DATA MINING VIA MATHEMATICAL PROGRAMMING AND MACHINE LEARNING

By

David R. Musicant

A dissertation submitted in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

(Computer Sciences)

at the

UNIVERSITY OF WISCONSIN – MADISON

2000

Abstract

This work explores solving large-scale data mining problems through the use of mathematical programming methods. In particular, algorithms are proposed for the support vector machine (SVM) classification problem, which consists of constructing a separating surface that can discriminate between points from one of two classes. An algorithm based on successive overrelaxation (SOR) is presented which can process very large datasets that need not reside in memory. Concepts from generalized SVMs are combined with SOR and with linear programming to find nonlinear separating surfaces. An "active set" strategy is used to generate a fast algorithm that consists of solving a finite number of linear equations of the order of the dimensionality of the original input space at each step. This ASVM active set algorithm requires no specialized quadratic or linear programming code, but merely a linear equation solver which is publicly available. An implicit Lagrangian for the dual of an SVM is used to lead to the simple linearly convergent Lagrangian SVM (LSVM) algorithm. LSVM requires the inversion at the outset of a single (typically small) matrix, and the full algorithm is given in 11 lines of MATLAB code.

Support vector regression problems are considered as well. The problem of tolerant data fitting by a nonlinear surface is formulated as a linear program with fewer variables than that of other linear programming formulations. A generalization of the linear programming chunking algorithm for arbitrary kernels is implemented wherein chunking is performed on both data points and problem variables. The robust Huber M-estimator, a differentiable cost function that is quadratic for small errors and linear otherwise, is modeled exactly in the original primal space of the problem by an easily solvable convex quadratic program for both linear and nonlinear support vector estimators. Experiments show that the above classification and regression techniques show strong performance in accuracy, speed, and scalability on both real-world datasets and synthetic ones. In some cases, datasets on the order of millions of points were utilized. These results indicate that SVMs, typically used on smaller datasets, can be used to solve massive data mining problems.

Acknowledgements

I owe thanks to a great many people, all of whom have played an important role in this culmination of significant long-term goals. My girlfriend Liz Olsen has been amazingly supportive of me, and listened unendingly to my excitements and frustrations. My parents Mitch and Jessie, and my sisters Lori and Karen, have backed all the changes in directions that I have taken and steadfastly encouraged me with their love and support to push myself further. Olvi Mangasarian, my advisor, introduced me to the exciting world of support vector machines and has helped me to find research opportunities. I am also indebted to Michael Ferris, from whom I learned the skills necessary to actually implement the ideas contained in this thesis; to Raghu Ramakrishnan, who impressed on me more than none other the importance of and challenges in adapting algorithms to apply to massive datasets; and Jude Shavlik, whose insights into machine learning helped me to form the bedrock for this work.

A number of colleagues in the UW Data Mining Institute have been of immense help. Of them, I thank Todd Munson, Yuh-Jye Lee, and my officemate Glenn Fung. I much appreciate the support I have received from my research "siblings" Paul Bradley, Kristin Bennett, and Nick Street. Finally, I owe a huge debt of gratitude to my friends who have supported me through the educational roller coaster, and made my life all the richer.

This research was partially supported by National Science Foundation Grants CCR-9729842 and CDA-9623632, by Air Force Office of Scientific Research Grants F49620-97-1-0326 and F49620-00-1-0085, and by the Microsoft Corporation.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

This dissertation explores solving data mining problems through the use of mathematical programming methods [5, 19, 59]. Such methods have become a rather active area of research in the past few years, and have been successfully used to solve a variety of machine learning problems. This work is concerned in particular with developing mathematical programming methods that scale well to solve the massively sized problems which are found in data mining. To that end, we begin with an overview of data mining, machine learning, and mathematical programming.

1.1 Data Mining

In recent years, massive quantities of business and research data have been collected and stored, partly due to the the plummeting cost of data storage [83]. Much interest has therefore arisen in how to "mine" this data to provide useful information. The phrases *data mining* and *knowledge discovery in databases* (or *KDD*) are both used to describe this process. More specifically, KDD can be defined as "the non-trivial process of identifying valid, novel, potentially useful, and ultimately understandable patterns in data" [32]. Data mining is likewise defined as "a step in the KDD process consisting of an enumeration of patterns (or models) over the data" [32]. There are many different data mining algorithms, each designed to address a specific KDD goal. Two particular types of data mining algorithms are those that address *classification* and *regression* problems [79]. The task of classification is that of "assigning things to categories or classes determined by their properties" [98]. Regression, on the other hand, attempts to predict a specific numerical quantity of something based on its properties [30, 117].

