DISSERTATION

TECHNIQUES IN SUPPORT VECTOR CLASSIFICATION

Submitted by

Shawn Martin

Department of Mathematics

In partial fulfillment of the requirements

for the degree of Doctor of Philosophy

Colorado State University

Fort Collins, Colorado

Spring 2001

COLORADO STATE UNIVERSITY

April 1, 2001

WE HEREBY RECOMMEND THAT THE DISSERTATION PREPARED UNDER OUR SUPERVISION BY SHAWN MARTIN ENTITLED "TECHNIQUES IN SUPPORT VECTOR CLASSIFICATION" BE ACCEPTED AS FULFILLING IN PART REQUIREMENTS FOR THE DEGREE OF DOCTOR OF PHILOSOPHY.

Committee on Graduate Work

Adviser

Department Head

ABSTRACT OF DISSERTATION

TECHNIQUES IN SUPPORT VECTOR CLASSIFICATION

This work falls into the field of Pattern Classification and more generally Artificial Intelligence. *Classification* is the problem of assigning a "pattern" $\mathbf{z}$ to be a member of a finite set ("class") X or a member of a disjoint finite set Y. In case $\mathbf{z} \in \mathbb{R}^n$ and $X, Y \subset \mathbb{R}^n$ we can solve this problem using Support Vector Machines. *Support Vector Machines* are functions of the form

$$f(\mathbf{z}) = \text{sign}(\textstyle\sum_i \alpha_i \kappa(\mathbf{x}_i, \mathbf{z}) + \sum_j \beta_j \kappa(\mathbf{y}_j, \mathbf{z}) + b), \qquad (*)$$

where $\kappa : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$ and $\mathbf{z}$ is classified as a member of $X = \{\mathbf{x}_i\}$ if $f(\mathbf{z}) > 0$ and a member of $Y = \{\mathbf{y}_j\}$ otherwise. We consider three problems in classification, two of which concern Support Vector Machines.

Our first problem concerns feature selection for classification. *Feature selection* is the problem of identifying properties which distinguish between the two classes X and Y. Color, for example, distinguishes between apples and oranges, while shape may not. Our method of feature selection uses a novel combination of a linear classifier known as Fisher's discriminant and a nonlinear (polynomial) map known as the Veronese map. We apply our method to a problem in materials design.

Our second problem concerns the selection of the *kernel* $\kappa : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$ in $(*)$. For kernel selection we use a kernel version of the classical Gram-Schmidt

orthonormalization procedure again coupled with Fisher's discriminant. We apply our method to the materials design problem and to a handwritten digit recognition problem.

Finally, we consider the problem of training Support Vector Machines. Specifically, we develop a fast method for obtaining the coefficients α_i and β_j in $(*)$. Traditionally, these coefficients are found by solving a constrained quadratic programming problem. We present a geometric reformulation of the SVM quadratic programming problem. We then present, using this reformulation, a modified version of Gilbert's Algorithm for obtaining the coefficients α_i and β_j. We compare our algorithm with the Nearest Point Algorithm and with Sequential Minimal Optimization.

Shawn Martin
Department of Mathematics
Colorado State University
Fort Collins, Colorado 80523
Spring 2001

ACKNOWLEDGEMENTS

Thanks to my advisor, Michael Kirby, and to the department head, Rick Miranda, for helpful discussions.

The Wisconsin Breast Cancer Data used in Chapter 4 was made available by Dr. William H. Wolberg at the University of Wisconsin Hospital, Madison.

This work has been supported by research grant DOD-USAF Office of Scientific Research F49620-99-1-0034.

TABLE OF CONTENTS

Chapter 1

INTRODUCTION

In this dissertation we consider one problem in Pattern Classification and two additional problems in Support Vector Classification. In Chapter 2 we consider the problem of feature selection using the idea of nonlinear preprocessing, in Chapter 3 we address the problem of kernel selection for Support Vector Machines, and in Chapter 4 we consider the problem of training Support Vector Machines.

The ideas in Chapter 2 were motivated by a particular problem in Materials Design (see Section 2.5) and a solution to that problem developed in [53] (see Section 2.6). The ideas in Chapter 3 were motivated by the ideas in Chapter 2 and a notable lack of methods of kernel selection for Support Vector Machines. The ideas in Chapter 4 were motivated by a lack of fast algorithms for training Support Vector Machines.

As indicated by the title, our solutions to these problems may all be considered "Techniques in Support Vector Classification." Obviously, Chapters 3 and 4 lie within the realm of Support Vector Classification. In addition, the technique in Chapter 2 may be considered a special case of the method in Chapter 3.

Before presenting our techniques in Support Vector Classification, we give a brief overview of Pattern Classification in Section 1.1 and a detailed presentation of Support Vector Machines in Section 1.2.

1.1 The Classification Problem

The *classification problem* occurs in its simplest form, the two class problem, whenever we have two disjoint finite sets X and Y and we ask: should an object $\mathbf{z} \notin X \cup Y$ be classified as a member of X or Y? As an example, let's say that X is a pile of apples and Y is a pile of oranges. We have an apple $\mathbf{z}$. Does our apple $\mathbf{z}$ belong to X or Y? The multi-class problem occurs when we have additional sets. Say we have another set W which corresponds to a pile of bananas. Does our apple $\mathbf{z}$ belong to W?

There are a large number of books on this topic. Here we cite two classics, [13] and [16], and two more recent works, [44] and [48]. The classics are both very readable and accurate. Of the more recent works [44] is readable and interesting while [48] is very comprehensive. Each of these books addresses both the two class and the multi-class problems. For ease of exposition (and because Support Vector Machines are fundamentally binary in nature) we consider only the two class problem in this introduction. In Section 3.7 we consider very briefly a ten-class problem.

An important part of solving a classification problem is identifying a salient feature (e.g. the color of the fruit) that distinguishes between the two classes X and Y. This aspect of classification, known as *feature selection*, dictates the working form of the problem. A chapter on feature selection can be found in [16], [44], and [48]. A survey of the topic can be found in [12].

Once we select good features, which we require to be real numbers, our database $X \cup Y$ is a subset of $\mathbb{R}^n$, where the coordinates $(z_1, \ldots, z_n)$ of $\mathbb{R}^n$ are the selected features. Feature selection is typically performed to reduce the dimension n of the problem and to obtain quality features $z_1, \ldots, z_n$. In addition, feature selection often helps to obtain insight into the underlying physical problem.

Another important part of solving a classification problem is constructing the decision function. When the features have been selected and the database $X \cup Y$ made to lie in $\mathbb{R}^n$, a solution to the classification problem is a *decision function* $f : \mathbb{R}^n \to \{\pm 1\}$ with $f(X) = 1$ and $f(Y) = -1$. The form of the decision function, known as the decision function's *architecture*, is the subject of much research in Pattern Classification.

Ideally, the architecture of the decision function f should take some optimal form, and when the entire statistical situation is known, such a form is the Bayesian classifier. In Bayesian classification, we know both the a priori probabilities $P(X)$ and $P(Y)$ that $\mathbf{z}$ should be classified as a member of X or Y (respectively) and the conditional probabilities $p(\mathbf{z}|X)$ and $p(\mathbf{z}|Y)$ that $\mathbf{z}$ will occur if $\mathbf{z}$ should be classified as a member of X or Y. Using Bayes' Rule we can then calculate the a posteriori probabilities $p(X|\mathbf{z})$ and $p(Y|\mathbf{z})$ that $\mathbf{z}$ should be classified as a member of X or Y. The Bayesian classifier has the decision function

$$f(\mathbf{z}) = \begin{cases} 1 & \text{if } p(X|\mathbf{z}) > p(Y|\mathbf{z}) \\ -1 & \text{otherwise.} \end{cases}$$

By making the decision which is probably correct, the Bayesian classifier minimizes average classification error. See [13] for details.

Of course, the Bayesian classifier is often difficult to apply since we seldom have such detailed statistical knowledge of our problem. Since, however, the Bayesian classifier takes an optimal form, a logical approach to classifier design is to approximate $P(X),\ P(Y),\ p(\mathbf{z}|X)$, and $p(\mathbf{z}|Y)$. Once these probability densities are known, we can certainly apply the Bayesian classifier. The main difficulty here lies in approximating $p(\mathbf{z}|X)$ and $p(\mathbf{z}|Y)$. This is usually accomplished by either parametric or non-parametric estimation.

In parametric estimation the forms of the probability densities are assumed to be known (e.g. normal) and the parameters which define the forms (e.g. the mean

μ and the variance σ^2) are then estimated using the data. Methods for parameter approximation include maximum likelihood estimation, Bayesian inference, and maximum a posteriori estimation. See [13] and [48].

The main trouble with parametric estimation, of course, is that often the probability densities do not have the assumed form. In non-parametric estimation the conditional probabilities $p(\mathbf{z}|X)$ and $p(\mathbf{z}|Y)$ are approximated directly. The simplest way to do this is using histograms, and in fact, the methods of non-parametric estimation are typically variations on this idea. These methods include Parzen windows and k nearest neighbor density estimation. Again see [13] and [48].

Another approach to classifier design is to skip Bayesian theory entirely and directly acquire a decision function f with some given architecture (e.g. linear). These "non-Bayesian" methods, however, can usually be interpreted [44] as attempting to estimate the a posteriori probabilities $p(X|\mathbf{z})$ and $p(Y|\mathbf{z})$ via regression on the pairs $X \times \{1\}$ and $Y \times \{-1\}$.

The simplest non-Bayesian classifiers are the *linear discriminants*. These are classifiers based on separating the two classes X and Y by a hyperplane H. Once this is done the decision function f classifies a point $\mathbf{z} \notin X \cup Y$ as a member of X if it lies on the "X" side of H and a member of Y otherwise. The linear discriminants are different only in their selection of the hyperplane H. Methods of determining a separating hyperplane include Fisher's linear discriminant, the perceptron algorithm, least mean squares, and many variants on these. See [13] and [48].

Finally, we have the nonlinear non-Bayesian classifiers. Included here are nearest neighbor algorithms, neural networks (multi-layer perceptrons), radial basis function networks, polynomial classifiers, Support Vector Machines, and various combinations of classifiers (e.g. classifier networks). See [48] and [44].

1.2 Support Vector Machines

Support Vector Machines are classifiers (there are also Support Vector Machines which perform regression [46]) which were originally developed in [50], [6], and [10]. The maximal margin hyperplane was presented in [50], the use of kernels was suggested in [6], and the soft margin generalization was given in [10]. They are very adaptable (able to assume polynomial, Radial Basis Function, and Neural Network forms) and have been applied successfully in [41], [19], [35], and [55] to problems in handwritten digit recognition, text categorization, face detection, and protein sequence extraction. For an excellent introduction to Support Vector Machines see [7]. For an encyclopedic treatment see [51]. For the latest work on SVMs see [47].

1.2.1 Linear Case

The prototypical Support Vector Machine is a linear discriminant which uses an optimal separating hyperplane. This hyperplane is known as the maximal margin hyperplane and is computed via a quadratic programming problem.

Following [7], suppose our two classes $X = \{\mathbf{x}_i\}$ and $Y = \{\mathbf{y}_j\}$ are nonempty, finite subsets of $\mathbb{R}^n$. Denote by $(\mathbf{x}, \mathbf{y})$ the inner product of $\mathbf{x}$ with $\mathbf{y}$ in $\mathbb{R}^n$. Assume that X and Y are *linearly separable*. That is, assume there exists a separating hyperplane H with equation $(\mathbf{x}, \mathbf{w}) + b = 0,\ \mathbf{w} \neq \mathbf{0}$ satisfying

$$\begin{array}{rll} (\mathbf{x}_i, \mathbf{w}) + b \geq 1 & \text{for all} & \mathbf{x}_i \in X \\ (\mathbf{y}_j, \mathbf{w}) + b \leq -1 & \text{for all} & \mathbf{y}_j \in Y, \end{array}$$

with equality for some i and j. Define the *margin* of H to be twice the distance from H to a nearest point in X (or, equivalently, a nearest point in Y). If we have $\mathbf{x}_{i_0}$ and $\mathbf{y}_{j_0}$ such that $(\mathbf{x}_{i_0}, \mathbf{w}) + b = 1$ and $(\mathbf{y}_{j_0}, \mathbf{w}) + b = -1$ we can compute the margin. It is

$$\frac{(\mathbf{w}, \mathbf{x}_{i_0})}{\|\mathbf{w}\|} - \frac{(\mathbf{w}, \mathbf{y}_{j_0})}{\|\mathbf{w}\|} = \frac{1-b}{\|\mathbf{w}\|} - \frac{-1-b}{\|\mathbf{w}\|} = \frac{2}{\|\mathbf{w}\|}.$$

The Support Vector Machine is a linear discriminant which uses the *maximal margin hyperplane*. Since the margin is $\frac{2}{\|\mathbf{w}\|}$, we compute the maximal margin hyperplane by solving

$$\begin{array}{rl} \min & \frac{1}{2}\|\mathbf{w}\|^2 \\ \text{subject to} & \begin{cases} (\mathbf{x}_i, \mathbf{w}) + b \geq 1 & \text{for all } \mathbf{x}_i \in X \\ (\mathbf{y}_j, \mathbf{w}) + b \leq -1 & \text{for all } \mathbf{y}_j \in Y. \end{cases} \end{array}$$

Of course, this formulation is very restrictive as it assumes perfect separation of X and Y. For linearly separable data with errors, known as *nonseparable* data, we use the *soft margin generalization* (due to [10])

$$\begin{array}{rl} \min & \frac{1}{2}\|\mathbf{w}\|^2 + C(\sum_i \phi_i + \sum_j \psi_j), \quad C > 0 \\ \text{subject to} & \begin{cases} (\mathbf{x}_i, \mathbf{w}) + b \geq 1 - \phi_i \\ (\mathbf{y}_j, \mathbf{w}) + b \leq -1 + \psi_j \\ \phi_i, \psi_j \geq 0, \end{cases} \end{array}$$

obtained by incorporating the slack variables ϕ_i and ψ_j into the problem. (C is a fixed constant.)

The soft margin generalization can in turn be reformulated via its Wolfe dual (see [26])

$$\begin{array}{cc} \max & \left(\min L \right) \\ \alpha_i, \beta_j, \eta_i, \gamma_j & \mathbf{w}, b, \phi_i, \psi_j \\ \text{subject to} & \alpha_i, \beta_j, \eta_i, \gamma_j \geq 0, \end{array}$$

where the Lagrangian

$$\begin{array}{rcl} L & = & \frac{1}{2}\|\mathbf{w}\|^2 + C(\sum_i \phi_i + \sum_j \psi_j) \\ & & - \sum_i \alpha_i((\mathbf{x}_i, \mathbf{w}) + b - 1 + \phi_i) \\ & & - \sum_j \beta_j(-1 + \psi_j - (\mathbf{y}_j, \mathbf{w}) - b) \\ & & - \sum_i \eta_i \phi_i - \sum_j \gamma_j \psi_j. \end{array}$$

The Wolfe dual is preferable to the original formulation because it allows us to express the problem in terms of inner products. This will be seen to be the key to the nonlinear generalization in Section 1.2.2.

Since our problem is convex we can use the Kuhn-Tucker conditions (again see [26])

$$\begin{gathered}\nabla_{\mathbf{w}} L = \mathbf{w} - \sum_i \alpha_i \mathbf{x}_i + \sum_j \beta_j \mathbf{y}_j = \mathbf{0} \\ \nabla_b L = -\sum_i \alpha_i + \sum_j \beta_j = 0 \\ \nabla_{\phi_i} L = C - \alpha_i - \eta_i = 0 \\ \nabla_{\psi_j} L = C - \beta_j - \gamma_j = 0 \\ \alpha_i((\mathbf{x}_i, \mathbf{w}) + b - 1 + \phi_i) = 0 \\ \beta_j(-1 + \psi_j - (\mathbf{y}_j, \mathbf{w}) - b) = 0 \\ \eta_i \phi_i = 0 \\ \gamma_j \psi_j = 0\end{gathered}$$

to rewrite the Wolfe dual as

$$\begin{array}{rl} \max \\ \alpha_i, \beta_j \end{array} \left\{ \begin{array}{l} \sum_i \alpha_i + \sum_j \beta_j - \frac{1}{2} \sum_{i,q} \alpha_i \alpha_q (\mathbf{x}_i, \mathbf{x}_q) \\ + \sum_{i,j} \alpha_i \beta_j (\mathbf{x}_i, \mathbf{y}_j) - \frac{1}{2} \sum_{j,q} \beta_j \beta_q (\mathbf{y}_j, \mathbf{y}_q) \end{array} \right. \qquad (1.1)$$

$$\text{subject to} \quad \left\{ \begin{array}{l} \sum_i \alpha_i = \sum_j \beta_j \\ 0 \leq \alpha_i, \beta_j \leq C \end{array} \right.$$

with solution

$$\begin{aligned} \mathbf{w} &= \sum_i \alpha_i \mathbf{x}_i - \sum_j \beta_j \mathbf{y}_j \\ b &= 1 - (\mathbf{x}_i, \mathbf{w}) \text{ for } \alpha_i \in (0, C) \\ &= -1 - (\mathbf{y}_j, \mathbf{w}) \text{ for } \beta_j \in (0, C) \end{aligned}$$

and corresponding linear discriminant decision function $f : \mathbb{R}^n \to \{0, \pm 1\}$ given by

$$f(\mathbf{z}) = \text{sign}((\mathbf{z}, \mathbf{w}) + b),$$

where $\mathbf{z}$ is classified as a member of X if $f(\mathbf{z}) > 0$ and a member of Y otherwise.

We remark that the solution $\mathbf{w}$ is global and unique; that the representation $\sum_i \alpha_i \mathbf{x}_i - \sum_j \beta_j \mathbf{y}_j$ of $\mathbf{w}$ may not be unique, but is generally sparse (many of the α_i and β_j are zero); and that the $\mathbf{x}_i$ and $\mathbf{y}_j$ which correspond to the nonzero α_i and β_j are known as *support vectors*. We provide in Figure 1.1 an illustration of linear SVMs in both the separable and nonseparable cases.

1.2.2 Nonlinear Case Using Kernels

In order to implement a Support Vector Machine that works with data which is nonlinearly separable, we first preprocess the data with an appropriate nonlinear

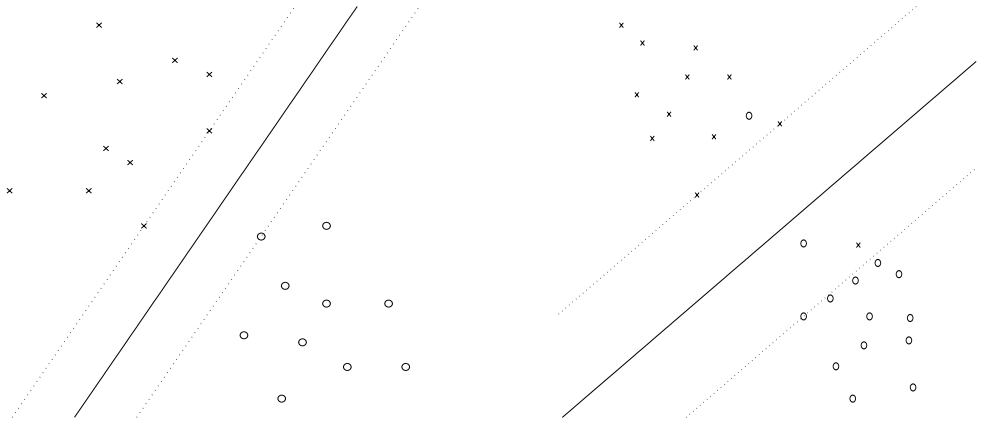

Figure 1.1: Linear Support Vector Machines. On the left (a) we show a linear Support Vector Machine in the case of separable data. The two classes X and Y are depicted by x's and o's, with the maximal margin hyperplane shown as a solid line separating them. The two dotted lines run through the support vectors and are separated by a distance equal to the margin. On the right (b) we show a linear Support Vector Machine in the case of nonseparable data.

map. Consider, for example, the exclusive-or problem illustrated in Figure 1.2(a). In this case the data is not linearly separable in $\mathbb{R}^2$, but if we apply the nonlinear map $\Phi : \mathbb{R}^2 \to \mathbb{R}^3$ given by

$$\Phi(x, y) = (x^2, \sqrt{2}xy, y^2), \tag{1.2}$$

we see that the data becomes linearly separable in $\mathbb{R}^3$, as shown in Figure 1.2(b).

This example provides a nice illustration of an appropriate application of nonlinear preprocessing, but it also hints at a fundamental flaw in the direct application of the technique. The map $\Phi : \mathbb{R}^2 \to \mathbb{R}^3$ results in an increase in the ambient dimension of the problem. This increase typically becomes more significant as the complexity of the problem increases. To avoid these problems we consider only a special class of maps for nonlinear preprocessing.

Specifically, we consider any map $\Phi : \mathbb{R}^n \to F$ which has an associated *kernel* function $\kappa : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$ with the property that

$$\kappa(\mathbf{x}, \mathbf{y}) = (\Phi(\mathbf{x}), \Phi(\mathbf{y})),$$

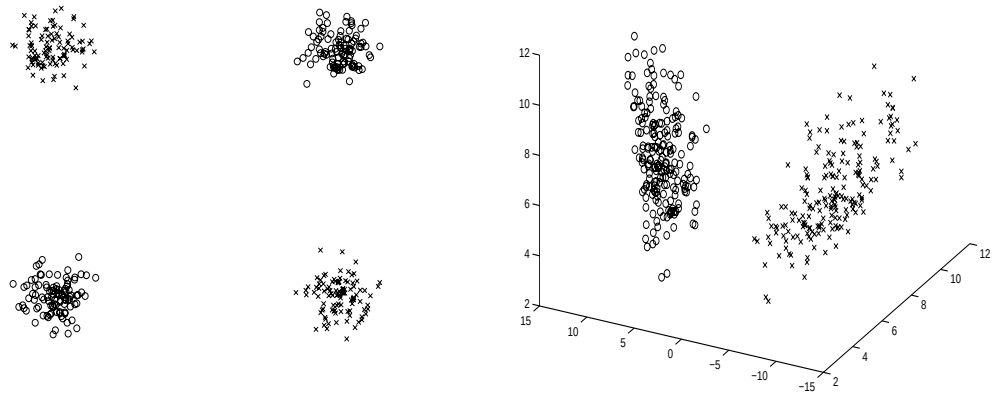

Figure 1.2: The Exclusive-Or Problem. On the left (a) we see that the exclusive-or problem is not linearly separable. On the right (b) the exclusive-or problem becomes linearly separable after nonlinear preprocessing by Φ in Equation (1.2).

where $(\Phi(\mathbf{x}), \Phi(\mathbf{y}))$ denotes the inner product of $\Phi(\mathbf{x})$ with $\Phi(\mathbf{y})$ in the Hilbert space F. It is, for example, easy to check that

$$\begin{aligned} \kappa(\mathbf{x}, \mathbf{y}) &= (\mathbf{x}, \mathbf{y})^2 \\ &= (x_1 y_1 + x_2 y_2)^2 \\ &= x_1^2 y_1^2 + 2 x_1 y_1 x_2 y_2 + y_2^2 x_2^2 \\ &= (\Phi(\mathbf{x}), \Phi(\mathbf{y})) \end{aligned}$$

is a kernel for Φ in (1.2). Other known kernels include (see [7])

- the Veronese kernel $\kappa(\mathbf{x}, \mathbf{y}) = ((\mathbf{x}, \mathbf{y}) + c)^d,\ c \geq 0,\ d \in \mathbb{Z}_{>0}$,
- the radial basis function (RBF) kernel $\kappa(\mathbf{x}, \mathbf{y}) = e^{\frac{-\|\mathbf{x}-\mathbf{y}\|^2}{2\sigma^2}},\ \sigma \neq 0$,
- the neural network kernel $\kappa(\mathbf{x}, \mathbf{y}) = \tanh(a(\mathbf{x}, \mathbf{y}) + b),\ a, b \geq 0$.

There are additional restrictions on the domain of the neural network kernel which can be found in [8].

Such kernels allow us to apply nonlinear preprocessing to our original data without actually applying a nonlinear map Φ. This is accomplished in practice by replacing inner products with kernels. In the case of the Support Vector Machine

problem (1.1) this gives

$$\begin{array}{ll} \max \\ \alpha_i, \beta_j \end{array} \quad \left\{ \begin{array}{l} \sum_i \alpha_i + \sum_j \beta_j - \frac{1}{2} \sum_{i,q} \alpha_i \alpha_q \kappa(\mathbf{x}_i, \mathbf{x}_q) \\ + \sum_{i,j} \alpha_i \beta_j \kappa(\mathbf{x}_i, \mathbf{y}_j) - \frac{1}{2} \sum_{j,q} \beta_j \beta_q \kappa(\mathbf{y}_j, \mathbf{y}_q) \end{array} \right. \tag{1.3}$$
$$\text{subject to} \quad \left\{ \begin{array}{l} \sum_i \alpha_i = \sum_j \beta_j \\ 0 \leq \alpha_i, \beta_j \leq C. \end{array} \right.$$

For more information on this approach see [7], [6], and [51].

We refer to the formulation in Equation (1.3) as the *Support Vector Machine Quadratic Programming problem.* If $C = \infty$ this is the *separable* SVM QP problem, and if $\kappa(\mathbf{x}, \mathbf{y}) = (\mathbf{x}, \mathbf{y})$ it is the *linear* SVM QP problem. The Support Vector Machine quadratic programming problem has decision function, officially known as *Support Vector Machine*

$$\begin{array}{rcl} f(\mathbf{z}) & = & \text{sign}((\mathbf{z}, \mathbf{w}) + b) \\ & = & \text{sign}(\sum_i \alpha_i \kappa(\mathbf{z}, \mathbf{x}_i) - \sum_j \beta_j \kappa(\mathbf{z}, \mathbf{y}_j) + b), \end{array}$$

where

$$\begin{array}{rcl} b & = & 1 - \sum_q \alpha_q \kappa(\mathbf{x}_i, \mathbf{x}_q) + \sum_j \beta_j \kappa(\mathbf{x}_i, \mathbf{y}_j) \text{ for } \alpha_i \in (0, C) \\ & = & -1 - \sum_i \alpha_i \kappa(\mathbf{y}_j, \mathbf{x}_i) + \sum_q \beta_q \kappa(\mathbf{y}_j, \mathbf{y}_q) \text{ for } \beta_j \in (0, C). \end{array}$$

The remarks made in the previous section also apply here. In particular, the solution f to the SVM QP problem is global and unique with many of the α_i and β_j zero. We make the additional observation that the architecture of the Support Vector Machine f is determined by the kernel κ, and to a lesser degree the constant C.

Chapter 2

FEATURE SELECTION FOR CLASSIFICATION

Here we consider the problem of feature selection. A chapter on feature selection is usually included in the more modern books on Pattern Classification (e.g. [48], [44], and [16]). A recent survey of the topic can also be found in [12].

To state the problem precisely, suppose that in addition to our database $X \cup Y$ we have a list $F_1, \ldots, F_m$ of possibly useful features. The goal of feature selection is a sub-list $F_{k_1}, \ldots, F_{k_n}$ of minimal length n which best distinguishes between our two classes X and Y. This sub-list should simplify the classification problem and give some insight into the underlying physical problem.

Our method of feature selection uses a nonlinear version of Fisher's discriminant to test the classification ability of a given feature. In Section 2.1 we describe Fisher's discriminant; in Sections 2.2 and 2.3 we introduce the Veronese map as a method of nonlinear preprocessing for Fisher's discriminant; in Section 2.4 we incorporate our nonlinear version of Fisher's discriminant into an exhaustive feature search; in Section 2.5 we present an application of our feature search to a problem in Materials Design; in Section 2.6 we compare our method with the method in [53]; and in Section 2.7 we place our method in the context of other methods. An alternate presentation of the material in this chapter may be found in [29].

