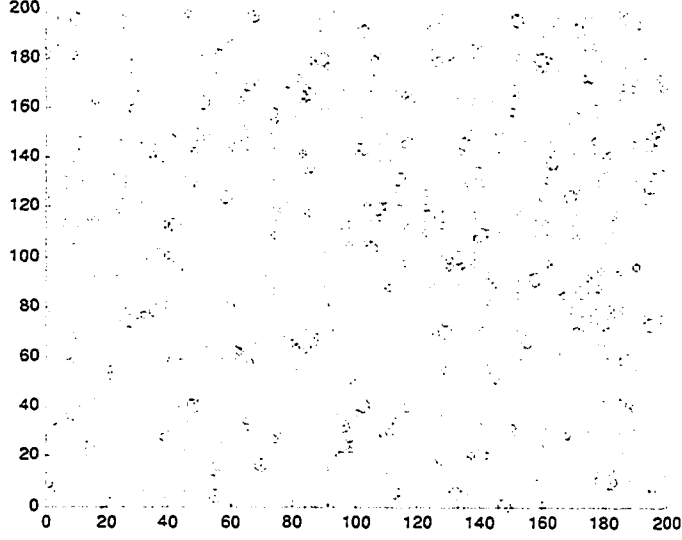


Figure 4.3: The checkerboard data set and 131 total clusters. The big circle shows the clusters. We choose only 20 % of data to represent the whole data set (650 points).

$$F(x) = c_1 f_1(x) + c_2 f_2(x) + \dots + c_n f_n(x), \quad (4.3)$$

where  $c_1, c_2, \dots, c_n \geq 0$  and  $\sum_{i=1}^N c_i = 1$ . We can select the  $c_i$ 's to be equal as follows

$$c_n = \frac{1}{N}, \quad n = 1, 2, \dots, N. \quad (4.4)$$

In this case, each machine is weighted equally. Bagging method may increase the stability of the algorithm [20, 48]. It can be shown [20] that the error bound for bagging method for any given  $\delta$  with probability  $1 - \eta$  with sample size  $\alpha$  of the training set and each having a stability  $\beta_\alpha$  is

$$E[(\theta(-yF(x)))] = \text{Rem}p^\delta(D_{n,\alpha}) + \frac{2\alpha}{l} \cdot \frac{\beta_\alpha}{\delta} + \sqrt{\frac{1}{2l} \cdot \left(\frac{4\alpha \cdot \beta_\alpha}{\delta} + 1\right) \ln\left(\frac{1}{\eta}\right)}, \quad (4.5)$$

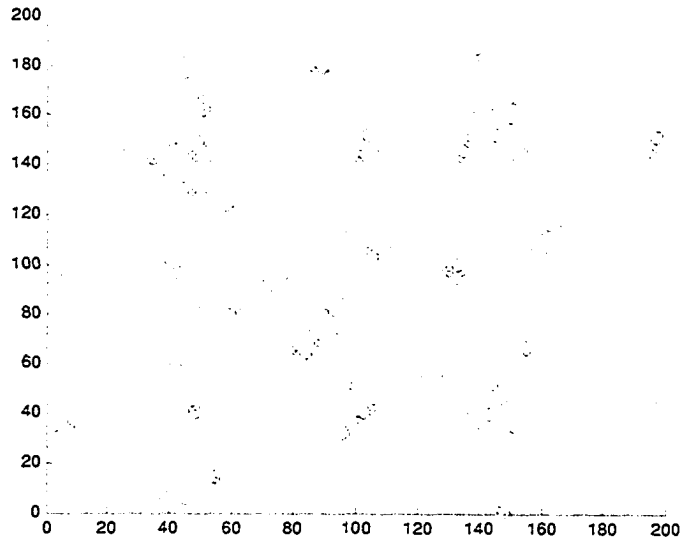


Figure 4.4: Support clusters which are found by using the SVM and original data that we extract by using those clusters. We use the clusters in figure 4.3. Support clusters are the clusters that we found by SVM.

where  $\beta_\alpha$  is assumed to be a non-increasing function of  $\alpha$ ,  $D_{n,\alpha}$  training sets of size  $\alpha$ ,  $\delta \geq 0$ , and  $l$  size of the whole dataset.

### 4.1.3 Hierarchical Training Methods

Both of the methods described above can be seen within a family of methods for fast training with very large datasets, namely hierarchical training methods. In both cases the original training problem is approached in a hierarchical way: first a set of smaller scale SVM problems is solved, and then based on the results of this iteration (in this case on the support vectors or support clusters found), a next set of SVM problems is solved. In the case of support clusters, the approach can be seen as solving the classification problem from a “low” to a “high resolution”: at the first iteration all



points belonging in the same cluster (points close to each other) are treated as one point. At each iteration we “increase the resolution of the space” by clustering the initial example points corresponding to the support clusters of the previous iteration using however a larger number of clusters - which is equivalent to grouping the example data at a higher spacial resolution. There has been no formal framework for developing and studying such hierarchical methods, which can be valuable for designing other such methods. We now turn to discussing some experimental results that show how these preliminary hierarchical training methods work.

## 4.2 $\gamma$ -SVM Algorithm

The main idea behind all the classification techniques is to find a hyperplane that separates two classes with very small error. Support vector machines identify two bounding hyperplanes such that

$$y_i \cdot (w \cdot x_i + b) = 1 \text{ for } i \in P \quad (4.6)$$

$$y_i \cdot (w \cdot x_i + b) = -1 \text{ for } i \in N$$

where  $P$  are the positive examples ( $y_i = +1$ ) and  $N$  are the negative examples ( $y_i = -1$ ).

If we find a way to obtain the data points that are close to those hyperplanes, then we reduce the number of data points by using an efficient pre processing algorithm. We propose an algorithm which is described in table 4.2.

SVM algorithm or other classification algorithms try to find a hyperplane that separates two classes of points with a minimum error. The solution of the SVM optimization

problem gives two parallel hyperplanes which are defined above in (4.6). The optimal hyperplane is in equal distance from each class. Suppose we have obtained the data points which are close to two hyperplanes defined as above. By obtaining those points we reduce the size of the training data in the SVM. That help us to solve the problem efficiently. The question is how can one identify those points. What kind of metric we need to use?

We can use any norms to compute the distance. Suppose we are given a set  $S$  of points  $x_i \in R^n$  with each  $x_i$  belonging to either of two classes defined by  $y_i \in \{-1, 1\}$ . The first step is to split the set  $S$  into two groups  $S_1, S_2$  such that  $S_1$  contains all the data points in one group and  $S_2$  contains the other one. First take a point from  $S_1$  and compute the distance from data points in  $S_2$ . Then compare the distance with a user defined number  $\gamma$ . If distance is less than  $\gamma$  put this point to a new set  $S^*$ . Repeat this computation for each point in  $S_2$ . After this process, the training set has been reduced to  $S^*$  which is much smaller in terms of cardinality than the set  $S$ . Then the SVM algorithm can be applied to the new data set  $S^*$ . The algorithm is given in table 4.2.

#### 4.2.1 $\gamma$ -SVM in feature space

The decision surface in input space  $X$  is usually nonlinear. Using the  $\gamma$ -SVM algorithm in input space does not always reduce the number of points in the final set. In addition to this, the distance between the centroids of two sets in input space  $X$  is not always a good approximation to the margin. Because of this flaw, we need to find a good threshold ( $\gamma$ ). This can be accomplished in the feature space  $H$ . We can compute the distance between two points in feature space and apply the  $\gamma$ -SVM algorithm in feature

space.

Let us explain how to compute norms in the feature space. The norm of a vector  $x \in R^d$  in input space can be computed as

$$\|x\|_X = \sqrt{x_1^2 + x_2^2 + \dots + x_d^2}. \quad (4.7)$$

Norm of a point in feature space can be expressed as

$$\|\phi(x)\|_H = \sqrt{\langle \phi(x), \phi(x) \rangle}. \quad (4.8)$$

As it can be seen from (4.8), norm can be expressed through a dot product. This can be computed by using kernel functions. Specifically,

$$\|\phi(x)\|_H = \sqrt{\phi(x) \cdot \phi(x)} \quad (4.9)$$

$$\|\phi(x)\|_H = \sqrt{k(x, x)}.$$

The distance between two points  $x, y$  in  $H$  can be computed as follows.

$$\begin{aligned} \|\phi(x) - \phi(y)\|_\phi &= \sqrt{\langle \phi(x) - \phi(y), \phi(x) - \phi(y) \rangle} \\ &= \sqrt{\phi(x) \cdot \phi(x) - \phi(x) \cdot \phi(y) - \phi(y) \cdot \phi(x) + \phi(y) \cdot \phi(y)} \\ &= \sqrt{\phi(x) \cdot \phi(x) - 2 \cdot \phi(x) \cdot \phi(y) + \phi(y) \cdot \phi(y)} \\ &= \sqrt{k(x, x) - 2 \cdot k(x, y) + k(y, y)}. \end{aligned} \quad (4.10)$$

If we use the radial basis kernel function to compute the distance, equation (4.10) can be simplified by using the fact that  $k(x, x)$  is equal to 1. Then we have the following expression

$$\|\phi(x) - \phi(y)\|_H = \sqrt{2 - 2 \cdot k(x, y)}. \quad (4.11)$$

Next, we have to compute the threshold in the feature space. This can be done in two ways. First find the centroids of two sets in feature space and then compute the distance between them. Second way is that centroids are computed in input space and then by using equation (4.11), distance is computed in feature space. To deal with overlapping classes (data points), threshold value should be adjusted. By doing that we make sure that final set contains non-overlapping data points also.

