UNIVERISTY OF OKLAHOMA

GRADUATE COLLEGE

Learning from Data with Uncertainty: Robust Multiclass Kernel-Based Classifiers and Regressors

A Dissertation

SUBMITTED TO THE GRADUATE FACULTY

In partial fulfillment of the requirements for the

degree of

Doctor of Philosophy

By

BUDI SANTOSA

Norman, Oklahoma

2005

UMI Number: 3170134

UMI Microform 3170134

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

Learning from Data with Uncertainty: Robust Multiclass Kernel-based Classifiers and Regressors

A Dissertation

APPROVED FOR THE

SCHOOL OF INDUSTRIAL ENGINEERING

BY

Dr Theodore B. Trafalis (Chair)

Dr Pakize Simin Pulat

Dr Randa L. Shehab

Dr Teri Reed Rhoads

Dr Michael B. Richman

Dr Tyrrel Conway

Preamble

Read! In the Name of your Lord Who has created (all that exists)
He has created man from a clot (a piece of thick coagulated blood)
Read! And your Lord is the Most Generous
Who has taught (the writing) by the pen
He has taught man that which he knew not
(Al Quran, al-'Alaq 96:1-5)

Say: Though the ocean became ink for the Words of my Lord, verily the sea would be used up before the words of my Lord were exhausted, even if we added another ocean like it, for its aid (Al Quran, 18:109)

Allah will exalt in degree those of you who believe, and those who have been granted knowledge (Al Quran, al-Mujaadilah 58:11)

Acknowledgments

I would like to thank to God (Allah) for His power to make this research possible. In performing this research and completing this dissertation, I received great help from many people. I would like to express my gratitude to Dr Theodore Trafalis as my chair advisor for his assistance, guidance, understanding and support throughout the entire program. I also like to express my thanks to Dr T. Conway, Dr P.S. Pulat, Dr R. Shehab, Dr M. Richman, and Dr T. Rhoads for serving as members of the dissertation committee. I also like to express my sincere thanks and gratitude to Dr T. Conway for his financial support during some part of my graduate program.

I also thank Dr Samir Alwazzi for fruitful discussions. A special debt of gratitude and affection is extended to my mother, my parents in law and all of my family members for their support and encouragement. Special thanks to my wife and children for their patience, support, help and encouragement during these long and stressful years of this program. Finally, I would like to acknowledge the support of the National Science Foundation under NSF Grant EIA-0205628.

Abstract

Motivated by the presence of uncertainty in real data, in this research we investigate a robust optimization approach applied to multiclass support vector machines (SVMs)and support vector regression. Two new kernel based-methods are developed to address data with uncertainty where each data point is inside a sphere of uncertainty. For classification problems, the models are called robust SVM (R-SVM) and robust feasibility approach (R-FA) respectively as extensions of SVM approach. The two models are compared in terms of robustness and generalization error. For comparison purposes, the robust minimax probability machine (MPM) is applied and compared with the above methods. From the empirical results, we conclude that the R-SVM performs better than robust MPM. For regression problems, the models are called robust support vector regression (R-SVR) and robust feasibility approach for regression (R-FAR). The proposed robust methods can improve the mean square error (MSE) in regression problems.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Overview

In real life, decision making problems are usually characterized by the presence of uncertainty. Any decision made by managers, engineers and other decision makers in any field is widely affected by the reality of uncertainty. Uncertainty in the stock market, for instance, complicates the task of prediction of the stocks price. Uncertainty is not a temporary deviation from well thought out long term plans. It is a basic structural character of the technological and business fields or environment [26]. It is very important to build a mathematical model in machine learning that incorporates the uncertainty such that our model can imitate closely the real problem.

Currently, incorporating uncertainty into a mathematical model formulation is active research in the machine learning community. Lanckriet et al.[27] developed a robust minimax probability machine (MPM) to predict the class of new observations in binary class problems. In their work, the mean and covariance matrix of the data in each class are assumed to belong in some specified set. In [17] the model that incorporates the uncertainty of the data is explored in a different way. The uncertainty of the data is characterized by interval uncertainties of the data within given hyper-rectangles. The problem then is addressed by minimizing the

worst-case value of a given loss function, over all possible choices of the data in the multi-dimensional intervals. Trafalis and Alwazzi [36] proposed a robust support vector machine (SVM) classifier that studies noisy data with bounded errors on the linear model of SVM. Their work investigated how the stability of the solution is affected by the noise of the data. In this research, a robust support vector machine approach is proposed which can improve the generalization error. The motivation is to increase the margin of separation by introducing noise. Different from the previous work, this research emphasizes how the generalization error improves with the data perturbation. The difference of our approach from Trafalis and Alwazzi [36] is that in their approach, the margin of separation decreases with the increase of the noise level and it approaches zero as the radius of the uncertainty sphere becomes equal to the margin. In our case, the margin increasing as as the level of uncertainty is increasing.

Street and Mangasarian [34] proved that the generalization error is improved when the training set is learned with less accuracy. They developed a linear model and trained it with several degrees of tolerances τ to investigate the influence of the noise to the test generalization. Their experiments on linear systems using nine sets of real-world data confirmed the improved generalization error.

1.2 Research Objectives

Robust optimization techniques recently have attracted several researchers. The reason is that there are some contradictions between the real-world data and the realm of traditional deterministic mathematical programming. Therefore, combination of these two issues becomes necessary. When operation researchers try to construct a model of a real-world system, they always find incomplete, noisy or uncertain data. On the other hand, in the world of mathematical programming, it is assumed that the model is deterministic, something that does not hold generally in the real world. It has been found that large error bounds arise when one solves mean value problems [8]. In this research, a robust optimization approach

applied to support vector machines is investigated. The motivation is to find a classifier that is "immune" against data uncertainties and has good generalization properties. Two new kernel based-methods are developed to handle the data with uncertainty where the data is inside a sphere. The models are called robust quadratic SVM (R-SVM) and robust feasibility-approach model (R-FA).

First, the models are developed for the binary class problem in linear cases. Building on ideas from [36] for data with bounded errors, the new models are built with the objective of being robust and improving the generalization error by perturbing the data with bounded perturbations. Next, the models are extended to nonlinear cases by utilizing the kernel method [31] and to more general multiclass problems. Finally, the results between the two models are compared to check which model is better in terms of robustness and generalization error. Comparison with other robust optimization methods such as the robust minimax probability machine is performed. The summary of model development in this research is depicted in Figure 1.1.

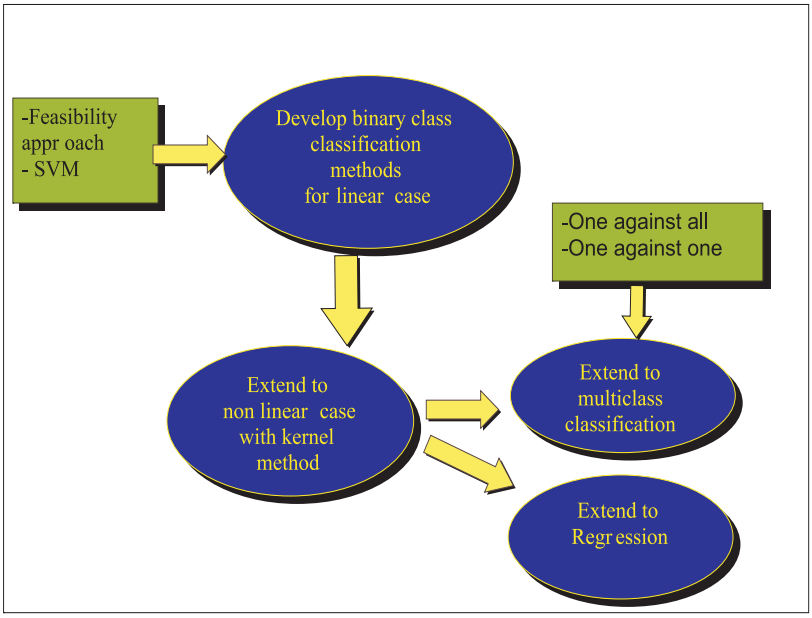


Figure 1.1: Model development process

1.3 Organization of the Dissertation

Chapter 2 describes some important concepts in machine learning. In chapter 3, a literature review on robust optimization, robust classification methods and multiclass SVM are provided. Chapter 4 provides mathematical formulation of the proposed models and the algorithm to solve the models. In chapter 5, the extension of the models into multiclass cases are described. Chapter 6 describes the extension of the models into regression analysis. Chapter 7 concludes the dissertation.

Chapter 2

Basic Concepts in Machine Learning

2.1 Binary Classification

Assume that we are given a set of training data $(x_i, y_i),\ i = 1, .., \ell$ with input data $X = \{x_1, x_2, .., x_\ell\} \subseteq \Re^N$and corresponding outputs $Y = \{y_1, .., y_\ell\} \subseteq \{\pm 1\}^\ell$. The goal of binary classification is to find a decision function *f(x)* that accurately predicts the class of *test* or *future* data points (x, y) coming from the same distribution function as that of the *training* data. This set of ℓ points $(x_i, y_i)\,, i = 1, .., \ell$ usually is called the training set, where the x_i correspond to the input parameters and the y_i refers to the output parameters. To solve this problem we can start from linear decision functions. The decision function can be represented as:

$$\begin{aligned} g(x) &:= sgn(f(x)) \\ \text{with } f(x) &= w^T x + b, \end{aligned} \tag{2.1}$$

where $x,\ w \in \Re^n$ and $b \in \Re$. The problem of learning from data can be stated as finding a set of (w, b) such that $f(x_i) =< w, x > +b = y_i$ for all i. An algorithm to cope with such a problem is what we call the perceptron. This iterative algorithm originally is designed to solve simple linear separable problems. A more difficult problem arises when the data are not linearly separable. In such a case, we have to find a nonlinear decision function to separate the data correctly. Working in the

feature space can simplify the classification task into a linear separation problem. The idea is to use the kernel method that we discuss in section 2.2.

2.2 The Kernel Method

Most machine learning algorithms are developed with the assumption of linearity. Then, the resulting algorithms are limited to linear discriminant functions. Hence, if for example a certain classification problem displays a nonlinear separating surface, algorithms such as the perceptron will not be able to account for this nonlinear behavior. In general, complex real-world problems require more expressive hypothesis spaces than linear functions. Kernel methods [31] offer an alternative solution by mapping a data point x in the input space into a higher dimensional feature space F through a feature map φ such that $\varphi : x \mapsto \varphi(x)$. Therefore the point x in the input space becomes $\varphi(x)$ in the feature space.

Unfortunately, very often the function $\varphi(x)$ is not available, can not be computed, or does not even exist. However, the dot product of two vectors can be computed, both in the input and feature space. In other words, while $\varphi(x)$ might not be available, the dot product $< \varphi(x_1), \varphi(x_2) >$ can still be computed in the feature space. In order to employ the kernel method, it is necessary to express the separation constraints in terms of inner products of the data vectors x_i. Consequently, the constraints describing the classification problem have to be reformulated, such that solely dot products are used. In the new space the dot product $< . >$ becomes $< \varphi(x), \varphi(x)' >$. A nonlinear kernel function, $k(x, x')$, can be used to substitute the dot product $< \varphi(x), \varphi(x)' >$. Then in the higher dimensional feature space, we can construct a linear decision function that represents a nonlinear function in the input space. Figure 2.1 describes an example of a feature mapping from a two dimensional input space to a two dimensional feature space. In the input space the data can not be separated linearly, but we can do so in the feature space. Hence by mapping the data into a feature space the classification task becomes simpler.

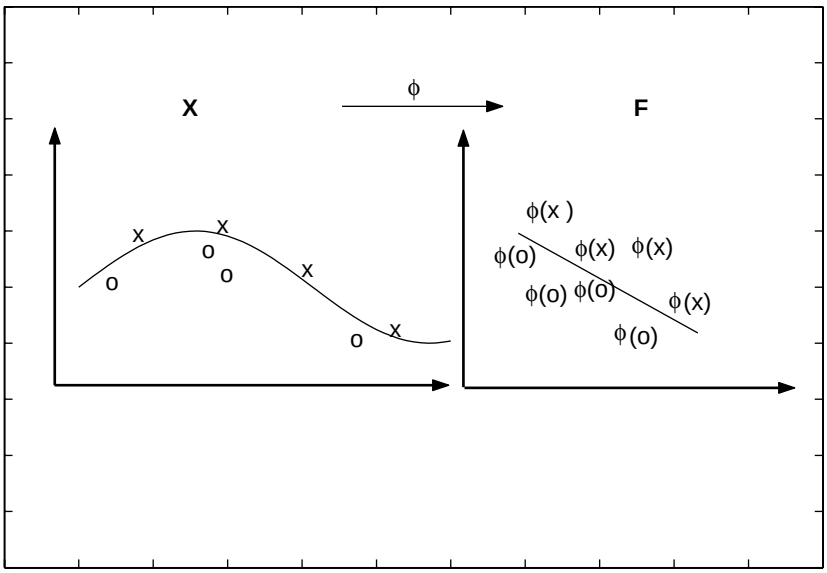


Figure 2.1: A kernel map converts a nonlinear problem into a linear problem

There are three nonlinear kernel functions usually used in the SVM literature [23]:

1. polynomial: $(x^T x_i + 1)^p$,
2. radial basis function (RBF): $exp(-\frac{1}{2\sigma^2}\|x - x_i\|^2)$,
3. tangent hyperbolic (sigmoid): $tanh(\beta x^T x_i + \beta_1)$, where $\beta, \beta_1 \in \Re$

The best kernel function which one can use to substitute for the dot products in the feature space depends on the data; usually one has to use cross-validation methods [22] to select the best kernel function.

2.3 Perceptron Algorithm

Frank Rosenblat's perceptron is a model aimed to solve visual perception tasks or to perform pattern recognition tasks [25]. The perceptron is based on the steepest descent method [1] and is used in the case of linear separability. The summary of the perceptron algorithm is given below [33].

Perceptron Algorithm

argument: training sample , $x = \{x_1, x_2..x_m\} \subset X$, $y = \{y_1, ..y_m\} \subset \{\pm 1\}$

learning rate : η

return: weight vectors w and threshold b

function perceptron(X,Y,η)

Initialize w,b=0

Repeat

for all i from i=1..,m

Compute $g(x_i) = sgn((w.x_i) + b)$

Update w, according to

$w' = w + \frac{\eta}{2}[y_i - g(x_i)]x_i$

$b' = b + \frac{\eta}{2}[y_i - g(x_i)]$

endfor

until for all $1 \leq i \leq m$ we have $g(x_i) = y_i$

return $f : x \mapsto (w.x) + b$

end

The perceptron is iterative in the sense that small changes are made iteratively to the weight vector in response to each given labeled example. The weight vector (w, b) is updated on a labeled example if only the (w, b) misclassifies this example. The geometry of the perceptron algorithm is described in Figure 2.2. It can be generalized in the case of nonlinear separability using the concept of the kernel function [32]. However it can not give the best decision function since it computes local minima. An effective method that computes global optima is described in the next section.

2.4 Support Vector Machines

A well known method in machine learning to find an optimal classifier (hyperplane) between two sets of points is the so called SVM [39]. This method has attracted

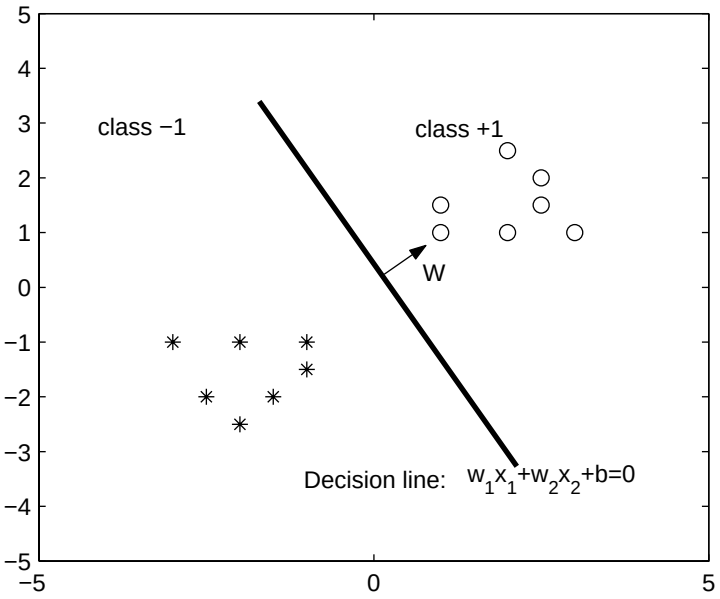


Figure 2.2: Geometric interpretation of Perceptron classifier

people in the machine learning and optimization community because of its impressive performance in generalization error of unseen data. In this method one seeks the best hyperplane among many possible hyperplanes to separate two sets of patterns. The optimal hyperplane is the one that is located mid-way between the two classes. This hyperplane is orthogonal to the shortest line connecting the convex hulls of the two classes. Seeking the best hyperplane is equivalent to maximizing the margin between the two classes. If $wx_1 + b = +1$ is on the supporting hyperplane of class $+1$ ($wx_1 + b = +1$) and $wx_2 + b = -1$ is on the supporting hyperplane of class -1 ($wx_2 + b = -1$), the margin between the two classes can be computed by computing the distance between the supporting hyperplanes of those classes. Specifically, the margin is computed as follows $(wx_1 + b = +1) - (wx_2 + b = -1) \Rightarrow w(x_1 - x_2)) = 2 \Rightarrow \left(\frac{w}{\|w\|}(x_1 - x_2)\right) = \frac{2}{\|w\|}$. Figure 2.3 shows how SVM works in finding a classifier with maximum margin. We need to show that maximizing the margin between the two set of points will increase the probability of correct classification of the testing points. Generally

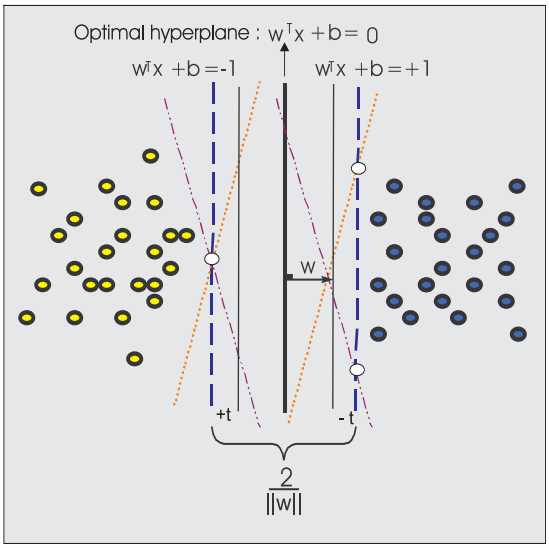


Figure 2.3: The idea of an optimal hyperplane for linear separable examples

there is an infinite number of separating hyperplanes. Suppose from this infinite number of separating hyperplanes, we investigate two hyperplanes $f_1(x)$ and $f_2(x)$ (see Figure 2.4). Hyperplane f_1 has a larger margin than hyperplane f_2. After finding these two hyperplanes, now a new testing point with label -1 is coming to our system. We have to classify this new point whether it is in class -1 or $+1$ using the hyperplanes that we obtained. Using f_1, we will classify this new point as being in class -1 which is classified correctly. Now, using f_2, we will classify the new point as being in class $+1$ which is incorrect. From this simple example, we see that increasing the margin can increase the probability of correct classification.