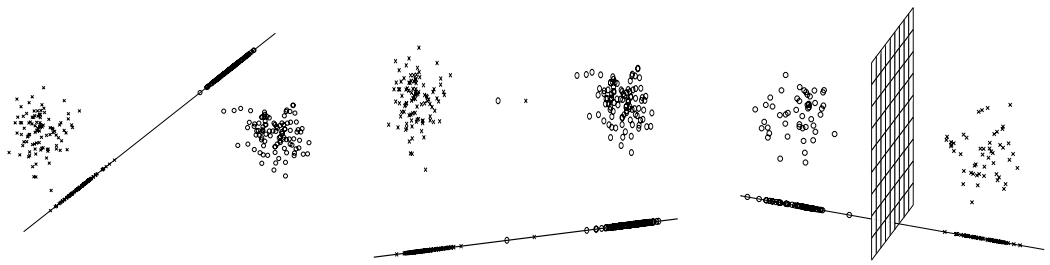

Figure 2.1: Fisher's Discriminant. On the left (a) is an example of Fisher's criterion. The two classes are represented by x's and o's, and the span of $\mathbf{a}$ is shown as a line. In this example $J(\mathbf{a}) = 77$. In the middle (b) is an example where $J(\mathbf{a}) = 64.5$ is a large number despite the fact that the two classes are not linearly separable. On the right (c) is the full version of Fisher's discriminant. In this case the line represents the span of $\mathbf{a}^*$ in Equation (2.1) and the separating plane has Equation (2.2).

2.1 Fisher's Linear Discriminant

The basis of our test of classification ability is Fisher's linear discriminant. Fisher's linear discriminant (here we follow [13]) locates a separating hyperplane by maximizing *Fisher's criterion* function $J : \mathbb{R}^n \to \mathbb{R}$ defined by

$$J(\mathbf{a}) = \frac{(m_1 - m_2)^2}{\sigma_1^2 + \sigma_2^2},$$

where m_1 and m_2 are the means of the (orthogonal) projections of our two classes X and Y, respectively, onto $\mathbf{a}$. Similarly, σ_1^2 and σ_2^2 are the variances of the projections of X and Y onto $\mathbf{a}$. An illustration of Fisher's criterion is provided in Figure 2.1(a).

Fisher's criterion function provides a useful measure of the linear separability of X and Y. By its definition, we see that larger values of J indicate better separation of X and Y. In particular, a value of one indicates a separation of the projected means by one standard deviation of the projected values of X plus one standard deviation of the projected values of Y. Similarly, if $J = 4$ the means are separated by approximately two standard deviations, et cetera. Of

course, high values of Fisher's criterion do not guarantee linear separability. This is demonstrated in Figure 2.1(b).

By maximizing Fisher's criterion we can locate a good separating hyperplane for X and Y. Maximizing J is accomplished by first rewriting Fisher's criterion as

$$J(\mathbf{a}) = \frac{\mathbf{a}^T S_b \mathbf{a}}{\mathbf{a}^T S_c \mathbf{a}},$$

where

$$\begin{aligned} S_b &= (\mathbf{m}_1 - \mathbf{m}_2)(\mathbf{m}_1 - \mathbf{m}_2)^T \\ S_c &= S_1 + S_2 \\ S_1 &= \tfrac{1}{N-1} \textstyle\sum_{i=1}^{N} (\mathbf{x}_i - \mathbf{m}_1)(\mathbf{x}_i - \mathbf{m}_1)^T \\ S_2 &= \tfrac{1}{M-1} \textstyle\sum_{j=1}^{M} (\mathbf{y}_j - \mathbf{m}_2)(\mathbf{y}_j - \mathbf{m}_2)^T \end{aligned}$$

with

$$\begin{aligned} X &= \{\mathbf{x}_i\}_{i=1}^{N} \\ Y &= \{\mathbf{y}_j\}_{j=1}^{M} \\ \mathbf{m}_1 &= \tfrac{1}{N} \textstyle\sum_{i=1}^{N} \mathbf{x}_i \\ \mathbf{m}_2 &= \tfrac{1}{M} \textstyle\sum_{j=1}^{M} \mathbf{y}_j. \end{aligned}$$

Next we calculate

$$\nabla J(\mathbf{a}) = \frac{2}{(\mathbf{a}^T S_c \mathbf{a})^2}((\mathbf{a}^T S_c \mathbf{a})(S_b \mathbf{a}) - (\mathbf{a}^T S_b \mathbf{a})(S_c \mathbf{a})).$$

Finally, we observe that $\nabla J(\mathbf{a})$ is zero when $S_c \mathbf{a} = (\mathbf{m}_1 - \mathbf{m}_2)$ so that $J(\mathbf{a}^*)$ is a maximum (according to [13]) when

$$\mathbf{a}^* = S_c^{-1}(\mathbf{m}_1 - \mathbf{m}_2). \tag{2.1}$$

The resulting separating hyperplane

$$H(\mathbf{x}) = (\mathbf{x}, \mathbf{a}^*) - \frac{\sigma_2 m_1 + \sigma_1 m_2}{\sigma_1 + \sigma_2}\|\mathbf{a}^*\| = 0 \tag{2.2}$$

yields a classifier known as *Fisher's linear discriminant*. (Here $(\mathbf{x}, \mathbf{a}^*)$ denotes the inner product of $\mathbf{x}$ with $\mathbf{a}^*$ in $\mathbb{R}^n$.) Fisher's discriminant is illustrated in Figure 2.1(c). Using Fisher's discriminant, a point $\mathbf{z} \in \mathbb{R}^n$ is classified as a member of X if $H(\mathbf{z})$ and $H(\mathbf{m}_1)$ have the same sign, and a member of Y otherwise.

2.2 The Veronese Map

To implement a nonlinear version of Fisher's discriminant we use nonlinear preprocessing as in Section 1.2.2. For the present we consider only the Veronese map for preprocessing. The map Φ used in Equation (1.2) to solve the exclusive-or problem is a particular example of a Veronese map. In general, the *Veronese map* $v_d : \mathbb{R}^n \to \mathbb{R}^N$, where $N = \binom{n+d}{d} - 1$, is given by

$$v_d(x_1, x_2, \ldots, x_n) = \begin{pmatrix} \text{nonconstant monomials} \\ \text{in } x_1, x_2, \ldots, x_n \text{ of degree} \leq d \end{pmatrix}.$$

It is a map borrowed from algebraic geometry and can be found in [45]. Φ in (1.2) is a homogeneous scaled version of this map. Other examples are

$$v_3(x, y) = (x^3, x^2y, xy^2, y^3, x^2, xy, y^2, x, y)$$

and

$$v_2(x, y, z) = (x^2, xy, xz, y^2, yz, z^2, x, y, z).$$

The essential property of the Veronese map is that polynomial hypersurfaces of degree at most d in $\mathbb{R}^n$ are hyperplanes in $\mathbb{R}^N$. Figure 2.2 illustrates, for example, that $xy = 1$ is a hyperbola in $\mathbb{R}^2$ but a plane in $\mathbb{R}^3$, again using Φ in (1.2). More explicitly, if Φ in (1.2) is described by $(u, v, w) = (x^2, \sqrt{2}xy, y^2)$ then the hyperbola $xy = 1$ in $\mathbb{R}^2$ yields the plane $v = \sqrt{2}$ in $\mathbb{R}^3$. This occurs in general and is precisely the property which makes the Veronese map useful in conjunction with Fisher's linear discriminant.

2.3 Symmetric Veronese Map

A variation of the Veronese map is the symmetric Veronese map. We will use this map in our application to the Materials Design problem in Section 2.5. The symmetric Veronese map may be implemented using the elementary symmetric

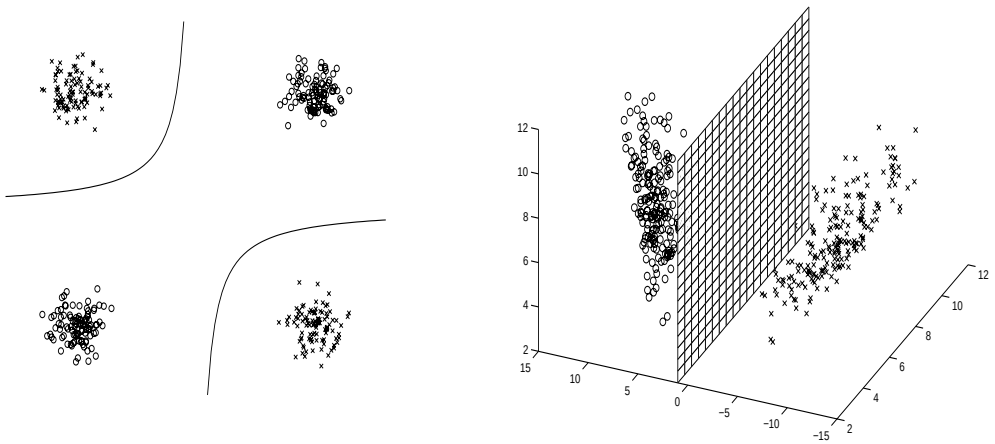

Figure 2.2: The Exclusive-Or Problem Revisited. On the left (a), the hyperbola $xy = 1$ separates X and Y. On the right (b), the same equation yields a plane which separates $\Phi(X)$ and $\Phi(Y)$.

polynomials. That is, we know that any symmetric polynomial in the variables $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ can be written as a polynomial in the elementary symmetric polynomials (see [33])

$$s_1(\mathbf{x}) = x_1 + x_2 + \cdots + x_n$$

$$s_2(\mathbf{x}) = x_1x_2 + x_1x_3 + \cdots + x_1x_n + x_2x_3 + \cdots + x_{n-1}x_n$$

$$\vdots$$

$$s_n(\mathbf{x}) = x_1x_2 \cdots x_n.$$

Thus, to obtain a symmetric Veronese map, we simply use the elementary symmetric polynomials to "symmetrize" our data before applying the regular Veronese map. In other words, the *symmetric Veronese map* is the composition

$$\begin{gathered} \mathbb{R}^n \xrightarrow{\mathbf{s}} \mathbb{R}^n \xrightarrow{v_d} \mathbb{R}^N \\ \mathbf{x} \mapsto \mathbf{s}(\mathbf{x}) = (s_1(\mathbf{x}), \ldots, s_n(\mathbf{x})) \mapsto v_d(\mathbf{s}(\mathbf{x})). \end{gathered}$$

2.4 Exhaustive Feature Search

Using Fisher's linear discriminant and the (symmetric) Veronese map we are in position to perform a search for the best feature combination. Our algorithm is as follows:

(1) For a given choice of feature(s) and a given degree d for the (symmetric) Veronese map we first preprocess our two class data X and Y using $v_d(\circ\mathbf{s}) : \mathbb{R}^n \to \mathbb{R}^N$. We then calculate

- The vector $\mathbf{a}^* \in \mathbb{R}^N$ in Equation (2.1) that maximizes Fisher's criterion $J : \mathbb{R}^N \to \mathbb{R}$ and the actual maximum $J(\mathbf{a}^*)$.
- The percentage of points in $v_d(X \cup Y)$ correctly classified using Fisher's linear discriminant in $\mathbb{R}^N$.

(2) We record the results of (1) for every set of $1, 2, 3, \ldots$ features and every degree $d = 1, 2, 3, \ldots$ of the (symmetric) Veronese map. This is practical because the optimization of J is computationally inexpensive.

(3) We rank the effectiveness of the feature combinations using the optimal values of J and the percentages recorded in step (2). In general a feature whose optimal J values and percentages are higher than another feature is a better discriminator.

To see why this algorithm provides a useful ranking of feature combinations, consider two fixed features F_1 and F_2. In this case our data X and Y lies in the plane $\mathbb{R}^2$. What does our algorithm measure? For each degree $d = 1, 2, 3, \ldots$ of the (symmetric) Veronese map we preprocess X and Y then use Fisher's criterion and Fisher's linear discriminant to measure the linear separability of our data after preprocessing. Thus we are measuring the linear separability of our data ($d = 1$), the quadratic separability of our data ($d = 2$), the cubic separability of our data ($d = 3$), et cetera. By using Fisher's linear discriminant in combination with the (symmetric) Veronese map, we are measuring the linear and increasingly nonlinear separability of our data for our given features F_1 and F_2.

We remark that this algorithm might be applied directly to a classification problem. Specifically, Fisher's discriminant coupled with the Veronese map yields a classifier. This classifier, however, while less expensive to train than a Support Vector Machine (see Chapter 4), may not be as good as a SVM. Thus we use our algorithm only for feature selection, not for classification.

2.5 Application to Materials Design

Here we present an application of our exhaustive feature search to a problem in Materials Design. Materials Design is the search for new materials (e.g. metals and ceramics) which have special properties (e.g. super-conductivity) or work better than older materials (e.g. metal alloys). The problem considered here comes from the Wright Patterson Air Force Base in Dayton, Ohio and concerns classifying chemical compounds. Specifically, a large number of chemical element combinations have been collected, see [52] and [18], which through different processes either form or fail to form compounds. We want to use these examples to predict when other chemical element combinations will form compounds.

In this section we perform feature selection for classifying three-element combinations into forming and non-forming categories. Here features are chemical properties (e.g. electronegativity) and our data consists of 4031 forming compounds and 2327 non-forming compounds (non-forming ternary from non-forming binary). Further, our data is symmetric in the order of the compounds, i.e., if elements (A, B, C) form a compound then elements (B, A, C) and (C, B, A), etc. should form the same compound. To exploit this observation we employ the symmetric Veronese map in Section 2.3 in our feature search. We perform a single feature search and a feature pair search with symmetric Veronese degrees $d = 1, \ldots, 10$ and $d = 1, \ldots, 5$ respectively. (Computer memory constraints kept us from considering higher degrees.)

The product of our test of separability for each feature against the symmetric Veronese maps of degree one through ten is Figure 2.3. Here we have identified the Pettifor Mendeleev number (feature 6) as the single best feature (in agreement with [53]), followed by the Alfred-Rochow and Pauling electronegativities (features 17 and 24) and the p-shell (feature 45). For the Mendeleev number we also include plots (Figure 2.4) of the optimal J value and the percentage of points correctly classified using Fisher's linear discriminant, both versus the symmetric Veronese degree.

The two feature test yields another intensity plot (Figure 2.5). The prominent black group occurs when one feature is the Mendeleev number. Within this group the feature which best complements the Mendeleev number is the valence electron number. For this pair we again include plots (Figure 2.6) of the optimal J and percentage correctly classified versus the symmetric Veronese degree.

2.6 Comparison to Villars' Work

Finally, we present Pierre Villars' [53] method of feature selection for the materials problem and show how our method generalizes his approach. Pierre Villars' method produces percentage separation plots for various arithmetic combinations (addition, subtraction, multiplication, division and maximum) of the feature data. Comparing the resulting plots allows selection of the best features. As an example, we include Villars' best result — the percentage separation plot for the Mendeleev number (Figure 2.7). To produce this plot we first calculated for $M_A < M_B < M_C$ the triples

$$(M_A + M_B + M_C, \frac{M_A}{M_B} + \frac{M_B}{M_C} + \frac{M_A}{M_C}, \max(M_A, M_B, M_C))$$

in both the forming and non-forming classes. We next calculated the distance between each pair of said triples and considered for each point its fifty nearest

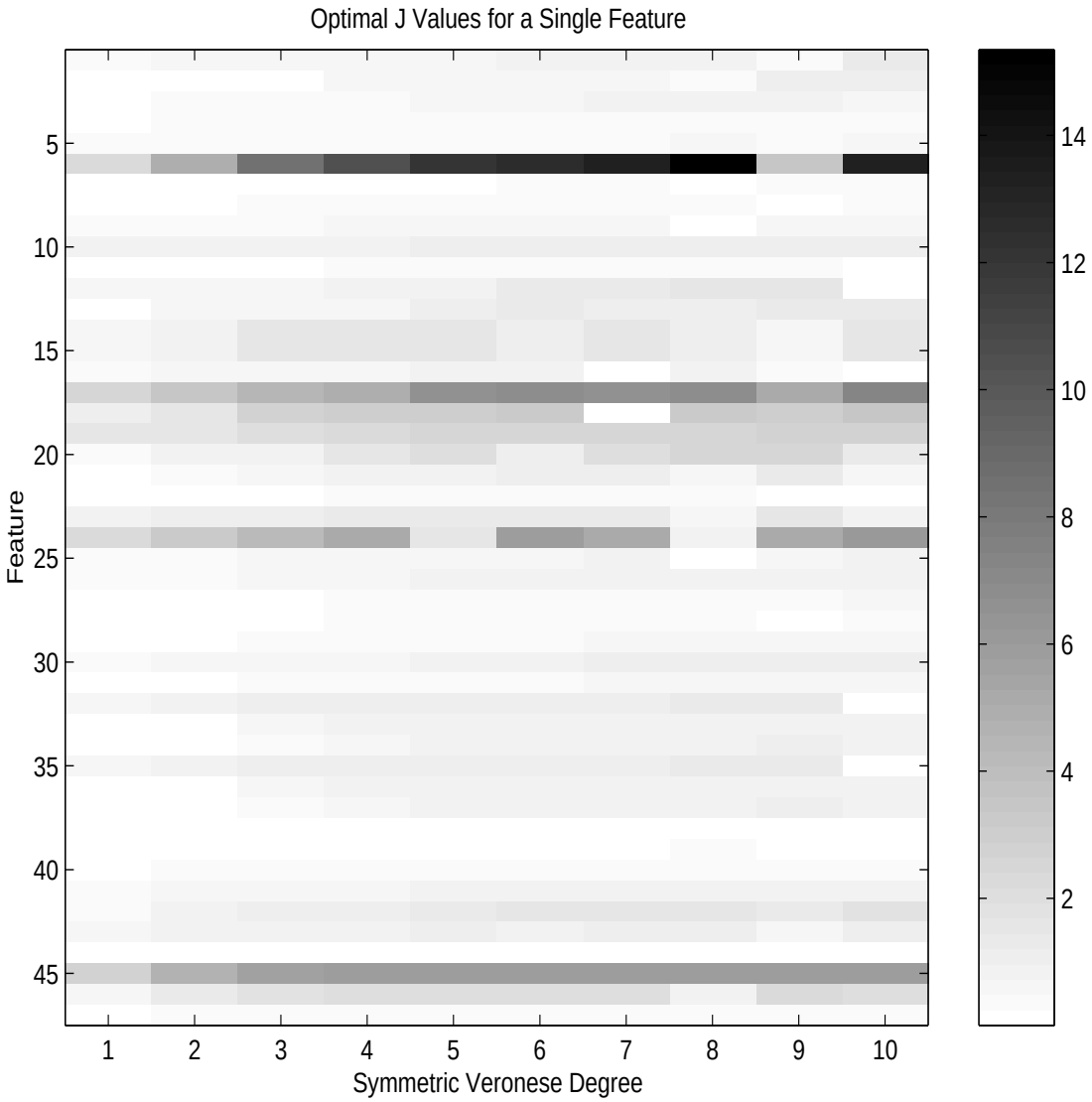

Figure 2.3: One Feature Test. This is an intensity plot of the optimal J value (whose range is given by the scale bar on the right) versus the symmetric Veronese degree for each feature. A dark horizontal bar indicates high optimal J values and thus identifies a good feature.

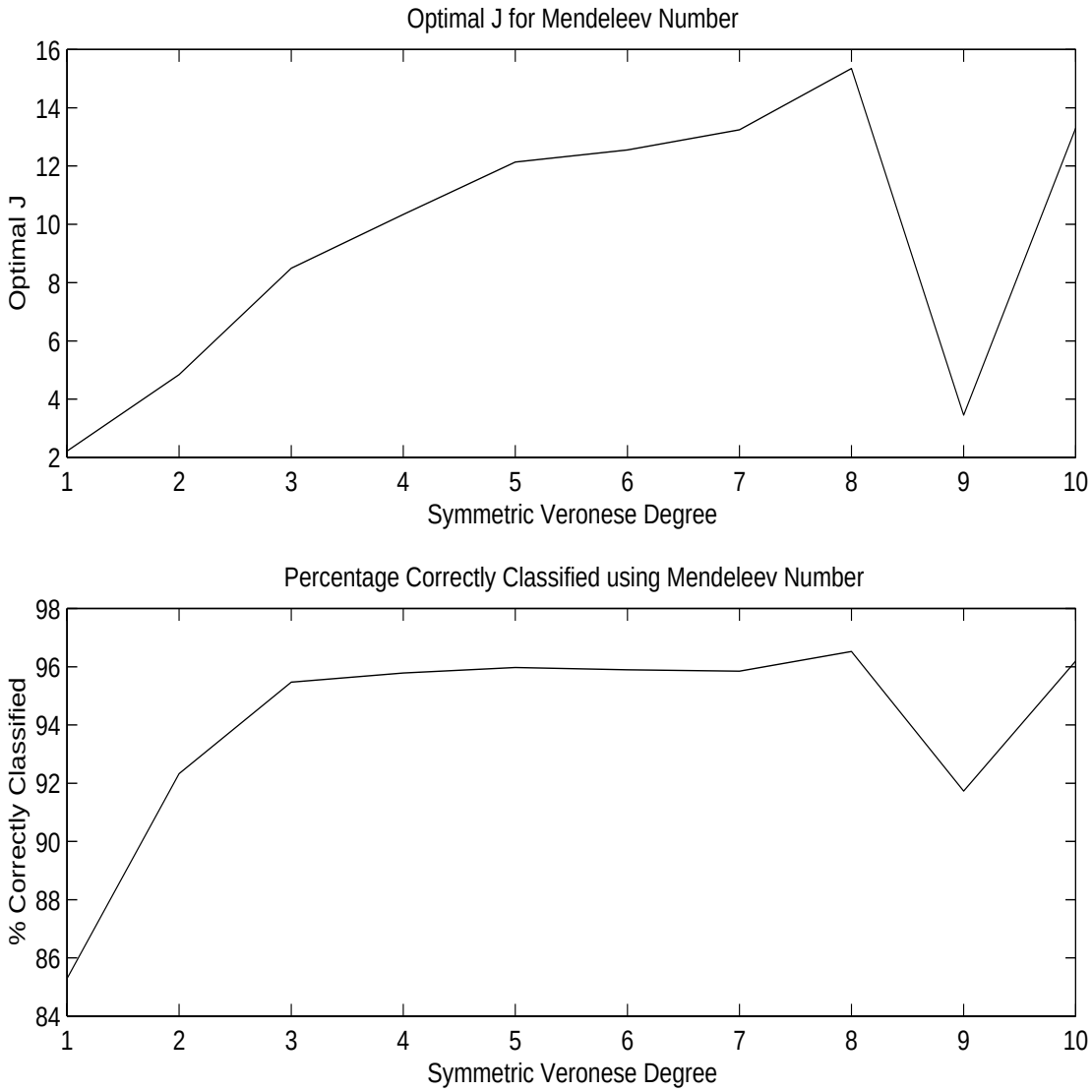

Figure 2.4: Mendeleev Number is the Best Feature. On the top (a) we plot the optimal J value and on the bottom (b) the percentage correctly classified both versus the symmetric Veronese degree for the Mendeleev number.

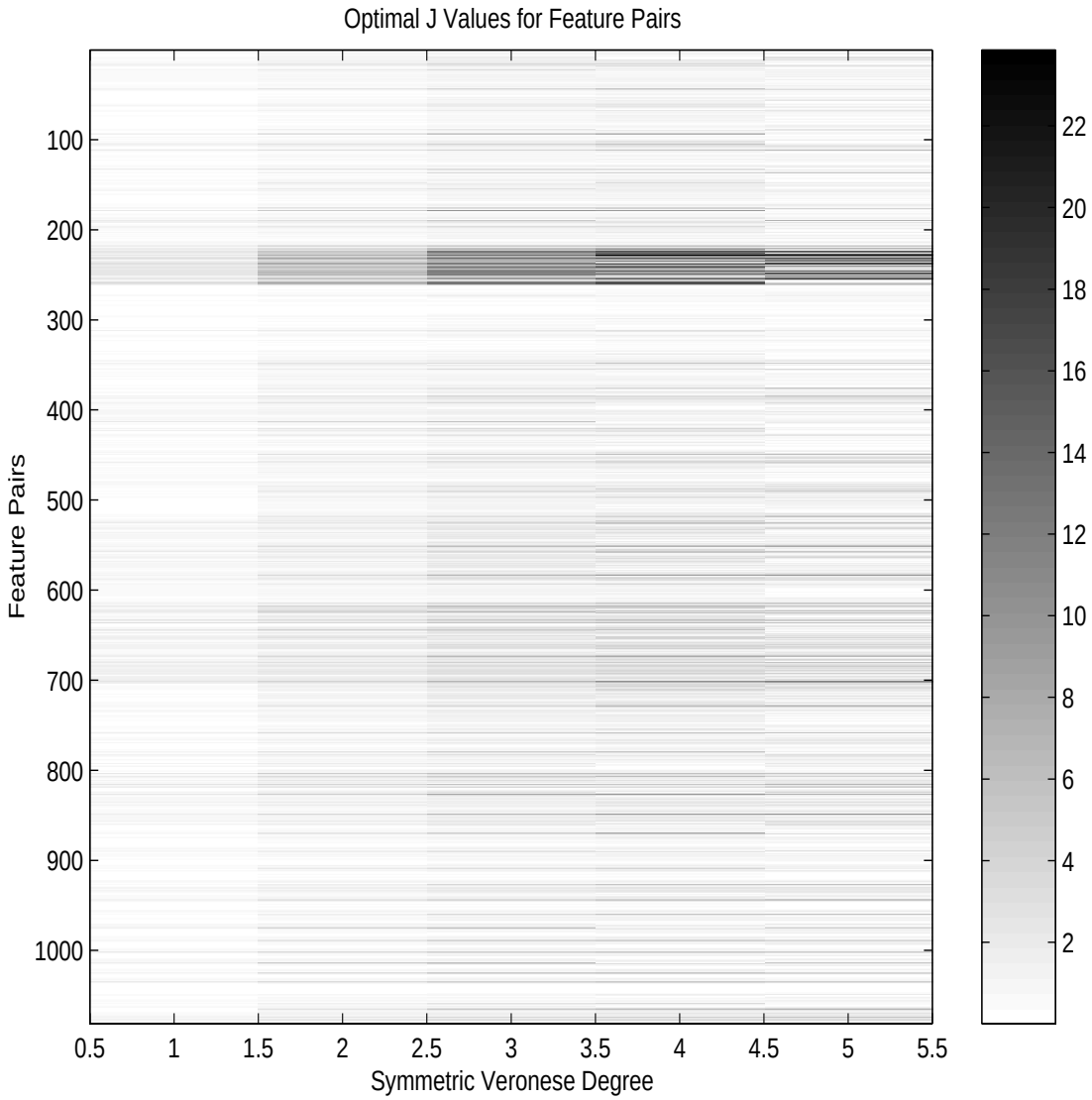

Figure 2.5: Two Feature Test. This is an intensity plot of optimal J versus the symmetric Veronese degree for every pair of features.