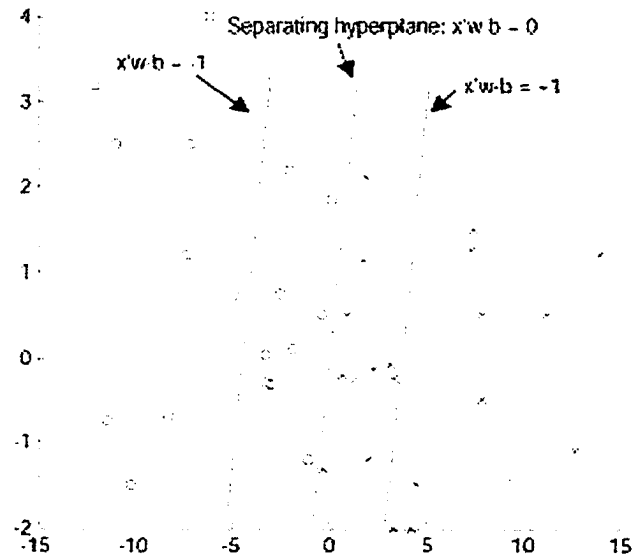


Figure 4.5: Solution of proximal support vector machine classifier

The relationship between this algorithm and the proximal support vector machines algorithm (PSVM) [22] is that PSVM finds two hyperplanes such that data points from each class clusters around them (see figure 4.5. On the other hand, our algorithm

identifies the data points from each class such that distance between them is less than some threshold value. Those points are possible candidate as support vectors.

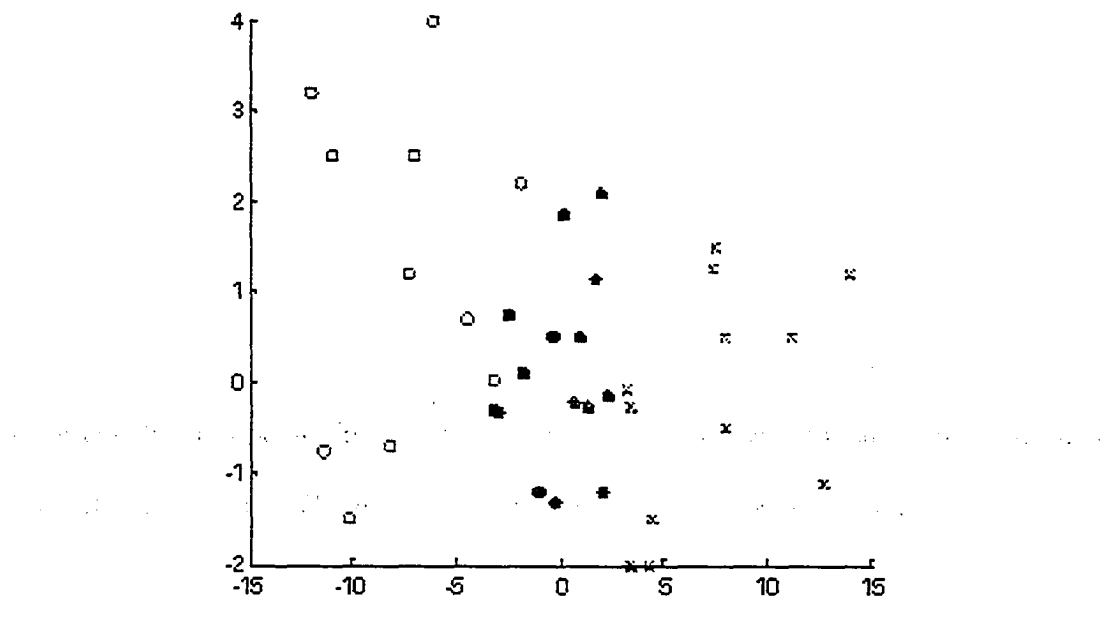


Figure 4.6: Identifying the closest points by using the algorithm we proposed. By exploring the sparsity of SVM theory, we do not need to whole dataset, we can use only filled data points shown in figure.

We will explore how our algorithm works with the following example (see figure 4.6). First, the number of support vectors,  $\alpha_i > 0$ , is less than or equal to the number of support vectors of other decomposition algorithms. This is what we expect from the algorithm, because the number of points in the final set is between 10-20 percent of the whole dataset.

The decision surface changes with different kernel functions. If the radial basis function is used (see figure 4.7), the SVM algorithm identifies some data points as

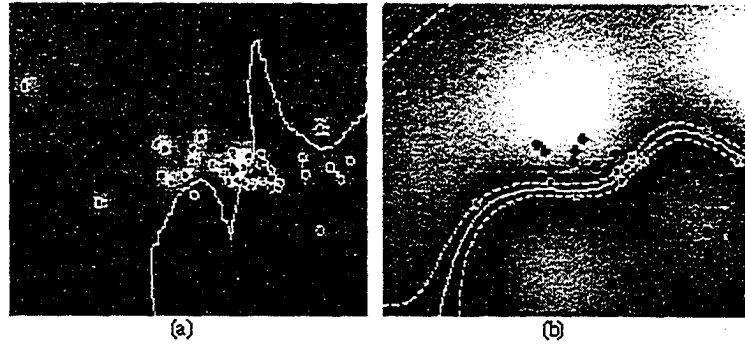


Figure 4.7: First figure(a) shows the decision surface of radial basis function kernel with  $\sigma = 1$ . In the second figure depicts the decision surface with  $\sigma = 3.2$ .

support vectors which are not supposed to be identified. This suggest that, the SVM with the radial basis overestimates the number of support vectors. This can be explained with the following reasoning. The algorithm fits the normal distribution to the dataset with a standard error  $\sigma$ . If the sigma is small, which depends on the data, then all the points will become important. Then, the decision surface will not classify the data correctly. On the other hand, if it is large, then less support vectors will be computed. This can be seen from figure 4.7. The optimal value of the  $\sigma$  can be found with trial and error.

If the linear kernel function is used, then the solution of the SVM and our algorithm are identical which is shown in figure 4.8.

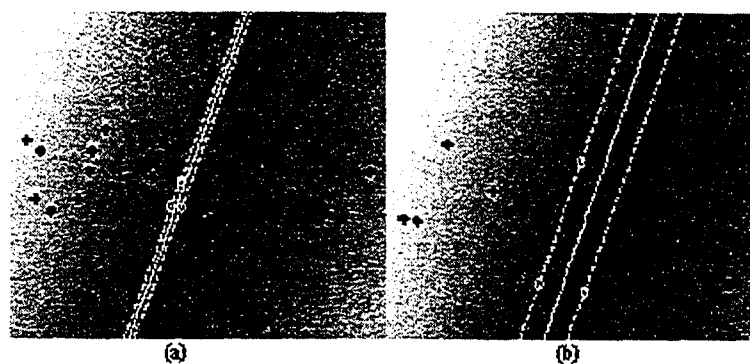


Figure 4.8: (a): In this figure, linear kernel function is used to train whole data set. (b): Linear kernel function is used with our algorithm. Decision surface are same for SVM and our algorithm

---

```

Begin
Given data set  $(x_1, y_1), \dots, (x_l, y_l)$  where  $x_i \in R^n$  and  $y \in \{-1, 1\}$ 
If  $y_i = 1$ , Put  $x_i$  in  $S_1$ , otherwise put in  $S_2$ 
Set  $l_1$  to number of data points in  $S_1$ 
Set  $l_2$  to number of data points in  $S_2$ 
For  $i = 1$  to  $l_1$ 
    For  $j = 1$  to  $l_2$ 
        Compute distance  $d(x_i, x_j)$  where  $x_i \in S_1$  and  $x_j \in S_2$ 
        If  $d(x_i, x_j) \leq \gamma$  Put  $x_j$  to set  $S^*$  with its label
    Next
Next
For  $i = 1$  to  $l_2$ 
    For  $j = 1$  to  $l_1$ 
        Compute distance  $d(x_i, x_j)$  where  $x_i \in S_2$  and  $x_j \in S_1$ 
        If  $d(x_i, x_j) \leq \gamma$  Put  $x_j$  to set  $S^*$  with its label
    Next
Next
Apply SVM algorithm to  $S^*$ 
End

```

---

Table 4.2:  $\gamma$ -SVM Algorithm



## Chapter 5

# Applications and Computational Results

This part consists of two main sections. In the first section we will compare Bender's decomposition technique, support cluster methods, subsampling and  $\gamma$ -SVM with other decomposition techniques. In the second section we apply the SVM method to S&P 500 daily return value prediction and option pricing model. SVM will be compared with neural networks and radial basis networks.

## 5.1 Comparison of the methods with other decomposition techniques

### 5.1.1 Classification case

Let us give some information about the data set we used for classification. For support clusters, subsampling method, and  $\gamma$ -SVM algorithm the following datasets have been used.

**Adult Dataset** comes from the UCI repository of machine learning databases. The task is to predict if an individual's annual income exceeds \$50,000 based on census data. There are 16 attributes changing from age to ethnical background. The total number of observation is 48000. Different sizes of training data will be used.

**Banana dataset** consists of artificial data derived from several Gaussian blobs in two dimensions. The training set has 4000 patterns and the test set contains 1300 points. For more information on the datasets see also the UCI Repository [9].

**Ford challenge** data is a time series of 50000 samples produced by a physical system (10-cylinder internal combustion engine). Each sample  $k$  of the time series consists of four inputs and one output ( $k = 1, 2, \dots, 50000$ ). The first input  $X1(k)$  represents a binary synchronization pulse related to a natural periodicity in the system (cylinder identifier). The second and third inputs,  $X2(k)$  and  $X3(k)$ , represent context (engine crankshaft speed in RPM and load). The fourth input  $X4(k)$  (crankshaft acceleration) has a more direct relationship with the bipolar output  $Y(k)$  (normal firing 1 or misfire +1, with normals dominating), but the relationship is complicated due to the dynamics of the system (torsional oscillations of crankshaft depend on speed and load as well as

presence of misfires prior to time  $k$ ).

**Splice Dataset:** The task in this dataset is to recognize two types of splice junctions in DNA sequences; exon/intron (EI) or intron/exon (IE) sites. A splice junction is a site in a DNA sequence at which 'superfluous' DNA is removed during protein creation. Intron refers to the portion of the sequence spliced out while exon is the part of the sequence retained. The number of cases is 3175.