The mathematical formulation of the SVM optimization problem for the linear separable case is given as

$$\min \frac{1}{2} \|w\|^2 \tag{2.2}$$

Subject to

$$y_i(wx_i + b) \geq 1, \; i = 1, .., \ell.$$

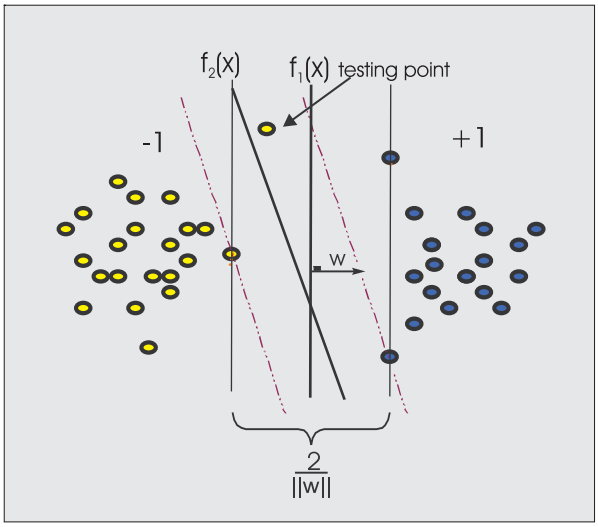


Figure 2.4: Increasing margin can increase the probability of correct classification

In the case of linear non-separable problems, the formulation of the SVM optimization problem is given as

$$\min \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{\ell} t_i \tag{2.3}$$

Subject to

$$y_i(wx_i + b) + t_i \geq 1$$

$$t_i \geq 0,\ i = 1, .., \ell.$$

By this formulation one wants to maximize the margin of separation of two classes by minimizing $\|w\|^2$ [23]. Other formulations based on LP are described in [4]. In all of those formulations one needs to minimize the misclassification errors that are described by the slack variables t_i while maximizing the margin. The slack variable t_i is used to handle the case of infeasibility of hard constraints $y_i(wx_i + b) \geq 1$ by penalizing points that do not satisfy the hard constraints. To minimize such deviations, we penalize those through a regularization constant C. The vector w is the normal to the separating hyperplane: $wx + b = 0$. The constant b determines its location relative to the origin.

To address the problem of nonlinearity that frequently occurs in real world problems, one can utilize kernel methods. Kernel methods [31] provide an alternative approach by mapping data points x in the input space into a higher dimensional feature space F through a map φ such that $\varphi : x \mapsto \varphi(x)$. Therefore a point x in the input space becomes $\varphi(x)$ in the feature space. The dual formulation of problem (2.3) is expressed in the feature space:

$$\min \frac{1}{2} \sum_{i,j=1}^{\ell} y_i y_j \alpha_i \alpha_j k(x_i, x_j) - \sum_{i=1}^{\ell} \alpha_i \tag{2.4}$$

Subject to

$$0 \leq \alpha_i \leq C, i = 1, ..\ell$$

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

where k is the kernel function described in section 2.2. The formulation in (2.4) is a linearly constrained quadratic programming. Training SVM is equivalent to solving the above convex optimization problem. Therefore the solution of SVM is unique (under the assumption that k is positive definite) and globally optimal, unlike other networks' training [23] which is equivalent to a nonconvex optimization problem with the danger of obtaining local optima solutions. Let $f(x) = \sum_{i=1}^{\ell} y_i \alpha_i^* k(x_i, x) + b^*$. The resulting optimal classifier is $g(x) = sign(\sum_{i=1}^{\ell} y_i \alpha_i^* k(x, x_i)) + b^*$, where $\alpha_i^*, i = 1, .., \ell$ are the optimal solutions of problem (2.4) and b^* is chosen so that $y_i f(x_i) = 1$ for any i with $C > \alpha_i^* > 0$ [15]. The points x_i for which $\alpha_i^* > 0$ are called *support vectors* and represent the training data points that are needed to represent the optimal decision function. In Figure 2.3, for example, the 3 white points represent the support vectors.

Chapter 3

Literature Review

3.1 Robust Optimization

The robust optimization methodology is a relatively new approach to deal with uncertain data. More recently, the so called robust optimization techniques have been investigated by several authors [3, 2, 10, 18, 19]. Those techniques are more meaningful in formulations with prior bounds on the size of the uncertainties on the data. Specifically, we consider the case where we have data with bounded errors. The solutions coming from robust optimization models are more stable and more appropriate for this kind of uncertainty.

Ben-Tal and Nemirovski [3] proposed the foundation of robust convex optimization based on previous work in robust control. Their assumption is that the data defining a convex optimization problem are not accurately specified, and the only knowledge about those is that they belong to a bounded uncertainty set U. The formulation of the resulting optimization problem is as follows

$$\begin{aligned} &\min f(x,\xi) \\ &\text{Subject to} \\ &g_i(x,\xi) \leq 0 \end{aligned} \tag{3.1}$$

$$x \in \Re^n, i = 1, .., \ell \ \forall \xi \in U \subseteq \Re^n,$$

where f, g_i are convex functions and ξ is an uncertain parameter vector.
They have shown, that when this set U is an ellipsoidal uncertainty set, then the robust convex program corresponding to some of the most important generic convex problems, such as linear programming, semi-definite programming and others, is a convex optimization problem which can be solved by an efficient algorithm, such as polynomial time interior point methods. More recently, Bertsimas et al.[7] have provided robust formulations with a similar complexity as that of the original uncertain optimization problem.

3.2 Robust SVM Classifier

In [36, 37], a robust SVM classifier development is described. This research assumes noisy data with bounded errors on the linear programming (LP) SVM formulation. Specifically, this approach assumes that a data point can be represented through a sphere with a known radius. Accordingly, the supporting hyperplane resulting from the model will be on the boundary of the sphere that contains the data closest to the separating hyperplane (classifier) in one side and on the boundary of the sphere from the separating hyperplane in the other side. In other words, the training data points (represented through the centers of the corresponding uncertainty spheres) can be modified through a new set of data points that are obtained by shifting the points labeled as $+1$ along $-w$ and the points labeled -1 along w, respectively to its boundary of uncertainty (see Figure 3.1).
The optimization problem formulation is given as

$$\min_{w,b,t} \|w\|_1 + C \sum_{i=1}^{\ell} t_i \tag{3.2}$$

Subject to

$$y_i \langle w, \tilde{x_i} \rangle - \sqrt{\eta} \|w\| + y_i b + t_i \geq 1$$

$$t_i \geq 0, \ i = 1..\ell,$$

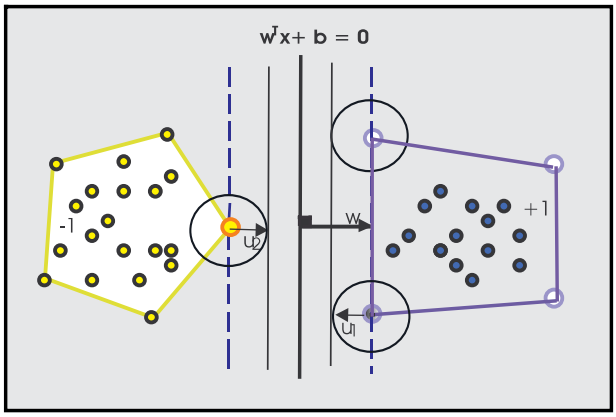


Figure 3.1: Geometric illustration of robust SVM using the approach in [36, 37]

where $\sqrt{\eta}$ is the radius and $\tilde{x}_i$ is the center of the uncertainty sphere. Setting $w = \sum_{i=1}^{\ell} y_i \alpha_i x_i$ and linearizing the objective function, the above problem can be formulated as follows:

$$\min \sum_{i=1}^{\ell} \alpha_i + C \sum_{i=1}^{\ell} t_i \tag{3.3}$$

Subject to

$$\sqrt{\eta}\sqrt{\alpha^t \tilde{k} \alpha} - y_i \sum_{i=1}^{\ell} y_j \alpha_j k(\tilde{x_j}, \tilde{x_i}) - y_i b - t_i + 1 \leq 0$$

$$t_i \geq 0, \alpha_i \geq 0, \ i = 1, .., \ell,$$

where $\tilde{k} = \tilde{k}(x_i, x_j) = y_i y_j < x_i, x_j >$. It is shown that the resulting SVM classifier is *robust* to the noise of the data [36, 37].

3.3 Robust Minimax Probability Machines (MPM)

The MPM approach was introduced by Lanckriet et al.[27]. The objective of this approach is to solve the problem of binary classification by minimizing the maximum probability of misclassification of the future data points. The problem can be defined as follows. Let x and y denote random vectors with x in class one and y in class two, with means and covariance matrices given by $(\overline{x}, \Sigma_x)$ and $(\overline{y}, \Sigma_y)$ respectively, where $x, \overline{x}, y, \overline{y} \in \Re^n$ and $\Sigma_x, \Sigma_y \in \Re^{nxn}$ and both are symmetric

and positive semidefinite. Then, a hyperplane $H(a,b) = \{z|a^T z = b\}$, where $a \in \Re^n \backslash \{0\}$ and $b \in \Re$ which separates the two classes of points with maximal probability with respect to all distributions having these mean and covariance matrices is determined. Mathematically the resulting optimization problem can be formulated as:

$$\begin{aligned} &\max_{\alpha, a \neq 0, b} \alpha \\ &\text{Subject to} \\ &\inf_{x \sim (\overline{x}, \Sigma_x)} Pr\left\{a^T x \geq b\right\} \geq \alpha \\ &\inf_{y \sim (\overline{y}, \Sigma_y)} Pr\left\{a^T y \leq b\right\} \geq \alpha \end{aligned} \tag{3.4}$$

After some manipulations by exploiting the powerful theorem by Popescu and Bertsimas [28] that states that:

$$\sup_{y \sim (\overline{y}, \Sigma_y)} Pr\left\{y \in S\right\} = \frac{1}{1+d^2}, \tag{3.5}$$

with $d^2 = \inf_{y \in S}(y - \overline{y})\Sigma_y^{-1}(y - \overline{y})$, the above problem can be simplified as [27]:

$$\begin{aligned} &\min_{\alpha} \left\| \Sigma_x^{\frac{1}{2}} a \right\|_2 + \left\| \Sigma_y^{\frac{1}{2}} a \right\|_2 \\ &\text{Subject to} \\ &a^T(\overline{x} - \overline{y}) = 1 \end{aligned} \tag{3.6}$$

and setting b to the value

$$b_* = a_*^T \bar{x} - \kappa_* \left\| \Sigma_x a \right\|, \tag{3.7}$$

where $\kappa(\alpha) = \sqrt{\frac{\alpha}{1-\alpha}}$ and a_*, κ_* are the optimal parameters. The geometric interpretation of MPM binary class classification problems is shown in Figure 3.2.

The robustness of this approach is created by giving the estimation of mean and covariance of each class of the data. It is assumed that the mean and covariance matrix of each class are only known within some specified set. In particular, it is assumed that $(\bar{x}, \Sigma_x) \in X$, where X is a subset of $\Re^n \times S_n^+$, where S_n^+ is the set

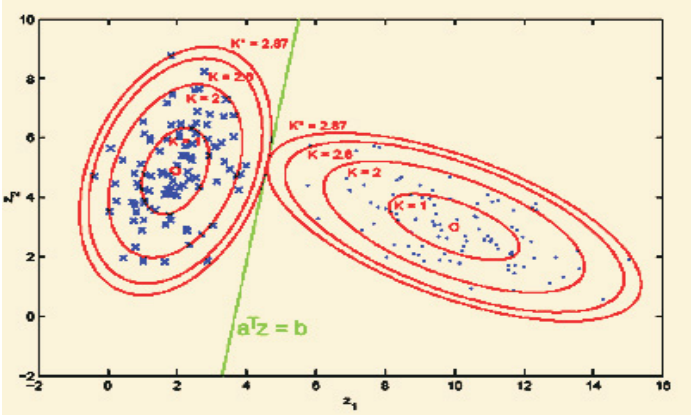


Figure 3.2: MPM classifier for binary-class classification problem [27]

of $n \times n$ symmetric, positive semidefinite matrices. Likewise a set Y is describing uncertainty of the mean and covariance matrix of the random variable y. The robust version of MPM is formulated as:

$$\max_{\alpha, a \neq 0, b} \alpha \tag{3.8}$$

Subject to

$$\inf_{x \sim (\overline{x}, \Sigma_x)} Pr\left\{a^T x \geq b\right\} \geq \alpha \forall (\bar{x}, \Sigma_x) \in X$$

$$\inf_{y \sim (\overline{y}, \Sigma_y)} Pr\left\{a^T y \leq b\right\} \geq \alpha \forall (\bar{y}, \Sigma_y) \in Y$$

Geometric interpretation of robust MPM for a binary class classification problem is shown in Figure 3.3. In the case of nonlinear separability, the kernel method is used by mapping all the data into a higher dimensional feature space. In the feature space, we want to find a hyperplane $H(a,b) = \{\varphi(z) | a^T \varphi(z) = b\}$ that corresponds to a nonlinear decision boundary $D(a,b) = \{z \in \Re^n | a^T \varphi(z) = b\}$ in the input space $\Re^n$ ($a \in \Re^n \setminus \{0\}$ and $b \in \Re$).

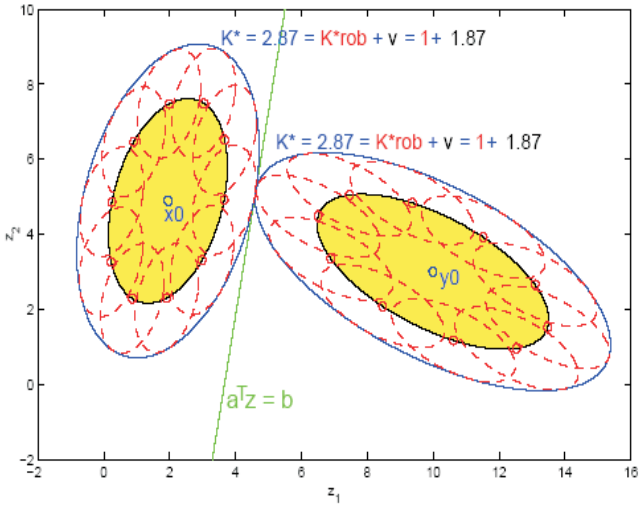


Figure 3.3: Robust MPM classifier for binary-class classification problem [27]

3.4 Robust Classification with Interval Data

El Ghaoui et al.[17] developed a binary linear classification model to cope with the problems where the data points are assumed to be unknown, but bounded within given hyper-rectangles, i.e. the covariates are bounded within intervals explicitly given for each data point separately. Let X denote a $n \times N$ matrix of N nominal data points $x_i \in \Re^n$, with corresponding label vector $y \in \{-1, +1\}^N$, Σ is a $n \times N$ matrix of positive numbers, with columns $\sigma_i, i = 1, .., N$. The data then can be expressed in an interval matrix model as: $X(\rho) = \{Z \in \Re^{n \times N} : X - \rho\Sigma \leq Z \leq X + \rho\Sigma\}$, where inequalities are understood componentwise. The standard error matrix Σ reflects the amplitude of the uncertainty for every covariate.The scalar ρ is a global measure of uncertainty. The problem then is addressed by designing a robust classifier, $w^T b + b$. In this setting the worst-case value of a given loss function, over all possible choices of the data in these multi-dimensional intervals is minimized. To measure the performance of the classifier on the uncertain training data, the *robust loss function* λ, is introduced, which depends on the classifier parameters w, b, the uncertain training set $X(\rho)$, as well as on the label vector y.

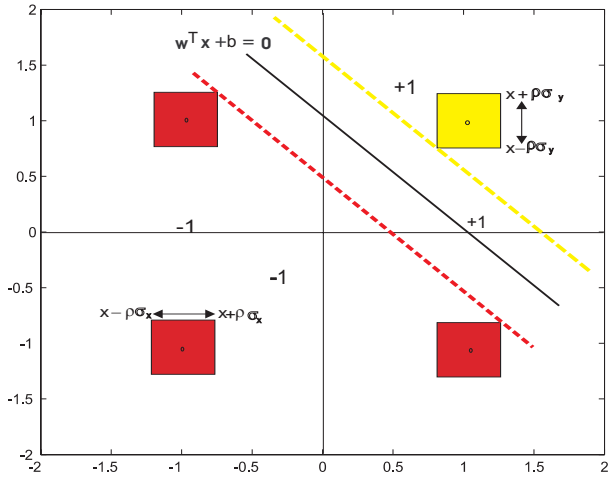


Figure 3.4: Linear classifier for AND problem with interval data

The linear programming SVM problem formulation on Hinge loss function [17] is given as:

$$\lambda_{SVM}(w,b) = \max_{Z \in X(\rho)} \sum_{i=1}^{N} (1 - y_i(w^T z_i + b))_+ = \sum_{i=1}^{N} (1 - y_i(w^T x_i + b)) + \rho \sigma_i^T |w|_+), \tag{3.9}$$

where Hinge loss function is defined as:

$$L_{SVM}(w,b,Z,y) = \sum_{i=1}^{N} (1 - y_i(w^T z_i + b)). \tag{3.10}$$

In their paper [17], two different formulation of robust models are developed by applying *logistic regression* and *MPM loss functions*.