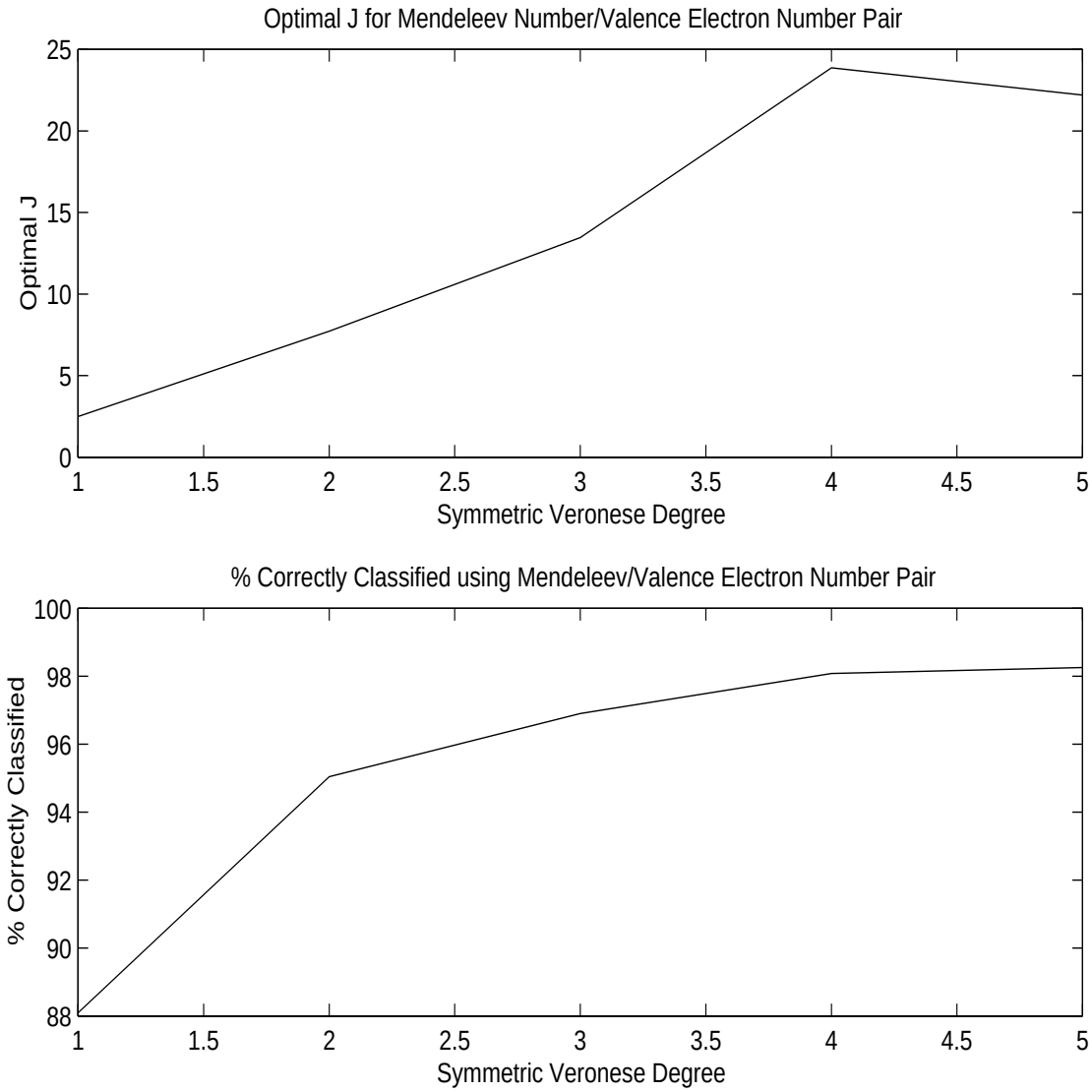

Figure 2.6: Mendeleev/valence electron numbers are the Best Feature Pair. On the top (a) we plot the optimal J value and on the bottom (b) the percentage correctly classified both versus the symmetric Veronese degree for the Mendeleev/valence electron number pair.

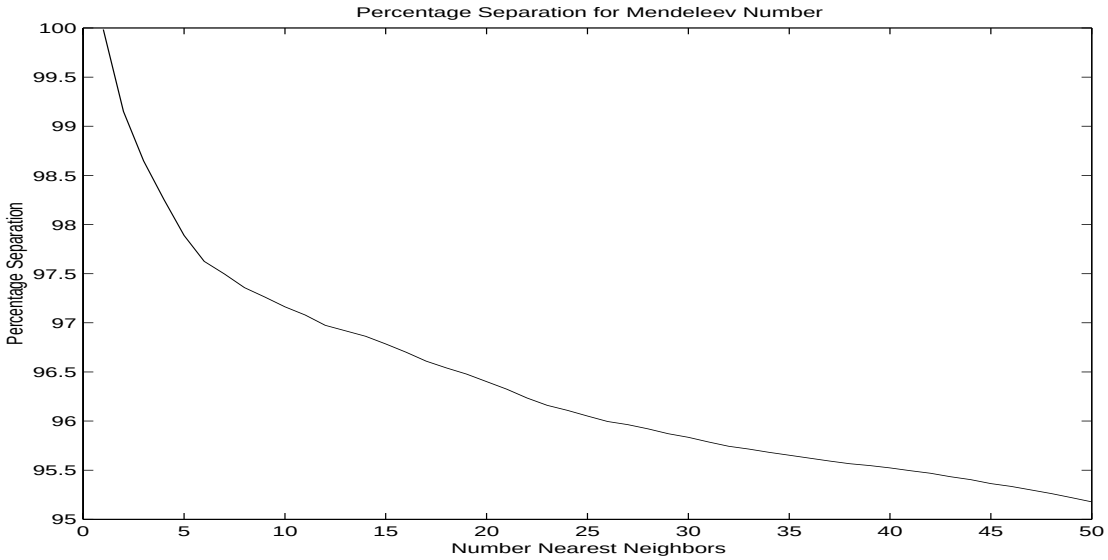

Figure 2.7: Percentage Separation Plot. Pierre Villars' percentage separation plot for the Mendeleev number.

neighbors. The average over every point (in both classes) of the percent of neighbors of the same type was then plotted versus the number (up to fifty) of nearest neighbors considered.

It is no coincidence that our results agree so well with Villars' work. In the Mendeleev number example, the percentage separation plot provides a rough measure of linear separability in a space with coordinates given by the aforementioned triples. Thus a high percentage separation indicates the existence of a separating plane in this space with equation

$$\alpha(M_A + M_B + M_C) + \beta(\frac{M_A}{M_B} + \frac{M_B}{M_C} + \frac{M_A}{M_C}) + \gamma\max(M_A, M_B, M_C) = \delta.$$

Neglecting the $\gamma\max(M_A, M_B, M_C)$ term, this is a cubic polynomial in the variables M_A, M_B, and M_C — a hyperplane in the degree three Veronese space $\mathbb{R}^{19}$.

Furthermore, with the exception of the max function, all of the coordinate systems used by Villars result in hyperplanes in different Veronese spaces. By testing for linear separability in these (and other) Veronese spaces, we thus extend Villars' method to a much more general situation.

2.7 Conclusions

With respect to the Materials Design problem, our method has successfully explained and generalized Pierre Villars' work [53] and has confirmed his result in identifying the Mendeleev number as the best single feature. In addition we have discovered a new result, namely that the p-shell is a good single feature, and we have extended feature selection information for the materials problem into the realm of two features, where we have identified the Mendeleev number in combination with the valence electron number as the best feature pair. Simultaneously, we have eliminated many features and feature pairs as inferior.

More generally, our method is a novel combination of Fisher's discriminant and the Veronese map applied to feature selection. In the parlance of [12], we have invented a new evaluation function. An evaluation function is a function which evaluates the effectiveness of a given feature or combination of features. Fisher's criterion is often used as an evaluation function (as well as the Mahalanobis distance and the Bhattacharya distance — see [13] and [16]), but has not been combined with the Veronese map for this purpose.

Combined with an exhaustive search (certainly not the most efficient search), our method is a filter method (see [12] and [21]). A filter method is a method which is independent of the final classifier. It is interesting to note, however, that our method results in a nonlinear version of Fisher's discriminant (see also [31]), which is a classifier in itself. In addition, this version of Fisher's discriminant occupies the same space as a Support Vector Machine with the Veronese kernel (see Sections 1.2.2 and 3.2). Thus our method may not be entirely (a priori) independent of the final classifier, especially if the final classifier is a Support Vector Machine.

Chapter 3

KERNEL SELECTION FOR SUPPORT VECTOR MACHINES

In this chapter we consider the problem of kernel selection for Support Vector Machines. (Refer to Section 1.2.) Until recently, kernel selection for Support Vector Machines has been performed by simply training a variety of Support Vector Machines and using the best of those trained. This method, however, is unwieldy and slow. Thus efforts have been made towards automatic kernel selection. Such efforts appear in [11], [9], and [42]. In all of these efforts, a "Statistical Risk Minimization" bound on the generalization error for Support Vector Machines from Vapnik-Chervonenkis (VC) theory [51] is employed. This bound depends on the radius of the smallest ball containing the preprocessed (via kernel) training data. In [11], this bound is computed to select a one parameter kernel while training a Support Vector Machine using the Kernel Adatron Algorithm [14]. In [9], an effort is made to rescale the preprocessed training data to fill the Statistical Risk Minimization ball, thus resulting in a more accurate bound. In [42], the distribution of the preprocessed training data within the SRM ball is analyzed using a technique called kernel PCA [43].

In this chapter we adopt a more empirical approach to kernel selection. Specifically, we generalize the approach in Chapter 2 for feature selection using kernels. Our method again uses Fisher's discriminant, but this time in combination with

a technique which we call kernel Gram-Schmidt. In Section 3.1 we introduce kernel Gram-Schmidt; in Sections 3.2 and 3.3 we generalize the method of feature selection from Chapter 2 using the Veronese kernel; in Section 3.4 we provide symmetric extensions for the RBF and neural network kernels; in Section 3.5 we present our method of kernel selection for Support Vector Machines; in Section 3.6 we apply our method to the Materials Design problem; in Section 3.7 we compare our method with the methods in [11], [9], and [42] using the United States Postal Service data set of handwritten digits; and in Section 3.8 we offer some concluding remarks. For an alternate presentation of the material in this chapter see [28].

3.1 Kernel Gram-Schmidt

Our primary vehicle for extending the ideas in Chapter 2 to the problem of kernel selection for Support Vector Machines is a kernel version of the classical Gram-Schmidt orthonormalization procedure.

Suppose we have data $Z = \{\mathbf{z}_i\}_{i=1}^m \subset \mathbb{R}^n$ (say $Z = X \cup Y$). Denote $\Phi(\mathbf{z}_i)$ by $\widetilde{\mathbf{z}_i}$ for $i = 1, \ldots, m$, where $\Phi : \mathbb{R}^n \to F$ is our nonlinear preprocessing map, and F is a Hilbert space with inner product $(\bullet, \bullet)$. For initial notational ease, we assume that $\{\widetilde{\mathbf{z}_i}\}_{i=1}^m$ is a linearly independent set of vectors in F. We want to produce a matrix

$$B = \begin{pmatrix} (\widetilde{\mathbf{z}_1}, \mathbf{u}_1) & \cdots & (\widetilde{\mathbf{z}_m}, \mathbf{u}_1) \\ \vdots & \ddots & \vdots \\ (\widetilde{\mathbf{z}_1}, \mathbf{u}_m) & \cdots & (\widetilde{\mathbf{z}_m}, \mathbf{u}_m) \end{pmatrix},$$

where $\mathbf{u}_1, \ldots, \mathbf{u}_m$ form an orthonormal basis for the subspace in F spanned by $\widetilde{\mathbf{z}_1}, \ldots, \widetilde{\mathbf{z}_m}$.

Using the classical Gram-Schmidt orthonormalization procedure (see [49]) we form

$$\begin{aligned}
\mathbf{u}_1 &= \tfrac{\widetilde{\mathbf{z}_1}}{\|\widetilde{\mathbf{z}_1}\|} \\
\mathbf{p}_1 &= (\widetilde{\mathbf{z}_2}, \mathbf{u}_1)\mathbf{u}_1 \\
\mathbf{u}_2 &= \tfrac{\widetilde{\mathbf{z}_2}-\mathbf{p}_1}{\|\widetilde{\mathbf{z}_2}-\mathbf{p}_1\|} \\
\mathbf{p}_2 &= (\widetilde{\mathbf{z}_3}, \mathbf{u}_2)\mathbf{u}_2 + (\widetilde{\mathbf{z}_3}, \mathbf{u}_1)\mathbf{u}_1 \\
\mathbf{u}_3 &= \tfrac{\widetilde{\mathbf{z}_3}-\mathbf{p}_2}{\|\widetilde{\mathbf{z}_3}-\mathbf{p}_2\|} \\
&\vdots \\
\mathbf{p}_{m-1} &= (\widetilde{\mathbf{z}_m}, \mathbf{u}_{m-1})\mathbf{u}_{m-1} + \cdots + (\widetilde{\mathbf{z}_m}, \mathbf{u}_1)\mathbf{u}_1 \\
\mathbf{u}_m &= \tfrac{\widetilde{\mathbf{z}_m}-\mathbf{p}_{m-1}}{\|\widetilde{\mathbf{z}_m}-\mathbf{p}_{m-1}\|}.
\end{aligned}$$

We then calculate

$$\begin{aligned}
(\widetilde{\mathbf{z}_1}, \mathbf{u}_1) &= \|\widetilde{\mathbf{z}_1}\| = \tfrac{(\widetilde{\mathbf{z}_1}, \widetilde{\mathbf{z}_1})}{(\widetilde{\mathbf{z}_1}, \mathbf{u}_1)} \\
(\widetilde{\mathbf{z}_2}, \mathbf{u}_1) &= \tfrac{(\widetilde{\mathbf{z}_2}, \widetilde{\mathbf{z}_1})}{\|\widetilde{\mathbf{z}_1}\|} = \tfrac{(\widetilde{\mathbf{z}_2}, \widetilde{\mathbf{z}_1})}{(\widetilde{\mathbf{z}_1}, \mathbf{u}_1)} \\
(\widetilde{\mathbf{z}_2}, \mathbf{u}_2)^2 &= \|\widetilde{\mathbf{z}_2} - \mathbf{p}_1\|^2 = \|\widetilde{\mathbf{z}_2}\|^2 - \|\mathbf{p}_1\|^2 \\
(\widetilde{\mathbf{z}_2}, \mathbf{u}_2) &= \tfrac{1}{(\widetilde{\mathbf{z}_2}, \mathbf{u}_2)} \left[(\widetilde{\mathbf{z}_2}, \widetilde{\mathbf{z}_2}) - (\widetilde{\mathbf{z}_2}, \mathbf{u}_1)^2\right] \\
(\widetilde{\mathbf{z}_3}, \mathbf{u}_1) &= \tfrac{(\widetilde{\mathbf{z}_3}, \widetilde{\mathbf{z}_1})}{(\widetilde{\mathbf{z}_1}, \mathbf{u}_1)} \\
(\widetilde{\mathbf{z}_3}, \mathbf{u}_2) &= \tfrac{1}{(\widetilde{\mathbf{z}_2}, \mathbf{u}_2)} \left[(\widetilde{\mathbf{z}_3}, \widetilde{\mathbf{z}_2}) - (\widetilde{\mathbf{z}_2}, \mathbf{u}_1)(\widetilde{\mathbf{z}_3}, \mathbf{u}_1)\right] \\
(\widetilde{\mathbf{z}_3}, \mathbf{u}_3) &= \tfrac{1}{(\widetilde{\mathbf{z}_3}, \mathbf{u}_3)} \left[(\widetilde{\mathbf{z}_3}, \widetilde{\mathbf{z}_3}) - ((\widetilde{\mathbf{z}_3}, \mathbf{u}_2)^2 + (\widetilde{\mathbf{z}_3}, \mathbf{u}_1)^2)\right],
\end{aligned}$$

and in general

$$(\widetilde{\mathbf{z}_i}, \mathbf{u}_j) = \frac{1}{(\widetilde{\mathbf{z}_j}, \mathbf{u}_j)} \left[(\widetilde{\mathbf{z}_i}, \widetilde{\mathbf{z}_j}) - \begin{pmatrix} (\widetilde{\mathbf{z}_j}, \mathbf{u}_{j-1})(\widetilde{\mathbf{z}_i}, \mathbf{u}_{j-1}) + (\widetilde{\mathbf{z}_j}, \mathbf{u}_{j-2})(\widetilde{\mathbf{z}_i}, \mathbf{u}_{j-2}) \\ + \cdots + (\widetilde{\mathbf{z}_j}, \mathbf{u}_1)(\widetilde{\mathbf{z}_i}, \mathbf{u}_1) \end{pmatrix}\right].$$

Finally, we allow linearly dependent $\{\widetilde{\mathbf{z}_i}\}$ by successively ignoring the cases $\widetilde{\mathbf{z}_{i_0}} = \mathbf{p}_{(i_0-1)}, \widetilde{\mathbf{z}_{i_1}} = \mathbf{p}_{(i_1-2)}$, et cetera. Thus we get a recursive algorithm for computing the matrix

$$B = \begin{pmatrix} (\widetilde{\mathbf{z}_1}, \mathbf{u}_1) & \cdots & (\widetilde{\mathbf{z}_m}, \mathbf{u}_1) \\ \vdots & \ddots & \vdots \\ (\widetilde{\mathbf{z}_1}, \mathbf{u}_q) & \cdots & (\widetilde{\mathbf{z}_m}, \mathbf{u}_q) \end{pmatrix},$$

where $\mathbf{u}_1, \ldots, \mathbf{u}_q$ form an orthonormal basis for the subspace in F spanned by $\widetilde{\mathbf{z}_1}, \ldots, \widetilde{\mathbf{z}_m}$.

This method, which we call *kernel Gram-Schmidt*, has several important properties. First and foremost, it is expressed entirely in terms of inner products in F

so can be computed using kernels. Second, it entirely avoids actually computing the orthonormal set $\{\mathbf{u}_1, \ldots, \mathbf{u}_q\}$. This is necessary since F may in fact be infinite dimensional. Third and last, but very important, the generated matrix B represents the data $\{\widetilde{\mathbf{z}}_i\}$ in F in exactly the same manner that it would be represented in $\mathbb{R}^q$. Thus we can apply any standard algorithm from linear algebra to B.

It is also worth noting that kernel Gram-Schmidt represents a significant philosophical deviation from the typical kernel based methods, including Support Vector Machines, kernel PCA [43], and kernel Fisher's Discriminant [31]. Kernel Gram-Schmidt makes an effort to use the actual dimension of the preprocessed data, i.e. the orthonormal basis $\mathbf{u}_1, \ldots, \mathbf{u}_q$, while Support Vector Machines, kernel PCA, and kernel Fisher's Discriminant all perform calculations using the entire data set as a spanning set (see Section 1.2.2). For this reason kernel Gram-Schmidt can be used on much larger data sets than kernel PCA and kernel Fisher's Discriminant. (Special algorithms exist for using Support Vector Machines for large data sets — see Chapter 4.)

Finally, we should remark that the classical Gram-Schmidt method is numerically unstable [49]. Thus it may be that kernel Gram-Schmidt is also numerically unstable. On the other hand, since it is the computation of the orthonormal basis $\mathbf{u}_1, \ldots, \mathbf{u}_q$ that is unstable when using classical Gram-Schmidt, and since kernel Gram-Schmidt does not actually compute said basis, it may be that kernel Gram-Schmidt is in fact numerically stable. Certainly the situation requires further study. See Chapter 5 for other topics which also require further study.

3.2 The Veronese Kernel

Our first application of kernel Gram-Schmidt is to provide a direct generalization of the method of feature selection in Chapter 2. To this end, we observe that

the Veronese map, as defined in Section 2.2, is not known to have an associated kernel, but that

$$\kappa(\mathbf{x}, \mathbf{y}) = ((\mathbf{x}, \mathbf{y}) + 1)^d$$

is a kernel (see [7]) for a scaled version of the map. The scaled (or weighted) Veronese map is given by

$$w_d(\mathbf{x}) = (\text{ scaled monomials } a_{\mathbf{j}}\mathbf{x}^{\mathbf{j}}),$$

where the sequences $\mathbf{j} = (j_1, j_2, \ldots, j_n) \in \mathbb{Z}^n$ range over $\{\mathbf{j} : 0 \leq \sum \mathbf{j} = j_1 + j_2 + \cdots + j_n \leq d\}$, the $\mathbf{x}^{\mathbf{j}}$ are monomials $x_1^{j_1} x_2^{j_2} \cdots x_n^{j_n}$, and $a_{\mathbf{j}}^2 = \binom{d}{\sum \mathbf{j}} \binom{\sum \mathbf{j}}{\mathbf{j}} = \binom{d}{\sum \mathbf{j}} \frac{(\sum \mathbf{j})!}{j_1! j_2! \cdots j_n!}$. Examples of the scaled Veronese map are

$$w_3(x, y) = (x^3, \sqrt{3}x^2 y, \sqrt{3}xy^2, x^3, \sqrt{3}x^2, \sqrt{6}xy, \sqrt{3}y^2, \sqrt{3}x, \sqrt{3}y, 1),$$

and

$$w_2(x, y, z) = (x^2, \sqrt{2}xy, \sqrt{2}xz, y^2, \sqrt{2}yz, z^2, \sqrt{2}x, \sqrt{2}y, \sqrt{2}z, 1).$$

We also observe that we may implement a symmetric Veronese kernel using the symmetric polynomials as in Section 2.3.

3.3 Feature Search Revisited

Using the (symmetric) Veronese kernel, we can generalize our method for feature selection as follows:

(F1) For a given choice of features and a given degree d for the (symmetric) Veronese kernel we apply kernel Gram-Schmidt to our data X and Y to obtain a $q \times m$ matrix B. We then calculate using B

- The vector $\mathbf{a}^* \in \mathbb{R}^q$ that maximizes Fisher's criterion $J : \mathbb{R}^q \to \mathbb{R}$ and the actual maximum $J(\mathbf{a}^*)$. See Equation (2.1) in Section 2.1.

- The percentage p of points in B correctly classified using Fisher's discriminant with separating hyperplane determined by $\mathbf{a}^*$. See Equation (2.2) in Section 2.1.
- The percentage t of points in test sets T_X and T_Y (see below) correctly classified. (T_X and T_Y are first preprocessed using kernel Gram-Schmidt.)

(F2) We record the results of (F1) for different feature combinations and every degree $d = 1, 2, \ldots, r$ of the (symmetric) Veronese kernel. Generally we take r between 4 and 10.

(F3) We rank the effectiveness of the feature combinations using the optimal values of J and the percentages recorded in step (F2). More precisely, we order the features by averaging the values $J(\mathbf{a}^*)$, p, and t calculated in step (F1) over the Veronese degrees $1, 2, \ldots, r$ in step (F2). We denote these averages by $\overline{J(\mathbf{a}^*)}_r$, $\overline{p}$, and $\overline{t}$ and we use $\overline{J(\mathbf{a}^*)}_r$ to rank the features.

In this version of our method, we have included test sets and we have specified our system for ranking features using average values of $J(\mathbf{a}^*)$. The mention of test sets requires explanation. In practice, a given classification problem consists of not only the sets X and Y, but also includes two other sets T_X and T_Y, all finite and pairwise disjoint. We use X and Y as "training" data and T_X and T_Y as "test" data. That is, we use X and Y to determine our classifiers while we use T_X and T_Y to test the performance of our classifiers. Using T_X and T_Y we can calculate in addition to p above the percentage t of test data correctly classified using Fisher's discriminant. The values of t then give us additional information to use in our comparison of features.

In addition, we remark that other kernels could also be used for feature selection, although they may result in different feature rankings. In addition to being

more easily interpreted (see Section 2.4) than the RBF and neural network kernels (see Section 1.2.2), we chose the (symmetric) Veronese kernel because it generally yields the fastest computations.

Finally, we remark that our use of the Veronese kernel to generalize the feature search in Section 2.4 allows us to use higher dimensional data and larger degrees of the Veronese map. Specifically, the Veronese kernel reduces our use of computer memory by allowing us to use the Veronese map without actually forming monomials. Now instead of storage constraints we are constrained by the scaling properties of the Veronese map. Specifically, high dimensional data and large degrees of the Veronese map can result in very small numbers, i.e. $x^7y^3z^2$ is very small when x, y, and z are small. These scaling effects can make calculations on the computer inaccurate.

3.4 Symmetric Kernels

Here we provide symmetric versions of the RBF and neural network kernels for use in the Materials Design problem in Section 3.6. In fact, we provide a symmetric generalization for any kernel $\kappa : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$ which can be expressed in terms of norms and inner products, i.e. any kernel of the form $k(\mathbf{x}, \mathbf{y}) = f(\|\mathbf{x}\|^2, (\mathbf{x}, \mathbf{y}), \|\mathbf{y}\|^2)$, where $f : \mathbb{R}^3 \to \mathbb{R}$. In the case of the RBF kernel, $f(u, v, w) = \exp(-(u + 2v + w)/2\sigma^2)$ and in the case of the neural network kernel $f(u, v, w) = \tanh(av + b)$.

To write our symmetric kernel we denote by $\sigma_1, \ldots, \sigma_{n!}$ the various permutations of $(x_1, \ldots, x_n)$. In the case of $\mathbb{R}^2$, for example, we have $\sigma_1(x_1, x_2) = (x_1, x_2)$ and $\sigma_2(x_1, x_2) = (x_2, x_1)$. Our symmetric kernel $\kappa_s : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$ is then defined by

$$\kappa_s(\mathbf{x}, \mathbf{y}) = \kappa(\sigma_1(\mathbf{x}), \mathbf{y}) + \kappa(\sigma_2(\mathbf{x}), \mathbf{y}) + \cdots + \kappa(\sigma_{n!}(\mathbf{x}), \mathbf{y}).$$

To see that κ_s is indeed symmetric, we observe that

$$\kappa_s(\sigma_i(\mathbf{x}), \mathbf{y}) = \kappa_s(\mathbf{x}, \mathbf{y})$$

and that

$$\kappa_s(\mathbf{x}, \mathbf{y}) = \kappa_s(\mathbf{y}, \mathbf{x})$$

for any permutation σ_i. The second equality is true because

$$\begin{gathered}\|\mathbf{x}\|^2 = \|\sigma_i(\mathbf{x})\|^2 \\ \{(\sigma_i(\mathbf{x}), \mathbf{y})\} = \{(\mathbf{x}, \sigma_i(\mathbf{y}))\} \\ \|\mathbf{y}\|^2 = \|\sigma_i(\mathbf{y})\|^2.\end{gathered}$$

Again using $\mathbb{R}^2$ as an example (take $\mathbf{x}, \mathbf{y} \in \mathbb{R}^2$) we have

$$\begin{gathered}\kappa_s(\mathbf{x}, \mathbf{y}) = \kappa(\mathbf{x}, \mathbf{y}) + \kappa(\sigma_2(\mathbf{x}), \mathbf{y}) \\ \kappa_s(\mathbf{x}, \sigma_2(\mathbf{y})) = \kappa(\mathbf{x}, \sigma_2(\mathbf{y})) + \kappa(\sigma_2(\mathbf{x}), \sigma_2(\mathbf{y})) = \kappa(\sigma_2(\mathbf{x}), \mathbf{y}) + \kappa(\mathbf{x}, \mathbf{y}) \\ \kappa_s(\sigma_2(\mathbf{x}), \mathbf{y}) = \kappa(\sigma_2(\mathbf{x}), \mathbf{y}) + \kappa(\sigma_2(\sigma_2(\mathbf{x})), \mathbf{y}) = \kappa(\sigma_2(\mathbf{x}), \mathbf{y}) + \kappa(\mathbf{x}, \mathbf{y}) \\ \kappa_s(\sigma_2(\mathbf{x}), \sigma_2(\mathbf{y})) = \kappa(\sigma_2(\mathbf{x}), \sigma_2(\mathbf{y})) + \kappa(\sigma_2(\sigma_2(\mathbf{x})), \sigma_2(\mathbf{y})) = \kappa(\mathbf{x}, \mathbf{y}) + \kappa(\sigma_2(\mathbf{x}), \mathbf{y}),\end{gathered}$$

i.e. $\kappa_s : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$ is symmetric.

3.5 Kernel Search

To perform kernel selection, we mimic the feature selection procedure in Section 3.3. We execute the following:

(K1) For a given kernel and choice of kernel parameters, we calculate $\mathbf{a}^*$, $J(\mathbf{a}^*)$, p and t as in (F1).

(K2) We record the results of (K1) for different kernels and kernel parameters. In fact we use various discretizations of the parameters in regions where, based on our data, we expect to encounter good kernels. See, e.g., Sections 3.6 and 3.7.

(K3) We compare the kernels using the values of $J(\mathbf{a}^*)$, p and t recorded in step (K2). Here we generally use the maximum values of $J(\mathbf{a}^*)$, p and t to compare the kernels, as opposed to the averages used in step (F3).