Data mining as a discipline shares much in common with machine learning [81] and statistics, as all of these endeavors aim to make predictions about data. An important distinction is illustrated by another definition of data mining, namely "finding interesting trends or patterns in large datasets, in order to guide decisions about future activities" [99]. One of the distinguishing characteristics of the data mining viewpoint is consideration of managing the data itself. Algorithms which perform efficiently on small datasets (kilobytes or megabytes) do not necessarily scale well when applied to datasets of hundreds of megabytes or gigabytes in size. We have taken the data mining perspective in this work in that we strive to produce algorithms that will work on massive datasets, e.g. millions of data points. Our hope is that such algorithms will eventually be used for applications such as analyzing census data, World Wide Web logs, or retail sales databases.

1.2 Machine Learning

Machine learning is generally considered to be an area of study within the larger umbrella of artificial intelligence. One definition of machine learning is "the field of scientific study that concentrates on induction algorithms and on other algorithms that can be said to 'learn.' " [52]. Induction includes classification and regression, the two problems of interest for this work.

Example 1.1 *Classification Problem*

Age	Income	Years of Education	Software purchaser?
30	$56,000 / yr	16	Yes
50	$60,000 / yr	12	No
16	$2,000 / yr	11	Yes
35	$30,000 / yr	12	No

*Table 1.1: Classification example **training** set*

The dataset in Table 1.1 contains demographic information for five randomly selected people. These people were surveyed to determine whether or not they purchased software on a regular basis.

Age	Income	Years of Education	Software purchaser?
40	$48,000 / yr	17	?
29	$60,000 / yr	18	?

*Table 1.2: Classification example **test** set*

The dataset in Table 1.2 contains demographic information for people who may or may not be good targets for software advertising. Question: Which of these people purchase software on a regular basis?

Example 1.2 *Regression Problem*

Age	Income	Years of Education	$ Spent / Year
30	$56,000 / yr	16	$800
50	$60,000 / yr	12	$0
16	$2,000 / yr	11	$200
35	$30,000 / yr	12	$0

*Table 1.3: Regression example **training** set*

The dataset in Table 1.3 contains demographic information for five randomly selected people. These people were surveyed to determine how much money they typically spent annually on software.

Age	Income	Years of Education	$ Spent / Year
40	$48,000 / yr	17	?
29	$60,000 / yr	18	?

*Table 1.4: Regression example **test** set*

The dataset in Table 1.4 contains demographic information for people who may or may not be good targets for software advertising. Question: How much money do these people typically spend annually on software?

The above two examples demonstrate toy examples of classification and regression as machine learning problems. In particular, both of these problems are considered examples of *supervised learning*. In supervised learning, a *training set* of examples is presented to an algorithm. The algorithm then uses this training set to find a rule to be used in making predictions on future data [105]. In the above examples, Tables 1.1 and 1.3 are training sets. The quality of a rule is typically determined through the use of a *test set*. The test set is another set of data with the same attributes as the training set, but which is *held* *out* from the training process. The values of the *output attributes*, which are indicated by question marks in Tables 1.2 and 1.4 are hidden away and "pretended" not to be known. After training is complete, the rule is used to predict values for the output attribute for each point in the test set. These output predictions are then compared with the (now revealed) known values for these attributes, and the difference between them is measured. Success in a classification problem is typically measured by the fraction of points which are classified correctly. Success in a regression problem is measured by some kind of error criterion which reflects the deviation of the predicted output value from the actual output value. These measurements provide an estimate as to how well the algorithm will perform on data where the value of the output attribute is truly unknown.