Figure 3.4 depicts the classification problem with interval data for the AND problem.

3.5 Multi-Class Support Vector Machines

Originally SVMs were designed for binary classification. However, how to extend the SVM model effectively to multiclass classification is still an on-going research issue [24]. Currently, there two main approaches for multi class SVM. One is

by constructing and combining several binary classifiers. The other one is by directly considering all the data into a single optimization formulation. The first approach where several binary classifiers are constructed and combined includes two methods: One-against-all (OAA), and One-against-one (OAO) [24]. Some research utilizing the second approach is proposed by Vapnik [40], Weston and Watkins [41], Bredensteiner and Bennet [11], Crammer and Singer [14].

3.5.1 One-against-all (OAA) Method

By this method, for k-class classification problems, we construct k classifiers where k is the number of classes. Let's call our classifier ρ. In this method, ρ^i is trained with all of the examples in the ith class with positive labels +1) and all other examples with negative labels (-1).

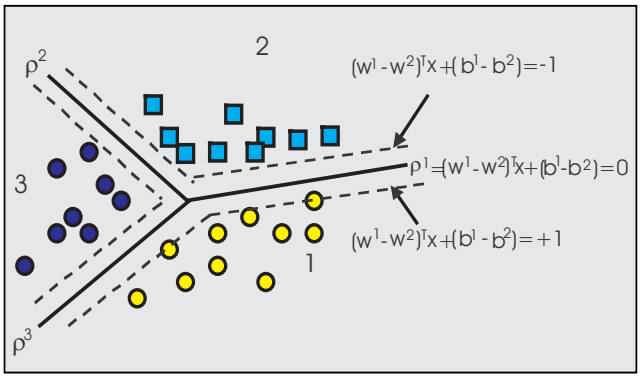


Figure 3.5: SVM classifier for three-class classification problem

In a three-class classification problem as shown in Figure 3.5, for example, when we train ρ^1, all points in class 1 are labeled with +1 and the other points from class 2 and 3 are labeled with -1. Likewise, when we train ρ^2 all points in class 2 we label with +1 and all other points from class 1 and 3 are labeled with -1. We do this for all $i = 1, 2, 3$. Given ℓ training data $(x_1, y_1), .., (x_\ell, y_\ell)$ where $x_i \in R^n, i = 1, 2, .., \ell$ and $y_i \in S = \{1, .., k\}$ is the class of x_i, the ith classifier

solves the following optimization problem [24]

$$\min_{w^i,b^i,t^i_j} \frac{1}{2}(w^i)^T w^i + C\sum_{j=1}^{\ell} t^i_j \tag{3.11}$$

Subject to

$$w^i\varphi(x_j) + b^i \geq 1 - t^i_j,\ if\ y_j = i$$

$$w^i\varphi(x_j) + b^i \leq -1 + t^i_j,\ if\ y_j \neq i$$

$$t_j \geq 0, j = 1, .., \ell, i = 1, .., k$$

After solving (3.11), there are k decision functions $w^1\varphi(x)+b^1, w^2\varphi(x)+b^2, .., w^k\varphi(x)+b^k$. Then, the class of point x is determined by the largest value of the decision function:

$$j = class\ of\ x = arg \max_{i=1,..,k} w^i\varphi(x) + b^i, where\ j \in S \tag{3.12}$$

Practically, we solve the dual problem of (3.11) as in (2.4) whose number of variables is the same as the number of data in the problem. Hence k l-variable quadratic programming problems are solved.

3.5.2 One-against-One (OAO) Method

This method constructs $k(k-1)/2$ classifiers where each one is trained on data from two classes. For example, if we have a three-class classification problem we have to construct 3 classifiers: ρ^{12}, ρ^{13} and ρ^{23}. When we train ρ^{12}, all points from class 1 are labeled with $+1$ and all points from class 2 are labeled with -1. The same approach is applied when we train ρ^{13} and ρ^{23}. For training data from ith and jth classes, we solve the following binary classification problem [24]:

$$\min_{w^{ij},b^{ij},t^{ij}} \frac{1}{2}(w^{ij})^T w^{ij} + C\sum_{r} t^{ij}_r \tag{3.13}$$

Subject to

$$w^{ij}\varphi(x_r) + b^{ij} \geq 1 - t^{ij}_r,\ if\ y_r = i$$

$$w^{ij}\varphi(x_r) + b^{ij} \leq -1 + t^{ij}_r,\ if\ y_r = j$$

$$t^{ij}_r \geq 0,$$

where r is referring to the data point index for each class. After all classifiers $k(k-1)/2$ are constructed, there are different methods for doing the future testing. One strategy is max-vote. Based on this strategy, for classifier ρ^{ij}, if the sign of a new point x is in the ith class, then the vote for the ith class is added by one. Otherwise, the vote for the jth class is increased by one. We repeat this step for all classifiers. Then, we predict x as being in the class with the largest vote. In the case where two classes have identical votes, we select the one with the smallest index. Practically, we solve the dual problem of (3.13) as in (2.4) whose number of variables is the same as the number of data in the two classes. Hence if on the average, each class has l/k data points, we have to solve $k(k-1)/2$ quadratic programming problems where each of them has $2l/k$ variables.

3.5.3 One Optimization Problem

The second approach of multiclass SVM is the one that considers all the data in one optimization problem. Weston and Watkins [41] proposed the following problem formulation to solve multiclass-SVM:

$$\min \frac{1}{2}\sum_{m=1}^{k} w_m^T w_m + C\sum_{i=1}^{\ell}\sum_{m\neq y_i}^{\ell} t_i^m \tag{3.14}$$

Subject to

$$w_{y_i}^T x_i + b_{y_i} \geq w_m^T x_i + b_m + 2 - t_i^m$$

$$t_i^m \geq 0, i = 1..,\ell\ m \in \{1,..,k\}\backslash y_i$$

This gives the decision function:

$$f(x) = arg\ \max_k\ [w_i^T x + b_i],\ i = 1,..,k \tag{3.15}$$

After some manipulations, we have the following dual problem to solve:

$$\max\ 2\sum_{i,m}\alpha_i^m + \sum_{i,j,m}[-\frac{1}{2}y_i y_j + \alpha_i^m \alpha_j^{y_i} - \frac{1}{2}\alpha_i^m \alpha_j^m](x_i, x_j) \tag{3.16}$$

Subject to

$$\sum_{i=1}^{\ell}\alpha_i^n = \sum_{i=1}^{\ell} c_i^n A_i,\ n = 1,..,k$$

$$0 \leq \alpha_i^m \leq C, i = 1, ..\ell$$
$$\sum_{i=1}^{\ell} \alpha_i^{y_i} = 0$$
$$i = 1.., \ell \; m \in \{1, .., k\} \backslash y_i,$$

where $A_i = \sum_{i=1}^{k} \alpha_i^m$ and $c_i^n = \begin{cases} 1 \; if \; y_i = n \\ 0 \; if \; y_i \neq n \end{cases}$

This gives the decision function:

$$f(x) = arg \; \max_n [\sum_{i:y_i=n} (c_i^n A_i - \alpha_i^n)(x_i^T x) + b_n] \tag{3.17}$$

As usual the inner products $(x_i^T x)$ in (3.17) and (3.16) can be replaced by the kernel $k(x_i, x_j)$.

In a different way, Bredensteiner and Bennet[11] proposed multicategory classifiers using SVMs. Two new hybrid approaches, k-Robust Linear Programming (k-RLP) and multi category-SVM (M-SVM), are developed as a combination of Multicategory Discrimination Method(M-RLP) as described in detail in [5, 6] and SVM. M-RLP constructs a piecewise-linear discriminant for a k-class problem using a single linear program. k-RLP uses one-against all method to construct k RLP-classifiers. Like in Robust Linear Programming (RLP) [5, 6], M-RLP does not include any terms for maximizing the margin and it does not directly permit the use of generalized inner products or kernels to allow extension to the nonlinear case. M-SVM is developed by adding regularization terms to M-RLP.

In [14], Crammer and Singer proposed a direct method for training multiclass classifiers. By using the dual of the optimization problem, kernels are incorporated with a compact set of constraints and decompose the dual problem into a set of multiple optimization problems of reduced size. An efficient fixed-point algorithm for solving the reduced optimization problems is described and its convergence is proved.

Chapter 4

Robust Kernel Feasibility-Approach and Robust SVM

4.1 Feasibility-Approach as An Optimization Problem

In this chapter and the next two chapters, we develop our models. We begin with the robust feasibility approach (R-FA) for binary classification problems.Then, this is followed by robust support vector machine (R-SVM). In the next chapter, we extend our models to multiclass classification problems. The next step is extending our models for function approximation or regression problems.

As the SVM algorithm, the feasibility-approach algorithm can be formulated through an optimization problem. Our motivation comes from the following argument. Suppose that we have a set of ℓ samples $\{x_1, x_2, .., x_\ell\}$ and we want a weight vector w and a bias b that satisfies $y_i(wx_i + b) \geq 1$ for all $i = 1, .., \ell$. This feasibility problem can be expressed as an LP problem [13] by introducing an artificial variable $t \geq 0$ and solving the following

$$\min t \tag{4.1}$$

Subject to

$$y_i(w^T x_i + b) + t \geq 1, i = 1, .., \ell$$
$$t \geq 0,$$

where $w \in \Re^n$ and b and t are scalar variables. If the optimal value $\hat{t} = 0$, then the samples are linearly separable and we have a solution. If $\hat{t} > 0$, there is no separating hyperplane and we have a proof that the samples are non-separable. By minimizing the slack variable t we can decide if the separation is feasible. In contrast to the SVM approach, we keep the same slack variable t constant over the separation constraints. As shown in Figure 2.3, here we minimize the deviation of points from the supporting hyperplanes $wx + b = \pm 1$.

4.2 R-FA and R-SVM Formulation

Now consider that our data are perturbed. Instead of having the input data point x_i we have $x_i = \tilde{x_i} + u_i$ where u_i is a bounded perturbation with $||u_i|| \leq \sqrt{\eta}$ where η is a positive number, and $\tilde{x_i}$ is the center of the uncertainty sphere where our data point is located. Therefore, the constraints in (4.1) become

$$y_i(< w, x_i > + b) + t \geq 1 \Leftrightarrow \tag{4.2}$$
$$y_i(< w, \tilde{x_i} > + < w, u_i > + b) + t \geq 1, i = 1, .., \ell$$
$$t \geq 0$$

Our concern is the problem of classification with respect to two classes. In order to have the best separating hyperplane we try to minimize the dot product of w and u_i in one side of the separating hyperplane (class -1) and maximize the dot product of w and u_i in the other side (class 1) subject to $||u_i|| \leq \sqrt{\eta}$. By this logic we are trying to maximize the distance between the classifier to both points on different sides (see Figure) 4.1.

Therefore, we have to solve the following problem

$$\max < w, u_i > \tag{4.3}$$
$$\text{Subject to } ||u_i|| \leq \sqrt{\eta}$$

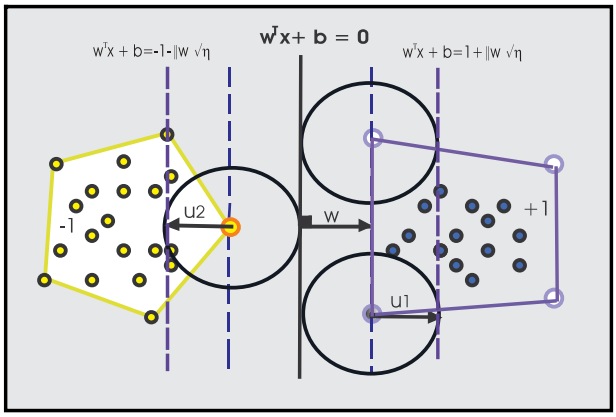


Figure 4.1: Finding the best classifier for data with uncertainty. The bounding planes are moved to the boundary of the spheres to obtain maximum margin

Using Cauchy's Schwarz inequality ($| < w, u > | \leq \|w\| \, . \, \|u\| \Rightarrow - \|w\| \, . \, \|u\| \leq < w, u > \leq \|w\| \, . \, \|u\|$), the maximum of $< w, u_i >$ is equal to $\|w\| \, . \, \|u\|$. Hence, referring to (4.3) the maximum of the dot product of $< w, u_i >$ will be $\sqrt{\eta} \, \|w\|$. By substituting this maximum value in (4.2), we have

$$\min t \tag{4.4}$$

Subject to

$$\sqrt{\eta} \, \|w\| - w\tilde{x}_i - b + t \geq 1, for \; y_i = -1$$

$$\sqrt{\eta} \, \|w\| + w\tilde{x}_i + b + t \geq 1, for \; y_i = +1$$

$$t \geq 0, i = 1, .., \ell$$

If we map the data from the input space to the feature space F,we can represent w in the space F as

$$w = \sum_{i=1}^{\ell} \alpha_i \varphi(\tilde{x}_i) \tag{4.5}$$

By substituting w with the above representation and substituting $< \varphi(\tilde{x}), \varphi(\tilde{x})' >$ with $k(\tilde{x}, \tilde{x}')$, we have the following R-FA formulation:

$$\min t \tag{4.6}$$

Subject to

$$\sqrt{\eta}\sqrt{\alpha^T K\alpha} - K_i\alpha - b + t \geq 1, for\ y_i = -1$$
$$\sqrt{\eta}\sqrt{\alpha^T K\alpha} + K_i\alpha + b + t \geq 1, for\ y_i = +1$$
$$t \geq 0, i = 1, .., \ell,$$

where K_i is the 1xℓ vector corresponding to the ith line of the kernel matrix K. Note that we reorder the rows of the matrix K based on the label. It is important to note that most of the time we do not need to know explicitly the map φ. The important idea is that we can replace $< \varphi(x), \varphi(x)' >$ with any suitable kernel $k(x, x')$.

Combining the constraints which include uncertainty term as in the feasibility-approach in the SVM model (2.3), we have the following R-SVM formulation:

$$\min \frac{1}{2}\alpha^T K\alpha + C\sum_{i=1}^{\ell} t_i \tag{4.7}$$

Subject to

$$\sqrt{\eta}\sqrt{\alpha^T K\alpha} - K_i\alpha - b + t_i \geq 1, for\ y_i = -1$$
$$\sqrt{\eta}\sqrt{\alpha^T K\alpha} + K_i\alpha + b + t_i \geq 1, for\ y_i = +1$$
$$t_i \geq 0$$

By the margin(η), we define the margin of separation when the level of uncertainty is η. Then

$$\begin{aligned} margin(\eta) &= \frac{(1 + \|w\|\sqrt{\eta} - b) - (-1 - b + \sqrt{(}\eta)\,\|w\|)}{\|w\|} \\ &= \frac{2 + 2\sqrt{\eta}\,\|w\|}{\|w\|} = \frac{2}{\|w\|} + 2\sqrt{\eta} = margin(0) + 2\sqrt{\eta}. \end{aligned} \tag{4.8}$$

Note that the margin of separation is increasing. In the case of robust optimization formulation [36], margin(η) = margin(0)-$2\sqrt{\eta}$.