**Mushroom Dataset** is from Audobon Society Field Guide. Attributes of the data describe the physical characteristics of a mushroom. The task is to identify the edible or poisonous mushrooms. The number of instance is 8124.

**Diabetes:** This dataset also came from UCI repository [9] and consists of a binary classification task with 8 input attributes. The size of the dataset is 844.

Three different datasets are used to compare support clusters, bagging methods and subsampling with the conventional SVM algorithm. The datasets we used are Banana, Diabetes and Ford-challenge data. Banana and Ford contain 4000 training data points and 1300 for testing. Diabetis has 468 points for training and 300 points for testing. All the support vector problems are solved by using Svm-Fu [52] or SVMTorch [16]. Training error is used as a stopping criterion. Specifically, we stop when training error is smaller than 0.001.

We have compared the speed and performance of the algorithms that we propose in this paper with decomposition techniques proposed by [44, 52, 16].

As it can be seen from table 5.1, support clusters method is superior to other methods. The reason is that finding the clusters of the dataset does not take much more time, and the corresponding SVM is smaller size than the other methods.

	BANANA	DIABETIES	FORD
Support Cluster	335.3	98.7	325.5
Bagging	571	107.3	395.3
Subsampling	758	137.4	493.75
SVM	998.3	427.5	823.3

Table 5.1: Comparison of CPU time (seconds).

The next two tables (5.3, 5.4) show the comparison of  $\gamma$ -SVM and SVMTorch algorithms. We make two comparisons in terms of speed and generalization. Table 5.3 shows that our proposed algorithm is much faster than the SVMTorch. The reason for that, is that we preprocess the data with a small amount of time. By doing that we identify the possible support vectors corresponding  $\lambda_i > 0$ . The final step of the algorithm finds only the optimal values. This step is not time consuming.

Since the  $\gamma$ -SVM algorithm is based on the same theory, we expect that the quality of the solution has to be as good as other decomposition techniques proposed to solve the SVM classification problem. As it can be seen from table 5.4, generalization error is either same as in SVM solution or better than that. For example, for ring dataset and ford challenge dataset, we have better results.

The comparison of Support clusters, bagging and  $\gamma$ -SVM with each other shows that  $\gamma$ -SVM is more efficient in terms of speed and generalization error.

Test performance shows different results than the speed up performance. When we use the whole data set, without any preprocessing, SVM algorithm finds the global optimum since the SVM optimization problem is convex. On the other hand, if we

	BANANA	DIABETIES	FORD
Support Cluster	0.855	0.717	0.808
(Std. Error)	(0.034)	(0.067)	(0.027)
Bagging	0.899	0.717	0.906
(Std. Error)	(0.019)	(0.055)	(0.029)
Subsampling	0.887	0.737	0.837
(Std. Error)	(0.037)	(0.062)	(0.036)
SVM	0.906	0.737	0.907
(Std. Error)	(0.034)	(0.062)	(0.029)

Table 5.2: Comparison of test performance

preprocess the data like using the clusters, or take the subsample to represent the whole dataset, the solution is not the optimal solution of conventional SVM but it is very close to that. There is a trade-off between speed and performance. We can speed up the process by compromising for the generalization performance. This difficulty can be achieved by using  $\gamma$ -SVM because we do not sacrifice any information lost for speeding up.

In this experiment, the banana data set is used. In table 5.5 the CLUSTERS column contains the total number of clusters for class -1 and class 1 respectively. As it can be seen from the above table, when we increase the number of clusters, CPU time is also increasing. On the other hand, test performance is getting better while we increase the number of clusters but, then performance starts decreasing. This suggests that there is an optimal number of clusters that we need to choose in order to get better results.

	$\gamma$ -SVM(CPU Time)	SVMTorch(CPU Time)
Banana	94.33	998.30
Ford Challenge	575.00	823.30
Ring Data	4.27	77.94
Adult	712.90	4471.80
Splice	2.08	2.17
Mushroom	40.61	51.69

Table 5.3: Comparison of  $\gamma$ -SVM with SVMTorch for CPU speeds in seconds

Since SVM depends on many free parameters such as kernel parameter and trade off value  $C$ , we might get better results for different values for  $C$  and kernel parameters.

### 5.1.2 Regression Case

Next we compare the Benders Decomposition algorithm with a standard quadratic programming algorithm and linear programming. Primal-Dual algorithm is used to solve LP and QP problems. We use the S&P 500 daily return value, Boston housing data and sin function. S&P 500 daily return values are computed by using

$$r_t = \frac{x_t - x_{t-1}}{x_{t-1}},$$

where  $r_t$  is the return at time  $t$ ,  $x_t$  and  $x_{t-1}$  are S&P 500 index values at  $t$  and  $(t-1)$  respectively. Different training sample sizes are used, from 30 to 300.

The following table shows the results of this experiment. We compare these two

	$\gamma$ -SVM	SVMTorch
Banana	90.33	90.66
Ford Challenge	92.31	90.70
Ring Data	98.75	98.00
Adult	76.00	76.00
Splice	60.00	54.00
Mushroom	93.27	95.25

Table 5.4: Comparison of  $\gamma$ -SVM with SVMTorch for generalization error which is the percentage of correctly classified data

algorithms in terms of speed. Benders Decomposition algorithm is approximately 10 times faster than quadratic programming [see tables 5.6, 5.7]. As it can be seen from the table below, when the sample size is getting larger, the time to solve the QP is increasing exponentially.

Boston data has 506 training data points. The results of our experiments are shown in tables 5.6, 5.7.

In order to see the speed we gain from decomposition, we need to compare with other existing algorithms. For comparison, an interior point QP algorithm and LP algorithm are chosen respectively. The following table (table 5.8) suggests that, Bender's Decomposition method is faster than the other two methods. We use the sin function with 104 data points.

CLUSTERS	TIME (CLUSTERS)	TOTAL TIME	PERFORMANCE
159	140.72	478.22	0.424615
320	178.95	704.18	0.854615
479	239.42	766.72	0.818462
999	414.3	941.6	0.733077

Table 5.5: Comparison of number of clusters

Number of Examples	Benders Decom.(CPU Time)	Std. QP Form.(CPU Time)
30	2.6	35.7
50	2.3	24.7
100	50.1	152.7
200	104.3	1489.3
300	107.6	3925.3

Table 5.6: Comparison of Benders Decomposition with standard QP algorithm for different sample sizes

## 5.2 Applications to Financial Markets

### 5.2.1 Prediction of S&P 500 Daily Return Value

Prediction of the economic indicators of a market is very crucial. If one has robust forecasting tools, then he/she will increase the return on investment. The financial forecasting problem is very complicated due to the number of factors that can influence the market. In addition to this, choosing the important factors is also very difficult.



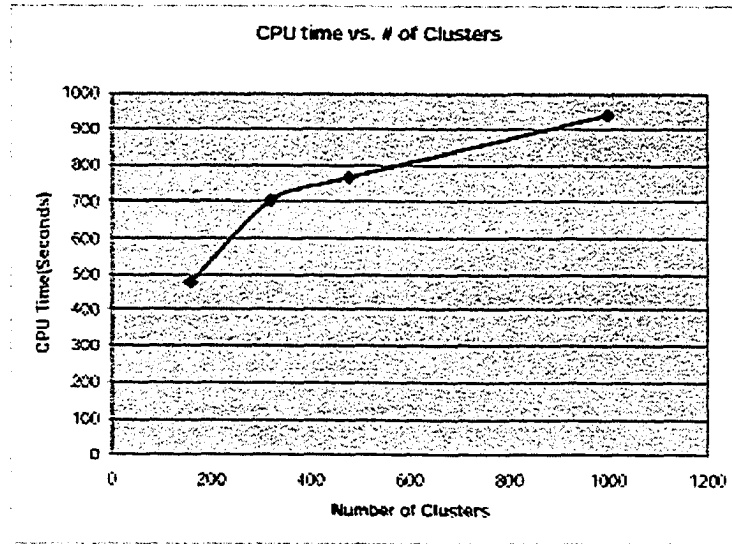


Figure 5.1: CPU time vs. Number of Clusters SVM.

Usually, one has to reduce the number of factors in his/her model. This can be done by using some statistical techniques.

Since, it is very difficult to predict the price of an individual stock, S&P 500 daily return data are used for the financial forecast application [58, 29]. Previous daily return values are used as input. Daily return will be computed by using the following equation

$$r_t = \frac{x_t - x_{t-1}}{x_t} \quad (5.1)$$

where  $r_t$  is the return at time  $t$ ,  $x_t$  is the S&P 500 index value at  $t$ , and  $x_{t-1}$  is S&P 500 index value at  $t-1$ .

Data are gathered from the Yahoo's financial web site. Daily return values are computed by using the above equation. Training data are sampled from June 9, 1999 to January 10, 2000. Test data are from January 11, 2000 to March 12, 2000. In our

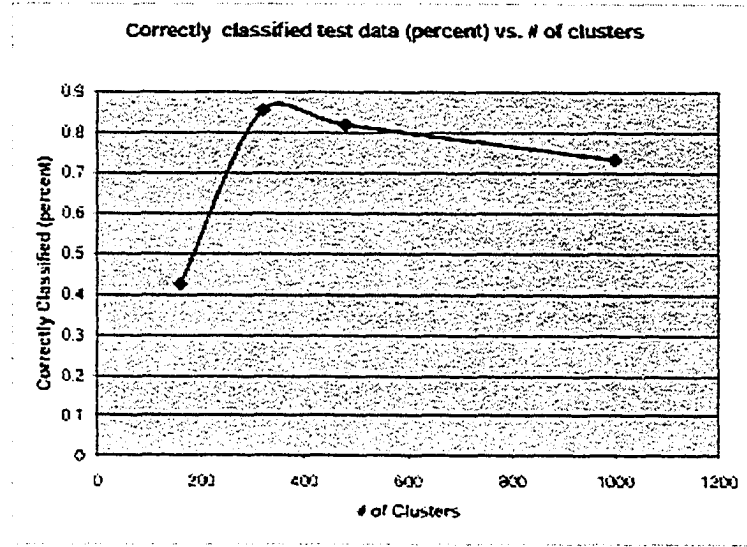


Figure 5.2: Correctly classified test data (percent) vs. number of clusters

experimentation, we approximate the following dynamical system.

$$x_t = f(x_{t-1}, x_{t-2}, x_{t-3}, x_{t-4}, x_{t-5}), \quad (5.2)$$

where  $x_t$  is S&P 500 index value at time  $t$ . A Matlab implementation of the SVR algorithm is used for quadratic programming. The primal dual interior point method is used for finding an optimal solution. For linear programming problem, Benders decomposition technique for support vector machines [59] is used. SV regression is compared with other techniques such as MLP, RBF networks and ARIMA model. There are several issues that we need to consider in the SVR application. First of all, we need to determine some parameters before running the particular algorithm. These parameters are ,  $C$  and kernel function. We set to  $C = 1000$ . Radial basis kernel function is used with different parameter values. Specifically, width is set to 0.1, 0.15, 0.20, 0.25, 0.5,

Number of Examples	Benders Decom.(CPU Time)
30	11.6
50	19.48
100	34.8
200	59.4
300	82.1

Table 5.7: Comparison of CPU time for different sample sizes. Boston housing data is used.