As a final step, we implement the Support Vector Machines corresponding to the best kernels found in steps (K1)-(K3) using the best features found in steps (F1)-(F3). We will consider the topic of SVM implementation in Chapter 4.

3.6 Application to Materials Design

Here we apply our methods to the Materials Design problem. In Section 2.5 we considered feature selection for classifying three element combinations into forming and non-forming categories. In this section we consider feature (using the symmetric Veronese kernel) and kernel selection for classifying two, three, and four element combinations into forming and non-forming categories. Our data set is an enlarged version of the data set used in Section 2.5 supplied by [52] and [37]. It includes 1333 binary examples, 4963 ternary examples, and 4278 quaternary examples along with a list of 88 possible features (in particular, we include the N1 and N2 orderings of the periodic table suggested in [36]). We also have a binary test set with 692 examples, a ternary test set with 2156 examples, and a quaternary test set with 2535 examples.

Instead of producing intensity plots for feature selection as in Section 2.5, we produced ranked lists using (F1)-(F3). Our generalized feature search in Section 3.3, using the symmetric Veronese map, allowed us to consider the larger data set described above. The increase in available features from the previous chapter to this chapter, however, forced us to consider different feature combinations. For the case of single feature selection, we again used every feature to produce our ranked lists, but for the case of feature pair and feature triple selection, we used only a subset of possible feature pairs and triples. Specifically, we required one of the features in each feature pair to be from the list of best single features, and two of the features in each feature triple to be from the list of best feature pairs.

Next, using the best features and feature pairs, we performed kernel selection, using symmetric kernels, as specified in (K1)-(K3). For this step we also scaled

our data to have mean zero and unit variance. For the symmetric Veronese kernel we used d from 1 to 10. We chose the symmetric radial basis function (RBF) kernel and symmetric neural network kernel parameter search region to have values between 0 and 2. (Recall from Section 1.2.2 that the RBF kernel is $\kappa(\mathbf{x}, \mathbf{y}) = \exp(-\|\mathbf{x} - \mathbf{y}\|^2/2\sigma^2)$ and the neural network kernel is $\kappa(\mathbf{x}, \mathbf{y}) = \tanh(a(\mathbf{x}, \mathbf{y}) + b)$.) This worked well for the RBF kernel, presumably because σ corresponds to the width of the RBF, but not as well for the neural network kernel. Thus we also tested values of a and b from 0 to 20 for the neural network kernel. Our final selections of the parameters for each kernel were chosen using plots such as those in Figures 3.1 and 3.2.

In presenting our results, we use the structure and notation of (F1)-(F3) and (K1)-(K3). Specifically, we use $J(\mathbf{a}^*)$ to denote the optimal value of Fisher's criterion obtained using training data, p to denote the percentage of training data correctly classified using Fisher's discriminant, and t to denote the percentage of test data correctly classified. In addition, we use $\overline{J(\mathbf{a}^*)}_r$, $\overline{p}$, and $\overline{t}$ to denote the averages of the values $J(\mathbf{a}^*)$, p, and t over the Veronese degrees $1, 2, \ldots, r$. Once these values are calculated, we use them to provide ranked lists of feature combinations and kernel comparisons, again as in (F1)-(F3) and (K1)-(K3). Finally, the best Support Vector Machines (using various values of C) were implemented using Joachims' SVM^{light} [20] with convergence tolerance $\epsilon = .001$. (We will discuss SVM implementation in Chapter 4.)

3.6.1 Binary Case

Feature Selection

The following tables result from steps (F1)-(F3) in Section 3.3 using the symmetric Veronese kernel with degrees $1, \ldots, 10$ in the case of single features; degrees $1, \ldots, 7$ in the case of feature pairs; and degrees $1, \ldots, 5$ in the case of feature

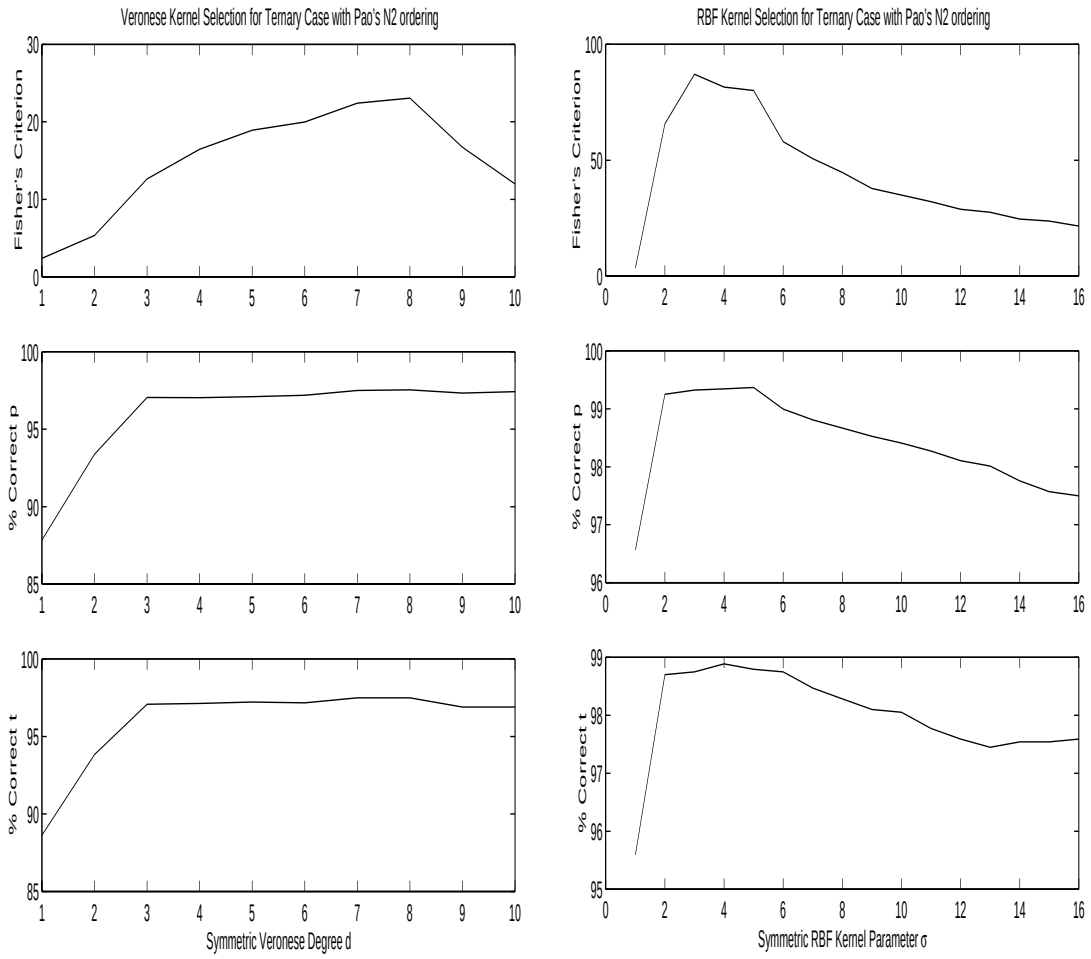

Figure 3.1: Symmetric Veronese and RBF Kernel Selection for Materials Design. Here we display kernel selection plots for the ternary case using Pao's N2 ordering. On the left are plots of Fisher's criterion, the percentage p of points correctly classified, and the percentage t of points correctly classified (from top to bottom) versus the symmetric Veronese degree d. Using these plots we selected $d = 3$. On the right are plots of the same values versus the symmetric RBF kernel parameter σ. Using the RBF plots we selected $\sigma = .25$.

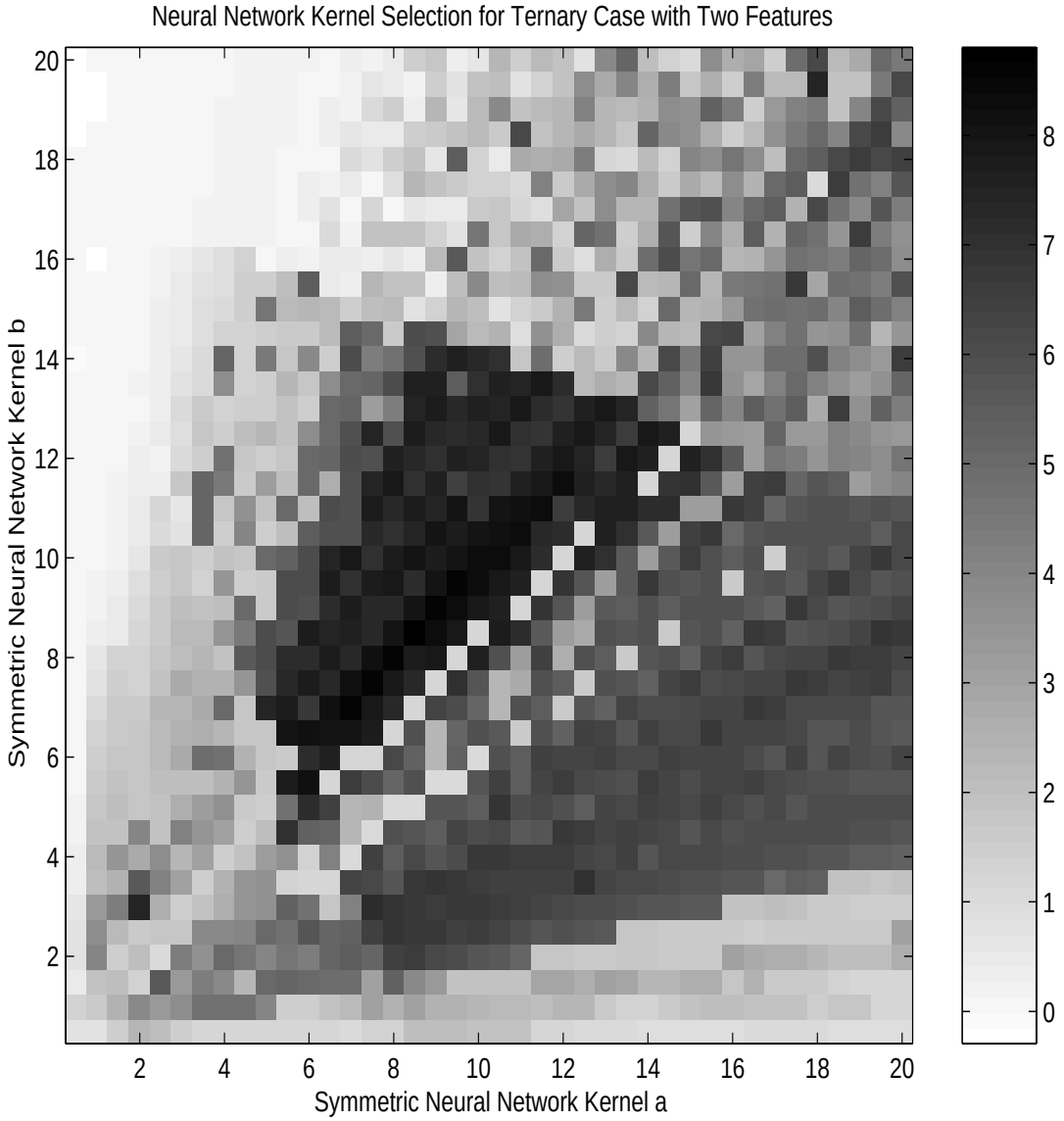

Figure 3.2: Symmetric Neural Network Kernel Selection for Materials Design. Here we display a kernel selection plot for the ternary case using Pao's N2 ordering and Pauling's Electronegativity (the third ranked feature pair). This is an intensity plot of the values of Fisher's criterion versus the symmetric neural network kernel parameters a and b. The values of Fisher's criterion are indicated by the scale bar on the right. Using this plot we selected $a = 8.5$ and $b = 8$.

triples. For the case of single features we've also included the means and standard deviations of the values $\overline{J(\mathbf{a}^*)}_r$, $\overline{p}$, and $\overline{t}$. (For two features, one of the features is from the list of best features. Thus the means and standard deviations for feature pairs are skewed and therefore not included. Similarly for feature triples.)

	Top 3 Features *Name*	$\overline{J(\mathbf{a}^*)}_{10}$	$\overline{p}$	$\overline{t}$
1	M13 Paos N2 ordering	3.14	85.76	83.82
2	M2 Mendeleev H t-d start right	2.77	85.27	84.78
3	M6 Mendeleev Pettifor regular	2.69	85.03	83.77
	Feature Mean Values	.67	65.60	63.62
	Feature Standard Deviations	.86	9.65	9.97

	Top 3 Feature Pairs *Names*	$\overline{J(\mathbf{a}^*)}_7$	$\overline{p}$	$\overline{t}$
1	M13 Paos N2 ordering G2 valence electron number (/)	6.40	90.56	88.58
2	M4 Mendeleev H d-t start right G2 valence electron number (/)	5.44	90.28	89.49
3	M5 Mendeleev Pettifor G2 valence electron number (/)	5.41	90.05	87.88

	Top 3 Feature Triples *Names*	$\overline{J(\mathbf{a}^*)}_5$	$\overline{p}$	$\overline{t}$
1	M13 Paos N2 ordering G2 valence electron number (/) M8 Mendeleev t-d start left	4.08	88.69	87.66
2	M8 Mendeleev t-d start left G2 valence electron number (/) I3 magnetic susceptibility (m3 kg-1)	3.65	88.73	87.72
3	M8 Mendeleev t-d start left G2 valence electron number (/) M10 Mendeleev d-t start left	3.03	87.23	85.43

Kernel Selection

The tables in this section result from steps (K1)-(K3) in Section 3.5 using the symmetric versions of the Veronese, RBF, and neural network kernels (see Sections 1.2.2 and 3.4). We include the best kernel of each type for the best three features,

the best three feature pairs, and the best three feature triples. These kernels were chosen using plots similar to those in Figures 3.1 and 3.2.

	Kernel Comparisons for Top 3 Features *Feature*	*Best Classifier*	$J(\mathbf{a}^*)$	p	t
1	M13 Paos N2 ordering	Veronese $d = 3$	2.42	84.77	82.80
		RBF $\sigma = .5$	5.61	92.27	91.47
		Net $a = 7.7, b = 5.2$	1.35	76.22	73.41
2	M2 Mendeleev H t-d rt.	Veronese $d = 3$	2.18	85.22	84.68
		RBF $\sigma = .25$	7.38	92.12	91.47
		Net $a = 4.4, b = 4.7$	1.18	71.94	70.81
3	M6 Mend. Pettifor reg.	Veronese $d = 3$	2.10	84.44	82.51
		RBF $\sigma = .45$	5.08	91.82	90.03
		Net $a = 18.1, b = 3$	1.15	76.97	75.87

	Kernel Comparisons for Top 3 F. Pairs *Feature Pair*	*Best Classifier*	$J(\mathbf{a}^*)$	p	t
1	M13 Paos N2 ordering G2 valence electron #	Veronese $d = 3$	3.97	91.00	89.88
		RBF $\sigma = .65$	12.1	95.87	93.06
		Net $a = 14.8, b = 6.4$	1.50	83.57	82.94
2	M4 Mendeleev H d-t rt. G2 valence electron #	Veronese $d = 3$	3.71	90.02	90.46
		RBF $\sigma = .25$	26.2	98.35	92.20
		Net $a = 16.8, b = 7$	.739	76.67	76.88
3	M5 Mendeleev Pettifor G2 valence electron #	Veronese $d = 3$	3.50	89.00	88.87
		RBF $\sigma = .2$	35.0	95.42	91.91
		Net $a = 7, b = 5$	1.07	85.25	80.64

	Kernel Comparisons for Top 3 F. Triples *Feature Triple*	*Best Classifier*	$J(\mathbf{a}^*)$	p	t
1	M13 Paos N2 ordering	Veronese $d = 5$	7.92	94.22	92.05
	G2 valence electron #	RBF $\sigma = .6$	36.4	99.02	93.50
	M8 Mendeleev t-d left	Net $a = 6, b = 12$	1.6	82.67	81.36
2	M8 Mendeleev t-d left	Veronese $d = 5$	7.54	94.90	92.49
	G2 valence election #	RBF $\sigma = .35$	31.1	97.75	92.34
	I3 magnetic susceptibility	Net $a = 9.5, b = 14.5$	.984	77.79	77.89
3	M8 Mendeleev t-d left	Veronese $d = 5$	5.29	91.30	89.45
	G2 valence electron #	RBF $\sigma = .55$	33.6	98.80	93.50
	M10 Mendeleev d-t left	Net $a = 6, b = 14$	.938	80.65	80.49

SVM Classifiers

Using Joachims' SVM^{light} [20] we trained (see Chapter 4) three Support Vector Machines on the binary data. The first SVM was trained using Pao's N2 ordering

with a symmetric RBF kernel of $\sigma = .5$ and a training value of $C = 10$. This SVM achieved a success rate of $p = 92.72\%$ on the training set and $t = 91.33\%$ on the test set. The second SVM was trained using Pao's N2 ordering and the valence electron number with a symmetric RBF kernel of $\sigma = .65$ and a training value of $C = 100$. Success rates of $p = 96.92\%$ on the training set and $t = 94.36\%$ on the test set were achieved in this case. The last SVM was trained using Pao's N2 ordering, the valence electron number, and the Mendeleev t-d left feature with a symmetric RBF kernel of $\sigma = .6$ and a training value of $C = 10$. This SVM yielded $p = 97.82\%$ on the training set and $t = 94.88\%$ on the test set.

Remark

In our search for good binary features we discovered that the M2 feature was the negative of the M3 feature, and that the G1 and G2 features were identical. Thus we selected only one of M2 or M3 and one of G1 or G2 in the final results. (This remark also applies to the Ternary and Quaternary cases.)

3.6.2 Ternary Case

Feature Selection

Here we again implemented steps (F1)-(F3) in Section 3.3 using the symmetric Veronese kernel. In this case we used degrees $1, \ldots, 7$ for single features and $1, \ldots, 5$ for feature pairs. As in the binary case, we include means and standard deviations in the case of single features.

	Top 3 Features *Name*	$\overline{J(\mathbf{a}^*)}_7$	$\overline{p}$	$\overline{t}$
1	M13 Paos N2 ordering	14.02	95.30	95.51
2	M3 Mendeleev H d-t start left	11.24	94.98	94.86
3	M5 Mendeleev Pettifor	10.86	94.64	94.98
	Feature Mean Values	2.29	75.32	74.82
	Feature Standard Deviations	3.27	11.77	12.08

	Top 5 Feature Pairs *Names*	$\overline{J(\mathbf{a}^*)}_5$	$\overline{p}$	$\overline{t}$
1	M13 Paos N2 ordering G1 group number (/)	21.0	95.76	95.59
2	M13 Paos N2 ordering I25 delta H interface O in M Miedema	19.1	95.99	95.81
3	M13 Paos N2 ordering E2 electronegativity (Pauling) (/)	18.7	96.07	96.09

Kernel Selection

Here are kernel comparisons for the top 3 ternary features and feature pairs.

	Kernel Comparisons for Top 3 Features *Feature*	*Best Classifier*	$J(\mathbf{a}^*)$	p	t
1	M13 Paos N2 ordering	Veronese $d = 3$	12.62	97.05	97.08
		RBF $\sigma = .25$	80.1	99.37	98.79
		Net $a = 12.6, b = 14.8$	7.31	95.56	95.96
2	M3 Mendeleev H d-t left	Veronese $d = 3$	9.96	96.70	96.61
		RBF $\sigma = .3$	46.3	98.92	98.24
		Net $a = 18.2, b = 19$	8.90	95.86	95.92
3	M5 Mendeleev Pettifor	Veronese $d = 3$	9.91	96.24	96.38
		RBF $\sigma = .25$	56.8	99.23	98.42
		Net $a = 11.8, b = 15.4$	2.55	92.87	92.90

	Kernel Comparisons for Top 3 F. Pairs *Feature Pair*	*Best Classifier*	$J(\mathbf{a}^*)$	p	t
1	M13 Paos N2 ordering G1 group number (/)	Veronese $d = 5$	46.8	99.02	97.77
		RBF $\sigma = .45$	143	99.86	98.65
		Net $a = 12, b = 12.5$	3.6	94.87	91.56
2	M13 Paos N2 ordering I25 Δ H interface O in M	Veronese $d = 5$	39.4	98.97	97.96
		RBF $\sigma = .2$	115	99.67	98.98
		Net $a = 11, b = 14.5$	5.3	94.11	94.43
3	M13 Paos N2 ordering E2 Pauling Electroneg.	Veronese $d = 5$	38.5	98.85	98.47
		RBF $\sigma = .2$	87.8	99.65	98.75
		Net $a = 8.5, b = 8$	7.9	96.12	96.57

SVM Classifiers

In the ternary case we trained two Support Vector Machines. The first was trained using Pao's N2 ordering with a symmetric RBF kernel of $\sigma = .25$ and a training value of $C = 10$. The resulting SVM had a success rate of $p = 99.18\%$ on

the training data and $t = 98.79\%$ on the test data. The second SVM was trained using Pao's N2 ordering and the G1 group number (same as the valence electron number — see remark in Section 3.6.1) with a symmetric RBF kernel of $\sigma = .45$ and a training value of $C = 10$. This SVM had a success rate of $p = 99.93\%$ on the training data and $t = 99.07\%$ on the test data.

3.6.3 Quaternary Case

Feature Selection

In the quaternary case we used the symmetric Veronese kernel with degrees $1, \ldots, 5$ for single feature selection. We again include means and standard deviations.

	Top 5 Features *Name*	$\overline{J(\mathbf{a}^*)}_5$	$\overline{p}$	$\overline{t}$
1	M5 Mendeleev Pettifor	53.1	97.37	97.16
2	M13 Paos N2 ordering	51.3	97.24	97.10
3	E8 chemical potential Miedema (/)	40.4	97.32	97.60
	Feature Mean Values	12.0	88.33	88.02
	Feature Standard Deviations	12.4	10.53	10.79

Kernel Selection

Kernel comparisons for the quaternary case were implemented for the best 3 features.

	Kernel Comparisons for Top 3 Features *Feature*	*Best Classifier*	$J(\mathbf{a}^*)$	p	t
1	M5 Mendeleev Pettifor	Veronese $d = 2$	27.2	98.97	98.93
		RBF $\sigma = .45$	508	99.96	99.88
		Net $a = 15.6, b = 13.6$	25.3	98.51	98.34
2	M13 Paos N2 ordering	Veronese $d = 3$	52.0	99.88	99.80
		RBF $\sigma = .4$	656	100	100
		Net $a = 7.6, b = 18.4$	28.9	98.73	99.21
3	E8 chem. pot. Miedema	Veronese $d = 2$	38.7	99.56	99.49
		RBF $\sigma = .55$	299	99.88	99.80
		Net $a = 4.4, b = 17.8$	34.8	99.03	98.74

SVM Classifier

For the quaternary case we trained only one Support Vector Machine. We used the Mendeleev Pettifor number with a symmetric RBF kernel of $\sigma = .45$ and a training value of $C = 100$. In this case the resulting SVM had a 100% success rate on both the training and test data.

3.7 Application to USPS Handwritten Digit Recognition

Here we use the United States Postal Service database of handwritten digits to compare our method of kernel selection with the methods in [11], [9], and [42]. (We do not consider feature selection for this problem.) The United States Postal Service database consists of 16×16 images of the digits 0 through 9. There are 7291 training examples and 2007 test examples. The USPS database can be obtained from [47].

In our first example, we perform an experiment found in [11]. Specifically, we have selected σ for an RBF kernel designed to separate the digits 0 from 3. Plots of $J(\mathbf{a}^*)$, $100 - p$, and $100 - t$ versus σ are provided on the left side of Figure 3.3. Based on these plots we selected σ from 5 to 8. Our plots of $100 - p$ and $100 - t$ are similar (in shape and with similar values) to the plot of the generalization error of the actual Support Vector Machine in [11].

In our second example, we have again selected σ for an RBF kernel. In this case the kernel is designed to separate the digits 0 to 4 from 5 to 9. To provide a more direct comparison of our method with the methods in [9] and [42] we first randomly permuted the entire USPS data set (training and test sets together) and then divided the training set into 23 subsets of size 317 (this was originally proposed in [42] and was also performed in [9]). We then executed our method of kernel selection. We include plots of the average values of $J(\mathbf{a}^*)$, $100 - p$, and $100 - t$ on the right side of Figure 3.3. The error bars show standard deviations

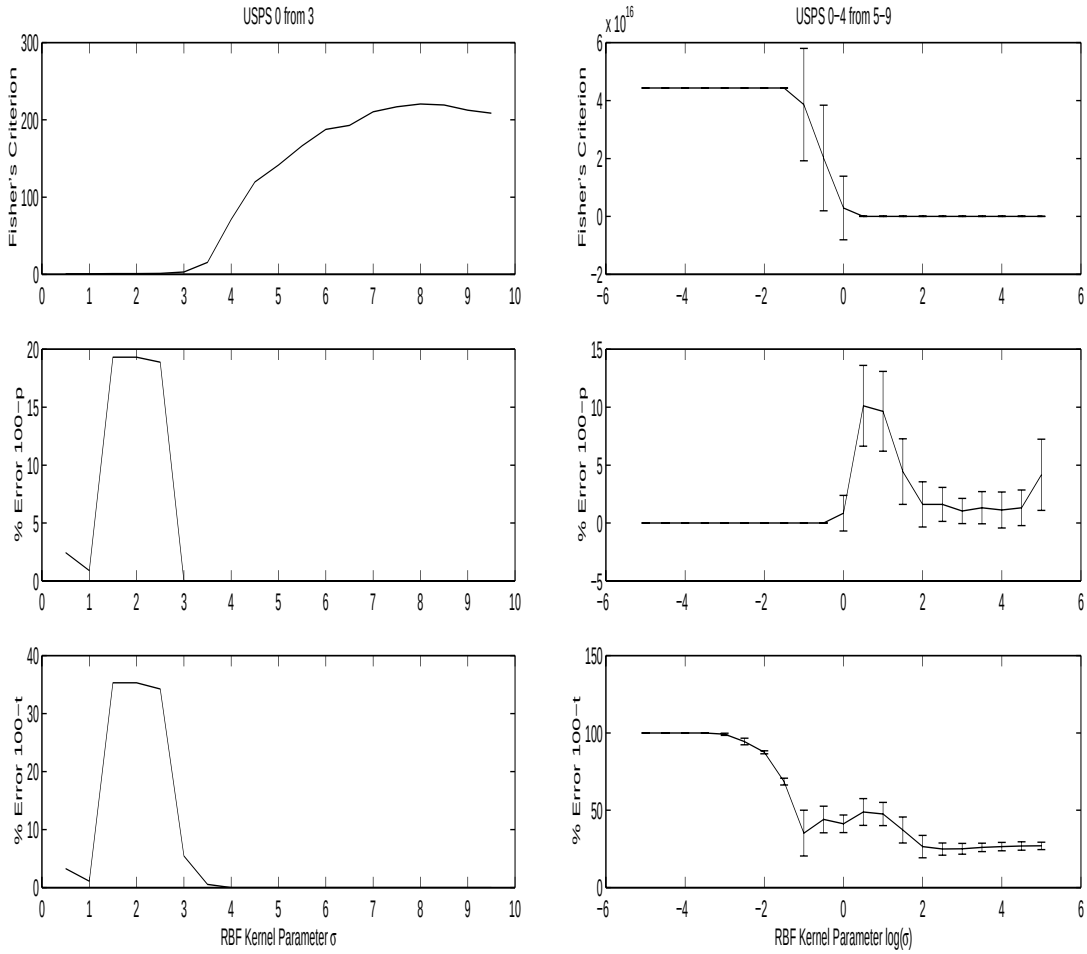

Figure 3.3: USPS Partial Problems. On the left are the plots of $J(\mathbf{a}^*)$, $100-p$, and $100-t$ (from top to bottom) versus RBF kernel σ for the problem of separating 0 from 3 in the USPS data set. On the right are the plots of $J(\mathbf{a}^*)$, $100-p$, and $100-t$ versus $\log(\sigma)$ for the problem of separating the digits 0 to 4 from 5 to 9.

over the 23 subsets. For this example our method selected $\sigma = .368$. In addition, our plot of $100-t$ is similar (in shape only) to the plots of generalization error in [9] and [42].