4.3 Optimization Approach for R-FA and R-SVM

The formulation in equation (4.6) is obviously a nonlinear programming problem. This problem is in the class of nonlinear constrained optimization problems. In

constrained optimization, the general aim is to transform the problem into an easier subproblem that can then be solved and used as the basis of an iterative process. The general form of a constrained optimization problem is

$$\begin{aligned} \min f(x) \\ \text{Subject to} \\ g_i(x) = 0,\ i = 1, .., m_e \\ g_i(x) \leq 0,\ i = m_e + 1, .., m \\ x_l \leq x \leq x_u, \end{aligned} \tag{4.9}$$

where at least one of the constraints is nonlinear. A characteristic of a large class of early methods is the conversion of the constrained problem to an equivalent unconstrained problem or into a problem with simple constraints. Basically, there are two alternative approaches. The first is called the penalty or the exterior penalty function method, in which a penalty term is added to the objective function for any violation of the constraints. This method generates a sequence of infeasible points, hence its name, whose limit is an optimal point to the original problem. The second method is called the barrier or interior penalty function method, in which a barrier term that prevents the points generated deviating from the feasible region is added to the objective function. This method generates a sequence of feasible points, whose limit is an optimal point to the original problem [1]. These methods are now considered relatively inefficient and have been replaced by methods that have focused on the solution of the Karush-Kuhn-Tucker (KKT) equations. The KKT equations are necessary conditions for optimality for a constrained optimization problem. Referring to the general problem in (4.9), KKT equations can be stated as

$$\begin{aligned} \nabla f(x*) + \sum_{i=1}^{m} \lambda_i^* \nabla g_i(x*) = 0 \\ \lambda_i^* \nabla g_i(x*) = 0,\ i = 1, .., m_e \end{aligned} \tag{4.10}$$

$$\lambda_i^* \geq 0,\ i = m_e + 1, .., m.$$

The solution of the KKT equations forms the basis to many nonlinear programming algorithms. These algorithms attempt to compute the Lagrange multipliers directly. One of the algorithms is Sequential Quadratic Programming (SQP) method. SQP belongs to the most powerful nonlinear programming algorithms we know today for solving differentiable nonlinear programming problems of the form (4.9). In this section we will describe the basic ideas of the SQP method for solving nonlinearly constrained optimization problems. In SQP, we solve QP subproblems sequentially to find the optimal solution for the original problem. SQP employs Newton's method (or quasi-Newton methods) to directly solve the KKT condition for the original problem. As a result, the accompanying subproblem turns out to be the minimization of quadratic function to the Lagrangian function optimized over a linear approximation to the constraints. The Lagrangian function is given as

$$L(x, \lambda) = f(x) + \sum_{i=1}^{m} \lambda_i g_i(x) \tag{4.11}$$

This method closely mimics Newton's method for constrained optimization as in the case of unconstrained optimization. At each major iteration, we solve a quadratic programming subproblem. The solution obtained from solving this subproblems then is used to form a new iterate. The QP subproblem of (4.11) is given as

$$\begin{aligned} \min \frac{1}{2} d^T H_k d + \nabla f(x_k)^T d \qquad (4.12) \\ \nabla g_i(x_k)^T d + g_i(x_k) = 0,\ i = 1, .., m_e \\ \nabla g_i(x_k)^T d + g_i(x_k) \leq 0,\ i = m_e + 1, .., m. \end{aligned}$$

In this subproblem formulation we are linearizing the nonlinear constraints. The term H_k in the formulation is a positive definite approximation of the Hessian matrix of the Lagrangian function (4.11), the vector d is the direction of descent and the variable we are trying to compute. This formulation can be solved using

any QP algorithm by using a starting point $x_k = x_0$. This solution is then used to form the next iterate

$$x_{k+1} = x_k + \alpha_k d_k$$

The step length α_k is determined using an appropriate line search technique so that a sufficient decrease of a *merit function(ψ)* is obtained. At each major iteration, a positive definite Hessian matrix is calculated using the BFGS method [29]. The updating rule is given as follows

$$H_{k+1} = H_k + \frac{q_k q_k^T}{q_k^T s_k} - \frac{H_k^T H_k}{s_k^T H_k s_k}, \tag{4.13}$$

where

$$s_k = x_k + 1 - x_k,\ q_k = \nabla f(x_{k+1}) + \sum_{i=1}^{m} \lambda_i^* \nabla g_i(x_{k+1}) - \left(\nabla f(x_k) + \sum_{i=1}^{m} \lambda_i^* \nabla g_i(x_k) \right).$$

Powell [29] suggests to keep the Hessian matrix H positive definite. To do this we have to maintain $q_k^T s_k$ positive and initialize H with a positive definite matrix. The step length parameter α_k is required to form a new iterate x_{k+1} to enforce the iterates of the SQP algorithm eventually to get closer to x^*, when starting from arbitrary initial values. The standard way to ensure that a reduction in ψ indicates progress is to construct ψ such that the solutions of 4.9 are unconstrained minimizers of ψ. This brings us to the idea of penalty functions. Another attribute of the merit function is that it should lead the iterates of the algorithm and provide a measure of progress by means of exhibiting a descent. In this implementation the merit function proposed by Han [21] and Powell [29] is used. The steplength α_k should satisfy at least a sufficient decrease of a merit function given by

$$\psi(x) = f(x) + \sum_{i=1}^{m_e} r_i g_i(x) + \sum_{i=m_e+1}^{m} r_i\ max[0, g_i(x)], \tag{4.14}$$

where x is function of α.

Powell [29] recommends setting the penalty parameter

$$r_i = (r_{k+1})_i = \max_i \left\{ \lambda_i, \frac{1}{2}((r_k)_i + \lambda_i) \right\}, i = 1, .., m. \tag{4.15}$$

In this implementation, the initial penalty parameter r_i is set to:

$$r_i = \frac{|\nabla f(x)|}{|\nabla g(x)|}. \tag{4.16}$$

This ensures that constraints with smaller gradients provide large contributions to the penalty parameter that would be the case for active constraints at the solution point. A further discussion on the implementation of SQP for solving nonlinear programming can be found in [20].

4.4 Experiments and Results

4.4.1 Experiments

In this study, some experiments were done for the AND and *Exclusive* OR (XOR) problems. Both R-SVM and R-FA were implemented on AND and *Exclusive* OR (XOR) problems. These experiments were to investigate how is the behavior of misclassification error with respect to the values of η, the feasibility-approach can still separate the data correctly by applying different level of uncertainty η. The AND problem is a two-class classification problem where two sets of points can be separated linearly (see Table 4.1). Because of this linearity, a kernelization is not necessary. To judge the performance of the kernel feasibility-approach, SVM as formulated in (2.2) was investigated. The SVM formulation is modified by replacing w with $w = \sum_{i=1}^{\ell} \alpha_i \varphi(x_i)$ to handle the case of nonlinearity in the case of the XOR problem (see Table 4.2).

The XOR problem can be viewed as a special case of a more general problem where we want to find a nonlinear classifier that classifies two sets of data points. Different from the AND problem, the XOR problem is nonlinear separable. Because of the nonlinearity of the XOR problem, a mapping of the original data

Table 4.1: AND Problem

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

Table 4.2: XOR Problem

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

into a higher dimensional feature space through a kernel mapping is necessary in order to simplify the classification task by finding a linear classifier in the feature space. We use a polynomial kernel function with degree 2 which is formulated as $K(x_i, x_i) = (x_i x_i' + 1)^2$ [23]. By this kernel function, we can compute the matrix K with dimension ℓ by ℓ, where ℓ is the number of data points. For example $K(1, 1)$, using the data in Table 4.1, can be computed as follows:

$x_1 = (1, 1)$

$x_1 x_1' = [1\ 1]\,[1\ 1]' = 2$

$(x_1 x_1' + 1)^2 = 9$

With the same procedure for all x_i, we obtain matrix K as follows:

$$\begin{bmatrix} 9 & 1 & 1 & 1 \\ 1 & 9 & 1 & 1 \\ 1 & 1 & 9 & 1 \\ 1 & 1 & 1 & 9 \end{bmatrix} \tag{4.17}$$

4.4.2 Results of AND Problem

Using the data in Table 4.1, for the feasibility approach, we have the following optimization problem for the AND problem.

$$\min t \tag{4.18}$$

Subject to

$$\begin{aligned} \sqrt{\eta}\sqrt{(w_1^2 + w_2^2)} + w_1 + w_2 + b + t &\geq 1 \\ \sqrt{\eta}\sqrt{(w_1^2 + w_2^2)} + w_1 - w_2 - b + t &\geq 1 \\ \sqrt{\eta}\sqrt{(w_1^2 + w_2^2)} - w_1 + w_2 - b + t &\geq 1 \\ \sqrt{\eta}\sqrt{(w_1^2 + w_2^2)} + w_1 + w_2 - b + t &\geq 1 \\ t &\geq 0 \end{aligned}$$

The corresponding SVM optimization problem is:

$$\min \frac{1}{2} w^T w + C \sum_{i=1}^{\ell} t_i \tag{4.19}$$

Subject to

$$\begin{aligned} \sqrt{\eta}\sqrt{{w_1}^2 + {w_2}^2} + w_1 + w_2 + b + t_i &\geq 1 \\ \sqrt{\eta}\sqrt{{w_1}^2 + {w_2}^2} + w_1 - w_2 - b + t_i &\geq 1 \\ \sqrt{\eta}\sqrt{{w_1}^2 + {w_2}^2} - w_1 + w_2 - b + t_i &\geq 1 \\ \sqrt{\eta}\sqrt{{w_1}^2 + {w_2}^2)} + w_1 + w_2 - b + t_i &\geq 1 \\ t_i &\geq 0 \end{aligned}$$

The results of experiments on AND problem using R-FA and R-SVM are shown in Tables 4.3 and 4.4. The results show that R-FA and R-SVM robust to the change

of the level of uncertainty η. By changing the vlue of η both classifiers still classify the points correctly.

Table 4.3: w and b values for R-FA on AND problem, with different uncertainties η

			η			
Variable	0.0	0.1	0.5	0.6	1.1	1.2
w_1	1	0.6910	0.5000	0.4772	0.4027	0.3950
w_2	1	0.6910	0.5000	0.4772	0.4027	0.3950
b	-1	-0.6910	-0.5000	-0.4772	-0.4027	-0.3881
t	0	0	0	0	0	0
result($\hat{y}$)	correct	correct	correct	correct	correct	correct

Table 4.4: w and b values with R-SVM approach for AND problem, with different uncertainties η, $C = 1$

			η			
variable	0.0	0.1	0.5	0.6	1.1	1.2
w_1	1	0.6910	0.5000	0.4772	0.4027	0.3923
w_2	1	0.6910	0.5000	0.4772	0.4027	0.3923
b	-1	-0.6910	-0.5000	-0.4772	-0.4027	-0.3923
margin	0.7071	1.0233	1.4142	1.4817	1.7559	1.8026
result($\hat{y}$)	correct	correct	correct	correct	correct	correct

4.4.3 Results of XOR Problem

Using the data in Table 4.2, we have the following feasibility-approach optimization problem for XOR problem.

$$\min t \tag{4.20}$$

Subject to

$$(\sqrt{\eta}\sqrt{(}9\alpha_1^2 + 9\alpha_2^2 + 9\alpha_3^2 + 9\alpha_4^2 + 2\alpha_1\alpha_2 + 2\alpha_1\alpha_3 + 2\alpha_1\alpha_4 +$$

$$
\begin{aligned}
&2\alpha_2\alpha_3 + 2\alpha_2\alpha_4 + 2\alpha_3\alpha_4) - 9\alpha_1 - \alpha_2 - \alpha_3 - \alpha_4 - b + t \geq 1 \\
&(\sqrt{\eta}\sqrt{(}9\alpha_1^2 + 9\alpha_2^2 + 9\alpha_3^2 + 9\alpha_4^2 + 2\alpha_1\alpha_2 + 2\alpha_1\alpha_3 + 2\alpha_1\alpha_4 + \\
&2\alpha_2\alpha_3 + 2\alpha_2\alpha_4 + 2\alpha_3\alpha_4) + \alpha_1 + 9\alpha_2 + \alpha_3 + \alpha_4 + b + t \geq 1 \\
&(\sqrt{\eta}\sqrt{(}9\alpha_1^2 + 9\alpha_2^2 + 9\alpha_3^2 + 9\alpha_4^2 + 2\alpha_1\alpha_2 + 2\alpha_1\alpha_3 + 2\alpha_1\alpha_4 + \\
&2\alpha_2\alpha_3 + 2\alpha_2\alpha_4 + 2\alpha_3\alpha_4) + \alpha_1 + \alpha_2 + 9\alpha_3 + \alpha_4 + b + t \geq 1 \\
&(\sqrt{\eta}\sqrt{(}9\alpha_1^2 + 9\alpha_2^2 + 9\alpha_3^2 + 9\alpha_4^2 + 2\alpha_1\alpha_2 + 2\alpha_1\alpha_3 + 2\alpha_1\alpha_4 + \\
&2\alpha_2\alpha_3 + 2\alpha_2\alpha_4 + 2\alpha_3\alpha_4) - \alpha_1 - \alpha_2 - \alpha_3 - 9\alpha_4 - b + t \geq 1 \\
&t \geq 0
\end{aligned}
$$

The solutions of solving problem (4.20) with some different η values are shown in Table 4.5. Table 4.6 shows the results of running SVM for the same problem with different uncertainty values.

Table 4.5: Alpha and b values with feasibility-approach for different η for XOR problem

	η					
Variable	0.0	0.1	0.5	2	2.8	2.9
α_1	-0.125	-0.125	-0.125	0.125	-0.1142	- 0.1130
α_2	0.125	0.125	0.125	0.125	0.1142	0.1130
α_3	0.125	0.125	0.125	0.125	0.1142	0.1130
α_4	-0.125	-0.125	-0.125	0.125	-0.1142	-0.1130
b	0	0	0	0	0	0
result($\hat{y}$)	correct	correct	correct	correct	correct	correct

4.5 Implementation on Real Data

For the implementation, the Iris data, breast bancer data [4], and tornado data [38] were used. Iris data is a well known data set in machine learning which consists of 150 data points. There are three classes in this data set. A subset of data were taken from the Iris data that contain only two classes. Then, twenty different

Table 4.6: Alpha and b values with SVM approach with different η for XOR problem

			η		
Variable	0.0	.1	1	2	2.1
α_1	-0.125	-0.1022	-0.0732	-0.0625	-0.0617
α_2	0.125	0.1022	0.0732	0.0625	0.0617
α_3	0.125	0.1022	0.0732	0.0625	0.0617
α_4	-0.125	-0.1022	-0.0732	0.0625	-0.0617
b	0	0	0	0	0
margin	1.4142	1.7304	2.4142	2.8284	2.8634
result($\hat{y}$)	correct	correct	correct	correct	correct

samples for training and testing were withdrawn from this subset. There were 50 data points in each training set and 50 data points in each testing set. The second experiments were done on breast cancer data with 683 data points and 9 attributes. This breast cancer databases was obtained from the University of Wisconsin Hospitals, Madison. There are two labels indicating the type of cancer. For the experiment, first, R-SVM and R-FA were trained with the whole training set which contains 410 data points and tested on the rest of 273 data points. The third data set was a tornado data set. The data set is the outputs from WSR-88D radar. The lables of the data are tornado and non-tornado. The data consist of 23 attributes. There are 749 data points for the training set that was selected randomly and 18,202 data point for the testing set. The experiments were done by varying the degree of uncertainty η to investigate how the generalization error changes with increasing uncertainty. The experiments were done both for the feasibility-approach and SVM with RBF kernel with spread 1. To solve the resulting nonlinear optimization problems, computer codes written in MATLAB using SNOPT solver which is embedded in the TOMLAB software package, were used. SNOPT employs a sparse SQP algorithm with limited-memory quasi-Newton approximations to the Hessian of Lagrangian. An augmented Lagrangian merit

function provides convergence from an arbitrary point [20]. The results of the experiments are presented in Tables 4.7, 4.8 and 4.9. For comparisons the Minimax Probability Machine (MPM) and robust MPM are run on the same data set.

Table 4.7: Percentage of average misclassified points for R-FA and R-SVM with different η values on Iris data, training=50 points, testing=50 points, polynomial, d=2, C=100

	η					
Method	0.0	0.01	0.05	0.5	1	2
R-SVM	5.1	5.1	5.5	5.5	5.6	5.6
R-FA	6.9	7.0	7.0	8.6	8.7	10.7

Table 4.8: Percentage of average misclassified points for R-FA and R-SVM with different η values on Breast Cancer data, training=410, testing=273, linear,C=10

	η			
Method	0.0	0.0001	0.001	0.01
Robust-SVM	3.19	3.15	3.10	4.82
R-FA	56.74	56.74	56.74	57.49

From Tables 4.7, 4.8 and 4.9, which show the impact of applying uncertainty on the generalization error, we can make some conclusions. First, by applying uncertainty η, the generalization error can be improved for some data. On a specific optimal value of uncertainty level η, the generalization reaches the minimum value. This can be seen from the plot of misclassification error versus η value on Tornado data in Figure 4.2. Second, R-SVM is better than R-FA in terms of generalization errors. In Table 4.10, we compare the results of R-SVM and R-FA and robust MPM. From the table we observe that R-SVM is better than robust-MPM for the three data sets: Iris, breast cancer and tornado. The slight difference value

Table 4.9: Percentage of misclassified points for R-FA and R-SVM with different η values on Tornado data, training=749 points, testing=18,202 points, polynomial, d=2, C=100

	η							
Method	0.0	0.1	0.5	1	1.5	2	2.5	3
Robust-SVM	1.21	1.8	3.9	1.0	1.3	0.9	2.7	27.6
Robust-FA	5.34	83.75	22.30	14.89	2.05	91.03	88.58	1.97

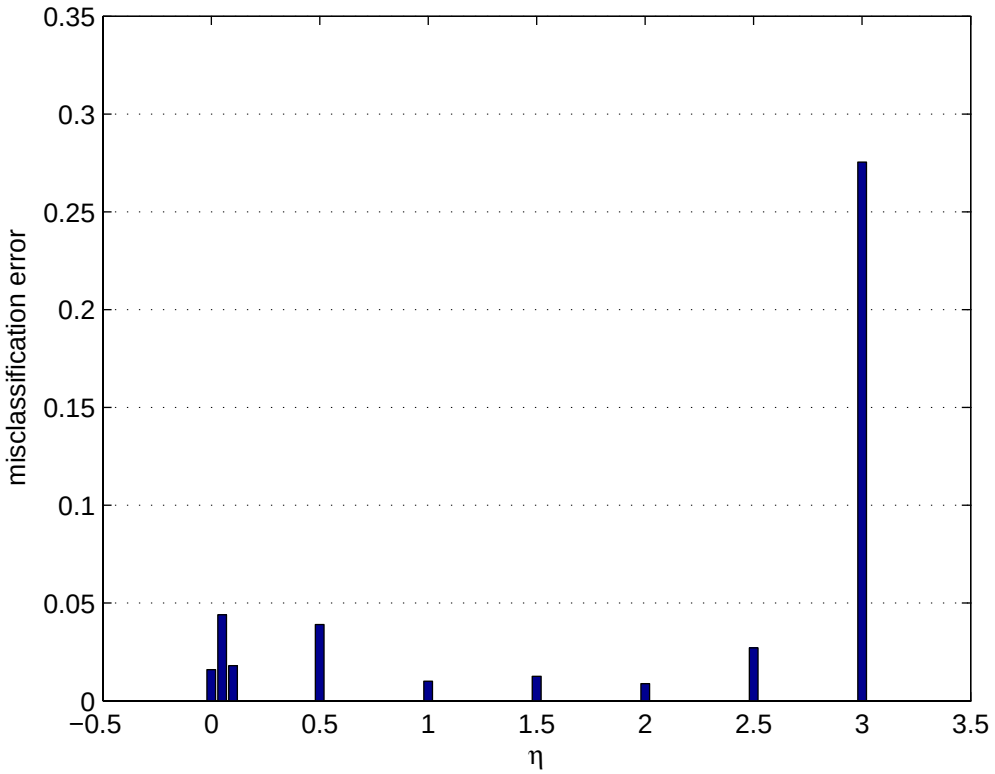


Figure 4.2: Plot of misclassification error vs uncertainty η for R-SVM classifiers on Tornado data

Table 4.10: Percentage of misclassified errors on three data sets for different methods

	Method					
Data	FA	SVM	MPM	Robust FA	Robust SVM	Robust MPM
Iris	6.9	5.1	11.0	7.0	5.1	8.7
Breast Cancer	56.74	3.19	3.0	56.74	3.10	3.6
Tornado	5.34	1.6	1.2	1.9	0.9	1.1

of misclassification errors between regular SVM and R-SVM results is meaningful especially when the testing size is large. As an example let's take the case of tornado prediction problem. When we decrease the misclassification error from 0.016 to 0.009, it means that we decrease the rate of predicting incorrectly by 0.007. If we have 18,202 observations in the testing set, this number is equivalent with 127 observations. This value is a significant improvement in the case of tornado prediction.