Methods	CPU Time
Bender's Decomposition	4
Standard LP( Without Decomposition)	11.8
Standard QP ( Without Decomposition)	240

Table 5.8: Comparison of CPU time for different methods( Bender's Decomposition, Standard QP and LP algorithms .

0.75, 1, 2.5, 5, 10, 50 and 100. As the width of the radial basis kernel increases from 0.1 to 100, we get worse results in terms of the prediction. In addition to this, we have very good forecast results for training period when we use small width. On the other hand, we obtain poor results in testing, period. As we increase , the accuracy of the prediction also increases. This can be seen from figures 5.4 and 5.5.

For the MLP case, different network architectures are used in order to find the best solution in terms of mean square error. The problem is that the back-propagation

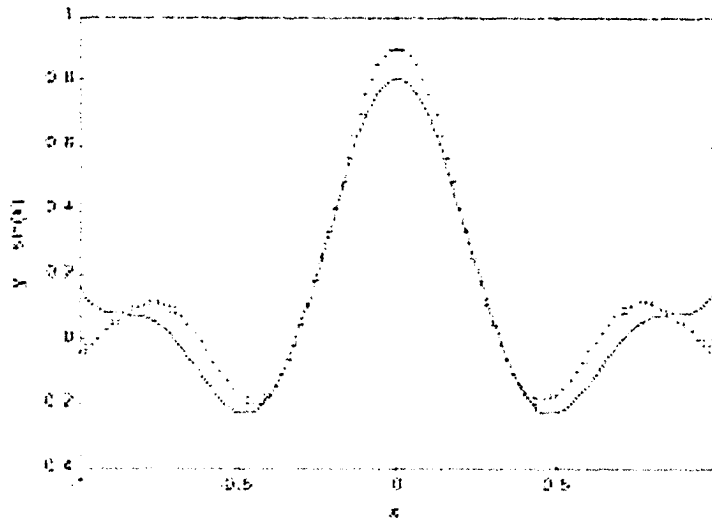


Figure 5.3: Estimated and actual value of  $\sin x$  function (+ is actual, solid estimated).

algorithm does not find the global minimum because the objective function is not convex. Therefore, the results we have are not as good as in the SVR case. Figure 5.6 illustrates the mean square error of the different MLP architectures.

For RBF networks and ARIMA model, we use the same structure. For each method, the best model is chosen. We compare these four models in terms of MSE's. The table 5.9 is shown the results.

According to the results shown in table 5.9, Linear Programming formulation of SVR is better than the other methods. Overall, the SVR approach is better than the time series methods and neural networks approaches.

### 5.2.2 Option Pricing Model

One of the goals of financial methods is asset evaluation which means determining the market price of an asset, predicting what this price will be in the future, and how it

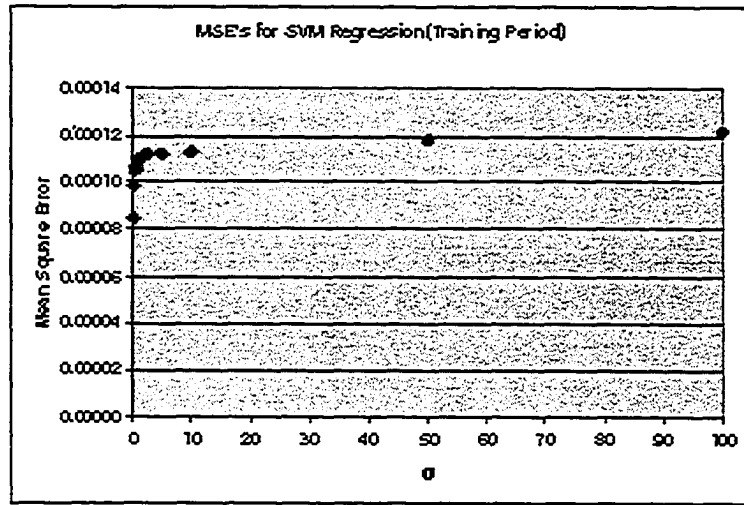


Figure 5.4: MSE's for SVM Regression (MSE vs. RBF Parameter)

will move with other indicators. This information is valuable for minimizing the risk of a portfolio. Most of the theoretical work on option pricing has focused on Black-Scholes option pricing model after the seminal papers of Black-Scholes[8]. The model was derived under strict assumptions. There has been a lot of research that focused on what would happen when those assumptions are relaxed [36, 55, 25, 23].

An option pricing model can be used for hedging; that is building a basket of assets in such a way that their payoff is without risk and equating the price of such baskets to that of the risk free assets. Combination of basic asset and derivatives will be included in this basket [3]. In that way, option models control the risk which is related to underlying assets.

An option is the right to buy or sell an asset in a determined time and price. Option price depends on the underlying asset price, time to maturity, volatility and risk free interest rate. A lot of research focus on how one creates a portfolio so that the risk

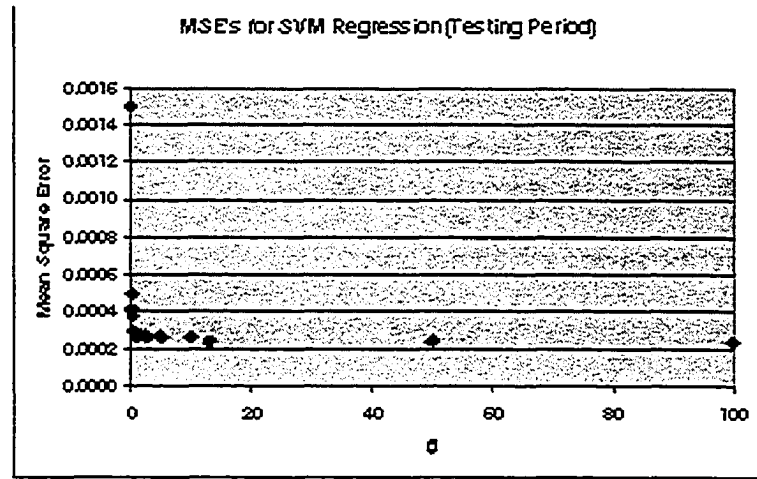


Figure 5.5: MSE's for SVM Regression (MSE vs. RBF Parameter)

can be minimized. There are several approaches for option pricing models such as binomial options and Black-Scholes option model[8, 25, 28]. We will focus on Black-Scholes option pricing model which has been applied to securities ranging from stock options to future options. It is a closed-form model that is obtained by using a dynamic hedging argument and no-arbitrage condition. Since the closed-form expression is not available in many cases, pricing formulas may still be obtained numerically. The price of the derivatives depends on the underlying asset's price dynamics. If the specification of this stochastic process of the asset's price is defined incorrectly, there could be a failure of the parametric model [29].

There has been a lot of work on non-parametric approaches the so called data-driven techniques. Some authors have proposed data-driven methods for pricing and hedging derivative securities by using minimum assumptions on the dynamics asset's price and the derivative models. Generally, these methods are called non-parametric

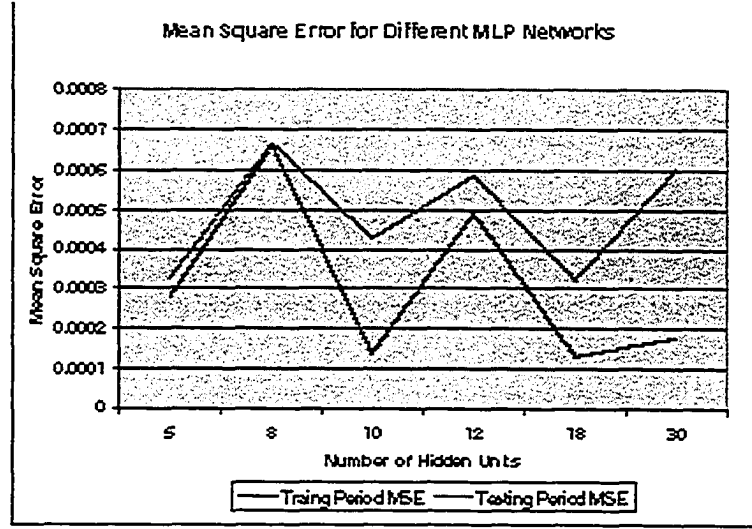


Figure 5.6: MSE vs. Number of Hidden Units for MLP Network

pricing models. For instance, radial basis function(RBF), multi-layer perceptron(MLP), projection pursuit regression(PPR) and linear regression models are used to compare B-S pricing models in [29, 3, 24, 60]. We propose another non-parametric method for option pricing which is called support vector regression(SVR). The benefit of SVR is that we solve a convex optimization problem.