One might ask: why bother with a test set? Why not measure the success of a training algorithm by comparing predicted output with actual output on the training set alone? The use of a test set is particularly important because it addresses the problem of *overfitting*. A learning algorithm may learn the training data "too well", i.e. it may perform very well on the training data, but very poorly on unseen testing data. For example, consider the following classification rule as a solution to the problem posed in Example 1.1:

Solution 1.1 *Overfitted Solution to Example 1.1*

If the data point is in Table 1.1, look it up in the table to find the class for which the point belongs. If the point is not in Table 1.1, classify it in the "No" category.

This solution is clearly problematic – it will yield 100% accuracy on the training set, but should do poorly on the test set since it assumes that all other points are automatically "No". An important aspect of developing supervised learning algorithms is in ensuring that overfitting does not occur.

In practice, training and test sets may be available a priori. Most of the time, however, only a single set of data is available. A random subset of the data is therefore held out from the training process in order to be used as a test set. This can introduce widely varying success rates, however, depending on which data points are held out. This is traditionally dealt with by using *cross-validation* [112]. The available data is randomly broken up into k disjoint groups of approximately equal size. The training process is run k times, each time holding out a different one of the groups to use as a test set and using all remaining points as the training set. The success of the algorithm is then measured by an average of the success over all k test sets. Usually we take $k = 10$, yielding the process referred to as tenfold cross-validation. There are some statistical issues with this approach, which are addressed in [28].

A plethora of methodologies can be found for solving classification and regression problems. These include the backpropagation algorithm for artificial neural networks [42, 78], decision tree construction algorithms [2, 12, 98], spline methods for classification [118, 120], probabilistic graphical dependency models [14, 41], least squares techniques for general linear regression [94, 103], and algorithms for robust regression [45, 46]. Support vector machines (SVMs) are another approach, rooted in mathematical programming [64, 117]. This dissertation can be considered a study in applying SVMs to massive databases.

1.3 Optimization, Mathematical Programming, and Support Vector Machines

Euler observed that "Nothing happens in the universe that does not have...a maximum or a minimum" [114]. Optimization techniques are used in industry in a wide variety of fields, typically in order to maximize profits or minimize expenses under certain constraints. For example, airlines use optimization techniques in finding the best ways to route airplanes. The use of mathematical models in solving optimization problems is referred to as *mathematical programming* [106]. The use of the word "programming" is now somewhat antiquated, and is used to mean "scheduling." Mathematical programming techniques have recently become attractive to use in solving machine learning problems [63], as they perform well while also providing a sound theoretical basis that some other popular techniques do not provide. Additionally, they offer some novel approaches in addressing the problem of overfitting [18, 21, 117].

It is not surprising that ideas from mathematical programming should find application in machine learning. After all, one can summarize the classification and regression problems as "Find a rule that *minimizes* the errors made in predicting an output." One formalization of these ideas as an optimization problem is referred to as the *support vector machine* (SVM) [18, 21, 117]. In the next chapter, we will review the relevant background material about SVMs.

1.4 Notation

- All vectors will be column vectors unless transposed to a row vector by a prime superscript $'$.

- A vector of ones in a real space of arbitrary dimension will be denoted by e.
- The identity matrix in a real space of arbitrary dimension will be denoted by I.

Let $x, y \in R^n, z \in R^m$, and $A \in R^{m \times n}$. Then:

- A' will denote the transpose of A, A_i will denote the i-th row of A, and $A_{\cdot j}$ will denote the jth column of A.
- $|x|$ will denote a vector of absolute values of the components $x_i,\ i = 1, \dots, n$ of x.
- x_+ will denote the vector in R^n with components $\max\{0, x_i\}$. This corresponds to projecting x onto the nonnegative orthant.
- x_* will denote the step function, defined as a vector in R^n of minus ones, zeros and plus ones, corresponding to negative, zero and positive components of x respectively. Similarly A_* will denote an $m \times n$ matrix of minus ones, zeros and plus ones.
- For $1 \le p < \infty$, the norm $\|x\|_p$ will denote the p-norm:

  $$\|x\|_p = \left(\sum_{i=1}^{n} |x_i|^p \right)^{\frac{1}{p}},$$

  and

  $$\|x\|_\infty = \max_{1 \le i \le n} |x_i|.$$

- $\|A\|$ will denote the 2-norm of a matrix A.
- We shall employ the MATLAB "dot" notation [76] to signify application of a function to all components of a matrix or a vector. For example, $A^2_\bullet \in R^{m \times n}$ will denote the matrix of elements of A squared.