Chapter 5

Robust Multiclass Kernel Feasibility Approach and SVM

5.1 Robust Multiclass Classification

In the previous chapter, we have discussed the case of binary classification where the set of labels $y = \{y_1, .., y_m\} \subseteq \{\pm 1\}$. In this chapter, the discussion will be extended to the cases where the set of labels $y = \{y_1, .., y_m\} \subseteq \{1, .., k\}$. Currently there are two main approaches to extend the binary into a multiclass classification. One is by constructing and combining several binary classifiers while the other is by directly considering all data in one optimization formulation [24]. In the case of multiclass SVMs, the idea of casting a multiclass problem as a single constrained optimization problem with a quadratic objective function was proposed by Weston and Watkins [41], and Bredensteiner and Bennet [11]. However, the size of the resulting optimization problems devised in the above papers is typically large and complex [14]. Hereby, we adopt the first approach where we construct and combine several binary classifiers [40, 24]. In this approach there are two procedures: One-against-all (OAA) and One-against-one (OAO) [24]. In the next sections we

develop robust multiclass classifiers based on these two procedures.

5.2 One-against-All (OAA) Method

As explained in section 3.5.1, by the OAA method, for k-class classification problems, we construct k classifiers where k is the number of classes. Now suppose we are given m training data $(x_1, y_1), .., (x_m, y_m)$ where $x_i \in R^n, i = 1, 2.., m$ and $y_i \in S = \{1, .., k\}$ is the class of x_i. Then the ith classifier solves the following robust feasibility-approach optimization problem [24]

$$\begin{aligned} \min t^i \qquad & (5.1) \\ \text{Subject to} & \\ \sqrt{\eta}||w^i|| + w^i\varphi(\tilde{x_j}) + b^i + t^i \geq 1, if\ y_j = i & \\ \sqrt{\eta}||w^i|| - w^i\varphi(\tilde{x_j}) - b^i + t^i \geq 1, if\ y_j \neq i & \\ t^i \geq 0. & \end{aligned}$$

After solving (5.1), there are k decision functions $w^1\varphi(x)+b^1, w^2\varphi(x)+b^2...w^k\varphi(x)+b^k$. Then, the class of point x is determined by the largest value of the decision function:

$$j = class\ of\ x = \arg \max_{i=1,..,k} w^i\varphi(x) + b^i, where\ j \in S = \{1, .., k\} \qquad (5.2)$$

The following example is an illustration of how this method works.

Example 1

Suppose we have a three-class problem. After all 3 classifiers are constructed, new four points (x_1, x_2, x_3, x_4) are coming for testing. The prediction values of each classifier are given in the following matrix. Each column of the matrix represents

the prediction values for each classifier.

$$\hat{y} = \begin{bmatrix} \mathbf{18.5965} & 14.7108 & -33.3073 \\ -12.3010 & 4.6637 & \mathbf{7.6373} \\ -9.4610 & \mathbf{8.4008} & 1.0602 \\ -20.0123 & 2.6552 & \mathbf{17.3571} \end{bmatrix}$$

The maximum value of $\hat{y}$ for each data point are:

$$\begin{bmatrix} 18.5965 \\ 7.6373 \\ 8.4008 \\ 17.3571 \end{bmatrix}$$

Then, based on these maximum $\hat{y}$ values, the classes for these four new points respectively are 1, 3, 2, and 3. For robust multiclass SVM(R-MSVM), we use exactly the same procedure as explained above. The difference is that in R-MSVM, for the ith R-SVM classifier, we solve the following problem:

$$\min_{w^i, b^i, t^i} \frac{1}{2}(w_i)^T w_i + C \sum_{j=1}^{\ell} t_j^i \tag{5.3}$$

Subject to

$$\sqrt{\eta}||w^i|| + w^i \varphi(\tilde{x_j}) + b^i \geq 1 - t_j^i, \; if \; y_j = i$$

$$-\sqrt{\eta}||w^i|| + w^i \varphi(\tilde{x_j}) + b^i \leq -1 + t_j^i, \; if \; y_j \neq i$$

$$t_j \geq 0, j = 1, .., \ell$$

5.3 One-against-One (OAA) Method

As explained in section 3.5.2, this method constructs $k(k-1)/2$ classifiers where each one is trained on data from two classes. For training data from ith and jth classes, we solve the following robust feasibility-approach binary classification

problem [24]:

$$\min t^i \tag{5.4}$$

Subject to

$$\sqrt{\eta}||w^{ij}|| + w^{ij}\varphi(\tilde{x_r}) + b^{ij} + t^{ij} \geq 1, if\ y_r = i$$

$$\sqrt{\eta}||w^{ij}|| - w^{ij}\varphi(\tilde{x_r}) - b^{ij} + t^{ij} \geq 1, if\ y_r = j$$

$$t^{ij} \geq 0$$

After all classifiers $k(k-1)/2$ are constructed, there are different methods for doing the future testing. One strategy is max-vote. Based on this strategy, for classifier ρ^{ij} if a new point x is in ith class,then the vote for the ith class is added by one. Otherwise, the vote for jth is increased by one. We repeat this step for all classifiers. Then, we predict x is in the class with the largest vote. In the case where two classes have identical votes, we select the one with the smaller index. To illustrate this method an example is given below.

Example 2

We are applying this method on the same problem as in example 1. After constructing all classifiers, we obtain $\hat{y}$ from classifier ρ^{12} as

$$\hat{y} = \begin{bmatrix} 1.4244 \\ -3.8363 \\ -3.4960 \\ -5.2525 \end{bmatrix}$$

Then the vote is given as :

class 1	class 2	class 3
1	0	0
0	1	0
0	1	0
0	1	0

The $\hat{y}$ from classifier ρ^{13} is

$$\hat{y} = \begin{bmatrix} -6.7500 \\ -0.8855 \\ 2.7612 \\ -7.5913 \end{bmatrix}$$

After considering classifier $\rho^{12}$and ρ^{13} the vote becomes:

class 1	class 2	class 3
1	0	1
0	1	1
1	1	0
0	1	1

$\hat{y}$ from classifier ρ^{13} is

$$\hat{y} = \begin{bmatrix} -24.2676 \\ 1.4833 \\ -3.6762 \\ 7.3307 \end{bmatrix}$$

After considering classifier ρ^{12}, ρ^{13} and $\rho^{23}$the vote becomes:

class 1	class 2	class 3
1	0	**2**
0	**2**	1
1	1	1
0	**2**	1

Then, based on these $\hat{y}$ values, the classes for these four new points respectively are 3, 2, 1 and 2. For the R-MSVM we use exactly the same procedure as explained above. The difference is that in R-MSVM, for each pair of classes, we solve the following robust binary classification problem:

$$\min_{w^{ij}, b^{ij}, t^{ij}} \frac{1}{2}(w^{ij})^T w^{ij} + C \sum_r t_r^{ij} \tag{5.5}$$

Subject to

$$\sqrt{\eta}||w^{ij}|| + w^{ij}\varphi(\tilde{x_r}) + b^{ij} + t_r^{ij} \geq 1,\ if\ y_r = i$$

$$\sqrt{\eta}||w^{ij}|| - w^{ij}\varphi(\tilde{x_r}) - b^{ij} + t_r^{ij} \geq 1,\ if\ y_r = j$$

$$t_r^{ij} \geq 0$$

5.4 Implementation

In this section, we apply the robust feasibility-approach for multiclass (R-MFA) and robust multiclass SVM (R-MSVM) on data sets where we introduce uncertainties and using the two aforementioned procedures: One-against-One and One-against-All. The Iris data with three classes, Balance Scale data with three classes, Dermatology data with six classes and Glass data with six classes [9] and phase-flow data with three classes [35] are used . The data sets are split into two sets: training set and testing set. For the Iris data, the training size is 100 data and the testing size is 50 data. Twenty different training and testing samples are chosen randomly from the data set. The Balance Scale data consists of 625 data. For the Balance Scale and Dermatology data, the ratio of training size and the testing size is 70:30. There are six different training and testing samples that are selected randomly. For flow phase data, there are four training and testing samples. The experiments are done by using an RBF kernel with $\sigma = 1$ and $C = 1$. The results of the experiments are presented in Tables 5.1, 5.2, 5.3, 5.4 and 5.5. For comparison purposes, the multiclass Minimax Probability Machine (MMPM) and robust MMPM [30] are run on the same data sets.

From Tables 5.1, 5.2, 5.3, 5.4 and 5.5, we see that the computation times of R-MSVM using OAO are lower than those given by R-MSVM using OAA. In the case that OAA and OAO have the same misclassification error, we can use computation time as a second criterion to choose the best approach. We observe from Table

Table 5.1: Percentage of average misclassification error and computation time for Robust-MFA and Robust-MSVM with different η values with OAA and OAO approach on Iris data, polynomial, d=2, C=100

		η				
Approach	Method	0.0	0.01	0.1	1	2
OAA	R-MFA	6.1(0.6)	6.1(0.8)	6.1(0.9)	6.3(0.9)	13.9(0.9)
	R-MSVM	5.1(2.4)	4.8(3.8)	4.8(5.2)	6.2(68.8)	6.2(67.4)
OAO	R-MFA	6.3(0.4)	6.3(0.4)	6.3(0.5)	6.7(0.5)	11.0(0.5)
	R-MSVM	5.1(2.8)	5.2(3.8)	4.5(2.4)	6.2(2.1)	6.2(1.6)

Table 5.2: Percentage of average misclassification error and computation time for Robust-MFA and Robust-MSVM with different η values with OAA and OAO approach on Dermatology data

		η				
Approach	Method	0.0	0.001	0.01	0.1	1
OAA	R-MFA	3.5(83.3)	3.5(115.2)	3.5(101.8)	3.5(109.9)	3.9(4.8)
	R-MSVM	3.1(787.6)	2.7(68.8)	3.1(63.7)	3.3(64.6)	3.5(66.2)
OAO	R-MFA	3.7(8.3)	3.7(8.7)	3.7(8.0)	3.7(8.3)	3.9(1.7)
	R-MSVM	3.5(122.2)	3.1(9.8)	3.5(9.5)	3.7(9.0)	3.7(8.0)

Table 5.3: Percentage of average misclassification error and computation time for Robust-MFA and Robust-MSVM with different η values with OAA and OAO approach on Balance Scale data

		η			
Approach	Method	0.0	0.1	0.5	1
OAA	R-MFA	0.0(30.9)	0.0(69.1)	0.0(65.4)	0.0(4.6)
	R-MSVM	0.0(22.6)	0.2(126)	0.2(176.5)	0.2(207.4)
OAO	R-MFA	0.0(24.3)	0.0(50.1)	0.0(45.2)	2.6(3.5)
	R-MSVM	0.0(14.6)	0.0(38.7)	0.0(65.6)	0.0(58.5)

Table 5.4: Percentage of average misclassification error and computation time for Robust-MFA and Robust-MSVM with different η values with OAA and OAO approach on Glass data

		η			
Approach	Method	0.0	0.1	0.5	1
OAA	R-MFA	42.8(169.3)	41.6(194.6)	42.2(136.9)	43.4(173.9)
	R-MSVM	34.4(110.6)	41.3(88.0)	35.9(107.1)	40.6(103.7)
OAO	R-MFA	37.8(24.7)	37.8(20.6)	36.3(6.8)	40.6(4.1)
	R-MSVM	45.0(37.4)	42.8(25.8)	43.1(32.9)	44.4(27.8)

Table 5.5: Percentage of average misclassification error and computation time for Robust-MFA and Robust-MSVM with different η values with OAA and OAO approach on Phase Flow data

		η				
Approach	Method	0.0	0.005	0.01	0.05	0.1
OAA	R-MFA	2.41(6)	2.41(28.35)	2.41(27.4)	2.41(32.58)	2.41(33.38)
	R-MSVM	2.41(58.68)	2.75(30.00)	1.60(25.83)	2.41(39.3)	2.41(35.2)
OAO	R-MFA	4.59(3.28)	4.59(7)	4.59(6.68)	4.59(7.10)	4.59(6.38)
	R-MSVM	3.32(22.75)	2.75(38.58)	1.60(24.9)	2.41(24.85)	2.41(27.43)

Table 5.6: Percentage of average misclassification errors on four data sets for different methods

		Method					
	Data	MFA	MSVM	MMPM	R-MFA	R-MSVM	R-MMPM
OAA	Iris	6.1	5.1	3.5	6.1	4.8	6.9
	Dermatology	3.5	3.1	6.1	3.5	2.7	2.6
	Balance Scale	0.0	0.0	6.7	2.6	0.2	23.2
	Glass	42.8	34.4	40.6	41.6	35.9	41.6
	Flow phase	2.41	2.41	4.24	2.41	1.60	4.01
OAO	Iris	6.3	5.1	3.5	6.3	4.5	10.0
	Dermatology	3.7	3.5	11.9	3.7	3.1	3.3
	Balance Scale	0.0	0.0	3.6	0.0	0.0	7.9
	Glass	37.8	45.0	40.9	36.3	42.8	39.4
	Flow phase	4.59	3.32	5.38	4.59	1.60	5.16

5.6 that for Iris, Dermatology and Phase Flow, R-MSVM is giving the minimum misclassification error. This value is better than that given by regular MSVM. For Balance Scale the R-MSVM performs the same as regular MSVM. For Glass data, the uncertainty causes the R-MSVM performs worse than the regular MSVM. We also investigate that R-MSVM outperforms R-MMPM for all data sets.

Chapter 6

Robust Kernel Feasibility-approach and SVM for Regression

6.1 Basic Idea

By the introduction of Vapnik's ϵ-insensitive loss function, SVM has been generalized for function approximation or regression [31]. Established on the unique theory of Structural Risk Minimization principle to estimate a function by minimizing an upper bound of the generalization error, SVM is shown to be very resistant to the over-fitting problem, eventually achieving high generalization performance. Suppose we have been given ℓ training data, $(x_i, y_i),\ i = 1, .., \ell$ with input data $x = \{x_1, x_2.., x_\ell\} \subseteq \Re^N$and corresponding outputs $y = \{y_1, .., y_\ell\} \subseteq \Re$. By support vector regression, one wants to find a function $f(x)$ that has at most ϵ deviation from the actual target y_i for all training data. Suppose we have the

following function as a regressor:

$$f(x) = w^T \varphi(x) + b, \tag{6.1}$$

where $\varphi(x)$ denotes a point in the high dimensional feature space F which is the mapping of a point x in the input space. The coefficients w and b are estimated by minimizing the regular risk function defined in equation (6.2).

$$\min \frac{1}{2} \|w\|^2 + C \frac{1}{\ell} \sum_{i=1}^{\ell} L_\epsilon(y_i, f(x_i)) \tag{6.2}$$

Subject to

$$y_i - w\varphi(x_i) - b \leq \epsilon$$

$$w\varphi(x_i) - y_i + b \leq \epsilon, i = 1, .., \ell,$$

where

$$L_\epsilon(y_i, f(x_i)) = \begin{cases} |y_i - f(x_i)| - \epsilon |y_i - f(x_i)| \geq \epsilon \\ 0, \quad \text{otherwise} \end{cases} \tag{6.3}$$

The term $\|w\|^2$ is called the regularization term. Minimizing $\|w\|^2$ will make a function as flat as possible, thus playing the role of controlling the function capacity. The second term is the empirical error measured by the ϵ-insensitive loss function. Using the idea of ϵ-insensitive loss function [39], one should seek to minimize the norm of w in order to accomplish good generalization properties for the regressor f. Therefore, we have to solve the following optimization problem in the primal weight space:

$$\min \frac{1}{2} \|w\|^2 \tag{6.4}$$

Subject to

$$y_i - w\varphi(x_i) - b \leq \epsilon$$

$$w\varphi(x_i) - y_i + b \leq \epsilon, i = 1, .., \ell$$

(6.5)

We assume that there is a function f that approximates all pairs (x_i, y_i) with precision ϵ. In this case, we assume that the problem is feasible. In the case of infeasibility, where some points might deviate from $f \pm \epsilon$, one can introduce slack variables t, t^* to cope with infeasible constraints of the optimization problem. Then, the above problem can be formalized as [39]:

$$\min \frac{1}{2} \|w\|^2 + C \sum_{i=1}^{\ell} (t_i + t_i^*) \tag{6.6}$$

Subject to

$$y_i - w^T \varphi(x_i) - b - t_i \leq \epsilon, i = 1, .., \ell$$

$$w\varphi(x_i) - y_i + b - {t^*}_i \leq \epsilon, i = 1, .., \ell$$

$$t_i, {t^*}_i \geq 0,$$

The constant $C > 0$ determines the trade off between the flatness of function f and the amount up to which deviations larger than ϵ are tolerated. Any deviation more than ϵ will be penalized with C. Figure 6.1 depicts the situation graphically. Only the points outside the shaded region contribute to the cost insofar, as the deviations are penalized in a linear fashion. In Support vector Regression (SVR), ϵ is equivalent to the approximation accuracy placed on the training data points.

A small ϵ corresponds to a large slack variable $t_i^{(*)}$ and high approximation accuracy. On the contrary, a large ϵ corresponds to a small slack variable $t_i^{(*)}$ and low approximation accuracy. According to equation (6.6), a large slack variable will make the empirical error having a large impact relatively to the regularized term. In SVR, support vectors are the training data points lying on or outside the ϵ-bound of the decision function. Therefore, the number of support vectors decreases as ϵ increases. Finally, by introducing Lagrange multipliers and exploiting the optimality constraints, the decision function is explicitly given as:

$$f(x) = \sum_{i=1}^{\ell} (\alpha_i - \alpha_i^*) K(x_i, x) + b, \tag{6.7}$$

where $K(x_i, x)$ is defined through the kernel function k.