### Monte-Carlo Simulations of Black-Scholes Formula

Non parametric approaches such as neural network and support vector regression can approximate complex nonlinear functions with very small errors. We are looking for an answer to the following questions as in [29]. Our contribution is that by using SVR we solve a convex optimization problem and find the optimal architecture of the resulting RBF network , while in [29] a nonconvex optimization problem is solved that does not provide necessarily the best architecture. Can the SVR technique recover/learn the B-S

	MSE (Training)	MSE (Testing)
SVR	0.0000846	0.0002357
SVR Bender's	0.000122	0.000233
MLP	0.0001317	0.0003217
RBF	0.0000779	0.0002436
ARIMA	0.0001093	0.0002379

Table 5.9: Best Models for Each Technique

formula?

The B-S option prices are generated by using Monte-Carlo simulation and trained by using SVR. The results of the SVR are compared with B-S formula to see how close the results of the method are to those of B-S formula. First, the stock price(S) is assumed to follow a geometric Brownian motion which is given by

$$dS(t) = \mu S(t)dt + \sigma S(t)dz(t) \quad (5.3)$$

where  $z$  is a Wiener process. We draw 506 pseudo random variates  $Z_t$  from the distribution  $N(\mu/253, \sigma^2/253)$  to obtain two years of daily continuously compounded returns which are converted to prices with

$$S(t) = S(0) \cdot \exp\left(\sum_{i=1}^t Z_i\right), t > 0. \quad (5.4)$$

As it can be seen from the following equation, the B-S formula is a function of the stock prices, time to maturity, the risk-free interest rate  $r$ , and the volatility of the underlying asset's continuously compounded return.



$$C(t) = S(t)\phi(d_1) + Xe^{-r(T-t)}\phi(d_2), \quad (5.5)$$

where

$$d_1 = \frac{\ln S(t)/X + (r + \sigma^2/2)(T - t)}{\sigma\sqrt{T - t}}, \quad d_2 = d_1 - \sigma\sqrt{T - t},$$

$X$  is the strike/exercise price, and  $\phi(\cdot)$  is the standard normal cumulative distribution function. We will keep  $r$  and  $\sigma$  fixed through the training process. By doing that, we reduce the number of inputs in B-S formula to two, stock price and time to maturity. One can make  $\sigma$  as an input variance by using the implied standard deviation from the observed option prices as the current estimate of the volatility [15].

Given simulated stock prices, option call prices are constructed according to CBOE rules. Expiration date is the third Friday of each month. CBOE puts each stock option on one of the cycle, which is January, February, or March. For instance, January cycle consists of January, April, July and October. If the expiration date for the current month has not been reached, options trade with the current month, next month and two months in its cycle. If the expiration date has passed, options trade with the next month, the next but one-month and next two months in its cycle. For example, Stock Y is January cycle. At the beginning of January, options are traded with January, February, April and July expiration dates. At the end of January, the expiration dates are February, March, April and July. At the beginning of June, they are traded with June, July, October, January and so on. The CBOE sets the strike prices at multiple of \$5 for stock prices in the \$25 to \$200 range. If the stock price moves outside of

the current strike-price range, another strike-price is added for all expiration dates to bracket that price. We set  $\mu = 0.10$  and  $\sigma = 0.20$ . Then,  $Z(t)$  which is normally distributed with  $(\mu/253, \sigma/253)$  is simulated for 2 years. Then  $Z(t)$  is used to compute stock price  $S(t)$  by using (5.4) where  $S(0)$  is 50. The B-S option call prices, and other variables,  $d_1, d_2$  are simulated through 2 years. Figure 5.7 shows the sample stock price path.

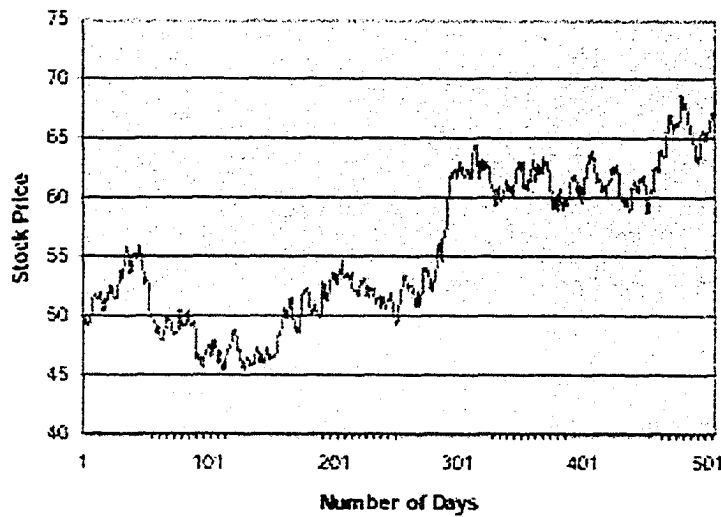


Figure 5.7: Sample path of the stock price

### Application of SVR to the B-S Formula

Since the optimization problem that we solve in SVR formulation is convex, we find a global optimal solution. Other methods, such as neural networks and RBF have a non-convex error function to minimize, and therefore the solution is not a global optimum solution. SVR has been applied to several areas such as biotechnology and meteorology.

In this paper, we try to investigate the behavior of the SVR model in option pricing. Then, we compare with MLP and RBF networks. We compare the testing error for those methods.

We consider 6000 data points generated from the B-S formula. We divide the data into two parts; the first part is used for training SVR, and it consists of 4500 data points. The second part, consisting of 1500 data points is the set of testing data.

Option call prices depend on the underlying stock price( $S(t)$ ), time to maturity, volatility( $\sigma$ ) and risk free interest rate( $r$ ). In order to keep the relation simple we assume as in [29] that the call price ( $C(t)$ ) is a function of  $S(t)$  and time to maturity ( $T-t$ ).  $C(t)$  and  $S(t)$  are normalized by dividing by  $X$ .

$$C(t)/X = f(S(t)/X, (T - t)) \quad (5.6)$$

In this experiment, we would like to see how SVR methods works for out of sample data points. The prediction is compared with the B-S call prices. We need to specify some parameters for SVR methods. Specifically, the kernel function, its parameters and trade-off constant  $C$  have to be chosen before finding the solution. Radial Basis Kernel function (RBF) with different  $\sigma$  is used and  $C$  is set to 10000. The following table shows the mean square error(MSE) and standard deviation for different  $\sigma$  values. Error is defined as the difference between the B-S call price and predicted value.

As it can be seen from table 5.10, when we increase  $\sigma$  in radial basis function, the mean square error of test data becomes smaller. Predicted call prices are very close to the B-S call price. At this point, we need to compare with other non parametric

$\sigma$	MSE	Std. Deviation
0.25	10.7311	43.6406
0.50	27.2218	22.1256
0.75	21.136	42.4486
1	1.67979	14.1565
1.5	1.00869	5.32272
3	2.74875	9.81067
7	0.443882	0.581088
50	0.014091	0.0877269
100	0.0143457	0.088257

Table 5.10: Test error for different configurations of SVR

approaches such as neural networks and radial basis networks.

	MSE	Std. Deviation
SVR	0.01409	0.0857
MLP	0.015039	0.0843
RBF	0.0145	0.0877

Table 5.11: Comparison of SVM and MLP

We have compared SVR with multi layer perceptron(MLP) and RBF. One hidden layer with different number of inputs are chosen. In this experiment, we choose a network with minimum MSE which is a  $(2 - 30 - 1)$  network structure. Then, we compare this one with the best SVR configuration. The MSE s of these two methods

are very close. However, as it can be seen from table 5.11, the SVR method provides the minimum MSE. This suggest that the SVR method is better than MLP in terms of generalization.

	MSE
Out of Money	0.04392
In the Money	0.00008
At the Money	0.01047

Table 5.12: Comparison of Error for out of money, in the money and at the money

Table 5.12 shows how SVR method behaves with different options such as out of money, in the money and at the money. With in the money option, the result of SVR is very close to exact B-S calculations. If the option is in out of money, this method makes more error. Therefore, it is safe to say that, this method can be used for in the money option.

## Chapter 6

# Summary, Conclusions and Recommendations

### 6.1 Summary

In this study, we used different decomposition technique to train SVM and SVR. The Benders Decomposition technique has been applied to regression problems. The sub-problem, arising from this algorithm has very nice structure. We showed that it can be solved without any optimization algorithms. We also applied clustering techniques to reduce the number of data points in the training set. Splitting and bagging techniques that are stable in terms of generalization are also used. Finally, for the classification problem, very efficient pre-processing algorithm has been developed and applied to various large datasets.

Comparison with other decomposition techniques (SVMFu and SVMTorch) shows that support clusters, bagging and subsampling methods are more efficient in terms

of speed. On the other hand, generalization error is same or larger for support clusters, bagging and subsampling methods.  $\gamma$ -SVM method is compared with SVMTorch.  $\gamma$ -SVM outperforms the SVMTorch in terms of speed and generalization. Since the estimation of the margin is used as a threshold,  $\gamma$ -SVM is sensitive. If we chose large threshold then the final set contains more points and if it is small then the set contains only overlapping points.

## 6.2 Recommendations for Future Research

Future research for Benders decomposition technique would be to develop an algorithm for the classification problem. The structure of the master problem in Benders decomposition method can be investigated for employing more efficient optimization algorithms. Interior point methods or cutting plane algorithm can be investigated to solve this problem. In addition to this, matrix approximation technique can be investigated to reduce the size of the kernel matrix [53, 21].

Hierarchical methods for support vector machines can be investigated in context of clustering or bagging methods. Instead of using one iteration to reduce the size of the training data, two or more iterations can be used. Theory of the clustering algorithm can be investigated for finding better partitioning algorithms.

Finally, more efficient and stable technique can be investigated for finding a good threshold for  $\gamma$ -SVM algorithm. This can be done by using the formulation of margin or radius in SVM.