- $x'y$ will denote the scalar (inner) product of x and y.
- $x \perp y$ will denote orthogonality, that is $x'y = 0$.
- $[x; z]$ will denote a column vector in R^{n+m}.
- For $u \in R^m$, $Q \in R^{m\times m}$ and $B \subset \{1, 2, \dots, m\}$, u_B will denote $u_{i\in B}$, Q_B will denote $Q_{i\in B}$ and Q_{BB} will denote a principal submatrix of Q with rows $i \in B$ and columns $j \in B$.
- The notation $\text{argmin}_{x\in S}\ f(x)$ will denote the set of minimizers in the set S of the real-valued function f defined on S.
- We use $:=$ to denote definition.
- For $A \in R^{m\times n}$ and $B \in R^{n\times \ell}$, the **kernel** $K(A, B)$ maps $R^{m\times n} \times R^{n\times \ell}$ into $R^{m\times \ell}$. In particular if x and y are column vectors in R^n then, $K(x', A')$ is a row vector in R^m, $K(x', y)$ is a real number and $K(A, A')$ is an $m \times m$ matrix. Note that for our purposes here $K(A, A')$ will be assumed to symmetric, that is $K(A, A') = K(A, A')'$.

1.5 Thesis Overview

We now provide an overview for the remainder of this thesis. Chapter 2 provides an introduction to support vector machines (SVMs) for classification. The basic ideas of separating surfaces, nonlinear kernels, and solution algorithms are presented and explored.

Chapter 3 describes successive overrelaxation (SOR), an algorithm for solving SVMs. Because SOR handles one point at a time, similar to Platt's sequential minimal optimization (SMO) algorithm [96] which handles two constraints at a time and Joachims'

SVM^{light} [49] which handles a small number of points at a time, SOR can process very large datasets that need not reside in memory.

Chapter 4 uses concepts from generalized SVMs [64] and linear programming to find nonlinear separating surfaces. Numerical results on a number of datasets show improved testing set correctness when comparing nonlinear separating surfaces to linear separating surfaces.

Chapter 5 proposes ASVM, an active set method for solving the SVM problem. This is a fast algorithm that consists of solving a finite number of linear equations on the order of the number of features in the original input space.

Chapter 6 introduces LSVM (Lagrangian SVM), an iterative algorithm which is notable for its simplicity and brevity as well as its performance. As with ASVM, no special optimization tools are required for the algorithm apart from a freely available equation solver.

Chapters 7 and 8 contain techniques for support vector regression. In particular, Chapter 7 presents a new linear programming based formulation for doing support vector tolerant regression that performs faster than previous techniques. Additionally, a row-column chunking scheme for handling massive datasets is illustrated.

Chapter 8 shows that the regression problem based on the Huber M-estimator loss function [45] can be formulated in a straightforward manner as a quadratic program. This quadratic program yields superior performance when compared to other proposed algorithms. Nonlinear regression surfaces can be handled through the use of kernel functions.

Chapter 9 provides summary remarks and directions for future research.

Chapter 2

Support Vector Machines for Classification

Support vector machines (SVMs) are an optimization-based approach for solving supervised machine learning problems. We present here an overview of support vector machines. Most of the information in this chapter can be found in more detail in [16, 18, 21, 117].

2.1 Statement of Problem

We consider the problem of classifying points into two classes, referred to as $A+$ and $A-$. The training data consists of m points in the n-dimensional real space R^n, as well as a class assignment for each point. We represent the points by the $m \times n$ matrix A, where each row of the matrix corresponds to a point in the classification problem. To indicate class membership, we use the $m \times m$ diagonal matrix D which contains $+1$'s and -1's along its diagonal. A $+1$ in a given row indicates that the corresponding point in the same row of A belongs to class $A+$, and a -1 indicates that the corresponding point in A belongs to class $A-$. For example, if we consider class "Yes" as $A+$ and "No" as $A-$