Finally, we have selected Veronese, RBF, and neural network kernels for use with the entire USPS data set. To be precise, we selected Veronese, RBF, and neural network kernels for separating each digit from the other nine (0 from $1,\ldots,9$; 1 from $0,2,\ldots,9$; 2 from $0,1,3,\ldots,9$; et cetera). We then trained (using the selected kernels) Support Vector Machines with Joachims' SVM^{light} [20] on each

of the above cases and constructed a classifier for the entire USPS data set by using the SVM with the largest output. That is, we classified a digit as zero if the SVM designed to separate 0 from $1, \ldots, 9$ had the largest value (before applying the sign function) of all the SVMs. We classified the digits one to nine similarly. For the Veronese kernel we used $c = 1$ and for most digits found $d = 3$ to be the best parameter. For the RBF kernel we found $\sigma = 10$ to be the best value for every digit. For the neural network kernel we selected a variety of values of a and b for the different digits, but in each case found the values of Fisher's criterion to be distributed in a similar manner. Figure 3.4 shows some plots representative of our findings for the three kernels. Our Support Vector Machine classifiers using the Veronese kernel (we did not mix kernel types) obtained an error rate of 4.5% on the test set, as did the SVMs using the RBF kernels. For the neural network kernel we obtained an error rate of 9.1% on the test set.

We have also compared our results to the results of the original application of Support Vector Machines to the USPS data [41]. Our kernel selections for the Veronese and RBF kernels agree very well with the selections in [41], as do our error rates. To be precise, the error rates in [41] were 4.0% and 4.3% for the Veronese and RBF kernels. (Also in the interest of precision, we should mention that in [41] the USPS images were smoothed via a "Gaussian kernel of width .75" prior to SVM implementation. We performed no smoothing or scaling.) Our selection for the neural network kernel, on the other hand, is in disagreement, as is our error rate. In [41] the neural network kernels used have negative values of b, which is interesting since such values are in theory invalid (see Section 1.2.2 and [8]). In addition, the error rate in [41] for the neural network kernel is 4.1%, less than half our error rate.

3.8 Conclusions

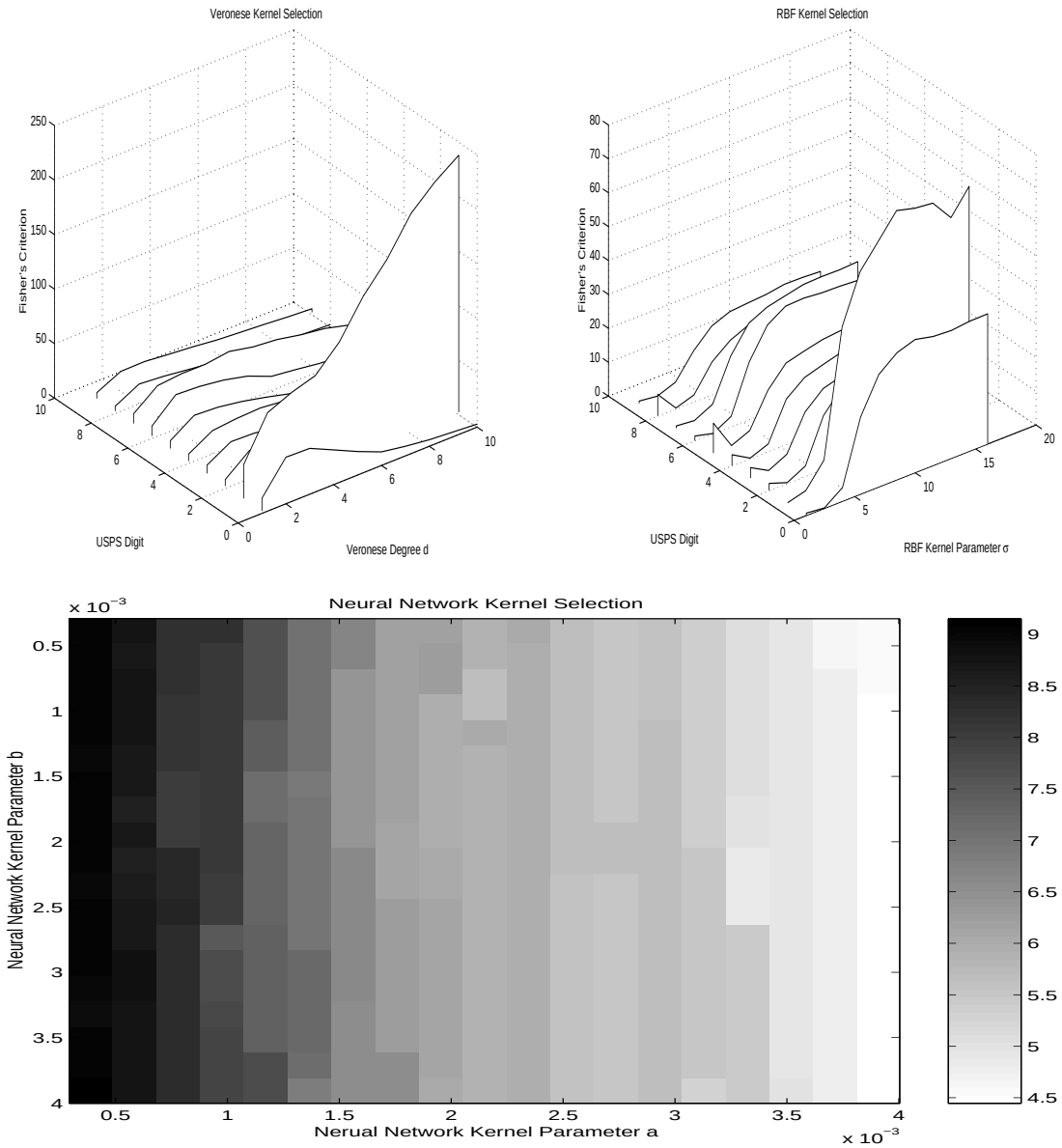

Figure 3.4: Entire USPS Problem. On the upper left is kernel selection information for the Veronese kernel for separating each of the USPS digits from the other nine. On the upper right is selection information for the RBF kernel. On the bottom is selection information for the neural network kernel for separating the digit 5 from the other nine digits. The neural network plot is an intensity plot with the values of Fisher's Criterion shown on the scale bar to the right.

In this chapter we have developed a method of kernel selection for Support Vector Machines. We have applied it the the Materials Design problem and we have used the USPS database of handwritten digits to compare our method to existing methods. We remark that

- Our method selects good kernels for use with Support Vector Machines, both in the Materials Design problem and in agreement with [11], [9], [42], and [41] on the USPS data.

- Our method works on large problems (e.g. the Materials problem and the entire USPS data set). This distinguishes our method from the method in [42] and also distinguishes our use of Fisher's discriminant from the kernel Fisher's Discriminant in [31].

- Our method works with all SVM kernels. This distinguishes our method from the method in [11].

Finally, we remark that our method of kernel Gram-Schmidt may be useful outside of kernel selection. It could, for example, be applied in conjunction with the Singular Value Decomposition (see also [43]), Least Squares regression (see also [46]), other methods of classification (see also [31]), et cetera.

Chapter 4

TRAINING SUPPORT VECTOR MACHINES

In this chapter we consider a new method for training Support Vector Machines. Support Vector Machines are traditionally obtained by solving the Support Vector Machine Quadratic Programming problem in Equation (1.3) from Section 1.2.2. Unfortunately, standard Quadratic Programming solvers typically require computer storage of a square matrix with $\#\{\mathbf{x}_i\} + \#\{\mathbf{y}_j\}$ entries to a side, i.e. a matrix with $(N + M)^2$ entries, where N and M are the number of points in our two classes X and Y. For large problems ($N + M \geq 5000$) it becomes impossible to accommodate such a matrix. Thus efforts have been made towards developing SVM specific training algorithms. In particular, the work in [34], [39], [20], [14], [27], and [24] has addressed this problem. In [34] a decomposition of the full SVM QP problem into subproblems is proposed, in [39] this decomposition is used with subproblems of the smallest possible size, and in [20] a very clean and computationally efficient version of the procedure in [34] is implemented. In [14] a variant of the traditional SVM QP problem is solved using an algorithm from Statistical Mechanics known as the Adatron algorithm, in [27] another variant of the traditional SVM QP problem is solved using successive overrelaxation, and in [24] the variant in [14] is solved using a geometric reformulation of the SVM QP problem.

Here we consider a new method for training Support Vector Machines and more generally the solution of the linear separable Support Vector Machine Quad-

ratic Programming problem (see Section 1.2.1)

$$\begin{array}{rl} \max \\ \alpha_i, \beta_j \end{array} \quad \left\{ \begin{array}{l} \sum_i \alpha_i + \sum_j \beta_j - \frac{1}{2} \sum_{i,q} \alpha_i \alpha_q (\mathbf{x}_i, \mathbf{x}_q) \\ + \sum_{i,j} \alpha_i \beta_j (\mathbf{x}_i, \mathbf{y}_j) - \frac{1}{2} \sum_{j,q} \beta_j \beta_q (\mathbf{y}_j, \mathbf{y}_q) \end{array} \right. \tag{4.1}$$

$$\text{subject to} \quad \left\{ \begin{array}{l} \sum_i \alpha_i = \sum_j \beta_j \\ 0 \leq \alpha_i, \beta_j. \end{array} \right.$$

In Section 4.1 we provide an entirely different formulation of the linear separable SVM QP problem (also considered in [24]); in Section 4.2 we use our reformulation to implement a variant of Gilbert's Algorithm [17] (also considered in [24]) for training linear separable Support Vector Machines; in Sections 4.3 and 4.4 we generalize our algorithm to the full (nonlinear, nonseparable) SVM; in Section 4.5 we compare our algorithm to both the Nearest Point Algorithm [24] and Sequential Minimal Optimization [39]; and in Section 4.6 we offers our concluding remarks.

4.1 Geometric SVM Problem

To set the stage for our reformulation of the linear separable SVM QP problem we first generalize the definition of the SVM margin from Section 1.2.1. Recall that for two nonempty, finite sets X and Y in $\mathbb{R}^n$ separated by a hyperplane H, the margin of H is twice the distance from H to a nearest point in X (or, equivalently, a nearest point in Y).

We extend this definition by considering any two nonempty, compact sets X and Y in $\mathbb{R}^n$. They need not even be disjoint. We then write the margin of a hyperplane $H = \{\mathbf{x} \in \mathbb{R}^n : (\mathbf{x}, \mathbf{w}) + b = 0, \mathbf{w} \neq \mathbf{0}\}$ as

$$m(\mathbf{w}) = d(P_{\mathbf{w}} C_X, P_{\mathbf{w}} C_Y),$$

where C_X and C_Y are the convex hulls of X and Y respectively, $P_{\mathbf{w}}$ is the orthogonal projection $\frac{\mathbf{w}\mathbf{w}^T}{\|\mathbf{w}\|^2}$ onto the one-dimensional subspace spanned by $\mathbf{w}$, and $d(\bullet, \bullet)$ denotes the distance between two sets, i.e. $d(A, B) = \min\{\|\mathbf{a} - \mathbf{b}\| : \mathbf{a} \in A, \mathbf{b} \in B\}$,

where $\|\bullet\|$ denotes the Euclidean norm in $\mathbb{R}^n$. (Note that we may use min because X and Y are compact.)

If H does not actually separate X and Y (which would be the case if $X \cap Y \neq \emptyset$), then the margin is simply zero. In addition, since $m(\mathbf{w}) = m(c\mathbf{w})$ for $c \neq 0$ we may restrict the domain of m to the unit sphere $S^{n-1} = \{\mathbf{x} \in \mathbb{R}^n : \|\mathbf{x}\| = 1\}$ with no loss of information.

As we will verify in Lemma 4.1, the margin $m : S^{n-1} \to \mathbb{R}$ is a continuous function on a compact set so by the Extreme Value Theorem achieves a global maximum at some point $\mathbf{w}^* \in S^{n-1}$. This is, of course, the point we wish to locate, as it is the normal to the maximal margin hyperplane.

Finally, let $S = X - Y = \{\mathbf{x} - \mathbf{y} : \mathbf{x} \in X, \mathbf{y} \in Y\}$ be the set of secants between our two classes, let C_S be the convex hull of S, and let $\mathbf{s}^*$ be the closest point in C_S to the origin. Since C_S is compact and convex, we are assured that $\mathbf{s}^*$ exists and is unique. For this and other interesting facts about convex sets see [25].

Lemma 4.1. *The margin $m : S^{n-1} \to \mathbb{R}$ is continuous.*

Proof. Suppose $\mathbf{w}, \mathbf{v} \in S^{n-1}$. Since $C_S = C_X - C_Y$ we have

$$\begin{aligned} m(\mathbf{w}) &= \min\{\|P_{\mathbf{w}}\mathbf{x} - P_{\mathbf{w}}\mathbf{y}\| : \mathbf{x} \in C_X, \mathbf{y} \in C_Y\} \\ &= \min\{\|\mathbf{w}\mathbf{w}^T(\mathbf{x} - \mathbf{y})\| : \mathbf{x} \in C_X, \mathbf{y} \in C_Y\} \\ &= \min\{|(\mathbf{s}, \mathbf{w})| : \mathbf{s} \in C_S\} \\ &= |(\mathbf{t}^*, \mathbf{w})| \end{aligned}$$

for some $\mathbf{t}^* \in C_S$ and similarly $m(\mathbf{v}) = |(\mathbf{u}^*, \mathbf{v})|$ for some $\mathbf{u}^* \in C_S$.

Now by the Cauchy-Schwarz inequality

$$\begin{aligned} m(\mathbf{w}) - m(\mathbf{v}) &= |(\mathbf{t}^*, \mathbf{w})| - |(\mathbf{u}^*, \mathbf{v})| \\ &\leq |(\mathbf{u}^*, \mathbf{w})| - |(\mathbf{u}^*, \mathbf{v})| \\ &\leq |(\mathbf{u}^*, \mathbf{w} - \mathbf{v})| \\ &\leq \|\mathbf{u}^*\| \|\mathbf{u} - \mathbf{v}\| \\ &\leq M\|\mathbf{w} - \mathbf{v}\|, \end{aligned}$$

where M bound C_S, i.e. $\|\mathbf{s}\| \leq M$ for all $\mathbf{s} \in C_S$. Similarly we have $m(\mathbf{w}) - m(\mathbf{v}) \geq -M\|\mathbf{w} - \mathbf{v}\|$ so that $m : S^{n-1} \to \mathbb{R}$ is continuous as claimed. *Q.E.D.*

Theorem 4.2. *If nonempty, compact sets X and Y in $\mathbb{R}^n$ are linearly separable then the margin $m : S^{n-1} \to \mathbb{R}$ attains a nonzero global maximum at $\mathbf{w}^* = \frac{\mathbf{s}^*}{\|\mathbf{s}^*\|}$.*

Proof. The hypotheses give $C_X \cap C_Y = \emptyset$ (see [25]). Thus $C_S = C_X - C_Y$ does not contain the origin so that $\mathbf{w}^*$ is well-defined. We wish to show that $m(\mathbf{w}^*) \geq m(\mathbf{w})$ for all $\mathbf{w} \in S^{n-1}$ and that $m(\mathbf{w}^*) > 0$. As in Lemma 4.1 and by the Cauchy-Schwarz inequality

$$\begin{aligned} m(\mathbf{w}) &= \min\{|(\mathbf{s}, \mathbf{w})| : \mathbf{s} \in C_S\} \\ &\leq |(\mathbf{s}^*, \mathbf{w})| \\ &\leq \|\mathbf{s}^*\| \|\mathbf{w}\| = \|\mathbf{s}^*\|. \end{aligned}$$

Since $\|\mathbf{s}^*\| \neq 0$, it is enough to show that $m(\mathbf{w}^*) = \|s^*\|$, i.e., that $|(\mathbf{s}, \mathbf{w}^*)| \geq |(\mathbf{s}^*, \mathbf{w}^*)| = \|\mathbf{s}^*\|$ for all $\mathbf{s} \in C_S$. For this we use contradiction: assume there is an $\mathbf{s} \in C_S$ such that $|(\mathbf{s}, \mathbf{w}^*)| < \|\mathbf{s}^*\|$. We will show that this implies the existence of a point $\mathbf{t} \in C_S$ with $\|\mathbf{t}\| < \|\mathbf{s}^*\|$, which contradicts the definition of $\mathbf{s}^*$.

Consider the line segment l from $\mathbf{s}^*$ to $\mathbf{s} \neq \mathbf{s}^*$ given by $(1 - \lambda)\mathbf{s}^* + \lambda\mathbf{s}$ with $0 \leq \lambda \leq 1$. Note that $l \subset C_S$ and in particular any point of l is in C_S. Let λ_0 be the scalar projection $\frac{-(\mathbf{s}^*, \mathbf{s} - \mathbf{s}^*)}{\|\mathbf{s} - \mathbf{s}^*\|^2}$ of $-\mathbf{s}^*$ onto $\mathbf{s} - \mathbf{s}^*$. Let $\mathbf{t} = (1 - \lambda_0)\mathbf{s}^* + \lambda_0\mathbf{s}$.

We claim that $\mathbf{t} \in l \subset C_S$ and that $\|\mathbf{t}\| < \|\mathbf{s}^*\|$. To see that $\mathbf{t} \in l$ it suffices to show that $0 < \lambda_0 < 1$. By assumption we have

$$|(\mathbf{s}, \mathbf{w}^*)| = \frac{|(\mathbf{s}, \mathbf{s}^*)|}{\|\mathbf{s}^*\|} < \|\mathbf{s}^*\|,$$

so that

$$(\mathbf{s}, \mathbf{s}^*) \leq |(\mathbf{s}, \mathbf{s}^*)| < \|\mathbf{s}^*\|^2$$
$$\frac{\|\mathbf{s}^*\|^2 - (\mathbf{s}, \mathbf{s}^*)}{\|\mathbf{s} - \mathbf{s}^*\|^2} = \lambda_0 > 0.$$

And since by definition of $\mathbf{s}^*$

$$(\mathbf{s}, \mathbf{s}^*) < \|\mathbf{s}^*\|^2 < \|\mathbf{s}\|^2,$$

we have

$$\|\mathbf{s}^*\|^2 - (\mathbf{s}, \mathbf{s}^*) < \|\mathbf{s} - \mathbf{s}^*\|^2 = \|\mathbf{s}\|^2 - 2(\mathbf{s}, \mathbf{s}^*) + \|\mathbf{s}^*\|^2,$$

or, equivalently, $\lambda_0 < 1$.

To see that $\|\mathbf{t}\| < \|\mathbf{s}^*\|$ we note that $\mathbf{t}$ is orthogonal to $-\lambda_0(\mathbf{s} - \mathbf{s}^*)$. Hence, we can apply the Pythagorean Theorem to get

$$\|\mathbf{t}\|^2 + \| - \lambda_0(\mathbf{s} - \mathbf{s}^*)\|^2 = \|\mathbf{t} - \lambda_0(\mathbf{s} - \mathbf{s}^*)\|^2 = \|\mathbf{s}^*\|^2.$$

Since $0 < \lambda_0 < 1$ and $\mathbf{s} \neq \mathbf{s}^*$, we have $\|\mathbf{t}\|^2 < \|\mathbf{s}^*\|^2$ as desired.

We have thus shown that $m(\mathbf{w}^*) = \|\mathbf{s}^*\| > 0$ and hence that $m : S^{n-1} \to \mathbb{R}$ attains a nonzero global maximum at $\mathbf{w}^*$. *Q.E.D.*

Corollary 4.3. *The maximal margin hyperplane is unique.*

Proof. In the proof of Theorem 4.2, we used the Cauchy-Schwarz inequality to assert that

$$m(\mathbf{w}) \leq |(\mathbf{s}^*, \mathbf{w})| \leq \|\mathbf{s}^*\|\|\mathbf{w}\| = m(\mathbf{w}^*)$$

for $\mathbf{w} \in S^{n-1}$. In fact the inequality is strict when $\mathbf{w}$ and $\mathbf{s}^*$ are linearly independent, i.e. the margin $m : S^{n-1} \to \mathbb{R}$ achieves global maximum only when

$\mathbf{w} = \pm\mathbf{w}^*$. As normal vectors, these yield the same hyperplane so the maximal margin hyperplane is indeed unique. It is given by

$$H^* = \{\mathbf{x} \in \mathbb{R}^n : 2(\mathbf{x}, \mathbf{s}^*) = \|\mathbf{x}^*\|^2 - \|\mathbf{y}^*\|^2\},$$

where $\mathbf{s}^* = \mathbf{x}^* - \mathbf{y}^*$. *Q.E.D.*

Theorem 4.4. *The linear separable Support Vector Machine Quadratic Programming problem in Equation (4.1) has solution* $\mathbf{w}^* = \frac{2\mathbf{s}^*}{\|\mathbf{s}^*\|^2}$.

Proof. First, it is instructive to observe that $\mathbf{s}^*$ solves a constrained version of the linear separable SVM QP problem. In particular, if $\mathbf{w} \in C_S$ we can write (see [25])

$$\mathbf{w} = \textstyle\sum_{ij} \gamma_{ij}(\mathbf{x}_i - \mathbf{y}_j),$$

where $\sum_{ij} \gamma_{ij} = 1$, and $\gamma_{ij} \geq 0$ for all ij. By setting $\alpha_i = \sum_j \gamma_{ij},\ \beta_j = \sum_i \gamma_{ij}$ we see that

$$\mathbf{w} = \textstyle\sum_i \alpha_i \mathbf{x}_i - \sum_j \beta_j \mathbf{y}_j,$$

with

$$\begin{gathered}\textstyle\sum_i \alpha_i = \sum_j \beta_j = 1, \\ \alpha_i, \beta_j \geq 0.\end{gathered}$$

Thus minimizing $\|\mathbf{w}\|$ when $\mathbf{w} \in C_S$ implies solving

$$\begin{array}{rl} \max\limits_{\alpha_i, \beta_j} & \left\{ \begin{array}{l} 2 - \frac{1}{2}\|\mathbf{w}\|^2 = \sum_i \alpha_i + \sum_j \beta_j - \frac{1}{2}\sum_{i,q} \alpha_i \alpha_q (\mathbf{x}_i, \mathbf{x}_q) \\ + \sum_{i,j} \alpha_i \beta_j (\mathbf{x}_i, \mathbf{y}_j) - \frac{1}{2}\sum_{j,q} \beta_j \beta_q (\mathbf{y}_j, \mathbf{y}_q) \end{array} \right. \\ \text{subject to} & \left\{ \begin{array}{l} \sum_i \alpha_i = \sum_j \beta_j = 1, \\ \alpha_i, \beta_j \geq 0. \end{array} \right. \end{array}$$

Next, we note that we can in fact generalize this approach by taking $c \geq 0$ and considering the convex hull C_{cS}, where $cS = \{c\mathbf{s} : \mathbf{s} \in S\}$. In this case we can write $\mathbf{w} \in C_{cS}$ as

$$\mathbf{w} = \sum_i \alpha_i \mathbf{x}_i - \sum_j \beta_j \mathbf{y}_j,$$

with $\sum_i \alpha_i = \sum_j \beta_j = c$ and $\alpha_i, \beta_j \geq 0$. Thus minimizing $\|\mathbf{w}\|$ when $\mathbf{w} \in C_{cS}$ implies solving

$$\begin{array}{ll} \max\limits_{\alpha_i, \beta_j} & \left\{ \begin{array}{l} 2c - \frac{1}{2}\|\mathbf{w}\|^2 = \sum_i \alpha_i + \sum_j \beta_j - \frac{1}{2}\sum_{i,q} \alpha_i \alpha_q (\mathbf{x}_i, \mathbf{x}_q) \\ + \sum_{i,j} \alpha_i \beta_j (\mathbf{x}_i, \mathbf{y}_j) - \frac{1}{2}\sum_{j,q} \beta_j \beta_q (\mathbf{y}_j, \mathbf{y}_q) \end{array} \right. \\ \text{subject to} & \left\{ \begin{array}{l} \sum_i \alpha_i = \sum_j \beta_j = c, \\ \alpha_i, \beta_j \geq 0. \end{array} \right. \end{array}$$

This problem (by Theorem 4.2) has solution $c\mathbf{s}^*$, the point in C_{cS} closest to the origin.

Finally, we finish the proof by observing that we have parameterized the linear separable SVM QP problem by c. Maximizing $2c - \frac{1}{2}\|c\mathbf{s}^*\|^2 = 2c - \frac{1}{2}c^2\|\mathbf{s}^*\|^2$ over $c \geq 0$ we obtain the optimal value $c^* = \frac{2}{\|\mathbf{s}^*\|^2}$ of c and hence the solution $\mathbf{w}^* = c^*\mathbf{s}^* = \frac{2\mathbf{s}^*}{\|\mathbf{s}^*\|^2}$ of the linear separable Support Vector Machine Quadratic Programming problem in Equation (4.1). *Q.E.D.*

The above results appear in various sources but with different proofs. In particular, results similar to Theorem 4.2 appear in [4] and [40], Corollary 4.3 appears in [51], and Theorems 4.2 and 4.4 appear in [24].

Theorem 4.2 is particularly useful for interpreting Support Vector Machines. It tells us that the maximal margin hyperplane is essentially the "perpendicular bisector" of a line segment connecting the two closest points of the convex hulls of X and Y. This is illustrated Figure 4.1(c). Figure 4.1 in its entirety gives a pictorial summary of Theorem 4.2 in its relation to SVMs.

4.2 Gilbert's Algorithm

Theorem 4.2 provides a geometric solution to the Support Vector Machine problem, but in order to actually implement a SVM we need an algorithm to

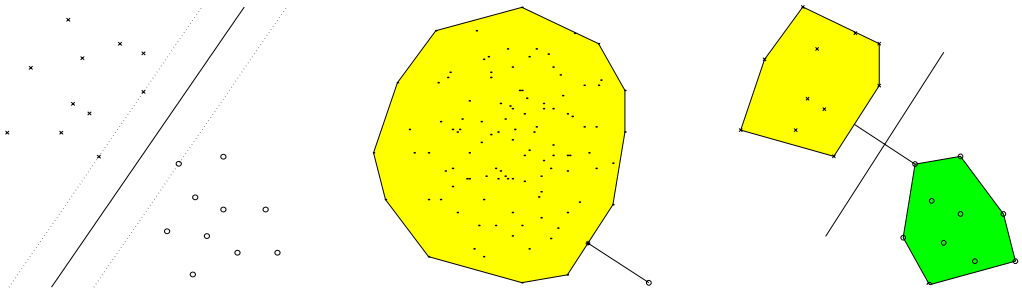

Figure 4.1: Pictorial summary of Theorem 4.2. On the left (a) is the typical illustration of a Support Vector Machine. The two classes X and Y are depicted using x's and o's, with the maximal margin hyperplane shown as a solid line separating them. The two dotted lines run through the support vectors and are separated by a distance equal to the margin. In the middle (b) is the corresponding secant hull with the origin marked by a circle, and with the point $\mathbf{s}^*$ in the secant hull closest to the origin marked by a star (and connected to the origin by a line segment). On the right (c) the maximal margin hyperplane is shown as the perpendicular bisector of a shortest line segment connecting the convex hulls of X and Y.

locate the point $\mathbf{s}^*$ in the secant hull closest to the origin. Gilbert's Algorithm [17] performs exactly this task.