# Bibliography

- [1] H. Almuallim, Y. Akiba, and S. Kaneda. On handling tree-structure attributes in decision tree learning. In *Proceedings of the Twelfth International Conference on Machine Learning*, pages 12–20. Morgan Kaufmann, 1995.
- [2] J. Aronis, V. Kolluri, F. Provost, and B. Buchanan. the world: Knowledge discovery from multiple distributed databases. In *Proceeding of Florida Artificial Intelligence Research Symposium (FLAIRS 97)*, 1997.
- [3] E. Barucci, L. Landi, and U. Cherubini. Computational methods in finance: Option pricing. *IEEE Computational Science and Engineering*, pages 66–80, 1996.
- [4] M. S. Bazaraa and H. D. Sherali. *Linear Programming and Network Flows*. John Wiley & Sons, 1990.
- [5] J. F. Benders. Partitioning precursors for solving mixed-variables programming problems. *Numer. Math.*, 4:238–252, 1962.
- [6] K. P. Bennett and C. Campbell. Support vector machines: Hype or hallelujah? *SIGKDD Explorations*, 2(2):1–6, 2000.
- [7] C. Bishop. *Neural Networks for Pattern Recognition*. Oxford UP, 1995.



- [8] F. Black and M. Scholes. The pricing options and corporate liabilities. *The Journal of Political Economy*, 81(3):637–654, 1973.
- [9] C.L. Blake and C.J. Merz. UCI Repository of machine learning databases, 1998.  
<http://www.ics.uci.edu/~mlearn/MLRepository.html>.
- [10] P.S. Bradley, U.M. Fayyad, and O.L. Mangasarian. Mathematical programming for data mining: formulations and challenges. *Journal on Computing*, 11:217–238, 1999.
- [11] P. Bradley and O.L. Mangasarian. Massive data discrimination via linear support vector machines. *Optimization Methods and Software*, 13:1–10, 2000.
- [12] L. Breiman. Bagging predictors. *Machine Learning*, 26(2):123–140, 1996.
- [13] C.J.C. Burges and B. Schölkopf. Improving the accuracy and speed of support vector machines. pages 375–381. MIT Press, Cambridge, MA.
- [14] C.J.C. Burges. A tutorial on support vector machines for pattern classification. *Data Mining and Knowledge Discovery*, 2(2):121–167, 1998.
- [15] M. Chesney and L. Scott. Pricing european currency options: A comparison of the modified black-scholes model and random variance model. *The Journal of Financial and Quantitative Analysis*, 24(3):267–284, 1989.
- [16] R. Collobert and S. Bengio. Svmtorch: Support vector machines for large-scale regression problems. *Journal of Machine Learning Research*, 1:143–160, 2001.

- [17] C. Cortes and V. Vapnik. Support vector networks. *Machine Learning*, 20:273–297, 1995.
- [18] G.B. Dantzig and P. Wolfe. The decomposition principle for linear programs. *Econometrica*, 29:767–778, 1961.
- [19] P. Domingos. Linear time rule induction. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, pages 96–101, 1996.
- [20] T. Evgeniou, M. Pontil, L. Perez-Breva, and T. Poggio. Bounds on the generalization performance of kernel machines ensembles. In *International Conference in Machine Learning*. Stanford, California, 2000.
- [21] S. Fine and K. Scheinberg. Efficient svm training using low-rank kernel representation. Technical report, IBM T.J. Watson Research Center, NY, December 11, 2000.
- [22] G. Fung and O.L. Mangasarian. Proximal support vector machine classifiers. KDD, 2001.
- [23] D. Galai. On the boness and black-scholes models for valuation of call options. *The Journal of Financial and Quantitative Analysis*, 13(1):15–27, 1978.
- [24] J. Galindo. A framework for comparative analysis of statistical and machine learning methods: An application to the black scholes option pricing equations. Technical report, Banco de Mexico, Mexico, DF, 04930, 1998.
- [25] R. Geske and R. Roll. On valuing american call options with the black-scholes european formula. *The Journal of Finance*, 39(2):443–455, 1984.

- [26] J. Gondzio, R. Sarkissian, and J.-P. Vial. Using an interior point method for the master problem in a decomposition approach. Technical report, Logilab, HEC, Section of Management Studies, University of Geneva, Switzerland, 1996.
- [27] R. Holte. Very simple classification rules perform well on most commonly used datasets. *Machine Learning*, 3:63–91, 1993.
- [28] J.C. Hull. *Options, Futures, and Other Derivative Securities*. Printice Hall, Englewood Cliffs, New Jersey, 2nd edition, 1993.
- [29] J. M. Hutchinson, A. W. Lo, and T. Poggio. A nonparametric approach to pricing and hedging derivative securities via learning networks. *The Journal of Finance*, XLIX(3):851–889, 1994.
- [30] ILOG. *CPLEX 6.5 Reference Manual*. ILOG, 1999.
- [31] T. Joachims. Making large-scale svm learning practical. In B. Schölkopf, C.J.C. Burges, and A.J. Smola, editors, *Advances in Kernel Methods: Support Vector Learning*, pages 169–184. MIT Press, 1999.
- [32] H. Kargupta and P. Chan. Kdd-98 workshop on distributed data mining. AAAI Press, 1998.
- [33] A. Demiriz K. Bennett and J. Shawe-Taylor. A column generation algorithm for boosting. In P. Langley, editor, *Proc. of 17. International Conference on Machine Learning*, pages 65–72. Morgan Kaufman, San Francisco, 2000.
- [34] Y.-J. Lee and O.L. Mangasarian. Rsvm: Reduced support vector machines. CD Proceedings of the First SIAM International Conference on Data Mining, 2001.

- [35] N. Littlestone and M. Warmuth. The weighted majority algorithm. *Information and Computation*, 108(2):212–261, 1994.
- [36] J. D. Macbeth and L. J. Merville. Tests of the black-scholes and cox call option valuation models. *The Journal of Finance*, 35(2):285–301, 1979.
- [37] O.L. Mangasarian and D.R. Musicant. Successive overrelaxation for support vector machine. *IEEE Transaction on Neural Networks*, 10:1032–1037, 1999.
- [38] O.L. Mangasarian, W.N. Street, and W.H. Wolberg. Breast cancer diagnosis and prognosis via linear programming. *Operations Research*, 43(4):570–577, 1995.
- [39] R.K. Martinson and J. Tind. An interior point method in dantzig-wolfe decomposition. *Computers and Operation Research*, 26:1195–1216, 1999.
- [40] M. Minoux. *Mathematical Programming: Theory and Algorithms*. John Willey and Sons, 1996.
- [41] A. Moore and M. Lee. Cached sufficient statistics for efficient machines learning with large datasets. *Journal of Artificial Intelligence Research*, 8:67–91, 1998.
- [42] R. Rifkin M. Pontil and T. Evgeniou. From regression to classification in support vector machines. Technical report, Massachusetts Institute of Technology, Artificial Intelligence Laboratory, 1997.
- [43] T. Oates and D. Jensen. The effects of training set size of decision tree complexity. In *Proceedings of the Fourteenth International Conference on Machine Learning*, 1997.

- [44] E. Osuna, R. Freund, and F. Girosi. Training support vector machines: An application to face detection. *Proc. Computer Vision and Pattern Recognition '97*, pages 130–136, 1997.
- [45] J. P. Pedroso and N. Murata. Support vector machines for linear programming: motivation and formulation. Technical Report 99-2.
- [46] J. Platt. Fast training of support vector machines using sequential minimal optimization. In B. Schölkopf, C.J.C. Burges, and A.J. Smola, editors, *Advances in Kernel Methods: Support Vector Learning*, pages 185–208. MIT Press, 1999.
- [47] M. Pontil and A. Verri. Properties of support vector machines. Technical report, Massachusetts Institute of Technology, Artificial Intelligence Laboratory, 1997.
- [48] M. Pontil. Stability of kernel machines and their ensembles. Technical report, Department of Information Engineering, University of Siena, Italy, 2002.
- [49] F. Provost and V. Kolluri. A survey of methods for scaling up inductive algorithms. *Machine Learning*, pages 1–42, 1999.
- [50] R. Holte P. Auer and W. Maas. Theory and applications of agnostic pac-learning with small decision trees. In A. Prieditis and S. Russell, editors, *Proceedings of the 12th International Conference on Machine Learning (ML'95)*, pages 21–29, 1995.
- [51] J. Quinlan. Learning efficient classification procedures and their application to chess endgames. In *Machine Learning: an AI approach*. Morgan Kaufmann, Los Altos, CA, 1983.

- [52] R. Rifkin. Svmfu a support vector machine package, 2000. <http://five-percent-nation.mit.edu/PersonalPages/rif/SvmFu/index.html>.
- [53] A. J. Smola and B. Schölkopf. Sparse greedy matrix approximation for machine learning. In *Proceedings of the 17th International Conference on Machine Learning*, pages 911–918. Morgan Kaufmann Publishers, 2000.
- [54] A. Smola and B. Schölkopf. A tutorial on support vector regression. Technical report, NeuroCOLT Technical Report NC-TR-98-030, Royal Holloway College, University of London, UK, 1998.
- [55] W. E. Sterk. Comparative performance of the black-scholes and roll-geske-whaley option pricing model. *The Journal of Financial and Quantitative Analysis*, 18(3):345–354, 1983.
- [56] J.P. Mesirov D. Slonim A. Verri S. Mukherjee, P. Tamayo and T.Poggio. Support vector machine classification of microarray data. Technical report, MIT Center for Biological and Computational Learning Paper 182 and MIT Artificial Intelligence Memo 1676, 1997.
- [57] T.B. Trafalis, T. Evgeniou, and H. Ince. Hierarchical methods for training support vector machines with very large datasets. In *Proceedings of the 30th International Conference on Computers and Industrial Engineering*. Tinos Island, Greece, 2002.
- [58] T.B. Trafalis and H. Ince. Support vector machine for regression and applications to financial forecasting. In *Neural Networks, 2000. IJCNN 2000, Proceedings of the*

- IEEE-INNS-ENNS International Joint Conference on*, volume 6, pages 348– 353. IEEE, 2000.
- [59] T.B. Trafalis and H. Ince. Benders decomposition technique for support vector regression. In *Neural Networks, 2002. IJCNN '02. Proceedings of the 2002 International Joint Conference on*, volume 3, pages 2767– 2772. IEEE, 2002.
- [60] R. Tsaih. Sensitivity analysis, neural networks and, the finance. In *IEEE International Joint Conference on Neural Networks*, volume 6, pages 3830–3835. IEEE, 1999.
- [61] G. Piatetsky-Shapiro U. Fayyad and P. Smyth. From data mining to knowledge discovery: an overview. In U. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, editors, *Advances in Knowledge Discovery and Data Mining*, pages 1–36. MIT Press, Cambridge, Mass., 1996.
- [62] V. Vapnik. *The Nature of Statistical Learning Theory*. Springer Verlag, 1995.
- [63] G. Zakeri, A. B. Philpott, and D. M. Ryan. Inexact cuts in benders decomposition. *SIAM J. Optim.*, 10(3):643–657, 2000.