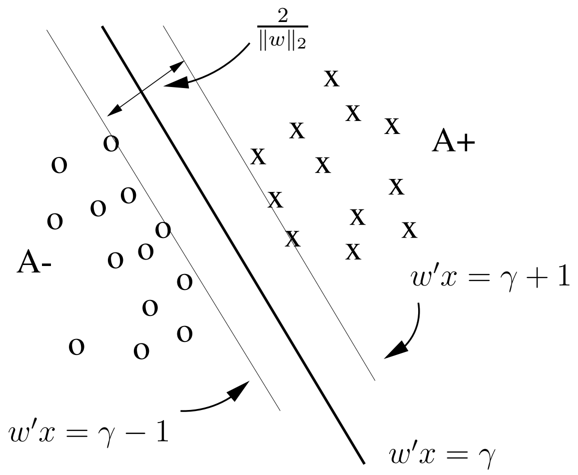

Figure 2.1: Linearly separable classification problem

we would represent the training set in Example 1.1 as:

$$A = \begin{bmatrix} 30 & 56000 & 16 \\ 50 & 60000 & 12 \\ 16 & 2000 & 11 \\ 35 & 30000 & 12 \end{bmatrix}, \quad D = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix} \tag{2.1}$$

Our goal is to find a hyperplane which will best separate the points into the two classes.

2.2 Linear Support Vector Machine Formulation

To solve this problem, let us visualize a simple example in two dimensions which is completely linearly separable.

Figure 2.1 shows a simple linearly separable classification problem, where the *separating hyperplane*, or *separating surface*

$$w'x = \gamma \tag{2.2}$$

separates the points in class $A+$ from the points in class $A-$. The goal then becomes one of finding the vector w and scalar γ such that the points in each class are correctly classified. In other words, we want to find w and γ such that the following inequalities are satisfied:

$$\begin{aligned} A_i w &> \gamma, \quad \text{for } D_{ii} = 1, \\ A_i w &< \gamma, \quad \text{for } D_{ii} = -1. \end{aligned} \tag{2.3}$$

In practice, however, we express these as non-strict inequalities. Therefore, we define $\delta > 0$ as:

$$\delta = \min_{1 \le i \le m} D_{ii}(A_i w - \gamma) \tag{2.4}$$

We then divide by δ, and redefine $w \to w/\delta$, $\gamma \to \gamma/\delta$ to yield the constraints:

$$\begin{aligned} A_i w &\geq \gamma + 1, \quad \text{for } D_{ii} = 1, \\ A_i w &\leq \gamma - 1, \quad \text{for } D_{ii} = -1. \end{aligned} \tag{2.5}$$

It turns out that we can write these two constraints as a single constraint:

$$D(Aw - e\gamma) \geq e, \tag{2.6}$$

where e is a vector of ones. Figure 2.1 shows the geometric interpretation of these two constraints. We effectively construct two *bounding planes*, one with equation $w'x = \gamma + 1$, and the other with equation $w'x = \gamma - 1$. These planes are parallel to the separating plane, and lie closer to the separating plane than any point in the associated class.

Any w and γ which satisfy constraint (2.6) will appropriately separate the class. The next task, then, is to determine how to find the best possible choices for w and γ. We want to find a plane that not only classifies the training data correctly, but will also perform well when classifying test data. Intuitively, the best possible separating plane is

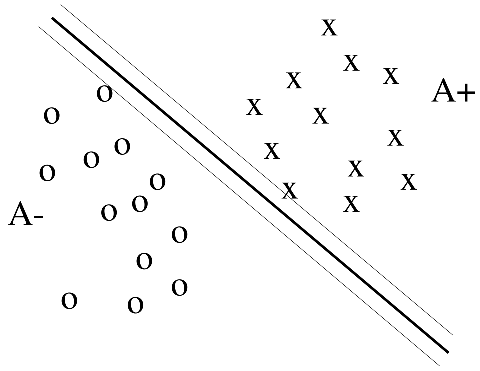

Figure 2.2: Alternative separating surface to the same data shown in 2.1. The bounding planes are closer together, and thus this plane is not expected to generalize as well.