To describe Gilbert's Algorithm we use the following notation. For $\mathbf{a}, \mathbf{b} \in \mathbb{R}^n$ we set

$$[\mathbf{a}, \mathbf{b}]^* = \begin{cases} \mathbf{a} & \text{if} \quad -(\mathbf{a}, \mathbf{b}-\mathbf{a}) \leq 0 \\ \mathbf{a} + \frac{-(\mathbf{a},\mathbf{b}-\mathbf{a})}{\|\mathbf{b}-\mathbf{a}\|^2}(\mathbf{b}-\mathbf{a}) & \text{if} \quad 0 < -(\mathbf{a}, \mathbf{b}-\mathbf{a}) < \|\mathbf{b}-\mathbf{a}\|^2. \\ \mathbf{b} & \text{if} \quad \|\mathbf{b}-\mathbf{a}\|^2 \leq -(\mathbf{a}, \mathbf{b}-\mathbf{a}) \end{cases}$$

Thus the point $[\mathbf{a}, \mathbf{b}]^*$ is the point on the line segment from $\mathbf{a}$ to $\mathbf{b}$ closest to the origin.

Assuming that X and Y are again finite and writing $S = X - Y = \{\mathbf{s}_m\}$, we define the *support function* $g : \mathbb{R}^n \to \mathbb{R}$ by

$$g(\mathbf{x}) = \max_m\{(\mathbf{x}, \mathbf{s}_m)\}$$

and the *contact function* $g^* : \mathbb{R}^n \to \mathbb{R}^n$ by

$$g^*(\mathbf{x}) = \mathbf{s}_{m_0},$$

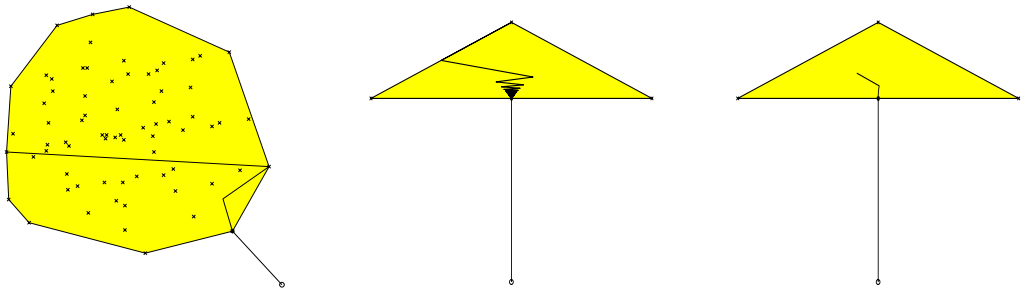

Figure 4.2: Convergence of Gilbert's Algorithm. Here we illustrate the behavior of Gilbert's Algorithm. On the left (a) $\mathbf{s}^* \in S$, and the algorithm, whose progress is tracked by the jagged line, converges to $\mathbf{s}^*$ in a finite number of steps. In the middle (b), $\mathbf{s}^* \notin S$ and the algorithm converges only asymptotically to $\mathbf{s}^*$. On the right (c), Gilbert's Algorithm for Support Vector Machines converges rapidly in angle to $\mathbf{s}^*$.

where m_0 corresponds to the the smallest indices i_0 and j_0 of X and Y such that $(\mathbf{x}, \mathbf{s}_{m_0}) \geq (\mathbf{x}, \mathbf{s}_m)$ for all m. This contact function gives an extreme point of S in the direction $\mathbf{x}$. (See [24] or [17].)

Finally, let $\mathbf{w}_1$ be a random point in S and let $\mathbf{w}_k = [\mathbf{w}_{k-1}, g^*(-\mathbf{w}_{k-1})]^*$ for $k > 1$. Gilbert showed in [17] that $\lim_{k\to\infty} \mathbf{w}_k = \mathbf{s}^*$, where $\mathbf{s}^*$ is the closest point in C_S to the origin. Gilbert also made some observations on the convergence properties of the sequence $\mathbf{w}_k$. We note merely that Gilbert's Algorithm will converge in a finite number of steps if $\mathbf{s}^* \in S$, but that asymptotic convergence is very likely when $\mathbf{s}^* \notin S$. Gilbert's Algorithm is illustrated in Figure 4.2 parts (a) and (b).

Unfortunately, the asymptotic behavior exhibited by Gilbert's Algorithm in Figure 4.2(b) is not only likely but very slow as well ($\sim 1/n$). It was for this reason that Gilbert's Algorithm was abandoned in [24]. Fortunately, Gilbert's Algorithm converges to $\mathbf{s}^*$ in angle much faster than in norm. That is, $(\mathbf{w}_k, \mathbf{s}^*)/(\|\mathbf{w}_k\|\|\mathbf{s}^*\|)$ converges to 1 much faster than $\|\mathbf{w}_k - \mathbf{s}^*\|$ converges to 0. Since the SVM maximal margin hyperplane is determined by direction and not by length (see e.g. Theorem 4.4), we can use this angle convergence criterion to greatly improve the performance of Gilbert's Algorithm for training Support Vector Machines.

In addition, we can improve the rate of convergence in angle of Gilbert's Algorithm by using an associated sequence of averages computed as follows. To compute $\overline{\mathbf{m}}_1$, we iterate Gilbert's Algorithm until $g^*(-\mathbf{w}_{k-1}) = g^*(-\mathbf{w}_{j-1})$ for some $j < k$. We set $\overline{\mathbf{m}}_1 = \frac{1}{k-j}\sum_{i=j+1}^{k} \mathbf{w}_i$. To compute $\overline{\mathbf{m}}_2$, we repeat this process as if we were restarting Gilbert's Algorithm from $\mathbf{w}_k$. In this manner, we obtain a sequence $\overline{\mathbf{m}}_1, \overline{\mathbf{m}}_2, \ldots$ of averages of points in Gilbert's Algorithm which also converge to $\mathbf{s}^*$ (since the points in Gilbert's Algorithm converge to $\mathbf{s}^*$). More importantly, the points $\overline{\mathbf{m}}_1, \overline{\mathbf{m}}_2, \ldots$ converge very rapidly in angle to $\mathbf{s}^*$. This is illustrated in Figure 4.2(c).

Algorithm 4.5. *Gilbert's Algorithm for Support Vector Machines.*

(1) Compute $\overline{\mathbf{m}}_1, \overline{\mathbf{m}}_2, \ldots$ as above.

(2) Stop when $\frac{(\overline{\mathbf{m}}_l, \overline{\mathbf{m}}_{l-1})}{\|\overline{\mathbf{m}}_l\| \|\overline{\mathbf{m}}_{l-1}\|} < \epsilon$, where ϵ is the convergence tolerance for the algorithm.

4.3 Nonlinear Generalization

To generalize Gilbert's Algorithm for Support Vector Machines to the nonlinear case we simply use SVM kernels. Recall from Section 1.2.2 that a kernel $\kappa : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$ can be used to implicitly preprocess data by an associated map $\Phi : \mathbb{R}^n \to F$, where F is a Hilbert space and

$$\kappa(\mathbf{x}, \mathbf{y}) = (\Phi(\mathbf{x}), \Phi(\mathbf{y})).$$

Kernels are used to make Support Vector Machines nonlinear (see Section 1.2.2). Hence, we use them to make Gilbert's Algorithm for SVMs nonlinear. This is accomplished by first rewriting Gilbert's Algorithm for SVMs in terms of inner products and then replacing the inner products with kernels.

We write

$$\mathbf{w}_{k-1} = \sum_i \alpha_i \mathbf{x}_i - \sum_j \beta_j \mathbf{y}_j.$$

for each $\mathbf{w}_{k-1}$ in Gilbert's Algorithm, where the α_i and β_j also depend on k. To implement Gilbert's Algorithm for Support Vector Machines we need to calculate $g^*(-\mathbf{w}_{k-1})$, $\mathbf{w}_k = [\mathbf{w}_{k-1}, g^*(-\mathbf{w}_{k-1})]^*$, and $\frac{(\overline{\mathbf{m}}_l, \overline{\mathbf{m}}_{l-1})}{\|\overline{\mathbf{m}}_l\| \|\overline{\mathbf{m}}_{l-1}\|}$.

To compute $g^*(-\mathbf{w}_{k-1})$ we observe that

$$\begin{aligned} g(-\mathbf{w}_{k-1}) &= \max_m\{-(\mathbf{w}_{k-1}, \mathbf{s}_m)\} \\ &= \max_i\{-(\mathbf{w}_{k-1}, \mathbf{x}_i)\} + \max_j\{(\mathbf{w}_{k-1}, \mathbf{y}_j)\}. \end{aligned}$$

(Recall S is indexed by m and X and Y are indexed by i and j.) If we denote $\max_i\{-(\mathbf{w}_{k-1}, \mathbf{x}_i)\}$ and $\max_j\{(\mathbf{w}_{k-1}, \mathbf{y}_j)\}$ by $g_X(-\mathbf{w}_{k-1})$ and $g_Y(\mathbf{w}_{k-1})$ we get

$$g^*(-\mathbf{w}_{k-1}) = g_X^*(-\mathbf{w}_{k-1}) - g_Y^*(\mathbf{w}_{k-1}).$$

Here we are using the notation in Section 4.2 to suggest that $g_X^*(\mathbf{x})$ is the point $\mathbf{x}_{i_0} \in X$ with smallest index i_0 such that $(\mathbf{x}, \mathbf{x}_{i_0}) \geq (\mathbf{x}, \mathbf{x}_i)$ for all i, and similarly for $g_Y^*(\mathbf{x})$.

Thus to compute $g^*(-\mathbf{w}_{k-1})$ we need only calculate $g_X^*(-\mathbf{w}_{k-1})$ and $g_Y^*(\mathbf{w}_{k-1})$. These are easily obtained if we keep an inner product cache containing the values $(\mathbf{w}_{k-1}, \mathbf{x}_i)$ and $(\mathbf{w}_{k-1}, \mathbf{y}_j)$ as we progress through the points in Gilbert's Algorithm. The idea of using such a cache was originated in [39] and adopted in [24].

Next we calculate $\mathbf{w}_k = [\mathbf{w}_{k-1}, g^*(-\mathbf{w}_{k-1})]^*$ from $-(\mathbf{w}_{k-1}, g^*(-\mathbf{w}_{k-1}) - \mathbf{w}_{k-1})$ and $\|g^*(-\mathbf{w}_{k-1}) - \mathbf{w}_{k-1}\|^2$. These quantities can be computed using the above inner product cache along with $\|\mathbf{w}_{k-1}\|^2$. To facilitate the computation of $\|\mathbf{w}_{k-1}\|^2$ we observe that

$$\|\mathbf{w}_k\|^2 = \|\mathbf{w}_{k-1}\|^2 - \frac{(\mathbf{w}_{k-1}, g^*(-\mathbf{w}_{k-1}) - \mathbf{w}_{k-1})^2}{\|g^*(-\mathbf{w}_{k-1}) - \mathbf{w}_{k-1}\|^2}$$

when $\mathbf{w}_k$ is neither $\mathbf{w}_{k-1}$ nor $g^*(-\mathbf{w}_{k-1})$. This recurrence allows us to compute $\|\mathbf{w}_k\|^2$ with a minimum of effort at each step of Gilbert's Algorithm.

Finally, we can maintain an inner product cache of the values $(\overline{\mathbf{m}}_l, \mathbf{x}_i)$ and $(\overline{\mathbf{m}}_l, \mathbf{y}_j)$ by adding the inner product caches containing $(\mathbf{w}_{k-1}, \mathbf{x}_i)$ and $(\mathbf{w}_{k-1}, \mathbf{y}_j)$ as we progress through the points of Gilbert's Algorithm. Using this new cache we can compute the values $\frac{(\overline{\mathbf{m}}_l, \overline{\mathbf{m}}_{l-1})}{\|\overline{\mathbf{m}}_l\| \|\overline{\mathbf{m}}_{l-1}\|}$.

4.4 Nonseparable Generalization

The final generalization of Gilbert's Algorithm for Support Vector Machines is to the nonlinear case with errors (the nonlinear nonseparable case). Unfortunately, Gilbert's Algorithm for SVMs cannot be used to solve the traditional SVM QP problem [10] in Equation (1.3) in Section 1.2.2. Fortunately, recent alternatives to the traditional SVM QP problem have been proposed in [15] and [27]. We use the approach in [15].

Specifically, we allow errors by introducing slack variables σ_i and τ_j into the SVM problem. We get

$$\begin{array}{rl} \min & \frac{1}{2}\|\mathbf{w}\|^2 + \frac{\widetilde{C}}{2}(\sum_i \sigma_i^2 + \sum_j \tau_j^2) \\ \text{subject to} & \left\{ \begin{array}{ll} (\mathbf{x}_i, \mathbf{w}) + b \geq 1 - \sigma_i & \text{for all } \mathbf{x}_i \in X \\ (\mathbf{y}_j, \mathbf{w}) + b \leq \tau_j - 1 & \text{for all } \mathbf{y}_j \in Y, \end{array} \right. \end{array} \tag{4.2}$$

where σ_i corresponds to the data point $\mathbf{x}_i$, τ_j corresponds to the data point $\mathbf{y}_j$, and $\widetilde{C}$ is a positive constant.

We use this formulation of the nonseparable Support Vector Machine because it can be solved via a separable problem (the traditional formulation [10] may be nonseparable). The separable problem is obtained as follows (see [15], [24]). Let $\mathbf{e}_i$ be a vector of length $\#\{\mathbf{x}_i\}$ (number of points in the X class) with a 1 in the ith position and zeros elsewhere. Similarly, let $\mathbf{f}_j$ be a vector of length $\#\{\mathbf{y}_j\}$ with a 1 in the jth position and zeros elsewhere. Let $\boldsymbol{\sigma}$ be the vector containing each σ_i and $\boldsymbol{\tau}$ be the vector containing each τ_j. Finally, let

$$\widetilde{\mathbf{w}} = \begin{pmatrix} \mathbf{w} \\ \sqrt{\widetilde{C}}\boldsymbol{\sigma} \\ \sqrt{\widetilde{C}}\boldsymbol{\tau} \end{pmatrix}, \ \widetilde{\mathbf{x}}_i = \begin{pmatrix} \mathbf{x}_i \\ \frac{1}{\sqrt{\widetilde{C}}}\mathbf{e}_i \\ \mathbf{0} \end{pmatrix}, \ \widetilde{\mathbf{y}}_j = \begin{pmatrix} \mathbf{y}_j \\ \mathbf{0} \\ -\frac{1}{\sqrt{\widetilde{C}}}\mathbf{f}_j \end{pmatrix}, \ \widetilde{b} = b.$$

Then $\{\widetilde{\mathbf{x}}_i\}$ and $\{\widetilde{\mathbf{y}}_j\}$ are linearly separable and we can solve

$$\begin{array}{rl} \min & \frac{1}{2}\|\widetilde{\mathbf{w}}\|^2 \\ \text{subject to} & \left\{ \begin{array}{ll} (\widetilde{\mathbf{x}}_i, \widetilde{\mathbf{w}}) + b \geq 1 & \text{for all } \widetilde{\mathbf{x}}_i \in X \\ (\widetilde{\mathbf{y}}_j, \widetilde{\mathbf{w}}) + b \leq -1 & \text{for all } \widetilde{\mathbf{y}}_j \in Y \end{array} \right. \end{array}$$

to obtain the solution of (4.2). Thus, to compute a Support Vector Machine in the case of nonseparable data we use $\{\widetilde{\mathbf{x}}_i\}$ and $\{\widetilde{\mathbf{y}}_j\}$ with Gilbert's Algorithm for SVMs.

For nonlinear nonseparable data, we replace the inner products $(\widetilde{\mathbf{x}}_i, \widetilde{\mathbf{y}}_j)$ in the dual problem

$$\begin{array}{rl} \max \\ \alpha_i, \beta_j \end{array} \left\{ \begin{array}{l} \sum_i \alpha_i + \sum_j \beta_j - \frac{1}{2}\sum_{i,q} \alpha_i \alpha_q (\widetilde{\mathbf{x}}_i, \widetilde{\mathbf{x}}_q) \\ + \sum_{i,j} \alpha_i \beta_j (\widetilde{\mathbf{x}}_i, \widetilde{\mathbf{y}}_j) - \frac{1}{2}\sum_{j,q} \beta_j \beta_q (\widetilde{\mathbf{y}}_j, \widetilde{\mathbf{y}}_q) \end{array} \right.$$

$$\text{subject to} \quad \left\{ \begin{array}{l} \sum_i \alpha_i = \sum_j \beta_j, \\ \alpha_i, \beta_j \geq 0 \end{array} \right.$$

with kernel products of the form $\widetilde{\kappa}(\widetilde{\mathbf{x}}, \widetilde{\mathbf{y}}) = \kappa(\mathbf{x}, \mathbf{y}) + \delta_{ij}/\widetilde{C}$, where δ_{ij} is 1 when $i = j$ and zero otherwise. This gives the dual

$$\begin{array}{rl} \max \\ \alpha_i, \beta_j \end{array} \left\{ \begin{array}{l} \sum_i \alpha_i + \sum_j \beta_j - \frac{1}{2}\sum_{i,q} \alpha_i \alpha_q \kappa(\mathbf{x}_i, \mathbf{x}_q) \\ + \sum_{i,j} \alpha_i \beta_j \kappa(\mathbf{x}_i, \mathbf{y}_j) - \frac{1}{2}\sum_{j,q} \beta_j \beta_q \kappa(\mathbf{y}_j, \mathbf{y}_q) \\ - \frac{1}{2\widetilde{C}}(\sum_i \alpha_i^2 + \sum_j \beta_j^2) \end{array} \right.$$

$$\text{subject to} \quad \left\{ \begin{array}{l} \sum_i \alpha_i = \sum_j \beta_j, \\ \alpha_i, \beta_j \geq 0. \end{array} \right.$$

of the nonseparable SVM problem in [15].

Hence, to implement the full nonlinear nonseparable version of Gilbert's Algorithm for Support Vector Machines we simply replace the kernel κ in the nonlinear version of Gilbert's Algorithm for SVMs with the kernel $\widetilde{\kappa}(\widetilde{\mathbf{x}}, \widetilde{\mathbf{y}}) = \kappa(\mathbf{x}, \mathbf{y}) + \delta_{ij}/\widetilde{C}$.

4.5 Examples/Comparisons

We now present some examples which we use to compare our algorithm with both the Nearest Point Algorithm [24] and Sequential Minimal Optimization [39]. (We should note that both NPA and Gilbert's Algorithm for SVMs use the above

nonseparable generalization while SMO uses the nonseparable generalization in [10].) We used `MATLAB` to code our algorithm, we obtained the NPA code from [23], and the SMO code from [2]. We used the different codes to train Support Vector Machines on the Two Spirals Data [22], the Wisconsin Breast Cancer Data [5], [3], and the Adult-4a Data [38], [5]. All three data sets were used in [24], and the Adult-4a set was used in [39].

To compare our algorithm with NPA and SMO, we ran the three algorithms on a random subset of each data set containing 90% of the original data. We used the remaining 10% of each data set as a test set. For each algorithm, we used a convergence tolerance of $\epsilon = .001$ and a Radial Basis Function kernel $\kappa(\mathbf{x}, \mathbf{y}) = \exp(\frac{-\|\mathbf{x}-\mathbf{y}\|^2}{2\sigma^2})$. As suggested in [24], we used $\sigma = 1/\sqrt{2}$ for the Two Spirals Data, $\sigma = 2$ for the Wisconsin Breast Cancer Data, and $\sigma = \sqrt{10}$ for the Adult-4a Data.

For each run we computed the following statistics: the number of kernel evaluations needed to train a SVM on the data; the percentage of support vectors found (number of support vectors found versus number of data vectors); and the percentage of the test set correctly classified. These statistics were selected to measure speed and quality of solution. The number of kernel evaluations performed was suggested as a measure of speed in [24] because the various algorithms considered (including NPA and SMO) were implemented using inner product caches. Thus, the cost of updating said caches (number of kernel evaluations performed) was the main computational expense.

The different statistics are plotted versus the solution margin of the resulting SVM for each algorithm. The statistics for the Two Spirals Data can be found in Figure 4.3; the statistics for the Wisconsin Breast Cancer Data can be found in Figure 4.4; and the statistics for the Adult-4a Data can be found in Figure 4.5.

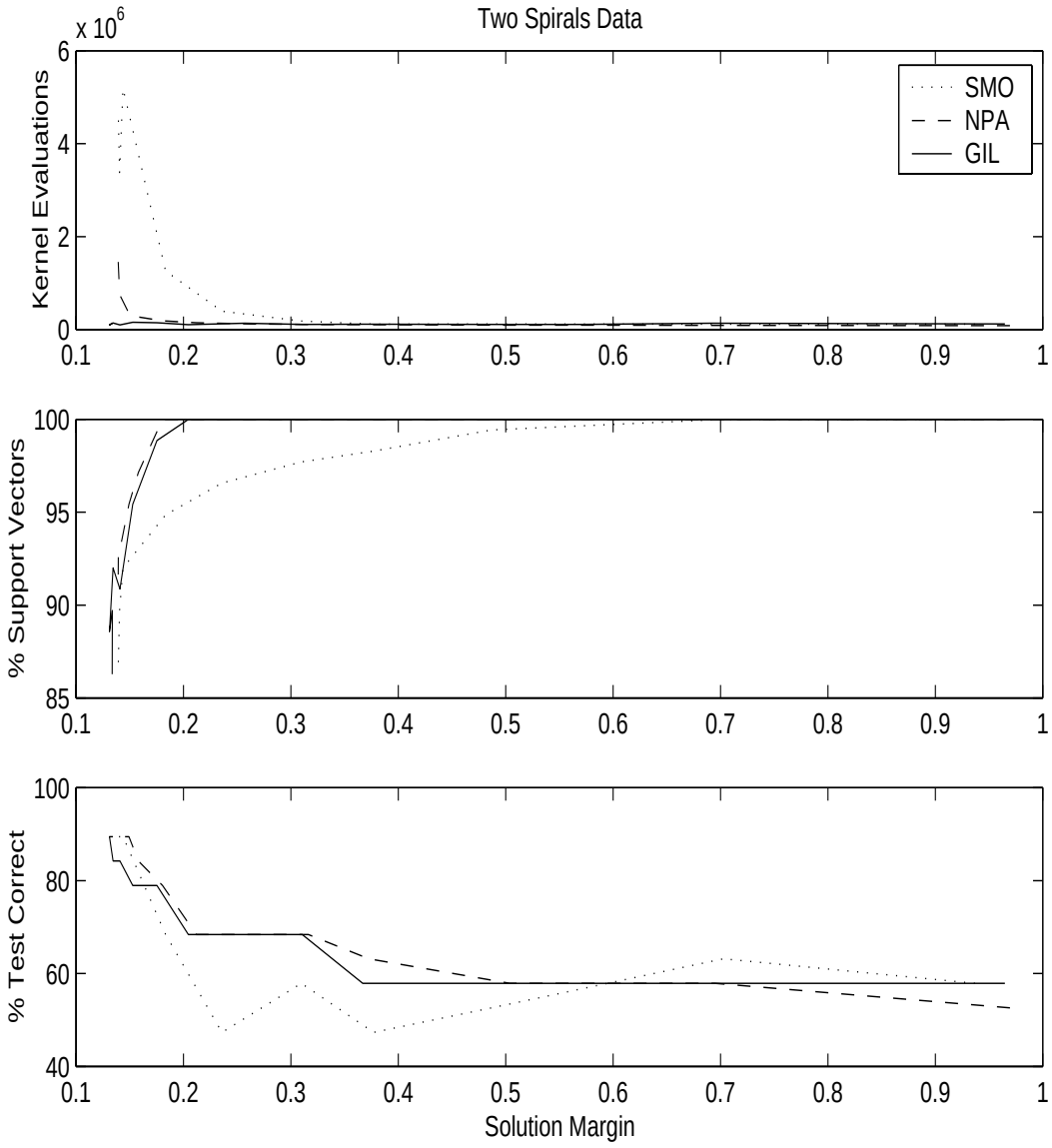

Figure 4.3: Two Spirals Data. Here we plot the performance statistics for SMO, NPA and Gilbert's Algorithm for SVMs on the Two Spirals Data. In each plot the horizontal axis gives the solution margin, dotted lines are used for SMO, dashed lines are used for NPA, and solid lines are used for Gilbert's Algorithm for SVMs. In the top plot (a) we see on the vertical axis the number of kernel evaluations performed, in the middle plot (b) we see the percentage of support vectors found, and in the bottom plot (c) we see the percentage of points in the test set correctly classified.

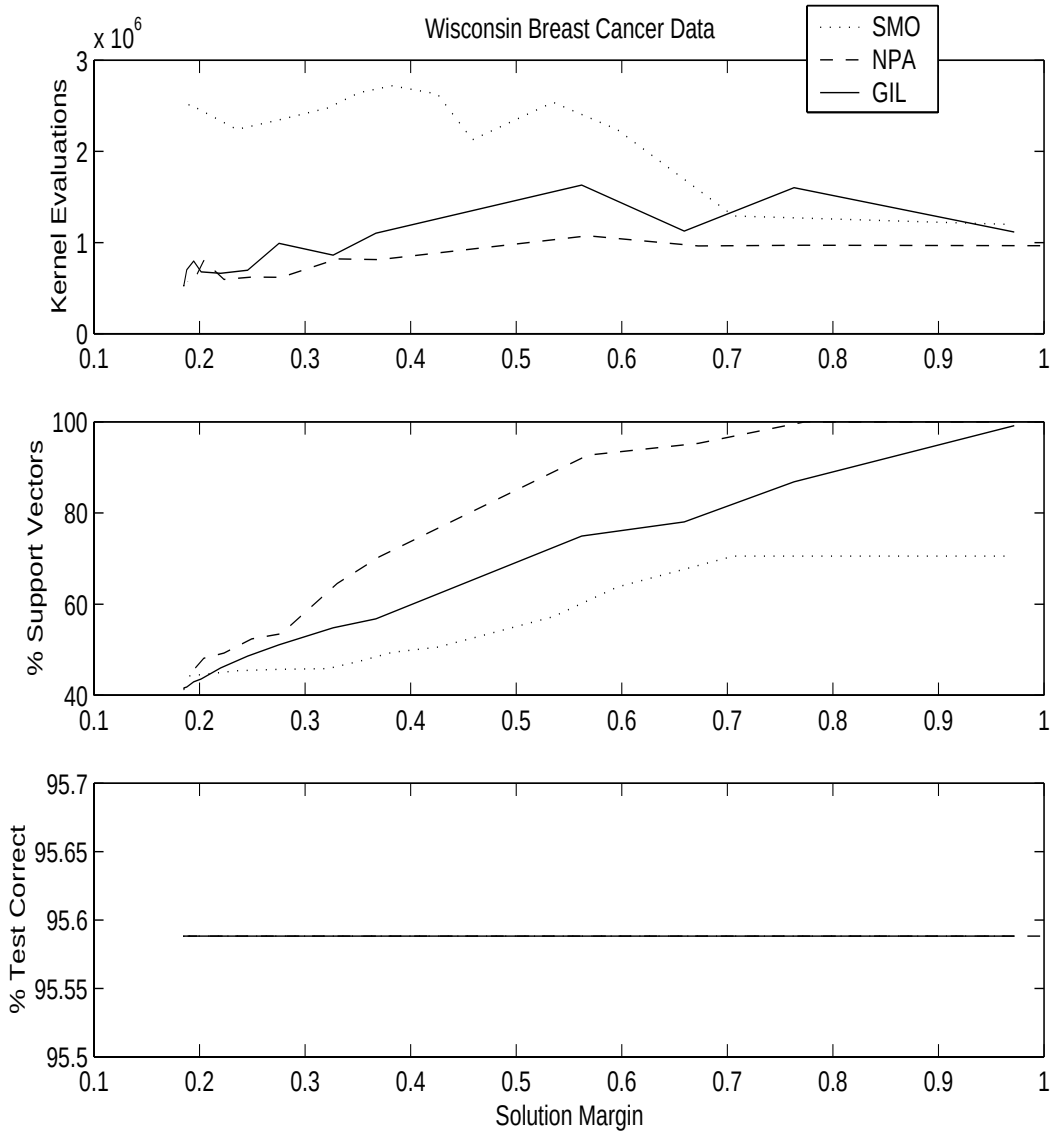

Figure 4.4: Wisconsin Breast Cancer Data. Here we plot the performance statistics for SMO, NPA and Gilbert's Algorithm for SVMs on the Wisconsin Breast Cancer Data. The data is presented as in Figure 4.3. The success rates on the test set are identical for all three algorithms.