## Appendix A

# Implementation of Benders Decomposition for Support Vector regression

```
/* This program uses the CPLEX to solve the master problem in  
Bender's Decomposition arising from SVR */
```

```
/*=====
```

```
HUSEYIN INCE
```

```
April 2002
```

```
UNIVERSITY OF OKLAHOMA
```

```
=====*/
```

```
#include <stdio.h>
```

```
#include <stdlib.h>
```



```

#include <fstream.h>

#include <iostream.h>

#include <assert.h>

#include <math.h>

#include <string.h>

#include <time.h>

#include "cplex.h"

int CplexAddrow(CPXENVptr env,CPXLPptr lp, double *gradient,
double *vd, int n); void Find_v(double *v, double *x); double
Sub1(double *v, double *gradient); int main () {

    //Initialize CPLEX main functions

    int      size;

    int iter = 1;

    double   objval,objval1,C;

    double   *x = NULL;

    double tot=0.0;

    double *c = NULL;

    double *lb = NULL;

    double *ub = NULL;

    int cur_numrows, cur_numcols;

    time_t start,end1;

    double dif;

    time (&start);

```

```

CPXENVptr      env = NULL;

CPXLPptr       lp = NULL;

static int      status = 0;

int            i,n;

env = CPXopenCPLEX(&status);

if(env == NULL)
{
    fprintf(stderr,"Could not open CPLEX environment,\n");

    //CPXgeterrorstring(env,status,errmsg);

    //fprintf(stderr,"%s",errmsg);

    goto TERMINATE;
}

lp = CPXcreateprob (env, &status, "bender_cplex");

if(lp == NULL)
{
    fprintf(stderr, "Failed to open LP,\n");

    goto TERMINATE;
}

cur_numrows = CPXgetnumrows (env, lp);

cur_numcols = CPXgetnumcols (env, lp);

cout << "\n Number of rows: " << cur_numrows << endl;

cout << "\n Number of cols: " << cur_numcols << endl;

double *v, *gradient;

```

```

cout << "\nPlease enter the number of data points: ";

cin >> n;

size = 2 * n + 2;

cout << "\nPlease enter the trade off value: ";

cin >> C;

v = (double *) malloc ((2*n) * sizeof(double));

x = (double *) malloc ((2*n + 2) * sizeof(double));

gradient = (double *) malloc ((2*n+2) * sizeof(double));

for (i= 0; i < (2*n+2); i++)
    x[i] = 0.0;

// Find the optimal solution of Sub problem given x;

Find_v(v, x); // This finds the opt. sol. of Sub problem given x.

double vd;

// Finds the current row of A matrix for

// Master problem with right hand side.

vd =Sub1(v, gradient);

gradient[size-1] = -1.0;

// Start Bender's Decomposition main loop here

tot = 0.0;

for(i = 0; i < 2*n+2; i++)

    tot = tot + gradient[i] * x[i];

// Here, initialize the CPLEX for the first time

// and write a function for adding new rows

```

```

// This part is initialization of the problem

//=====

c = (double *) malloc ((2*n+2) * sizeof(double));
lb = (double *) malloc ((2*n+2) * sizeof(double));
ub = (double *) malloc ((2*n+2) * sizeof(double));
for (i = 0; i < 2*n; i++)
{
    c[i] = 0;
    lb[i] = 0;
    ub[i] = C;
}
c[size-2] = 0;
c[size-1] = 1;
lb[size-2] = -C*1000000;
lb[size-1] = -C*1000000;
ub[size-1] = C*1000000;
ub[size-2] = C*1000000;
status = CPXnewcols(env,lp,size,c, lb, ub, NULL,NULL);
if(status) goto TERMINATE;

//=====

if ( status ) {
    fprintf (stderr, "Failed to add constraint to problem.\n");
    goto TERMINATE;
}

```

```

}

cur_numrows = CPXgetnumrows (env, lp);

cur_numcols = CPXgetnumcols (env, lp);

cout << "\n Number of rows: " << cur_numrows << endl;

cout << "\n Number of cols: " << cur_numcols << endl;

objval = -1000000000000;

objval1 = 1000000000000;

while ((fabs(objval - objval1) > 0.01) )

{

    cout << " tot - vd = " << (tot - vd) << endl;

    status =CplexAddrow(env, lp, gradient, &vd, size);

    objval1 = objval;

    status = CPXprimopt(env,lp);

    status = CPXgetobjval (env, lp, &objval);

    cout << " Objective value at iteration ;

    cout << iter << " : " << objval << endl;

    cout << " Difference at iteration ";

    cout << iter << " : " << fabs(objval - objval1) << endl;

    if ( status ) {

        fprintf (stderr, "Failed to optimize problem.\n");

        goto TERMINATE;

    }

    // Get the current optimal solution

```

```

status = CPXgetx(env,lp,x, 0, size-1);

if ( status ) {

    fprintf (stderr, "Failed to get x.\n");

    goto TERMINATE;

}

Find_v(v, x);    // Find the next solution for sub problem

// Finds the current row of A matrix of

// Master problem with right hand side.

vd =Sub1(v, gradient);

tot = 0.0;

for(i = 0; i < 2*n+2; i++)

    tot = tot +gradient[i] * x[i];

cur_numrows = CPXgetnumrows (env, lp);

cout << "\n Number of rows: " << cur_numrows << endl;

if ( status ) {

    fprintf (stderr, "Failed to add constraint to problem.\n");

    goto TERMINATE;

}

iter = iter + 1;

if(iter < 80 || fabs(objval - objval1) == 0.0)

    objval1 = 0.0;

}

status = CPXwritesol (env, lp, "myfile.txt", NULL); time (&end1);

```

```

    cout << "\nOptimal solution: \n";

    for(i = 0; i < size; i++)

        cout << x[i] << endl;

    dif = difftime (end1,start);

    printf ("CPU Time (seconds): %.2lf\n", dif );

TERMINATE:

    if ( lp != NULL ) {

        status = CPXfreeprob (env, &lp);

        if ( status ) {

            fprintf (stderr, "CPXfreeprob failed," %d.\n", status);

        }

    }

    /* Free up the CPLEX environment, if necessary */

    if ( env != NULL ) {

        status = CPXcloseCPLEX (&env);

        if ( status ) {

            char  errmsg[1024];

            fprintf (stderr, "Could not close CPLEX environment.\n");

            CPXgeterrorstring (env, status, errmsg);

            fprintf (stderr, "%s", errmsg);

        }

    }

```

```

    }

return status; }

void Find_v(double *v, double *x) {

// Solving the Sub problem arising from SVR bender's implementation

    int n, m,i,j;

    double result = 0.0;

    ifstream inFile2;

    char *filename4 = "cost.txt"; // rhs of original problem

    inFile2.open(filename4,ios::in);

    if (!inFile2)

    {

        cout<<"Unable to open "<<filename4;

        exit(1);

    }

    assert(!inFile2.eof());

    ifstream inFile1;

    char *filename3 = "matrix.txt"; // Kernel Matrix //

    inFile1.open(filename3,ios::in);

    if (!inFile1)

    {

        cout<<"Unable to open "<<filename3;

        exit(1);

    }

```



```

inFile1 >> n;

inFile1 >> m;

double *difference;

difference = (double *) malloc (n * sizeof(double));

double *gradient;

gradient = (double *) malloc ((2*n) * sizeof(double));

double *tot;

tot = (double *) malloc (n * sizeof(double));


// Computing d-Fy from bender's decomposition
//which is cost vector of sub problem then
// find the optimal solution of SUB problem
for( i = 0; i < n; i++)

    difference[i] = x[i] - x[n+i];

for(j = 0; j < n; j++)

{

    for(i=0; i < m; i++)

        inFile1 >> gradient[i] ;

    result = 0.0;

    for(i=0; i < m; i++)

        result = result + difference[i] * gradient[i];

    tot[j] = result + x[2*m];

}

```

```

    for(i= 0; i < m; i++)
    {
        v[i] = tot[i];
        v[i+m] = -tot[i];
    }

    // Finding the (d-F*y)

    double C;

    inFile2 >> C;

    for(i = 0; i < 2*m; i++)
    {
        inFile2 >> result;

        v[i] = result - v[i];

        if(v[i] < 0)
            v[i] = -C;

        else
            v[i] = 0;
    }

    inFile1.close();

    inFile2.close();

    return;
}

```

```

double Sub1(double *v, double *gradient) {

    // getting the cost vector

    int n, m,i,j;

    double result = 0.0;

    double vd = 0.0;

    ifstream inFile1;

    char *filename3 = "matrix.txt";

    inFile1.open(filename3,ios::in);

    if (!inFile1)

    {

        cout<<"Unable to open "<<filename3;

        exit(1);

    }

    inFile1 >> n;

    inFile1 >>m;

    double *difference;

    double *tot;

    difference = (double *) malloc (n * sizeof(double));

    tot = (double *) malloc (n * sizeof(double));

    // Computing (f - vF)

    for( i = 0; i < m; i++)

        difference[i] = v[i] - v[m+i];