therefore one where the bounding planes are as far as part as possible. Figure 2.2 shows a plane where the bounding planes are very close together, and thus is likely not a good choice. In order to find the best separating plane, one should spread the bounding planes as far as possible while retaining classification accuracy. This idea can be backed up quantitatively with concepts from statistical learning theory [16, 21, 117]. The distance between the bounding planes is given by $\frac{2}{\|w\|_2}$. Therefore, in order to maximize the distance, we construct an optimization problem where we minimize the magnitude of w subject to constraint (2.6):

$$\begin{aligned} &\min_{(w,\gamma)\in R^{n+1}} \frac{1}{2}\|w\|_2^2 \\ \text{s.t.} \qquad & D(Aw - e\gamma) \geq e \end{aligned} \tag{2.7}$$

We minimize over $\|w\|_2^2$ as it yields an equivalent and more tractable optimization problem than if we minimized over $\|w\|_2$. This optimization problem is a *quadratic program.* We next consider the case where the classes are not linearly separable, as shown in Figure 2.3. If the classes are not linearly separable, then we want to choose w and γ which will work in some optimal fashion. Therefore, we introduce a vector of slack variables y into

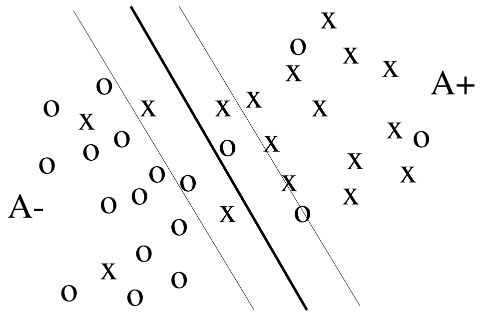

Figure 2.3: Linearly inseparable classification problem.

constraint (2.6) which will take on nonzero values only when points are misclassified, and we minimize the sum of these slack variables.

$$\begin{aligned} \min_{(w,\gamma,y)\in R^{n+1}} & \frac{1}{2}\|w\|_2^2 + \nu e'y \\ \text{s.t.} \qquad & D(Aw - e\gamma) + y \geq e \\ & y \geq 0 \end{aligned} \tag{2.8}$$

Note that the objective of this quadratic program now has two terms in it. The $\|w\|_2^2$ term attempts to maximize the distance between the bounding planes. The $e'y$ term attempts to minimize the classification errors made. Therefore, the parameter $\nu \geq 0$ is introduced to balance the emphasis of these two goals. A large value of ν indicates that most of the importance is to be placed on reducing classification error. A small value of ν indicates that most of the importance is to be placed on separating the planes and thus attempting to avoid overfitting. Finding the correct value of ν is typically an experimental task, accomplished via a tuning set and cross-validation. More sophisticated techniques for determining ν are a current topic of research [119].

Quadratic program (2.8) is referred to as a *support vector machine* (SVM). All points which lie on the "wrong" side of their corresponding bounding plane are called *support*

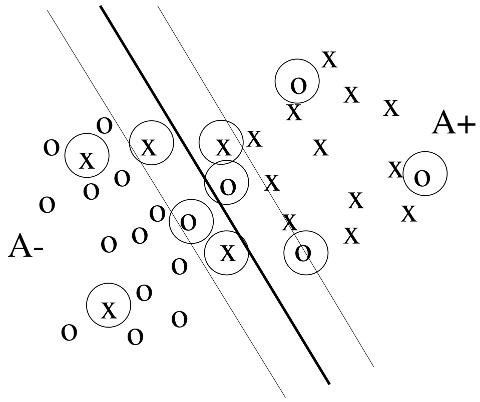

Figure 2.4: Sample two-dimensional dataset again, with support vectors indicated by circles.

vectors (see Figure 2.4), where the name "support vectors" comes from a mechanical analogy where the support vectors can be thought of as point forces keeping a stiff sheet in equilibrium [16]. It turns out that support vectors play an important role in the study of SVMs. If all the points which are not support vectors are removed from a dataset, the SVM optimization problem (2.8) yields the same solution as it would if all the points were included.

SVMs classify datasets with *numeric* attributes, as is clear by the formulation shown in (2.8). In practice, many datasets have *categorical* attributes. SVMs can handle such datasets if the categorical attributes are somehow transformed into numeric attributes. One common method for doing so is to create a set of new artificial binary numeric features, where each feature corresponds to a different possible value of the categorical attribute. For each data point, the values of all these artificial features are set to 0 except for the one feature that corresponds to the actual categorical value for the point. This one feature is assigned the value 1.