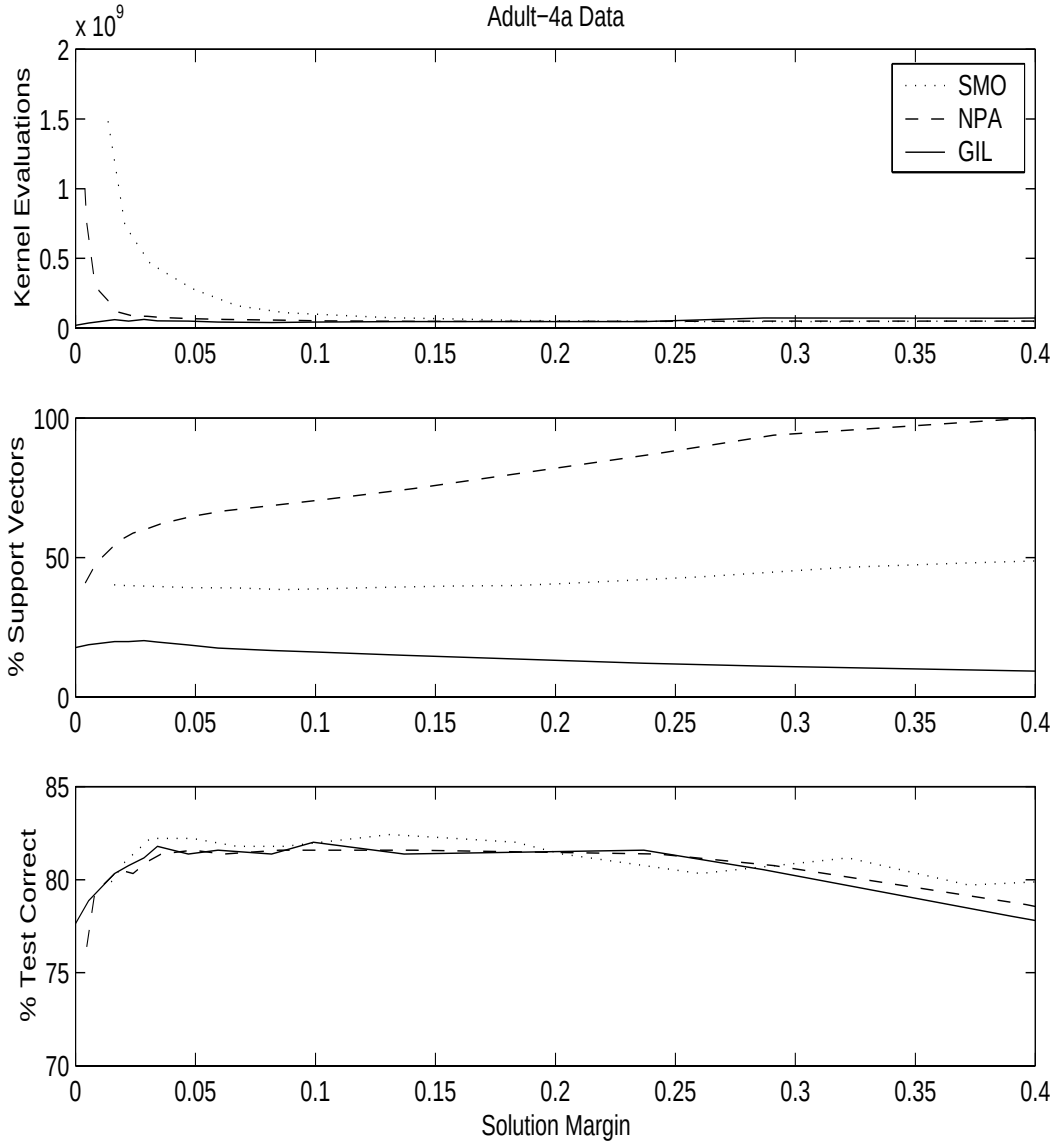

Figure 4.5: Adult-4a Data. Here we plot the performance statistics for SMO, NPA and Gilbert's Algorithm for SVMs on the Adult-4a Data.

It is interesting to note, as indicated by part (a) of Figures 4.3, 4.4, and 4.5, that our algorithm is similar in speed to both NPA and SMO for large margins (corresponding to small values of $\widetilde{C}$), and significantly faster than NPA and SMO for small margins (large values of $\widetilde{C}$). It is also interesting to note that our algorithm provides Support Vector Machines with representations which are similar (in terms of number of support vectors) to NPA and SMO in some cases (part (b) of Figures 4.3 and 4.4), but which can be much more efficient (fewer support vectors) in other cases (part (b) of Figure 4.5). Finally, we note that all three algorithms performed equally on the test set (part (c) of Figures 4.3, 4.4 and 4.5).

4.6 Conclusions

We must first acknowledge the similarity of our work to the work in [24]. The main theoretical foundation of our work (Theorems 4.2 and 4.4) was done in the summer of 1999, before we were aware of the same results in [24]. Gilbert's Algorithm was also considered in [24], but dismissed because of slow convergence in its final stages. We have, on the other hand, successfully applied Gilbert's Algorithm to Support Vector Machines via a simple modification. Our modification made an ailing algorithm into one of the fastest available for SVMs and might similarly accelerate existing algorithms.

Finally, Gilbert's Algorithm for Support Vector Machines has a number of advantages over both NPA and SMO. First, it is simpler (no heuristics) than SMO and much simpler than NPA. Second, Gilbert's Algorithm for SVMs runs almost as fast (in cpu time) coded in `MATLAB` as NPA in `Fortran` and SMO in `C++`. Thus, it may be faster than both NPA and SMO when coded in `C++`. (`MATLAB` is generally much slower than either `Fortran` or `C++`.) Third and last, the Support Vector Machines produced by Gilbert's Algorithm for SVMs are as accurate as those produced by NPA and SMO and may have more efficient representations (fewer support vectors).

Chapter 5

CONCLUSIONS

In this dissertation we proposed a solution to a problem in Pattern Classification and two additional solutions to problems in Support Vector Classification. In Chapter 2 we proposed a new method of feature selection for classification using Fisher's criterion and the Veronese map. In Chapter 3 we generalized our feature selection algorithm form Chapter 2 using the Veronese kernel and proposed a kernel selection algorithm for Support Vector Machines. Finally, in Chapter 4 we developed a new, fast algorithm for training Support Vector Machines. In this chapter we summarize more precisely our work. In Section 5.1 we recap the primary contributions of this dissertation, in Section 5.2 we give some possible extensions of our results, and in Section 5.3 we ask some questions raised during the course of this research.

5.1 Primary Contributions

The highlights of this research include

- The explanation and extension of Villars' method of feature selection in Section 2.6 using the Veronese map and Fisher's discriminant.
- The development of kernel Gram-Schmidt in Section 3.1.
- An algorithmic approach to both feature and kernel selection for Support Vector Machines.

- Theorem 4.2 in Section 4.1, a theorem on an optimal separation of convex sets, and the corresponding geometric solution of the linear separable Support Vector Machine Quadratic Programming problem (Theorem 4.4).
- A new angle convergence criterion for Gilbert's Algorithm in Section 4.2.
- The Two Point Algorithm in Appendix A.

5.2 Possible Extensions

Possible extensions of this work include

- The use of kernel Gram-Schmidt with other standard methods in linear algebra. Kernel Gram-Schmidt produces a matrix of nonlinearly preprocessed data. This matrix may be used in conjunction with Least Squares methods, the Singular Value Decomposition, linear regression, and other classification methods (e.g. linear perceptrons). This technique would offer an alternative to existing kernel methods, in particular kernel Principal Component Analysis [43], Support Vector Regression [46], and kernel Fisher Discriminant [31]. In particular, using kernel Gram-Schmidt would allow much larger data sets than kernel PCA and kernel Fisher Discriminant.
- Kernel selection using our kernel evaluation method in Chapter 3 combined with some type of gradient search algorithm. This would automate the process of kernel selection and most likely be more effective than our present exhaustive search.
- Use of the Two Point Algorithm in conjunction with the angle convergence criterion to find the point on a convex set closest to the origin. This method might be faster than the known methods for solving this problem in [17], [1], [32], and [54].

5.3 Open Questions

Some questions raised during the course of this research include

- Is there an effective way of running kernel Gram-Schmidt backwards? That is, how can we represent the results of an algorithm operating on the preprocessed data matrix in terms of our original data?
- Is kernel Gram-Schmidt numerically stable?
- How can we select C in the Support Vector Machine Quadratic Programming problem in Equation (1.3)? This question is addressed in [9] and [11].
- How does our kernel selection method compare to the fast modern algorithms for training Support Vector Machines (see Chapter 4)? In other words, would it be effective to use a fast modern SVM training algorithm to select kernels just by brute force (training many SVMs)? Or is our method still faster? Does our method yield more information about the final SVM?
- Can Theorems 4.2 and 4.4 in Chapter 4 be extended to work with the Support Vector Machine soft margin generalization (see Section 1.2.1)?
- How does the nonseparable SVM generalization in Section 4.4 compare to the standard soft margin generalization [10]?

Bibliography

[1] R. Barr. An efficient computational procedure for a generalized quadratic programming problem. *SIAM Journal of Control*, 7(3), 1969.

[2] M. Barros de Almeida. SMOBR – a SMO program for SVMs. http://www.litc.cpdee.ufmg.br/∼barros/svm/smobr, 2000. Computational Intelligence and Technology Laboratory.

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

[4] K. Bennett and E. J. Bredensteiner. Geometry in learning. In C. Gorini, editor, *Geometry at Work*, pages 132–145. Mathematical Association of America, Washington, D.C., 2000. Also at K. Bennett's Rensselaer Polytechnic Institute home page http://www.rpi.edu/∼bennek.

[5] C.L. Blake and C.J. Merz. UCI repository of machine learning databases. http://www.ics.uci.edu/∼mlearn/MLRepository.html, 1998. University of California, Irvine, Dept. of Information and Computer Sciences.

[6] B. E. Boser, I. M. Guyon, and V. N. Vapnik. A training algorithm for optimal margin classifiers. In D. Haussler, editor, *5th Annual ACM Workshop on COLT*, pages 144–152, Pittsburgh, PA, 1992. ACM Press.

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

[8] C. J. C. Burges. Geometry and invariance in kernel based methods. In B. Schölkopf, C. J. C. Burges, and A. J. Smola, editors, *Advances in Kernel Methods — Support Vector Learning*, pages 89–116, Cambridge, MA, 1999. MIT Press.

[9] O. Chapelle and V. Vapnik. Model selection for support vector machines. In Sara A. Solla, Todd K. Leen, and Klaus-Robert Müller, editors, *Advances in Neural Information Processing Systems 12*. Cambridge, Mass: MIT Press, 2000.

[10] C. Cortes and V. Vapnik. Support vector networks. *Machine Learning*, 20:273 – 297, 1995.

[11] N. Cristianini, C. Campbell, and J. Shawe-Taylor. Dynamically adapting kernels in support vector machines. In *Advances in Neural Information Processing Systems,*, volume 11, pages 204–210, 1999.

[12] M. Dash and H. Liu. Feature selection for classification. *Intelligent Data Analysis*, 1(3), 1997.

[13] R.O. Duda and P.E. Hart. *Pattern Classification and Scene Analysis*. Wiley Interscience, New York, 1973.

[14] T.-T. Frieß, N. Cristianini, and C. Campbell. The kernel adatron algorithm: A fast and simple learning procedure for support vector machines. In *15th Intl. Conf. Machine Learning*. Morgan Kaufmann Publishers, 1998.

[15] T.-T. Frieß and R. F. Harrison. Support vector neural networks: The kernel adatron with bias and soft margin. Technical Report AC&SE RR 725, University of Sheffield, England, 1998.

[16] K. Fukunaga. *Introduction to Statistical Pattern Recognition*. Academic Press, Inc., 1990.

[17] E. G. Gilbert. An iterative procedure for computing the minimum of a quadratic form on a convex set. *SIAM Journal of Control*, 4(1), 1966.

[18] A. G. Jackson, 1999. US Air Force Research Laboratory, Wright-Patterson Air Force Base, OH 45433-6523.

[19] T. Joachims. Text categorization with support vector machines: Learning with many relevant features. In Claire Nédellec and Céline Rouveirol, editors, *Proceedings of the European Conference on Machine Learning*, pages 137–142, Berlin, 1998. Springer.

[20] T. Joachims. Making large–scale SVM learning practical. In B. Schölkopf, C. J. C. Burges, and A. J. Smola, editors, *Advances in Kernel Methods — Support Vector Learning*, pages 169–184, Cambridge, MA, 1999. MIT Press.

[21] G. John, R. Kohavi, and K. Pfleger. Irrelevant features and the subset selection problem. In *Proc. of the 11th International Conference on Machine Learning ICML94*, pages 121–129, 1994.

[22] M. Kantrowitz. Carnegie Mellon Univeristy artificial intelligence repository. http://www.cs.cmu.edu/afs/cs/project/ai-repository/ai/areas/neural/bench/0.html, 1993. Carnegie Mellon University School of Computer Science.

[23] S. S. Keerthi. Sathiya Keerthi's home page. http://guppy.mpe.nus.edu.sg/~mpessk/npa.shtml, 1999. Dept. of Mechanical and Production Engineering, The National University of Singapore.

[24] S. S. Keerthi, S. K. Shevade, C. Bhattacharyya, and K. R. K. Murthy. A fast iterative nearest point algorithm for support vector machine classifier design. Technical Report TR-ISL-99-03, Indian Institute of Science, 1999.

[25] S. R. Lay. *Convex Sets and Their Applications*. Pure and Applied Mathematics. Wiley Interscience, New York, 1982.

[26] O. L. Mangasarian. *Nonlinear Programming*. Series in Systems Science. McGraw Hill, New York, 1969.

[27] O. L. Mangasarian and D. R. Musicant. Successive overrelaxation for support vector machines. *IEEE Transactions on Neural Networks*, 10:1032–1037, 1999.

[28] S. Martin, M. Kirby, and R. Miranda. Kernel/feature selection for support vector machines applied to materials design. In *IFAC Symposium on Artificial Intelligence in Real Time Control AIRTC-2000*, pages 29–34. Elsevier Science, Ltd., 2000. Budapest, Hungary, October 2-4, 2000.

[29] S. Martin, M. Kirby, and R. Miranda. Symmetric veronese classifiers with application to materials design. *Engineering Applications of Artificial Intelligence*, 13:513–520, 2000.

[30] `MATLAB`. MathWorks, Inc., Natick, Massachusetts, 1999.

[31] S. Mika, G. Rätsch, J. Weston, B. Schölkopf, and K.-R. Müller. Fisher discriminant analysis with kernels. In Y.-H. Hu, J. Larsen, E. Wilson, and S. Douglas, editors, *Neural Networks for Signal Processing IX*, pages 41–48. IEEE, 1999.

[32] B. F. Mitchell, V. F. Dem'yanov, and V. N. Malozemov. Finding the point of a polyhedron closest to the origin. *SIAM Journal of Control*, 12(1), 1974.

[33] W. Keith Nicholson. *Introduction to Abstract Algebra*. PWS-KENT Publishing, 1993.

[34] E. Osuna, R. Freund, and F. Girosi. An improved training algorithm for support vector machines. In J. Principe, L. Gile, N. Morgan, and E. Wilson, editors, *Neural Networks for Signal Processing VII — Proceedings of the 1997 IEEE Workshop*, pages 276 – 285, New York, 1997. IEEE.

[35] E. Osuna, R. Freund, and F. Girosi. Training support vector machines: An application to face detection. In *Proceedings of CVPR'97, Puerto Rico*, 1997.

[36] Yoh-Han Pao. Topological models of interacting systems and visualization (power point presentation). Case Western Reserve University, 1999.

[37] Yoh-Han Pao and Y. Zhao, 1999. Case Western Reserve University, Cleveland, OH 44106.

[38] J. Platt. John Platt's home page. http://www.research.microsoft.com/∼jplatt /smo.html, 1998. Microsoft Research.

[39] J. Platt. Fast training of support vector machines using sequential minimal optimization. In B. Schölkopf, C. J. C. Burges, and A. J. Smola, editors, *Advances in Kernel Methods — Support Vector Learning*, pages 185–208, Cambridge, MA, 1999. MIT Press.

[40] N. K. Sancheti and S. S. Keerthi. Computation of certain measures of proximity between convex polytopes: A complexity viewpoint. In *Proc. of IEEE International Conference on Robotics and Automation*, Nice, France, 1992.

[41] B. Schölkopf, C. Burges, and V. Vapnik. Extracting support data for a given task. In U. M. Fayyad and R. Uthurusamy, editors, *Proceedings, First International Conference on Knowledge Discovery & Data Mining*, Menlo Park, 1995. AAAI Press.

[42] B. Schölkopf, J. Shawe-Taylor, A. J. Smola, and R. C. Williamson. Generalization bounds via eigenvalues of the Gram matrix. Technical Report 99-035, NeuroCOLT, 1999. Short conference version: Kernel-dependent Support Vector error bounds, Ninth International Conference on Artificial Neural Networks, pp. 103 - 108, IEE Conference Publications No. 470, London.

[43] B. Schölkopf, A. Smola, and K.-R. Müller. Nonlinear component analysis as a kernel eigenvalue problem. *Neural Computation*, 10:1299–1319, 1998. Technical Report No. 44, 1996, Max Planck Institut für biologische Kybernetik, Tübingen.

[44] J. Schürmann. *Pattern Classification: A Unified View of Statistical and Neural Approaches*. John Wiley & Sons, 1996.

[45] I. R. Shafarevich. *Basic Algebraic Geometry 1*. Springer-Verlag, 1994.

[46] A. J. Smola and B. Schölkopf. A tutorial on support vector regression. NeuroCOLT Technical Report NC-TR-98-030, Royal Holloway College, University of London, UK, 1998. To appear in *Statistics and Computing, 2001*.

[47] A. J. Smola and B. Schölkopf. http://www.kernel-machines.org, 2000.

[48] S. Theodoridis and K. Koutroumbas. *Pattern Recognition*. Academic Press, 1999.

[49] L. Trefethen and D. Bau. *Numerical Linear Algebra*. SIAM, 1997.

[50] V. Vapnik. *Estimation of Dependences Based on Empirical Data [in Russian]*. Nauka, Moscow, 1979. (English translation: Springer Verlag, New York, 1982).

[51] V. Vapnik. *Statistical Learning Theory*. Wiley Interscience, New York, 1998.

[52] P. Villars. Interim report special project spc 98-4028 covering 15 march 1998 – 14 march 1999. Technical report, Material Phases Data Systems (MPDS), CH-6354, Vitznau, Switzerland, 1999.

[53] P. Villars. Is the mendeleev number the best elemental property parameter for the classification of ternary formers/nonformers? EP'99 presentation, Air Force Institute of Technology, WPAFB, Ohio, 1999.

[54] P. Wolfe. Finding the nearest point in a polytope. *Mathematical Programming*, 11:128–149, 1976.

[55] A. Zien, G. Rätsch, S. Mika, B. Schölkopf, T. Lengauer, and K.-R. Müller. Engineering support vector machine kernels that recognize translation initiation sites. *BioInformatics*, 2000. to appear.

Appendix A

TWO POINT ALGORITHM

While developing Gilbert's Algorithm for Support Vector Machines in Chapter 4 we discovered a similar algorithm which we call the Two Point Algorithm. To describe the Two Point Algorithm we use the notation we used to describe Gilbert's Algorithm in Section 4.2. Specifically, we set

$$[\mathbf{a},\mathbf{b}]^* = \begin{cases} \mathbf{a} & \text{if} \quad -(\mathbf{a},\mathbf{b}-\mathbf{a}) \leq 0 \\ \mathbf{a} + \frac{-(\mathbf{a},\mathbf{b}-\mathbf{a})}{\|\mathbf{b}-\mathbf{a}\|^2}(\mathbf{b}-\mathbf{a}) & \text{if} \quad 0 < -(\mathbf{a},\mathbf{b}-\mathbf{a}) < \|\mathbf{b}-\mathbf{a}\|^2 \\ \mathbf{b} & \text{if} \quad \|\mathbf{b}-\mathbf{a}\|^2 \leq -(\mathbf{a},\mathbf{b}-\mathbf{a}) \end{cases}$$

for $\mathbf{a},\mathbf{b} \in \mathbb{R}^n$. In addition, we say that

$$\begin{array}{ll} \mathbf{a} < \mathbf{b} & \text{if} \quad \mathbf{a} \neq \mathbf{b} \text{ and } [\mathbf{a},\mathbf{b}]^* = \mathbf{a} \\ \mathbf{a} \sim \mathbf{b} & \text{if} \quad \mathbf{a} \neq \mathbf{b} \text{ and } [\mathbf{a},\mathbf{b}]^* \neq \mathbf{a},\mathbf{b} \\ \mathbf{a} > \mathbf{b} & \text{if} \quad \mathbf{a} \neq \mathbf{b} \text{ and } [\mathbf{a},\mathbf{b}]^* = \mathbf{b}. \end{array}$$

and we use notation such as

$$\mathbf{a} \leq \mathbf{b} \quad \text{if} \quad \mathbf{a} = \mathbf{b} \text{ or } \mathbf{a} < \mathbf{b}.$$

With these definitions, $[\mathbf{a},\mathbf{b}]^*$ is again the point on the line segment from $\mathbf{a}$ to $\mathbf{b}$ closest to the origin, $\mathbf{a} < \mathbf{b}$ if $[\mathbf{a},\mathbf{b}]^* = \mathbf{a}$, and $\mathbf{a} \sim \mathbf{b}$ if $[\mathbf{a},\mathbf{b}]^*$ is between $\mathbf{a}$ and $\mathbf{b}$. Writing $S = \{\mathbf{s}_k\}_{k=1}^p$ and denoting the convex hull of S by C_S, we have the following lemmas.

Lemma A.1. *If* $\mathbf{a} < \mathbf{b}$ *then* $\|\mathbf{a}\| < \|\mathbf{b}\|$.

Proof. By hypothesis we have $\mathbf{a} \neq \mathbf{b}$ and $-(\mathbf{a}, \mathbf{b}-\mathbf{a}) \leq 0$. If $\mathbf{a}$ and $\mathbf{b}$ are linearly independent then using the Cauchy-Schwarz inequality we get

$$\|\mathbf{a}\| \leq \frac{(\mathbf{b}, \mathbf{a})}{\|\mathbf{a}\|} \leq \frac{\|\mathbf{b}\|\|\mathbf{a}\|}{\|\mathbf{a}\|} = \|\mathbf{b}\|.$$

If $\mathbf{a} = \lambda\mathbf{b}$ for some $\lambda \in \mathbb{R}$ then we have

$$\begin{gathered} -(\lambda\mathbf{b}, \mathbf{b}-\lambda\mathbf{b}) \leq 0 \\ -\lambda\|\mathbf{b}\|^2 + \lambda^2\|\mathbf{b}\|^2 \leq 0 \\ \lambda(\lambda-1) \leq 0 \\ 0 \leq \lambda \leq 1. \end{gathered}$$

Since $\mathbf{a} \neq \mathbf{b}$ we have $0 \leq \lambda < 1$ so that $\|\mathbf{a}\| = \lambda\|\mathbf{b}\| < \|\mathbf{b}\|$. *Q.E.D.*

Lemma A.2. *Suppose $\mathbf{s}^* \in C_S$. Then the following are equivalent:*

- *(a)* $\mathbf{s}^*$ *is the point in C_S closest to the origin, i.e $\|\mathbf{s}^*\| \leq \|\mathbf{s}\|$ for all $\mathbf{s} \in C_S$*
- *(b)* $\mathbf{s}^* \leq \mathbf{s}$ *for all $\mathbf{s} \in C_S$*
- *(c)* $\mathbf{s}^* \leq \mathbf{s}_k$ *for $k = 1, \ldots, p$.*

Proof. (a) $\Rightarrow$ (b). Fix $\mathbf{s} \in C_S \setminus \{\mathbf{s}^*\}$. Let $\lambda_0 = -\frac{(\mathbf{s}^*,\ \mathbf{s}-\mathbf{s}^*)}{\|\mathbf{s}-\mathbf{s}^*\|^2}$ and $\mathbf{t} = (1-\lambda_0)\mathbf{s}^* + \lambda_0\mathbf{s}$. If we assume that $\lambda_0 > 0$ then we can show using (a) that $\mathbf{t} \in C_S$ but that $\|\mathbf{t}\| < \|\mathbf{s}^*\|$. (See the proof of Theorem 4.2.) Thus $\lambda_0 \leq 0$ and $\mathbf{s}^* < \mathbf{s}$.

(c) $\Rightarrow$ (a). Let $H = \{\mathbf{x} : (\mathbf{s}^*, \mathbf{x}-\mathbf{s}^*) \geq 0\}$. Note that H is a closed half-space containing S (since $\mathbf{s}^* \leq \mathbf{s}_k$ for $k = 1, \ldots, p$). Since H is convex we see that $C_S \subset H$. Further, $\mathbf{h} \in H \setminus \{\mathbf{s}^*\}$ implies that $\mathbf{s}^* < \mathbf{h}$ so by Lemma A.1 $\|\mathbf{s}^*\| < \|\mathbf{h}\|$, i.e. $\|\mathbf{s}^*\| < \|\mathbf{h}\|$ for every $\mathbf{h} \in H \setminus \{\mathbf{s}^*\}$. This implies that $\|\mathbf{s}^*\| < \|\mathbf{s}\|$ for all $\mathbf{s} \in C_S \setminus \{\mathbf{s}^*\}$ since $C_S \setminus \{\mathbf{s}^*\} \subset H \setminus \{\mathbf{s}^*\}$. *Q.E.D.*

Lemma A.3. *Fix $\mathbf{s}_0 \in \mathbb{R}^n$. Then the map*

$$\mathbf{t} \mapsto [\mathbf{t}, \mathbf{s}_0]^*$$

is continuous on $\mathbb{R}^n$.

Proof. Denote the map $\mathbf{t} \mapsto [\mathbf{t}, \mathbf{s}_0]^*$ by f. Assume $\mathbf{s}_0 \neq \mathbf{0}$ and define

$$\begin{aligned} H &= \{\mathbf{x} : (\mathbf{s}_0, \mathbf{x} - \mathbf{s}_0) \geq 0\} \\ B &= \{\mathbf{x} : \|\tfrac{\mathbf{s}_0}{2} - \mathbf{x}\| \leq \|\tfrac{\mathbf{s}_0}{2}\|\} \\ E &= \mathbb{R}^n \setminus (H^\circ \cup B^\circ), \end{aligned}$$

where H° and B° denote the interiors of H and B. We note that H, B, and E are closed and that $\mathbb{R}^n = H \cup B \cup E$. To show that f is continuous it is enough to show that $f|_H, f|_B$, and $f|_E$ are continuous.

Suppose $\mathbf{t} \in H \setminus \{\mathbf{s}_0\}$. Then $(\mathbf{s}_0, \mathbf{t} - \mathbf{s}_0) \geq 0$ which implies that

$$\begin{gathered} (\mathbf{t}, \mathbf{s}_0) \geq \|\mathbf{s}_0\|^2 \\ -(\mathbf{t}, \mathbf{s}_0) + \|\mathbf{t}\|^2 \geq \|\mathbf{s}_0\|^2 - 2(\mathbf{t}, \mathbf{s}_0) + \|\mathbf{t}\|^2 \\ -(\mathbf{t}, \mathbf{s}_0 - \mathbf{t}) \geq \|\mathbf{s}_0 - \mathbf{t}\|^2. \end{gathered}$$

Thus $f(\mathbf{t}) = [\mathbf{t}, \mathbf{s}_0]^* = \mathbf{s}_0$, and since $f(\mathbf{s}_0) = \mathbf{s}_0$ we see that $f|_H \equiv \mathbf{s}_0$ is continuous.