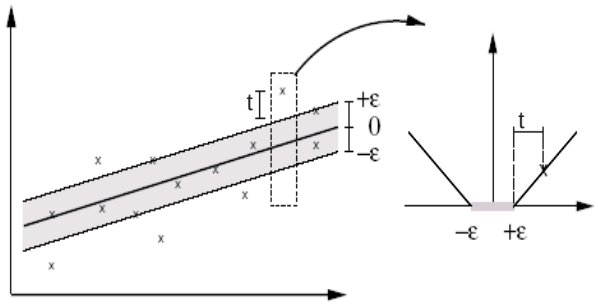


Figure 6.1: ϵ-insensitive loss function. The points outside the shaded region are penalized

Now, consider that our data are perturbed. Instead of having the input data point x_i we have $x_i = \tilde{x_i} + u_i$ where u_i is a bounded perturbation with $||u_i|| \leq \sqrt{\eta}$ where η is a positive number, and $\tilde{x_i}$ is the center of the uncertainty sphere where our data point is located. Therefore, equation (6.6) becomes

$$\min \frac{1}{2} \|w\|^2 + C \sum_{i=1}^{\ell} (t_i + t^*{}_i) \tag{6.8}$$

Subject to

$$y_i - < w, \tilde{x_i} > - < w, u_i > -b - t_i \leq \epsilon, i = 1, .., m$$

$$< w, \tilde{x_i} > + < w, u_i > -y_i + b - t^*{}_i \leq \epsilon, i = 1, .., m$$

$$t_i, t^*{}_i \geq 0$$

$$\forall u_i \subseteq \Re^d, \text{such that} ||u_i|| \leq \sqrt{\eta}.$$

By substituting w with $K_i\alpha$, and the term $< w, u >$ with its maximum value, $\sqrt{\eta}\sqrt{\alpha^T K \alpha}$, we obtain a robust support vector regression (R-SVR) formulation as:

$$\min \frac{1}{2} \alpha^T K \alpha + C \sum_{i=1}^{\ell} (t_i + t^*{}_i) \tag{6.9}$$

Subject to

$$y_i - K_i\alpha - \sqrt{\eta}\sqrt{\alpha^T K \alpha} - b - t_i \leq \epsilon, i = 1, .., m$$

$$K_i\alpha + \sqrt{\eta}\sqrt{\alpha^T K \alpha} - y_i + b - t^*{}_i \leq \epsilon, i = 1, .., m$$

$$t_i, t^*{}_i \geq 0$$

$$\forall u_i \subseteq \Re^d, \text{such that } ||u_i|| \leq \sqrt{\eta}.$$

The geometric illustration of robust SVR is given in Figure 6.2. Following the above SVR formulation, the regression model for the robust feasibility approach (R-FAR) is formulated as

$$\min t \tag{6.10}$$

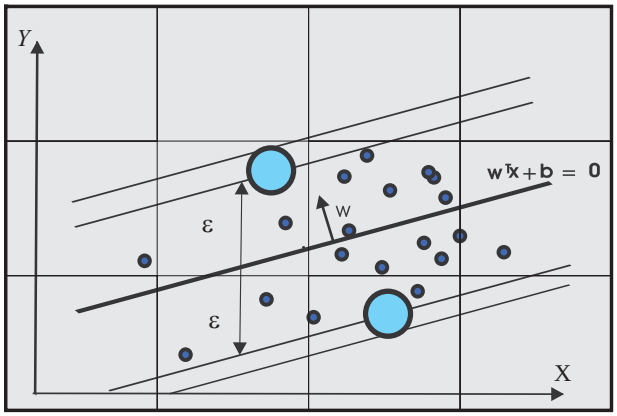


Figure 6.2: Geometric illustration of robust SVR

Subject to

$$y_i - K_i\alpha - \sqrt{\eta}\sqrt{\alpha^T K\alpha} - b - t \leq \epsilon, i = 1, .., m$$

$$K_i\alpha + \sqrt{\eta}\sqrt{\alpha^T K\alpha} - y_i + b - t \leq \epsilon, i = 1, .., m$$

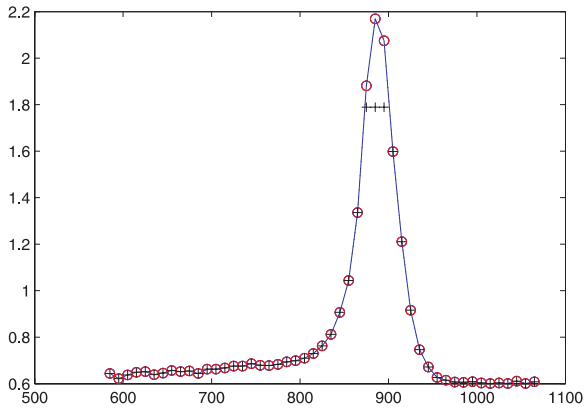
$$t \geq 0$$

$$\forall u_i \subseteq \Re^d, \text{such that } ||u_i|| \leq \sqrt{\eta}.$$

Figure 6.3 and 6.4 illustrates the performance of SVR and FA on Titanium data [16]. Table 6.1 shows the mean squared error (MSE) of FAR and SVR for different σ and C on Titanium data.

Table 6.1: MSE of FAR for regression and SVR on Titanium Data with RBF kernel

Method	MSE
FA, σ=1	4.44e-031
FA, σ=10	3.59e-027
SVR,σ=1,C=1	0.0048
SVR,σ=1,C=10	1.38e-004



— : actual data, + : $\sigma = 1$, C = 1, ○ : $\sigma = 1$, C = 10

Figure 6.3: Plots of the actual data and SVR results with RBF kernel

6.2 Implementation

In this implementation R-SVR and R-FAR are applied in time series Flour price data [12] and Abalone data [9]. Tables 6.2 and 6.4 show the performance of SVR and FAR with uncertainty on Flour price data and Abalone data. As shown in those tables, applying uncertainty can reduce the MSE of SVR significantly. This does not occur for FAR. Table 6.3 indicates that varying ϵ can improve the performance of SVR. Figure 6.5 shows the plots of actual and predicted values on Flour price data.

Table 6.2: MSE of R-SVR and R-FAR on flour price data with RBF kernel with η varied, $\epsilon = 0.0$

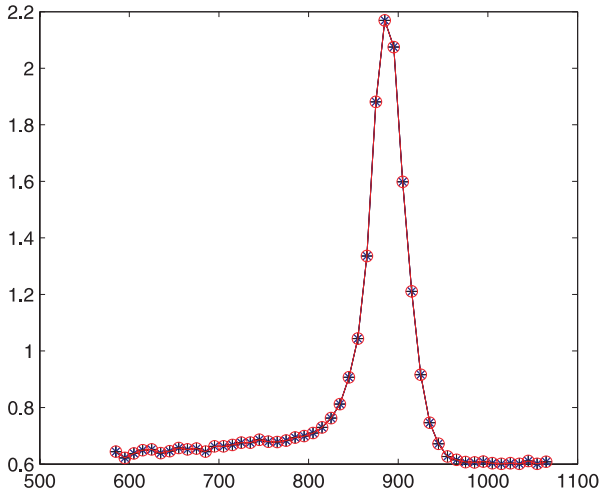
	η				
Method	0.0	0.00001	0.0001	0.001	0.01
R-FAR, σ=10	279.18	8125	10822	235130	731690
R-SVR, C=1000, σ=10	265.79	164.32	121.89	46.33	381.35

Table 6.3: MSE of R-FAR and R-SVR on flour price data with RBF kernel with ϵ varied

	ϵ				
Method	0	1	3	5	10
R-FAR,σ=10,η=0.0	279.18	279.18	279.18	279.18	98.03
R-SVR,C=1000,σ=10,η=0.001	46.33	45.77	41.0366	47.90	98.90

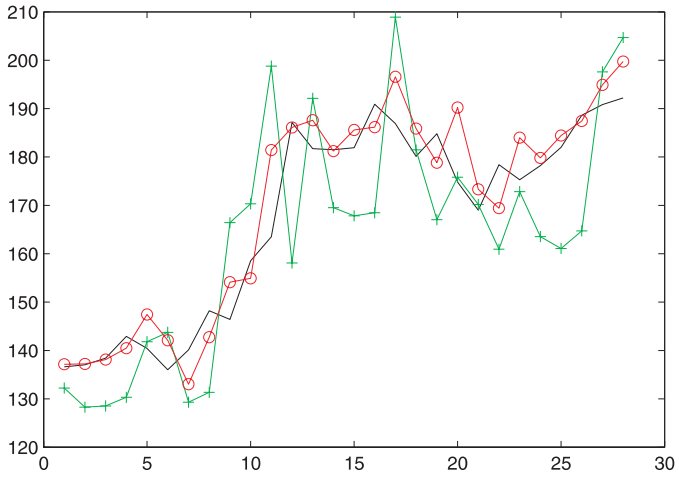
Table 6.4: MSE and computation time (CPU time) for R-FA and R-SVM with different η values on Abalone, training=750 points, testing=250 points, $\epsilon = 0.0$

	η				
Method	0.0	0.001	0.01	0.1	1
R-FAR	41.95(663.0)	16.76(517.4)	443.86(831.1)	32.17(814.5)	72(4205.5)
R-SVR	7.76(193.15)	29.54(32.03)	21.52(28.16)	7.56(30.56)	32.83(44.14)



-*- : actual data, -+- : $\sigma = 1$, -o- : $\sigma = 10$

Figure 6.4: Plots of the actual data and FA results with RBF kernel,



— :actual data, -+- : $\eta = 0$, -o- : $\eta = 0.001$

Figure 6.5: Plots of the actual data and R-SVR results with RBF kernel on flour price data

Chapter 7

Conclusions and Future Research

7.1 Conclusions

In this work, motivated by the presence of uncertainty in real data, robust classifiers (R-SVM, R-MSVM, R-FA and R-MFA) and regressors (R-SVR and R-FAR) have been developed. The impact of uncertainty on the data to the generalization error was investigated for binary classification, multiclass classification and regression problems. For classification, the computational results indicated that robust SVM (R-SVM and R-MSVM) performs better than regular SVM for both binary and multiclass classification problems. Robust SVM approach was better than robust feasibilty (R-FA and R-MFA) in terms of generalization error. For regression problems, robust SVR (R-SVR) improved the performance of regular SVR. R-SVR outperforms robust feasibility approach for regression (R-FAR).

7.2 Future Research

Preliminary experimentations demonstrated that robust classifiers possess a promising potential for generalization. Extending the techniques developed in the dissertation will address problems in a more realistic sense. Future research might focus on:

1. Different types of uncertainty such as in the case where the data are inside a box of uncertainty. We can adopt a similar way like in [17] to determine the uncertainty. Specifically, if X denote a $n \times N$ matrix of N nominal data points $x_i \in \Re^n$, with corresponding label vector $y \in \{-1, +1\}^N$, Σ is a $n \times N$ matrix of positive numbers, with columns $\sigma_i, i = 1, .., N$, then the data with uncertainty can be expressed in an interval matrix model as: $X(\rho) = \{Z \in \Re^{n \times N} : X - \rho\Sigma \leq Z \leq X + \rho\Sigma\}$,

2. Each data point has a different level of uncertainty. This means that each data point has different level of uncertainty u_i. In this research we assume each data point has the same level of uncertainty u.

3. Each attribute has a different level of uncertainty. This means that for every data point, the orientation of the uncertainty could be different for each attribute or dimension. In this research we assume that the orientation of the uncertainty is the same for each attribute or dimension.

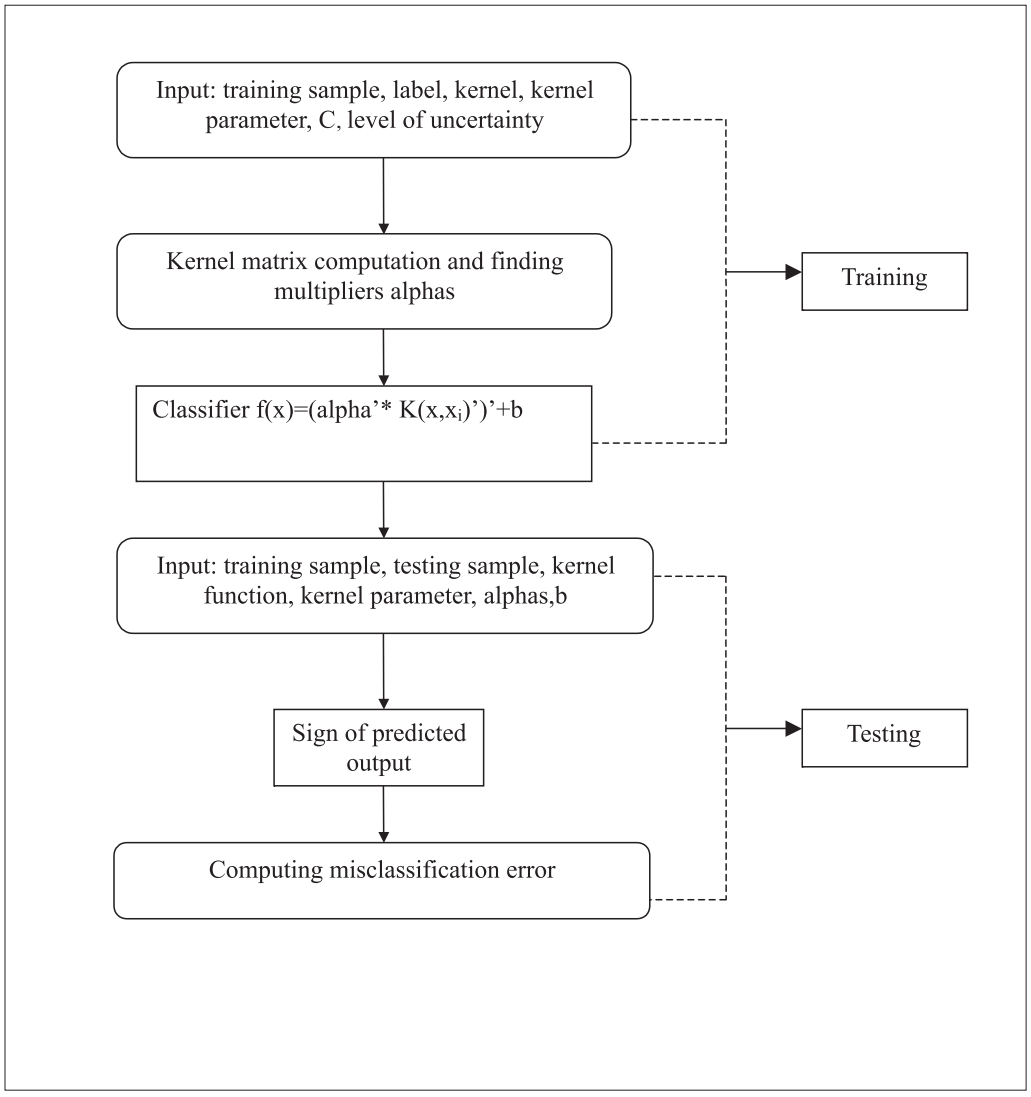
4. Using L_1 norm to develop a robust support vector machine as an alternative to the L_2 norm.

5. Investigate a theoritical error bound for the generalization error in the classification and regression case for the R-SVM approach.