```

```

for(j = 0; j < n; j++)
{
    for(i=0; i < m; i++)
        inFile1 >> gradient[i] ;

    result = 0.0;

    for(i=0; i < m; i++)
        result = result + difference[i] * gradient[i];

    tot[j] = result;
}

for(i= 0; i < m; i++)
{
    gradient[i] = tot[i];
    gradient[i+m] = -tot[i];
}

for(i= 0; i < 2*m; i++)
    gradient[i] = 1 - gradient[i];

result = 0.0;

for(i=0; i < n; i++)
    result =result + difference[i];

gradient[2*m] = -result;
gradient[2*m+1] = -1.0;

inFile1.close();

// Finding the vd right hand side of Master problem

```

```

    ifstream inFile2;

    char *filename4 = "cost.txt";

    inFile2.open(filename4,ios::in);

    if (!inFile2)

    {

        cout<<"Unable to open "<<filename4;

        exit(1);

    }

    assert(!inFile2.eof());

    double rhs = 0.0;

    inFile2 >> result;

    for(i= 0; i < 2*n; i++)

    {

        inFile2 >> rhs;

        vd = vd + rhs * v[i];

    }

    inFile2.close();

    return -vd;

}

int CplexAddrow(CPXENVptr env,CPXLPptr lp, double *gradient,
double *vd, int n) {

// gradient: coefficients that will be added to problem

```

```

// vd:          rhs of master problem

// n:           size of the problem or number of variables


//This contains only one variable and always zero

int *rmatbeg = NULL;

//Size of this one is equal to number of variables

int *rmatind = NULL;

// counter variable

int i, status = 0;

char sense[1];

rmatbeg = (int *) malloc (1 * sizeof(double));

rmatind = (int *) malloc (n * sizeof(double));

sense[0] = 'L';

rmatbeg[0] = 0;

for(i = 0; i < n; i++)

    rmatind[i] = i;

status = CPXaddrows (env, lp, 0, 1, n, vd, sense, rmatbeg,

                    rmatind, gradient, NULL, NULL);

if ( status ) goto TERMINATE;

TERMINATE:

return (status);

}

```

## Appendix B

# C++ implementation of e-SVM algorithm

```
//  
  
// HUSEYIN INCE  AUGUST 2002  
  
#include <stdio.h>  
  
#include <stdlib.h>  
  
#include <fstream.h>  
  
#include <iostream.h>  
  
#include <assert.h>  
  
#include <math.h>  
  
#include <string.h>  
  
#include <time.h>
```

```

int main() {

    double    *A = NULL,  *B = NULL;

    double *a = NULL, *b = NULL, *c1 = NULL, *c2= NULL;

    int *rhs1=NULL, *rhs2=NULL;

    double sigma, temp,dist1;

    int *list1, *list2;

    char filename[32],  filename2[32];

    int  i, n,m,t,k,n1,m1,j;

    time_t start,end1;

    double dif;

    // This part interacts with the user to get the required information

    cout << "\nInput File name for class 1(example: ban_pos.txt):";

    cin >> filename;

    cout << "\nInput File name for class -1(example: ban_minus.txt):";

    cin >> filename2;

    cout << "\nPlease enter the RBF kernel parameter (sigma):";

    cin >> sigma;

    ifstream inFile, inFile1;

    ofstream outFile;

    inFile.open(filename,ios::in);

    if (!inFile)

        cout<<"Unable to open "<<filename;

```



```

inFile >> n;

inFile >> m;

A = (double *) malloc ((m * n)* sizeof(double));

a = (double *) malloc (m * sizeof(double));

for (i = 0; i < m * n; i ++)

    inFile >> A[i];

inFile.close();

// For Class -1

inFile.open(filename2,ios::in);

if (!inFile)

    cout<<"Unable to open "<<filename;

inFile >> n1;

inFile >> m1;

B = (double *) malloc ((m1 * n1)* sizeof(double));

b = (double *) malloc (m * sizeof(double));

for (i = 0; i < m1 * n1; i ++)

    inFile >> B[i];

inFile.close();

list1 = (int *) malloc(n * sizeof(int));

list2 = (int *) malloc(n1 * sizeof(int));

for( i = 0; i < n; i++)

    list1[i] = 0;

for( i = 0; i < n1; i++)

```

```

        list2[i] = 0;

outFile.open("finalset.txt",ios::out);

if (!outFile)

        cout<<"Unable to open matrix.txt";

// Computing Centers for sets

c1 = (double *) malloc(m * sizeof(double));

c2 = (double *) malloc(m * sizeof(double));

for(i = 0; i < m; i++){

        c1[i] = 0.0;

        c2[i] = 0.0;

}

time (&start);

for(j = 0; j < n; j++){

        k = m * j;

        for( t = 0; t < m; t++)

                c1[t] = c1[t] + A[k+t];

}

for(j = 0; j < n1; j++){

        k = m * j;

        for( t = 0; t < m; t++)

                c2[t] = c2[t] + B[k+t];

}

for( i = 0; i < m; i++){

```

```

        c1[i] = c1[i]/n;

        c2[i] = c2[i]/n1;

    }

    dist1 = 0.0;

    for( t = 0; t < m; t++)

        dist1 = dist1 + (c1[t] - c2[t]) * (c1[t] - c2[t]);

    dist1 = dist1 / (2 * sigma * sigma);

    dist1 = exp(-1 * dist1);

    dist1 = sqrt(2 - 2* dist1);

    cout << "Distance for centers " << dist1 << endl;

    int count = 0;

    for( i = 0; i < n; i++){

        // You know this part get it from svm_cplex

        k = m * i;

        for( t = 0; t < m; t++)

            a[t] = A[k+t];

        for(j = 0; j < n1; j++){

            k = m * j;

            for( t = 0; t < m; t++)

                b[t] = B[k+t];

            temp = 0;

            for( t = 0; t < m; t++)

                temp = temp + (a[t] - b[t]) * (a[t] - b[t]);

```

```

temp = temp / (2 * sigma * sigma);

temp = exp(-1 * temp);

temp = sqrt(2 - 2 * temp);

if (temp < dist1){

    if(list1[i] == 0){

        list1[i] = 1;

        count = count + 1;

        for(k = 0; k < m; k++)

            outFile << a[k] << " ";

        outFile << 1 << endl;

    }

    if(list2[j] == 0){

        list2[j] = 1;

        count = count + 1;

        for(k = 0; k < m; k++)

            outFile << b[k] << " ";

        outFile << -1 << endl;

    }

}

cout << "Number of data points in final set: ";

count << endl;

}

```

```
    outFile.close();

    time (&end1);

    dif = difftime (end1,start);

    printf ("CPU Time (seconds): %.2lf\n", dif );

    cout << "The number of points in final set :";

    count << endl;

    cout << "Process is completeeeeeee....." << endl;

    return 0;

}
```

## Appendix C

# Bagging and Subsampling Codes in Matlab

```
\textbf{function [test_b, test_random] = bagging(trainD, testD,  
ker, C, p, split) }  
  
% This function computes the support vectors of large data sets  
% by splitting it smaller set without randomly and randomly.  
% and then finds the test results. Returns the test results  
% for each set.  
%  
% Explanation of Variables  
% trainD = Training data set. Last column must be Y values 1, or-1  
% testD = Test data without output columns
```

```

% ker    = Kernel function

% C      = Trade-off value

% p      = RBF function parameter or degree of polynomial

% split  = number of data in each split

% Output Variables

% test_b = test of each split(without random)

% test_random = test of each random split

%

% [test_b, test_random] = bagging(trainD, testD, ker, C, p, split)

%

%

% HUSEYIN INCE

% 08-15-2001

%

[n m] = size(trainD); [n1 m1] = size(testD); k = n/split; test_b =
zeros(n1,k); test_random = zeros(n1,k+1); j = 1; st = cputime; for
i = 1: k

    [nsv alpha bias] = svc(trainD(j:split*i,1:m-1),...
    trainD(j:split*i,m),ker,C,p);

    test_b(:,i) = svcoutput(trainD(j:split*i,1:m-1),...
    trainD(j:split*i,m), ...
    testD,ker,alpha,bias,p);

    j = j+split

```

```

        split*i
    end

    for i = 1: k+1

        splitD = randomsplit(trainD, split);

        [nsv alpha bias] = svc(splitD(:,1:m-1),splitD(:,m),ker,C,p);

        test_random(:,i) = svcoutput(splitD(:,1:m-1),splitD(:,m),...
            testD,ker,alpha,bias,p);

    end

    fprintf('Total execution time: %4.1f seconds\n',cputime - st);

\textbf{function [nsv, alpha, bias, final_data, predict_y] =
subsampling(trainD, testD, ker, C, p, split)}}

% This function computes the support vectors of large data sets
% by splitting it smaller set without randomly and randomly.
% and then finds the test results. Returns the test results
% for each set.
%
% Explanation of Variables
% trainD = Training data set. Last column must be Y values 1, or-1
% testD = Test data without output columns
% ker = Kernel function

```



```

% C      = Trade-off value

% p      = RBF function parameter or degree of polynomial

% split  = number of data in each split

% Output Variables

% test_b = test of each split(without random)

% test_random = test of each random split

%

% [nsv, alpha, bias, final_data, predict_y] =

% subsampling(trainD, testD, ker, C, p, split)

%

%

% HUSEYIN INCE

% 08-15-2001

%

[n m] = size(trainD); [n1 m1] = size(testD);

k = n/split; test_b =

zeros(n1,k); test_random = zeros(n1,k+1);

j = 1;

final_data = []; st = cputime; for i = 1: k

    [nsv alpha bias] = svc(trainD(j:split*i,1:m-1),...

    trainD(j:split*i,m),ker,C,p);

    a = find(alpha > abs(0.000001));

```

```

        final_data = [final_data; findsplit(a,trainD(j:split*i,:)) ];

        j = j+split;

        split*i;

    end

    if(size(final_data,1) > 1000)

        save final_data.txt final_data -ascii;

    else

        fprintf('FINAL SVM RESULTS:\n');

        [nsv alpha bias]=svc(final_data(:,1:m-1),final_data(:,m),ker,C,p);

        predict_y = svcoutput(final_data(:,1:m-1),final_data(:,m),...

        testD,ker,alpha,bias,p);

    end

    fprintf('Total execution time: %4.1f seconds\n',cputime - st);

```