Next suppose $\mathbf{t} \in B \setminus \{\mathbf{s}_0\}$. Then $\|\frac{\mathbf{s}_0}{2} - \mathbf{t}\| \leq \|\frac{\mathbf{s}_0}{2}\|$ which implies that

$$\begin{gathered} \|\tfrac{\mathbf{s}_0}{2} - \mathbf{t}\|^2 \leq \|\tfrac{\mathbf{s}_0}{2}\|^2 \\ -(\mathbf{t}, \mathbf{s}_0) + \|\mathbf{t}\|^2 \leq 0. \end{gathered}$$

Hence $f(\mathbf{t}) = [\mathbf{t}, \mathbf{s}_0]^* = \mathbf{t}$ and again $f(\mathbf{s}_0) = \mathbf{s}_0$ so that $f|_B(\mathbf{t}) = \mathbf{t}$ is continuous.

Finally, suppose $\mathbf{t} \in E \setminus \{\mathbf{s}_0\}$. Then $(\mathbf{s}_0, \mathbf{t} - \mathbf{s}_0) \leq 0$ and $\|\frac{\mathbf{s}_0}{2} - \mathbf{t}\| \geq \|\frac{\mathbf{s}_0}{2}\|$, which imply $0 \leq \lambda_0 = \frac{-(\mathbf{t}, \mathbf{s}_0 - \mathbf{t})}{\|\mathbf{s}_0 - \mathbf{t}\|^2} \leq 1$ as previously. Thus $f(\mathbf{t}) = [\mathbf{t}, \mathbf{s}_0]^* = (1 - \lambda_0)\mathbf{t} + \lambda_0 \mathbf{s}_0$ is continuous on $E \setminus \{\mathbf{s}_0\}$. To see that $f|_E$ is continuous at $\mathbf{s}_0$ we note that

$$\begin{aligned} \|f(\mathbf{s}_0) - f(\mathbf{t})\| &= \|\mathbf{s}_0 - (\lambda_0(\mathbf{s}_0 - \mathbf{t}) + \mathbf{t})\| \\ &= \|(\mathbf{s}_0 - \mathbf{t})(1 - \lambda_0)\| \\ &= \|\mathbf{s}_0 - \mathbf{t}\||1 - \lambda_0| \\ &\leq \|\mathbf{s}_0 - \mathbf{t}\|. \end{aligned}$$

We conclude that $\mathbf{t} \mapsto [\mathbf{t}, \mathbf{s}_0]^*$ is continuous as claimed. *Q.E.D.*

These lemmas assist in the proof of the following theorem, which we call the Two Point Algorithm.

Theorem A.4. *Let* $\mathbf{z} = (\mathbf{s}_1, \ldots, \mathbf{s}_p, \mathbf{s}_1, \ldots, \mathbf{s}_p, \mathbf{s}_1, \ldots, \mathbf{s}_p, \ldots)$, *and define* $(\mathbf{w}_k)_{k=1}^\infty$ *by* $\mathbf{w}_1 = \mathbf{z}_1, \mathbf{w}_k = [\mathbf{w}_{k-1}, \mathbf{z}_k]^*$ *for* $k \geq 2$. *Then* $\lim_{k \to \infty} \mathbf{w}_k = \mathbf{s}^*$, *where* $\mathbf{s}^*$ *is the closest point in* C_S *to the origin.*

Proof. Since $\mathbf{w}_k \in C_S$ for all k the sequence $(\mathbf{w}_k)_{k=1}^{\infty}$ is bounded so by the Bolzano-Weierstrass Theorem there exists a subsequence $(\mathbf{t}_j = \mathbf{w}_{k_j})_{j=1}^{\infty}$ converging to some $\mathbf{t} \in \mathbb{R}^n$. In fact, $\mathbf{t} \in C_S$ since C_S is closed.

Case I: $\mathbf{t} = \mathbf{s}^*$. By Lemma A.1, $\|\mathbf{w}_1\| \geq \|\mathbf{w}_2\| \geq \cdots \geq \|\mathbf{s}^*\|$ so that $(\|\mathbf{w}_k\|)_{k=1}^{\infty}$ converges. Since $\mathbf{w}_{k_j} \to \mathbf{t} = \mathbf{s}^*$ as $j \to \infty$ we see that $\|\mathbf{w}_k\| \to \|\mathbf{s}^*\|$ as $k \to \infty$. Now combining Lemma A.2(b) and the Cauchy-Schwarz inequality we get

$$\|\mathbf{s}^*\|^2 \leq (\mathbf{s}^*, \mathbf{w}_k) \leq \|\mathbf{s}^*\|\|\mathbf{w}_k\|$$

so that $(\mathbf{s}^*, \mathbf{w}_k) \to \|\mathbf{s}^*\|^2$ as $k \to \infty$ and hence

$$\|\mathbf{w}_k - \mathbf{s}^*\|^2 = \|\mathbf{w}_k\|^2 - 2(\mathbf{s}^*, \mathbf{w}_k) + \|\mathbf{s}^*\|^2 \to 0.$$

as $k \to \infty$, or $\lim_{k\to\infty} \mathbf{w}_k = \mathbf{s}^*$.

Case II: $\mathbf{t} \neq \mathbf{s}^*$. Let $(\mathbf{g}_m = \mathbf{s}_{i_m})_{m=1}^{q}$ be the subsequence of $(\mathbf{s}_i)_{i=1}^{p}$ of terms $\mathbf{s}_{i_m}$ with $\mathbf{t} \leq \mathbf{s}_{i_m}$, and let $(\mathbf{h}_r = \mathbf{s}_{i_r})_{r=1}^{p-q}$ be the subsequence of $(\mathbf{s}_i)_{i=1}^{p}$ of terms $\mathbf{s}_{i_r}$ with $\mathbf{t} \gtrsim \mathbf{s}_{i_r}$. Note that $\{\mathbf{g}_m\} \cap \{\mathbf{h}_r\} = \emptyset$ and $\{\mathbf{g}_m\} \cup \{\mathbf{h}_r\} = S$ as sets, and that $\{\mathbf{h}_r\} \neq \emptyset$ since $\|\mathbf{s}^*\| < \|\mathbf{t}\|$ so by Lemma A.2(c) there exists $\mathbf{s}_{i_0} \in S$ with $\mathbf{t} \gtrsim \mathbf{s}_{i_0}$.

For $r = 1, \ldots, p-q$ define

$$\begin{aligned} \mathbf{z}_r &= [\mathbf{t}, \mathbf{h}_r]^*, \\ \epsilon_r &= \|\mathbf{t}\| - \|\mathbf{z}_r\| > 0. \end{aligned}$$

Let $\epsilon = \min\{\epsilon_1, \ldots, \epsilon_{p-q}\}$. Since the maps $\mathbf{x} \mapsto [\mathbf{x}, \mathbf{h}_r]^*$ are continuous (Lemma A.3) with $[\mathbf{t}, \mathbf{h}_r]^* = \mathbf{z}_r$ for each r we have $\delta_r > 0$ such that

$$\|[\mathbf{x}, \mathbf{h}_r]^* - \mathbf{z}_r\| < \epsilon$$

when

$$\|\mathbf{x} - \mathbf{t}\| < \delta_r$$

for $r = 1, \ldots, p-q$. Let $\delta = \min\{\delta_1, \ldots, \delta_{p-q}\}$ and note that $\mathbf{x} \in B_\epsilon(\mathbf{z}_r)$ implies $\|\mathbf{x}\| < \|\mathbf{t}\|$, where we use $B_\epsilon(\mathbf{z}_r) = \{\mathbf{x} \in \mathbb{R}^n : \|\mathbf{x} - \mathbf{z}_r\| < \epsilon\}$ to denote the open ϵ ball centered about $\mathbf{z}_r$.

Next consider the sequence

$$\mathbf{v} = (\mathbf{v}_1, \ldots, \mathbf{v}_{2q-1}) = (\mathbf{g}_1, \ldots, \mathbf{g}_q, \mathbf{g}_1, \ldots, \mathbf{g}_{q-1}).$$

Starting backwards with $\mathbf{v}_{2q-1}$ we get $0 < \gamma_{2q-1} \leq \delta$ such that

$$\|[\mathbf{x}, \mathbf{v}_{2q-1}]^* - \mathbf{t}\| < \delta$$

when

$$\|\mathbf{x} - \mathbf{t}\| < \gamma_{2q-1}.$$

Continuing on to $\mathbf{v}_1$ we get $0 < \gamma_1 \leq \gamma_2 \leq \cdots \leq \gamma_{2q-1} \leq \delta$ with

$$\|[\mathbf{x}, \mathbf{v}_i]^* - \mathbf{t}\| < \gamma_{i+1}$$

when

$$\|\mathbf{x} - \mathbf{t}\| < \gamma_i$$

for $i = 1, \ldots, 2q-2$. If $\{\mathbf{g}_m\} = \emptyset$ let $\gamma_1 = \delta$.

Finally, pick N such that $\|\mathbf{t}_j - \mathbf{t}\| < \gamma_1$ for $j \geq N$ and consider the subsequence

$$\begin{array}{cccl} (1) & \mathbf{w}_{(k_N)+1} & = & [\mathbf{w}_{k_N}, \mathbf{z}_{(k_N)+1}]^* \\ (2) & \mathbf{w}_{(k_N)+2} & = & [\mathbf{w}_{(k_N)+1}, \mathbf{z}_{(k_N)+2}]^* \\ & & \vdots & \\ (q) & \mathbf{w}_{(k_N)+q} & = & [\mathbf{w}_{(k_N)+q-1}, \mathbf{z}_{(k_N)+q}]^* \\ (q+1) & \mathbf{w}_{(k_N)+q+1} & = & [\mathbf{w}_{(k_N)+q}, \mathbf{z}_{(k_N)+q+1}]^* \end{array}$$

of $(\mathbf{w}_k)_{k=1}^{\infty}$.

At step (1), $\mathbf{z}_{(k_N)+1} = \mathbf{s}_{i_0}$ for some $1 \leq i_0 \leq p$. If $\mathbf{s}_{i_0}$ is a term of $(\mathbf{g}_m)_{m=1}^{q}$, say $\mathbf{s}_{i_0} = \mathbf{g}_{m_0} = \mathbf{v}_{m_0}$, then $\mathbf{w}_{(k_N)+1} \in B_{\gamma_{m_0+1}}(\mathbf{t})$ since $\mathbf{w}_{k_N} \in B_{\gamma_1}(\mathbf{t}) \subset B_{\gamma_{m_0}}(\mathbf{t})$. If $\mathbf{z}_{(k_N)+2}$ is also a term of $\mathbf{v}$, i.e. $\mathbf{z}_{(k_N)+2} = \mathbf{v}_{m_0+1}$, then $\mathbf{w}_{(k_N)+2}$ in step (2) is contained in $B_{\gamma_{m_0+2}}(\mathbf{t})$.

Eventually, however, $\mathbf{z}_{(k_N)+c}$ is a term of $(\mathbf{h}_r)_{r=1}^{p-q}$ since $\{\mathbf{h}_r\} \neq \emptyset$ implies that q is the maximum length of a consecutive subsequence of $\mathbf{v}$ consisting of consecutive

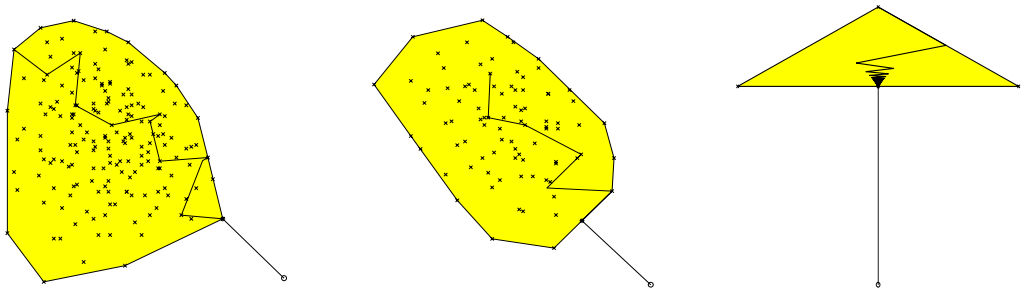

Figure A.1: Two Point Algorithm. Here we illustrate the behavior of the two point algorithm. On the left (a) $\mathbf{s}^* \in S$, and the algorithm, whose progress is tracked by the jagged line, converges to $\mathbf{s}^*$ in at most p steps (guaranteed). In the middle (b), $\mathbf{s}^* \notin S$ but the algorithm still converges to $\mathbf{s}^*$ (in less than p steps for this example — not guaranteed). On the right (c), the algorithm converges only asymptotically to $\mathbf{s}^*$.

terms in $\mathbf{z}$. We choose the smallest c in $1, \ldots, q+1$ such that $\mathbf{z}_{(k_N)+c}$ is a term of $(\mathbf{h}_r)_{r=1}^{p-q}$, say $\mathbf{z}_{(k_N)+c} = \mathbf{h}_{r_0} = \mathbf{s}_{i_{r_0}}$. Then we have $\mathbf{w}_{(k_N)+c-1} \subset B_\delta(\mathbf{t})$ so that

$$\mathbf{w}_{(k_N)+c} = \left[\mathbf{w}_{(k_N)+c-1}, \mathbf{z}_{(k_N)+c}\right]^*$$

is contained in $B_\epsilon(\mathbf{z}_{r_0})$. This implies that

$$\|\mathbf{t}\| > \|\mathbf{w}_{(k_N)+c}\| \geq \|\mathbf{w}_{(k_N)+c+1}\| \geq \ldots,$$

which implies that

$$\|\mathbf{t}\| > \|\mathbf{t}_{N+c}\| \geq \|\mathbf{t}_{N+c+1}\| \geq \ldots,$$

which contradicts the fact the $\mathbf{t}_j \to \mathbf{t}$. In other words, case II ($\mathbf{t} \neq \mathbf{s}^*$) never happens. *Q.E.D.*

In some instances, the Two Point Algorithm (the sequence in Theorem A.4) will converge in a finite number of steps. In particular this will occur if $\mathbf{s}^* \in S$. If $\mathbf{s}^* \notin S$ finite convergence may still occur, but it is not guaranteed. Figure A.1 illustrates the convergence of the two point algorithm in a variety of situations.

Figure A.1 also shows the similarity of the Two Point Algorithm to Gilbert's Algorithm (compare Figure 4.2). In particular, we note that both the Two Point

Algorithm and Gilbert's Algorithm become mired in the asymptotic convergence in Figure A.1(c). In that situation both algorithms also converge at a similar rate: they are both $O(p)$ (recall $S = \{\mathbf{s}_k\}_{k=1}^{p}$) for a single step.

Of course, the Two Point Algorithm and Gilbert's Algorithm are different as well. The main difference being that the Two Point Algorithm progresses randomly while Gilbert's Algorithm does not. Another difference is the cost of the initial progress of the two algorithms. Specifically, the Two Point Algorithm is less costly for its initial progress than Gilbert's Algorithm (the Two Point Algorithm is $O(1)$ for its initial steps while Gilbert's Algorithm is $O(p)$ for every step).

Appendix B

MATLAB CODE

Here we include the MATLAB (ver 5.3.1 [30]) code which was used to implement the various algorithms presented in this dissertation.

B.1 Veronese Map

```
function V = Veronese(X,d)
%VERONESE Veronese map
%  V = VERONESE (X,D) returns a matrix V whose columns are
%  the Veronese images of degree D of the columns of X.

[n,m] = size(X);
N = prod(2:n+d)/(prod(2:n)*prod(2:d)) - 1;
V = zeros(N,m);
c = [ones(1,d),n+1];
lc = d;
j = 1;

% c cycles through the monomials of degree d down to degree 1
for i = 1:N
  V(i,:) = prod(X(c(1:lc),:),1);
  if c(j) < n
    c(j) = c(j)+1;
  elseif c(j+1) < n-1
    c = [(c(j+1)+1)*ones(1,j+1),c(j+2:lc+1)];
    j = 1;
  elseif c(j+1)<n
    j = j+1;
    c(j) = c(j)+1;
  else
    c = [ones(1,j-1),n+1];
    lc = j-1;
```

```
    j = 1;
  end
end
```

B.2 Fisher's Discriminant

```
function [J,a] = Fisher (beta,gamma)
%FISHER calculates Fisher's criterion
%  [J,A] = FISHER (BETA,GAMMA) maximizes Fisher's
%  criterion.  Inputs: BETA is the X class with data points as
%  columns, and GAMMA is the Y class.  Outputs: J is the maximum
%  value of Fisher's criterion function, and A is the vector
%  normal to Fisher's hyperplane.

N = size(beta,2);
M = size(gamma,2);

S1 = (beta*beta' - (sum(beta,2)*sum(beta,2)')/N)/(N-1);
S2 = (gamma*gamma' - (sum(gamma,2)*sum(gamma,2)')/M)/(M-1);
Sc = S1 + S2;

m1 = sum(beta,2)/N;
m2 = sum(gamma,2)/M;
b = m1-m2;

a = Sc\b;
Sca = Sc*a;

atSba = (sum(a'*beta)/N - sum(a'*gamma)/M)^2;
atSca = a'*Sca;

J = atSba/atSca;
```

B.3 Kernel Gram-Schmidt

```
function Xtilde = kerGS (X,stol,mtol,ker,kp1,kp2,maxdim)
%KERGS performs implicit Gram-Schmidt via kernel.
%  XTILDE=KERGS(X,STOL,MTOL,KER,KP1,KP2,MAXDIM) finds the coeffs of
%  the data X with respect to an orthonormal basis implicitly
%  calculated using the kernel KER.  Inputs: data matrix X with
%  data points as columns, relative error STOL for subspace
%  inclusion (recommend .001), machine precision tolerance MTOL
%  (recommend 6), inline kernel KER with two parameters KP1, KP2,
%  and maximum allowed dimension MAXDIM of problem.  Outputs a data
```

```
%  matrix XTILDE with columns containing the coefficients of the
%  data in the orthonormal basis (dim is #rows).

n = size(X,2);

Xtilde = zeros(maxdim+1,n);
PivotIndices = zeros(maxdim,1);
PivotValues = zeros(maxdim,1);

c = (1-stol)^2;

dimtest = zeros(n,1);

d=1;
for i = 1:n
  if c*ker(X(:,i),X(:,i),kp1,kp2) > Xtilde(1:d,i)'*Xtilde(1:d,i)

    for j = 1:n
      dimtest(j) = sqrt(abs(ker(X(:,j),X(:,j),kp1,kp2) - ...
   Xtilde(1:d,j)'*Xtilde(1:d,j)));
    end

    [PivotValues(d),bestdim] = max(dimtest);
    PivotIndices(d) = bestdim;

    if d > mtol
      if log(PivotValues(d)) > log(PivotValues(d-mtol))
        d = d - mtol;
        break
      end
    end

    d=d+1;
    for j = 1:n
      Xtilde(d,j) = (ker(X(:,bestdim),X(:,j),kp1,kp2) - ...
  Xtilde(1:d-1,bestdim)'*Xtilde(1:d-1,j))/ ...
  dimtest(bestdim);
    end

    if d > maxdim
      break
    end

  end
```

```
end

Xtilde = Xtilde(2:d,:);
```

B.4 Gilbert's Algorithms for SVMs

```
function [w,b] = trainSVM(X,Y,ker,e,Ctilde)
%TRAINSVM trains support vector machines
%  [W,B] = TRAINSVM(X,Y,KER,E,CTILDE) trains a SVM using
%  Gilbert's Algorithm for Support Vector Machines.
%
%  INPUTS: X -- data matrix with points as columns
%          Y -- row vector containing data labels (1 or -1)
%          KER -- inline kernel function
%          E -- angle convergence criterion
%          CTILDE -- Friess' constant
%
%  OUTPUTS: W -- the point sstar closest to the origin
%                (the normal to the maximal margin hyperplane)
%           B -- bias: separating hyperplane has form x.w + b = 0
%           report -- time, flops, # kernel evaluations
%
%  NOTES: (1) KER takes two matrices of the same size
%             with points as columns and returns a row vector
%             of kernels.  The first entry of the vector is the
%             kernel of the 1st column of each input matrix, etc.
%         (2) W is (alpha1,...,alphan,-beta1, ...
%             -betam), where n = #class1, m = #class2.
%         (3) the points WK in Gilbert's Algorithm are kept
%             internally as structs, where the fields are:
%               .alphabeta -- as in (1),
%               .norm2 -- the norm^2 of WK,
%               .nonzeros -- a vector of length n+m with
%                1's where nonzero entries of WK occur.
%         (4) CACHE is an internally maintained vector which
%             contains the kernel (inner) products of the points
%             WK in Gilbert's Algorithm with the points in X.

st = cputime;
flops(0);

class1 = find(Y==1);
class2 = find(Y==-1);
```

```
n = length(class1);
m = length(class2);
N = n + m;

i = class1(round(rand(1)*(n-1))+1);
j = class2(round(rand(1)*(m-1))+1);

wk.alphabeta = zeros(1,N);
wk.alphabeta([i j]) = [1 -1];
wk.norm2 = ker(X(:,i),X(:,i)) - 2*ker(X(:,i),X(:,j)) + ...
           ker(X(:,j),X(:,j));
wk.norm2 = wk.norm2 + 2/Ctilde;
wk.nonzeros = zeros(1,N);
wk.nonzeros([i j]) = [1 1];
kerevals = 3;

cache = ker(X,repmat(X(:,i),1,N))-ker(X,repmat(X(:,j),1,N));
cache(i) = cache(i) + 1/Ctilde;
cache(j) = cache(j) - 1/Ctilde;
kerevals = kerevals + 2*N;

[wk,avgwk,cache,avgcache,CV,gilevals] = ...
        gilbert(X,class1,class2,wk,[i;j],ker,cache,Ctilde);
kerevals = kerevals + gilevals;

fprintf('.');

solution = 0;
while ~solution

  [wk,avgwksol,cache,avgsolcache,CV,gilevals] = ...
        gilbert(X,class1,class2,wk,CV,ker,cache,Ctilde);
  kerevals = kerevals + gilevals;
  fprintf('.');

  avgwknz = find(avgwk.nonzeros);

  coserr = avgwk.alphabeta(avgwknz)*(avgsolcache(avgwknz))'/...
        (sqrt(avgwk.norm2*avgwksol.norm2));
  fprintf(' %f\n',coserr);

  if coserr >= 1 - e
    solution = 1;
  else
```

```
    avgwk = avgwksol;
  end

end

w = avgwksol.alphabeta;

b = -(min(avgsolcache(class1)) + max(avgsolcache(class2)))/2;

fprintf('Report\n');
fprintf('------\n');
fprintf('Time: %f seconds\n',cputime-st);
fprintf('Flops: %d\n',flops);
fprintf('Kernel Evaluations: %d\n',kerevals);


function [wk,avgwk,cache,avgcache,CV,kerevals] = ...
        gilbert (X,class1,class2,wk,CV,ker,cache,Ctilde)
%GILBERT performs steps in Gilbert's algorithm
%  [WK,AVGWK,CACHE,AVGCACHE,CV,KEREVALS] = GILBERT(X, ...
%  CLASS1,CLASS2,WK,CV,KER,CACHE,CTILDE) performs steps in
%  Gilbert's algorithm in order to compute the associated
%  averages.
%
%  INPUTS: X -- data matrix with points as columns
%          CLASS1,CLASS2 -- class indices in X
%    WK -- last point in sequence
%          CV -- last contact vector used (column vector [i;j])
%          KER -- inline kernel function
%          CACHE -- inner product cache
%          CTILDE -- Friess' constant
%
%  OUTPUTS: WK -- new point in sequence
%           AVGWK -- average over repeated contact vectors
%           CACHE -- updated cache
%           AVGCACHE -- inner product cache of avgwk with X
%           CV -- newest contact vector
%           KEREVALS -- number of kernel evaluations performed
%

N = size(X,2);

avgwk.alphabeta = zeros(1,N);
avgwk.norm2 = 0;
```

```
avgwk.nonzeros = zeros(1,N);
avgcache = zeros(1,N);
avgcount = 0;

kerevals = 0;

CVs = CV;
i = 2;

cycle = 0;
while ~cycle

  CVs(:,i) = contact(class1,class2,cache);
  [wk,cache,braevals] = bracket(wk,CVs(1,i),CVs(2,i), ...
        X,ker,cache,Ctilde);
  kerevals = kerevals + braevals;

  wknz = find(wk.nonzeros);
  avgwk.alphabeta(wknz) = avgwk.alphabeta(wknz) + ...
        wk.alphabeta(wknz);
  avgwk.nonzeros(wknz) = ones(1,length(wknz));
  avgcache = avgcache + cache;
  avgcount = avgcount + 1;

  % update avgwk, avgcache

  if sum(sum(abs(repmat(CVs(:,i),1,i-1)-CVs(:,1:(i-1))),1)==0)
    cycle = 1;
  else
    i = i + 1;
  end

end

avgwknz = find(avgwk.nonzeros);
avgwk.alphabeta(avgwknz) = avgwk.alphabeta(avgwknz)/avgcount;
avgcache = avgcache/avgcount;
avgwk.norm2 = avgwk.alphabeta(avgwknz)*(avgcache(avgwknz))';

CV = CVs(:,i);


function CV = contact(class1,class2,cache);
%CONTACT computes the contact vector indices
```

```
%  CV = CONTACT(CLASS1,CLASS2,CACHE) computes the contact vector
%  indices CV = [I;J] for the data given the two classes CLASS1
%  and CLASS2 and the inner product CACHE.

[mU,i] = max(-cache(class1));
[mV,j] = max(cache(class2));
CV = [class1(i);class2(j)];


function [wk,cache,kerevals] = bracket(wk,i,j,X,ker,cache,Ctilde)
%BRACKET calculates [WK,X(I)-X(J)]* using kernel KER
%  [WK,CACHE,KEREVALS] = BRACKET(WK,I,J,X,KER,CACHE,CTILDE)
%  computes the closest point on the line segment from WK to
%  X(I) - X(J) to the origin after implicit remapping by KER.
%
%  INPUTS: WK -- previous point in Gilbert's Algorithm
%                (row vector with alphai's, -betaj's)
%          I,J -- determine the secant X(:,I)-X(:,J)
%          X -- two class data with points as columns
%          KER -- inline kernel function
%          CACHE -- the inner product cache
%          CTILDE -- Friess' constant
%
%  OUTPUTS: WK -- next point in Gilbert's Algorithm (closest pt)
%           CACHE -- the updated inner product cache
%           KEREVALS -- number of kernel evaluations

wksij = cache(i) - cache(j);
norm2sij = ker(X(:,i),X(:,i)) - 2*ker(X(:,i),X(:,j)) + ...
           ker(X(:,j),X(:,j));
norm2sij = norm2sij + 2/Ctilde;
kerevals = 3;

top = wk.norm2 - wksij;
bot = wk.norm2 - 2*wksij + norm2sij;

if top > eps
  N = size(X,2);
  if top < bot
    lambda = top/bot;
    wknz = find(wk.nonzeros);
    wk.alphabeta(wknz) = (1-lambda)*wk.alphabeta(wknz);
    wk.alphabeta([i j]) = wk.alphabeta([i j]) + lambda*[1 -1];
    wk.norm2 = wk.norm2 - top^2/bot;
```

```
    wk.nonzeros([i j]) = [1 1];
    cache = (1-lambda)*cache + ...
        lambda*(ker(X,repmat(X(:,i),1,N)) - ...
        ker(X,repmat(X(:,j),1,N)));
    cache(i) = cache(i) + lambda/Ctilde;
    cache(j) = cache(j) - lambda/Ctilde;
  else
    wk.alphabeta = zeros(1,N);
    wk.alphabeta([i j]) = [1 -1];
    wk.norm2 = norm2sij;
    wk.nonzeros = wk.alphabeta;
    wk.nonzeros(j) = 1;
    cache = ker(X,repmat(X(:,i),1,N)) - ker(X,repmat(X(:,j),1,N));
    cache(i) = cache(i) + 1/Ctilde;
    cache(j) = cache(j) - 1/Ctilde;
  end
  kerevals = kerevals + 2*N;
end
```