6. Investigate computational methods for large scale problems following the R-SVM optimization model.

Appendix A

Matlab Code for Classification Problems



The flowchart for classification code

```
function [alpha,b]=tomtrainperc(x,y,ker,par,eta);
%This code is to compute alpha and b with SNOPT
% SNOPT is nonlinear solver embedded in TOMLAB
% This is a comewrcial software package
%for Feasibility Approach
%R(1) ... R(m) : represents alpha
% R(m+1) : represents b
% input : x - training sample
% y - training label, should have the same number
% of rows as x
% ker - kernel function
% par - kernel parameter
% eta - level of uncertainty
% output: alpha - lagrange multiplier
% b - bias
[m,c] =size(x);
% Finding kernel matrix
k=kernel(x',ker,par);
z=y;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Optimization part
%%%%%%%%%%%%%%%%
x0=zeros(m+2,1);
lb1=-inf*ones(m+1,1);
ubc=-ones(m,1);
if eta~=0
%problem structure in Tomlab
Prob=conAssign('ofxor','ofxor_g',[],[],[lb1;0],[],'coba',x0,[],[],[],...
[],[],'perccon3','perccon3_dc',[],[],[],[ubc]);

    else
Prob=conAssign('ofxor','ofxor_g',[],[],[lb1;0],[],'coba',x0,[],[],[],...
[],[],'perccon2','perccon2_dc',[],[],[],[ubc]);
end
Prob.user.H=k;
Prob.user.z=z;
Prob.user.sqrteta=sqrt(eta);
R=tomRun('snopt', Prob, [], 2);
alpha=R.x_k(1:m);
b=R.x_k(m+1);
clear R;
%%%%%%%%%%%%%%%%%%%%%%%%
function  f=ofxor(x)
%Objective function
n=length(x)-2;
f=x(n+2);
```

```
%%%%%%%%%%%%%%%%%%%%%
function  g =ofxor_g(x)
%Gradient of the obj function
g=zeros(length(x),1);
g(length(x))=1;

%%%%%%%%%%%%%%%%%%%%%
function  [g]=perccon2(x,Prob);
%Constraint
k=Prob.user.H;
z=Prob.user.z;
n=length(x)-2;
kx = k'*x(1:n);
g=[z.*(-kx-x(n+1)) - x(n+2);kx'*x(1:n)];
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function  [J]=perccon2_dc(x,Prob);
%Gradient of the constraint
k=Prob.user.H;
z=Prob.user.z;
n = length(x)-2;
J=zeros(n,length(x));
J(1:n,1:n) = -diag(z)*k';
J(1:n,n+1) = -z;
J(1:n,n+2) = -1;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function  [g]=perccon3(x,Prob);
k=Prob.user.H;
z=Prob.user.z;
sqeta=Prob.user.sqrteta;
n = length(x) - 2;
x2k = x(1:n)'*k*x(1:n);
% s is uncertainty =(w,u)
s = sqeta*sqrt(x2k);
g=[-s + z.*(-k'*x(1:n)-x(n+1)) - x(n+2)];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function  [J]=perccon3_dc(x,Prob);
%Gradient of the constraint
k=Prob.user.H;
z=Prob.user.z;
sqeta=Prob.user.sqrteta;
n = length(x) - 2;
J=zeros(n,length(x));
x2k = x(1:n)'*k*x(1:n);
%This is to avoid zero matrix
x2k=x2k+1e-10*eye(size(x2k));
s = sqeta*sqrt(x2k);
kx = x(1:n)'*k;
```

```
J(1:n,1:n) = -diag(z)*k';
J(1:n,n+1) = -z;
J(1:n,n+2) = -1;
J(1:n,1:n) = J(1:n,1:n)-ones(n,1)*(sqeta/sqrt(x2k))*kx;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function  [alpha,b]=tomtrainsvm(x,y,ker,par,C,eta)
% This code is to compute alpha and b with SNOPT for svm problem
% R(1) ... R(m) : represents alpha
% R(m+1) : represents b
% input : x - training sample
% y - training label
% ker - kernel function
% par - kernel parameter
% C - cost constant or upper bound for alpha
% eta - level of uncertainty
% output: alpha - lagrange multiplier
% b - bias
[m,c]=size(x);
% Finding kernel matrix
k = zeros(m,m);
k=kernel(x',ker,par);
z=y;
% Optimization part
x0=zeros(2*m+1,1);
lb2=zeros(m,1);
lb1=-inf*ones(m+1,1);
ubc=-ones(m,1);
if eta ~=0
Prob = conAssign('svmof','svmof_g',[],[],[lb1;lb2],[],'coba2',x0,[],[],[],...
[],[],'svmcon','svmcon_dc',[],[],[],[ubc]);
else
Prob =conAssign('svmof','svmof_g',[],[],[lb1;lb2],[],'coba2',x0,[],[],[],...
[],[],'svmcon1','svmcon1_dc',[],[],[],[ubc]);
end
Prob.user.H=k;
Prob.user.z=z;
Prob.user.C=C;
Prob.user.sqrteta=sqrt(eta);
R=tomRun('snopt', Prob, [], 2);
alpha=R.x_k(1:m);
b=R.x_k(m+1);
clear R;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function  [f]=svmof(x,Prob);
%The obj function
k=Prob.user.H;
z=Prob.user.z;
```

```
C=Prob.user.C;
sqeta=Prob.user.sqrteta;
m=size(x,1);
n = 0.5*(m - 1);
x2k = x(1:n)'*k*x(1:n);
sumslack=C*sum(x(n+2:m));
f=[0.5*x2k + sumslack];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function  [J]=svmof_g(x,Prob);
%Gradient of the obj function
k=Prob.user.H;
C=Prob.user.C;
m=length(x);
n = 0.5*(m-1);
J=zeros(length(x),1);
J(1:n) = x(1:n)'*k;
J(n+1)=0;
J(n+2:m)=C*ones(n,1);

    function  [g]=svmcon(x,Prob);
%Constraint
k=Prob.user.H;
z=Prob.user.z;
sqeta=Prob.user.sqrteta;
m=length(x);
n = 0.5*(m-1);
x2k = x(1:n)'*k*x(1:n);
s = sqeta*sqrt(x2k);
kx = k'*x(1:n);
slack=zeros(n,1);
for j=1:n
slack(j)=x(n+1+j);
end
g=[-s + z.*(-kx-x(n+1))-slack];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [J]=svmcon_dc(x,Prob);
%Gradient of the constraint
k=Prob.user.H;
z=Prob.user.z;
sqeta=Prob.user.sqrteta;
m=length(x);
n = 0.5*(m-1);
J=zeros(n,length(x));
x2k = x(1:n)'*k*x(1:n);
x2k=x2k+1e-10*eye(size(x2k));
s = sqeta*sqrt(x2k);
kx = x(1:n)'*k;
```

```
J(1:n,1:n) = -diag(z)*k';
J(1:n,n+1) = -z;
J(1:n,n+2:m)=-eye(n);
J(1:n,1:n) = J(1:n,1:n)-ones(n,1)*(sqeta/sqrt(x2k))*kx;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [g]=svmcon1(x,Prob);
%constraint
k=Prob.user.H;
z=Prob.user.z;
m=size(x,1);
n =0.5*(m-1);
kx = k'*x(1:n);
slack=zeros(n,1);
for j=1:n
slack(j)=x(j+n+1);
end
g=[z.*(-kx-x(n+1)) -slack];


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function  [J]=svmcon1_dc(x,Prob);
%Gradient of the constraint
k=Prob.user.H;
z=Prob.user.z;
m=size(x,1);
n =0.5*(m-1);
J=zeros(n,m);
J(1:n,1:n) = -diag(z)*k';
J(1:n,n+1) = -z;
J(1:n,n+2:m)=-eye(n);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function  yt=testperc(x,tstx,ker,par,alpha,b)
%to predict the label of data point with feasibility approach or svm
% input : x - training sample
% tstx- testing sample
% ker - kernel function (should be the same as in training phase)
% par - kernel parameter (should be the same as in training phase
% output: yt - binary label 1 or -1
%run after tomtrainperc.m or tomtrainsvm.m
m = size(tstx,1);
k=kernel(tstx',x',ker,par);
yt=(alpha'*k')'+b;
for i=1:m
if yt(i)<0
yt(i)=-1;
else
yt(i)=1;
```

```
end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function  [alpha0,b0]=tommultiperc(x,y,ker,par,eta)
%This code is to compute alpha0 , b0 for multiclass kernel
% with feasibility approach with one-against-all approach
% input : x - training sample for training
% ker - kernel function
% y - training label
% eta - degree of uncertainty
% output: alpha0 - lagrange multiplier
% b0 - bias
xsup0=[];
alpha0=[];
b0=[];
st = cputime;
Mmax=max(y);label: 1, 2, 3, ..., M
Mmin=min(y);label: 1, 2, 3, ..., M
M = Mmax-Mmin + 1;
nbsv=zeros(1,M);
nbsuppvector=zeros(1,M);
for i=1:M
yone=(y==i)+(y ~=i)*(-1);
[alpha,b]=tomtrainperc(x,yone,ker,par,eta);
alpha=alpha';
alpha0=[alpha0;alpha];
b0=[b0;b];
end;
fprintf('Computation-time: %4.1f seconds\n',cputime - st);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function  [alpha0,b0,nbdata,xsup]=tommultiperconeaone(x,y,ker,par,eta)
%This code is to compute alpha0, b0 for multiclass kernel %with feasibility approach with
one-against-one approach
%input : x - training sample
% y - training label
% ker - kernel function
% par - kernel parameter
% eta - degree of uncertainty
% output: alpha0 - lagrange multiplier
% nbdata - number of data for each class
% b0 - bias
alpha0=[];
xsup=[];
b0=[];
classifier=[];
Mmax=max(y);label: 1, 2, 3, ..., M
Mmin=min(y);label: 1, 2, 3, ..., M
```

```
M = Mmax-Mmin + 1;
nbdata=zeros(1,M*(M-1)/2);
nbalpha=zeros(1,M*(M-1)/2);
st = cputime;
k=1;
for i=1:M-1
for j=i+1:M
indi=find(y==i);
indj=find(y==j);
xapp=[x(indi,:); x(indj,:)];
yone=[ones(length(indi),1);-ones(length(indj),1)];
[alpha,b]=tomtrainperc(xapp,yone,ker,par,eta);
xsup=[xsup;xapp];
n2=size(xapp,1);
alpha=alpha';
n1=size(alpha,2);
nbalpha(k)=n1;
nbdata(k)=n2;
classifier(k,:)=[i j];
alpha0=[alpha0;alpha'];
b0=[b0;b];
k=k+1;
end;
end;
fprintf('Computation-time: %4.1f seconds\n',cputime - st);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function  [alpha0,b0]=tommultisvm(x,y,ker,par,C,eta)
%This code is to compute alpha0 , b0 for multiclass svm
%with one-against-all approach
%input : x - training sample
% y - training label
% ker - kernel function
% par - kernel parameter
% C - constant cost(upper bound for alpha)
% eta - degree of uncertainty
% output: alpha0 - lagrange multiplier
% b0 - bias
xsup0=[];
alpha0=[];
b0=[];
st = cputime;
Mmax=max(y);label: 1, 2, 3, ..., M
Mmin=min(y);label: 1, 2, 3, ..., M
M = Mmax-Mmin + 1;
nbsv=zeros(1,M);
nbsuppvector=zeros(1,M);
for i=1:M
yone=(y==i)+(y~=i)*(-1);
```

```
[alpha,b]=tomtrainsvm(x,yone,ker,par,C,eta);
alpha=alpha';
alpha0=[alpha0;alpha];
b0=[b0;b];
end;
fprintf('Computation-time: %4.1f seconds\n',cputime - st);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function  [alpha0,b0,nbdata,xsup]=tommultisvmoneaone(x,y,ker,par,C,)
%This code is to compute alpha0, b0 for multiclass kernel svm
%using SNOPTwith one-against-one approach
%input : x - training sample
% y - training label
% ker - kernel function
% par - kernel parameter
% eta - degree of uncertainty
% output: alpha0 - lagrange multiplier
% b0 - bias
alpha0=[];
xsup=[];
b0=[];
classifier=[];
Mmax=max(y);label: 1, 2, 3, ..., M
Mmin=min(y);label: 1, 2, 3, ..., M
M = Mmax-Mmin + 1;
nbdata=zeros(1,M*(M-1)/2);
nbalpha=zeros(1,M*(M-1)/2);
st = cputime;
k=1;
for i=1:M-1
for j=i+1:M
indi=find(y==i);
indj=find(y==j);
xapp=[x(indi,:); x(indj,:)];
yone=[ones(length(indi),1);-ones(length(indj),1)];
[alpha,b]=tomtrainsvm(xapp,yone,ker,par,C,eta);
xsup=[xsup;xapp];
n2=size(xapp,1);
alpha=alpha';
n1=size(alpha,2);
nbalpha(k)=n1;
nbdata(k)=n2;
classifier(k,:)=[i j];
alpha0=[alpha0;alpha'];
b0=[b0;b];
clear alpha,b;
k=k+1;
end;
end;
```

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

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function  yt=testmultiperc(x,tstx,nbclass,ker,par,alpha0,b0)
%This to predict the label of data point with feasibility approach
%using one against all
%input : x - training set
%tstx - testing set
%nbclass-number of class
%ker - kernel function (should be the same as in training phase)
%par - kernel parameter (should be the same as in training phase
%alpha0,b0 - obtained from running trainmultiperc.m
% output : yt - predicted label for testing set
n1 = size(tstx,1)
H=kernel(tstx',x',ker,par);
size(H)
for i=1:nbclass
yt(:,i)=(alpha0(i,:)*H')'+b0(i);
end
[maxi,yt]=max(yt');
yt=yt';

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function  yt=testmultisvm(x,tstx,nbclass,ker,par,alpha0,b0)
%This to predict the label of data point with svm
%using one against all
%input : x - training set
%tstx - testing set
%nbclass-number of class
%ker - kernel function (should be the same as in training phase)
%par - kernel parameter (should be the same as in training phase
%alpha0,b0 - obtained from running trainmultisvm.m
% output : yt - predicted label for testing set
n = size(x,1);
H=kernel(tstx',x',ker,par);
for i=1:nbclass
yt(:,i)=(alpha0(i,:)*H')'+b0(i);
end
[maxi,yt]=max(yt');
yt=yt';

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function  yt=testmultiperconeaone(xsup,tstx,nbdata,ker,par,alpha0,b0)
%This code is to predict the label of data point with feasibility approach % for multiclass
with one against one
%input :
% xsup - obatained from training phase
%tstx - testing sample
%nbdata- number of data of each class
```

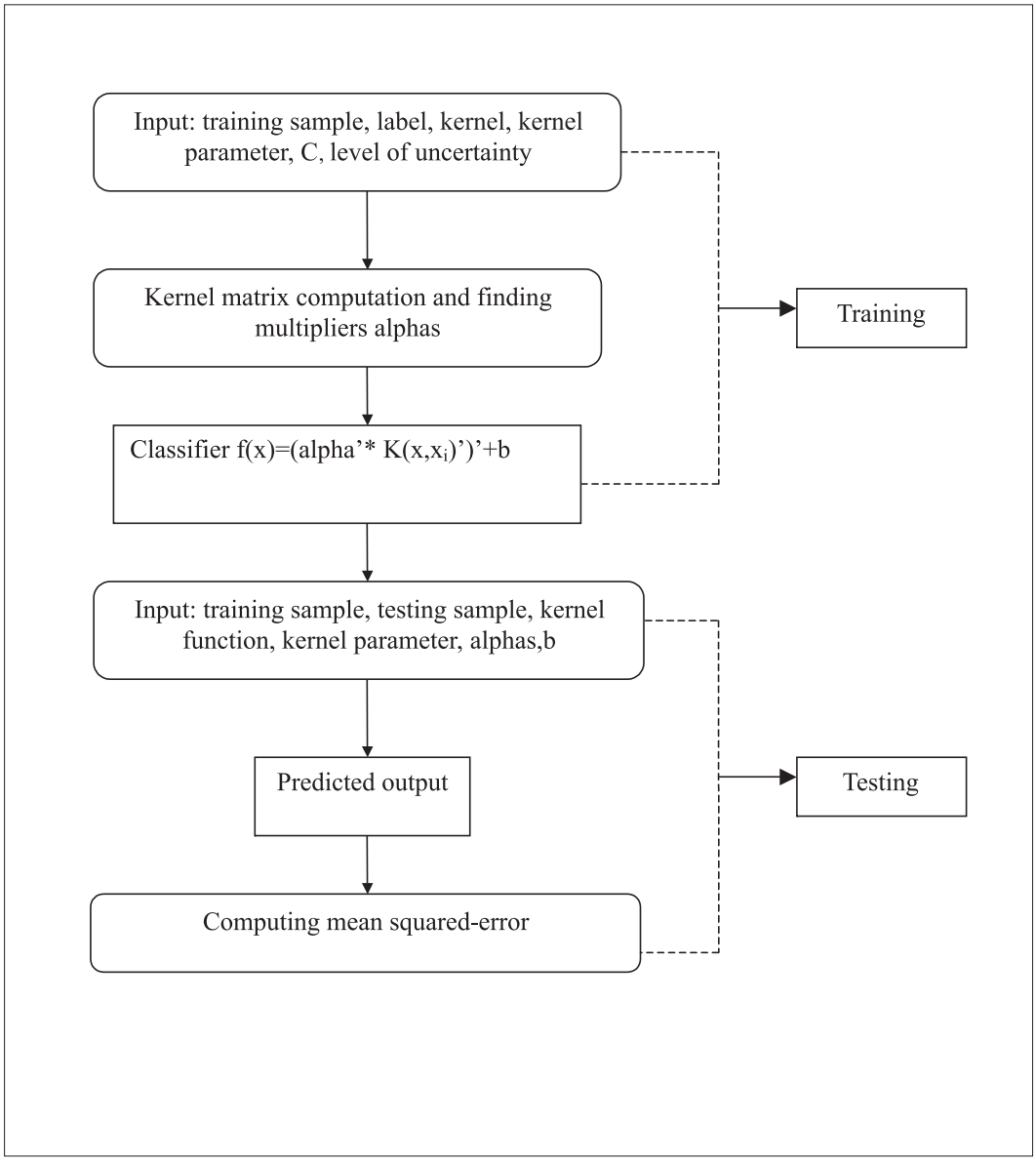
```
%ker - kernel function (should be the same as in training phase)
%par - kernel parameter (should be the same as in training phase
%alpha0,b0 - obtained from running trainmultiperconeaone.m
% output : yt - predicted label for testing set
%run after trainmultiperconeaone.m
[n1,n2]=size(tstx);
nbclass=(1+ sqrt(1+4*2*length(nbdata)))/2;
vote=zeros(n1,nbclass);
k=1;
nbdata=[0 nbdata];
aux=cumsum(nbdata);
for i=1:nbclass-1;
for j=i+1:nbclass;
xaux=xsup(aux(k)+1:aux(k)+nbdata(k+1),:);
alphaaux=alpha0(aux(k)+1:aux(k)+nbdata(k+1));
baux=b0(k);
H=kernel(tstx',xaux',ker,par);
yt=(alphaaux'*H')'+baux;
indi=find(yt≥0);
indj=find(yt<0);
vote(indi,i)=vote(indi,i)+1;
vote(indj,j)=vote(indj,j)+1;
k=k+1;
end
end
[maxi,yt]=max(vote');
yt=yt';

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function yt=testmultisvmoneaone(xsup,tstx,nbdata,ker,par,alpha0,b0)
%This code is to predict the label of data point with SVM
%for multiclass with one against one
%input : xsup - obatained from training phase
% tstx - testing sample
% nbdata- number of data of each class
% ker - kernel function (should be the same as in training phase)
% par - kernel parameter (should be the same as in training phase
% alpha0,b0 - obtained from running trainmultiperconeaone.m
% output : yt - predicted label for testing set
%run after trainmultisvmoneaone.m
[n1,n2]=size(tstx);
nbclass=(1+ sqrt(1+4*2*length(nbdata)))/2;
vote=zeros(n1,nbclass);
k=1;
nbdata=[0  nbdata];
aux=cumsum(nbdata);
for i=1:nbclass-1;
for j=i+1:nbclass;
xaux=xsup(aux(k)+1:aux(k)+nbdata(k+1),:);
```

```
alphaaux=alpha0(aux(k)+1:aux(k)+nbdata(k+1));
baux=b0(k);
H=kernel(tstx',xaux',ker,par);
yt=(alphaaux'*H')+baux;
indi=find(yt≥0);
indj=find(yt<0);
vote(indi,i)=vote(indi,i)+1;
vote(indj,j)=vote(indj,j)+1;
k=k+1;
end
end
[maxi,yt]=max(vote');
yt=yt';
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function K = kernel(varargin)
% kernel Computes kernel matrix.
% Synopsis:
% K = kernel( X, ker, arg )
% K = kernel( X1, X2, ker, arg )
% Description:
% K = kernel( X, ker, arg ) returns matrix K [n x n] where n is
% number of input data n=size(X,2). The matrix K contains
% values of kernel function for given data X, i.e.,
% K(i,j)=k(X(:,i),X(:,j)), i,j=1,...,n.
% K = kernel( X1, X2, ker, arg ) return matrix [n x m] where n
% is number of data in X1 and X2, i.e., n=size(X1,2) and
% m=size(X2,2). The kernel matrix K contains values of kernel
% functions for given data X1 and X2, i.e.,
% K(i,j)=k(X1(:,i),X2(:,j)), i = 1, ..., n, j = 1, ..., m.
% This function is only interface for C-code. To make executable
% file run mex kernelccode.c kernel.c
% Input:
% X [dim x n] Input data.
% or
% X1 [dim x n], X2 [dim x m] Input data.
% ker [string] Kernel identifier.
% arg [...] Argument of selected kernel.
% Output:
% K [n x n] Kernel matrix.
% or
% K [n x m] Kernel matrix.
% About: Statistical Pattern Recognition Toolbox
% (C) 1999-2003, Written by Vojtech Franc and Vaclav Hlavac
if nargin ==3,
[ker,arg]=kerarg(varargin2, varargin3);
K = kernelccode( varargin1, ker, arg );
else
[ker,arg]=kerarg(varargin3, varargin4);
K = kernelccode( varargin1, varargin2, ker, arg);
```

```
end
return;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [ker,arg]=kerarg(name,arg)
% kerarg Returns kernel identifier and adjusted arguments.
% [ker,arg]=kerarg(name,arg)
% Description:
% This function serves as an interface between C-code functions
% using library kernel.c and M-files.
% The input is a given kernel name and corresponding arguments.
% The output is the kernel identifier (integer) and kernel argument
% adjusted if necesary.
% Input:
% name [string] Kernel name.
% arg [...] Kernel argument.
% Output:
% id [int] Kernel identifier.
% arg [...] Kernel argument.
% About: Statistical Pattern Recognition Toolbox
% (C) 1999-2003, Written by Vojtech Franc and Vaclav Hlavac
% http://www.cvut.cz Czech Technical University Prague
% http://www.feld.cvut.cz Faculty of Electrical Engineering
% http://cmp.felk.cvut.cz Center for Machine Perception
% Modifications:
% 4-June-2003, VF
switch lower(name)
case 'linear'
ker = 0; arg = 0;
case 'poly'
ker = 1;
if isempty(arg), arg = [1,1];
elseif length(arg) < 2, arg=[arg,1]; end
case 'rbf'
ker = 2;
if isempty(arg), arg = 1; end
case 'sigmoid'
ker = 3;
if isempty(arg), arg = [1,1]; end
otherwise
error('Unknown kernel identifier.');
end
return;
```

Appendix B

Matlab Code for Regression Analysis



The flowchart for regression code

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function  [alpha,b]=percreg(x,y,ker,par,epsi,eta)
% This code is to compute alpha and b with SNOPT in feasibility
%approach for regression
% w(1) ... w(m) : represents alpha
% w(m+1) : represents b
% input : x - training sample
% y - training label
% ker - kernel function
% par - kernel parameter
% epsi- epsilon
% eta - level of uncertainty
% output: alpha - lagrange multiplier
% b - bias
[m, c] = size(x);
k = zeros(m,m);
k=kernel(x',ker,par);
z=y;
%%%%%%%%%%%%%%%%%
% Optimization part
x0=zeros(2+m,1);
lb2=zeros(1,1);
lb1=-inf*ones(m+1,1);
ubc1=epsi*ones(m,1);
ubc2=epsi*ones(m,1);
%This the structure of problem using Tomlab
if eta ~=0
Prob = conAssign('percregof','percregof_g',[],[],[lb1;lb2],[],'coba2',x0,[],[],[],...
[],[],'percregcon','percregcon_dc',[],[],[],[ubc1;ubc2]);
else
Prob = conAssign('percregof','percregof_g',[],[],[lb1;lb2],[],'coba2',x0,[],[],[],...
[],[],'percregcon1','percregcon1_dc',[],[],[],[ubc1;ubc2]);
end
Prob.user.H=k;
Prob.user.z=z;
Prob.user.sqrteta=sqrt(eta);
Prob.SOL.optPar(10)=0.0001;
Prob.SOL.optPar(30)=300000;
R=tomRun('snopt', Prob, [], 2);
alpha=R.x_k(1:m);
b=R.x_k(m+1);
clear R;
clear k;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function  [g]=percregcon1(x,Prob);
m=length(x);
n=(m-2);
```

```
k=Prob.user.H;
y=Prob.user.z;
x2k = x(1:n)'*k*x(1:n);
kx = k'*x(1:n);
g=[y-kx-x(n+1)-x(n+2);-y+kx+x(n+1)-x(n+2)];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function  [J]=percregcon1_dc(x,Prob);
k=Prob.user.H;
m=length(x);
n = (m-2);
J=zeros(2*n+1,length(x));
x2k = x(1:n)'*k*x(1:n);
x2k=x2k+1e-10*eye(size(x2k));
kx = x(1:n)'*k;
J(1:n,1:n) =-k';
J(1:n,n+1) = -1;
J(1:n,n+2)=-1;
J(n+1:2*n,1:n) =k';
J(n+1:2*n,n+1) = 1;
J(n+1:2*n,m)=-1;


%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function  [g]=percregcon(x,Prob);
m=length(x);
n=(m-2);
k=Prob.user.H;
y=Prob.user.z;
sqeta=Prob.user.sqrteta;
x2k = x(1:n)'*k*x(1:n);
s = sqeta*sqrt(x2k);
kx = k'*x(1:n);
g=[y-kx-s-x(n+1)-x(n+2);-y+kx+s+x(n+1)-x(n+2)];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function  [J]=percregcon_dc(x,Prob);
k=Prob.user.H;
y=Prob.user.z;
sqeta=Prob.user.sqrteta;
m=length(x);
n = (m-2);
J=zeros(2*n,length(x));
x2k = x(1:n)'*k*x(1:n);
x2k=x2k+1e-10*eye(size(x2k));
s = sqeta*sqrt(x2k);
kx = x(1:n)'*k;
J(1:n,1:n) =-k';
J(1:n,n+1) = -1;
```

```
J(1:n,n+2)=-1;
J(1:n,1:n) = J(1:n,1:n)-ones(n,1)*(sqeta/sqrt(x2k))*kx;
J(n+1:2*n,1:n) =k';
J(n+1:2*n,n+1) = 1;
J(n+1:2*n,m)=-1;
J(n+1:2*n,1:n) = J(n+1:2*n,1:n)+ones(n,1)*(sqeta/sqrt(x2k))*kx;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function  f=percregof(x,Prob) m=size(x,1);
n=(m-2);
sumslack=sum(x(n+2));
f=sumslack;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function  g=percregof_g(x,Prob)
m=length(x);
n = (m-2);
g=zeros(length(x),1);
g(1:n+1) =zeros(n+1,1);
g(n+2)=ones(1,1);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function  [alpha,b]=svmreg(x,y,ker,par,epsi,C,eta)
% This code is to compute alpha and b with SNOPT for sv regression
% R(1) ... R(m) : represents alpha
% R(m+1) : represents b
% input : x - training sample
% y - training label
% ker - kernel function
% par - kernel parameter
% epsi- epsilon
% eta - level of uncertainty
% output: alpha - lagrange multiplier
% b - bias
[m,c ]= size(x);
k=kernel(x',ker,par);
z=y;
% Optimization part
x0=zeros(3*m+1,1);
lb2=zeros(2*m,1);
lb1=-inf*ones(m+1,1);
ubc1=epsi*ones(m,1);
ubc2=epsi*ones(m,1);
if eta ~=0
Prob = conAssign('svmregof','svmregof_g',[ ],[ ],[lb1;lb2],[ ],'coba2',x0,[],[],[],...
[],[],'svmregcon','svmregcon_dc',[],[],[],[ubc1;ubc2]);
else
Prob = conAssign('svmregof','svmregof_g',[],[],[lb1;lb2],[],'coba2',x0,[],[],[],...
[],[],'svmregcon1','svmregcon1_dc',[],[],[],[ubc1;ubc2]);
```

```
end
Prob.user.H=k;
Prob.user.z=z;
Prob.user.C=C;
Prob.user.sqrteta=sqrt(eta);
Prob.SOL.optPar(10)=0.00001;
Prob.SOL.optPar(30)=300000;
R=tomRun('snopt', Prob, [], 2);
alpha=R.x_k(1:m);
b=R.x_k(m+1);
clear R;
clear k;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function  [g]=svmregcon(x,Prob);
k=Prob.user.H;
y=Prob.user.z;
sqeta=Prob.user.sqrteta;
m=length(x);
n = (1/3)*(m-1);
x2k = x(1:n)'*k*x(1:n);
s = sqeta*sqrt(x2k);
kx = k'*x(1:n);
slack1=zeros(n,1);
for j=1:n
slack1(j)=x(n+1+j);
end
slack2=zeros(n,1);
for j=1:n
slack2(j)=x(2*n+1+j);
end
g=[y-kx-s-x(n+1)-slack1;-y+kx+s+x(n+1)-slack2];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function  [J]=svmregcon_dc(x,Prob);
%gradient of constraints of svr formulation
k=Prob.user.H;
y=Prob.user.z;
sqeta=Prob.user.sqrteta;
m=length(x);
n = (1/3)*(m-1);
J=zeros(2*n,length(x));
x2k = x(1:n)'*k*x(1:n);
x2k=x2k+1e-10*eye(size(x2k));
s = sqeta*sqrt(x2k);
kx = x(1:n)'*k;
J(1:n,1:n) =-k';
J(1:n,n+1) = -1;
J(1:n,n+2:2*n+1)=-eye(n);
```

```
J(1:n,1:n) = J(1:n,1:n)-ones(n,1)*(sqeta/sqrt(x2k))*kx;
J(n+1:2*n,1:n) =k';
J(n+1:2*n,n+1) = 1;
J(n+1:2*n,2*n+2:m)=-eye(n);
J(n+1:2*n,1:n) = J(n+1,2*n)+ones(n,1)*(sqeta/sqrt(x2k))*kx;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function  [f]=svmregof(x,Prob);
%svr objective function
k=Prob.user.H;
z=Prob.user.z;
C=Prob.user.C;
sqeta=Prob.user.sqrteta;
m=size(x,1);
n = (1/3)*(m - 1);
x2k = x(1:n)'*k*x(1:n);
sumslack=sum(x(n+2:m));
f=[0.5*x2k +C*sumslack];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function  [J]=svmregof_g(x,Prob);
%gradient of svr objective function
k=Prob.user.H;
C=Prob.user.C;
m=length(x);
n = (1/3)*(m-1);
J=zeros(length(x),1);
J(1:n) = x(1:n)'*k;
J(n+1)=0;
J(n+2:m)=C*ones(2*n,1);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function  yt=testsvmreg(x,tstx,ker,par,alpha,b);
%This code is to predict the output of testing set point with SVR/feasibility approach
%used after running svmreg
m = size(tstx,1);
k=kernel(tstx',x',ker,par);
yt=(alpha'*k')'+b;
```

Bibliography

[1] M.Z. Bazaraa, H.D. Sherali, and C.M. Shetty. *Nonlinear Programming: Theory and Algorithms*. John Wiley & Sons, New York, 1993.

[2] A Ben-Tal and A. Nemirovski. Robust solutions to uncertain linear program via convex programming. *Operation Research Letters*, 25(1):1–17, 1996.

[3] A Ben-Tal and A. Nemirovski. Robust convex optimization. *Mathematics of Operation Research*, 23(4):769–805, 1998.

[4] K. Bennett and O.L. Mangasarian. Robust linear programming discrimination of two linearly inseparable sets. *Optimization Methods and Software*, 1:23–34, 1992.

[5] K. Bennett and O.L. Mangasarian. Multicategory discrimination via linear programming. *Optimization Methods and Software*, 3:27–39, 1994.

[6] K. Bennett and O.L. Mangasarian. Serial and parallel multicategory discrimination. *SIAM Journal on Optimization*, 4(4):722–734, 1994.

[7] D. Bertsimas, D. Pachamanova, and M. Sim. Robust linear optimization under general norms. *Operations Research Letters*, 2004.

[8] J.R. Birge. The value of statistic solution in stochastic linear programming with fixed resources. *Math. Programming*, 24:314–325, 1982.

[9] C.L. Blake and C.J. Merz. UCI Repository of machine learning databases, 1998. http://www.ics.uci.edu/∼mlearn/MLRepository.html.

[10] S. Boyd, M.S Lobo, and L. Vandenberghe. Application of second-order cone programming. *Linear Algebra and its Applications*, 284:193–226, 1998.

[11] Erin J. Bredensteiner and Kristin P. Bennett. Multicategory classification by support vector machines. *Computational Optimizations and Applications*, 12:53–79, 1999.

[12] K. Chakraborty, K. Mehrotra, C.K. Mohan, and S. Ranka. Forecasting the behavior of multivariate time series using neural network. *Neural Networks*, 5:961–970, 1992.

[13] V. Chvatal. *Linear Programming*. Freeman and Company, 1983.

[14] Koby Crammer and Yoram Singer. On the algorithmic implementation of multiclass kernel-based vector machines. *Journal of Machine Learning Research*, 2:265–292, 2001.

[15] N. Cristianini and J. Shawe-Taylor. *An Introduction to Support Vector Machines*. Cambridge University Press, 2000.

[16] P. Dierckx. *Curve and Surface Fitting with Splines. Monographs on Numerical Analysis*. Clarendon Press, Oxford, 1993.

[17] L. El Ghaoui, G.R.G. Lanckriet, and G. Natsoulis. Robust classification with interval data. Technical report, Technical Report CSD-03-1279, Division of Computer Science, University of California, Berkeley, 2003. http://robotics.eecs.berkeley.edu/ gert/.

[18] L. El Ghaoui and H. Lebret. Robust solutions to least square problems to uncertain data matrices. *SIAM Journal on Matrix Analysis and Applications,*, 18:1035–1064, 1997.

[19] L. El Ghaoui, F. Oustry, and H. Lebret. Robust solutions to uncertain semidefinite programs. *SIAM J. of Optimization*, 18:1035–1064, 1997.

[20] P.E. Gill, W. Murray, and M.A. Saunders. Snopt: An SQP algorithm for large-scale constrained optimization. *SIAM Journal on Optimization*, Vol. 12(4):9791006, 2002.

[21] S.P. Han. A globally convergent method for nonlinear programmings. *J. Optimization Theory and Applications*, 22:297, 1977.

[22] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning: data mining , inference, and prediction*. Springer-Verlag, New York, 2001.

[23] Simon Haykin. *Neural Network: A Comprehensive Foundation*. Prentice Hall, New Jersey, 1999.

[24] C.-W. Hsu and C.-J. Lin. A comparison of methods for multi-class support vector machines. *IEEE Transactions on Neural Networks*, 13:415–425, 2002.

[25] V. Kecman. *Learning and soft computing :support vector machines, neural networks, and fuzzy logic*. MIT Press, Cambridge, 2001.

[26] P. Kouvelis and Gang Yu. *Robust Discrete Optimization and Its Applications*. Kluwer Academic Publishers, Boston, 1997.

[27] G.R.G. Lanckriet, L. El Ghaoui, C. Bhattacharyya, and M.I. Jordan. A robust minimax approach to classification. *Journal of Machine Learning Research*, 3:555–582, 2002.

[28] I. Popescue and D. Bertsimas. Optimal inequalities in probability theory: A convex optimization approach. Technical report, Technical Report TM62, INSEAD, 2001.

[29] M.J.D. Powell. *A Fast Algorithm for Nonlinearly Constrained Optimization Calculations*, volume 630, chapter Numerical Analysis. Lecture Notes in Mathematics,Springer Verlag, G.A.Watson (ed.), 1978.

[30] B. Santosa and T.B. Trafalis. Multiclass procedure for minimax probability machine. In C.H. Dagli, A.L. Buczak, J. Ghosh, M Embrechts, O.Ersoy, and S. Kercel, editors, *Intelligent Engineering Systems Through Artificial Neural Networks 14*, pages 447–452. ASME Press, 2004.

[31] B. Schölkopf and A.J. Smola. *Learning with Kernels*. The MIT Press, Cambridge, Massachusetts, 2002.

[32] J. Shawe-Taylor and N. Cristianini. *Kernel Methods for Pattern Analysis*. Cambridge University Press, 2004.

[33] A.J. Smola, Peter L. Bartlett, B. Schölkopf, and D. Schuurmans. *Adavances in Large Margin Classifiers*. The MIT Press, 2000.

[34] W.N. Street and O.L. Mangasarian. Improved generalization error via tolerant training. *Journal of Optimization Theory and Applications*, 96(2):259–279, 1998.

[35] T. B. Trafalis, O. Oladunni, and D. V. Papavassiliou. Two-phase flow regime identification with a multi-classification svm model. *Industrial and Engineering Chemistry Research*, submitted, 2005.

[36] T.B. Trafalis and S.A. Alwazzi. Robust optimization in support vector machine training with bounded errors. In *Proceedings of the International Joint Conference On Neural Networks, Portland, Oregon*, pages 2039–2042. IEEE Press, 2003.

[37] T.B. Trafalis and S.A. Alwazzi. Robust support vector regression and applications. In C.H. Dagli, A.L. Buczak, J. Ghosh, M Embrechts, O.Ersoy, and S. Kerc, editors, *Intelligent Engineering Systems Through Artificial Neural Networks 13*, pages 181–186. ASME Press, 2003.

[38] T.B. Trafalis, B. Santosa, and M.B. Richman. Tornado detection with kernel-based methods. In C.H. Dagli, A.L. Buczak, J. Ghosh, M Embrechts, O.Ersoy, and S. Kercel, editors, *Intelligent Engineering Systems Through Artificial Neural Networks 13*, pages 677–682. ASME Press, 2003.

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

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

[41] J. Weston and C. Watkins. Support vector machines for multi-class pattern recognition. In *Proceeding of the Seventh European Symposium on Articial Neural Networks ICANN'97*, 1998.