2.3 Nonlinear Support Vector Machine Formulation

SVMs can be used to find nonlinear separating surfaces as well, which significantly expands their applicability. To see how to do so, we first look at the equivalent *dual problem* to the SVM (2.8). The dual [21, 62, 64, 102] is expressed as:

$$\begin{aligned} \min_{u \in R^m} \frac{1}{2} u' D A A' D u - e' u \\ \text{s.t.} \qquad e' D u = 0 \\ 0 \leq u \leq \nu e \end{aligned} \tag{2.9}$$

The variables (w, γ) of the *primal problem* which determine the separating surface (2.2) can be obtained from the solution of the dual problem as (see Chapter 3 for more details):

$$w = A' D u, \quad \gamma \in \underset{\alpha \in R}{\operatorname{argmin}} \; e'(e - D(AA'Du - e\alpha))_+ \tag{2.10}$$

The dual formulation (2.9) can be generalized to find nonlinear separating surfaces. To do this, we observe that problem (2.9) requires knowledge only of scalar products between different rows of A as indicated by the AA' term in the objective. We therefore replace the term AA' by a *kernel function*, which is a nonlinear function which maps AA' into another matrix of the same size.

Definition 2.1 *Kernel Function*

Let $S \in R^{m \times n}$ and $T \in R^{n \times \ell}$. The **kernel** $K(S, T)$ *maps $R^{m \times n} \times R^{n \times \ell}$ into $R^{m \times \ell}$.*

Example 2.1 *Polynomial Kernel:* $K(S, T) = (ST + ee')^d_\bullet$

Example 2.2 *Gaussian (Radial Basis) Kernel:*

$$[K(S, T)]_{i,j} = \exp(-\mu \|S_i' - T_{\cdot j}\|_2^2), i = 1, ..., m, \; j = 1, ..., \ell$$

Using a polynomial kernel in the dual problem is equivalent to mapping the original data matrix A into a higher order polynomial space, and finding a linear separating surface in that space. In general, using any kernel that satisfies *Mercer's condition* [16, 21, 117] to find a separating hyperplane corresponds to finding a linear hyperplane in a higher order (possibly infinite) *feature space*. Before we can state Mercer's condition, we must first assume that a function $k(s,t) : R^n \times R^n \to R$ is known such that each element $[K(S,T)]_{i,j}$ of the kernel matrix can be represented as

$$(K(S,T))_{i,j} = k(S'_i, T_{\cdot j}). \tag{2.11}$$

Mercer's condition then requires that for X a compact subset of R^n,

$$\int\limits_{X \times X} k(s,t) f(s) f(t)\, ds\, dt \geq 0, \; \forall f \in L_2(X), \tag{2.12}$$

where the Hilbert space $L_2(X)$ is the set of functions f for which

$$\int\limits_{X} f(x)^2\, dx < \infty. \tag{2.13}$$

This is equivalent to requiring that within any finite subset of X, the matrix $K(S,T)$ is positive semi-definite [21].

We can therefore express the dual SVM as a nonlinear classification problem:

$$\begin{aligned} \min_{u \in R^m} \frac{1}{2} u' D K(A, A') D u - e'u \\ \text{s.t.} \qquad e'Du = 0 \\ 0 \leq u \leq \nu e \end{aligned} \tag{2.14}$$

where the separating surface is given by the equation

$$K(x', A') D u = \gamma \tag{2.15}$$

and γ can be found via the optimization problem

$$\gamma \in \operatorname*{argmin}_{\alpha \in R} e'(e - D(K(A, A')Du - e\alpha))_+. \tag{2.16}$$

Therefore, a point $x \in R^n$ can be classified into class $A+$ or $A-$ according to the decision function

$$(K(x', A')Du - \gamma)_* \tag{2.17}$$

where a value of 1 indicates class $A+$ and a value of -1 indicates class $A-$. In the unlikely event that the decision function yields a 0, i.e. the case where the point is on the decision plane itself, an ad-hoc choice is usually made. Practitioners often assign such a point to the class with the majority of training points.

2.4 Support Vector Machine Classification Algorithms

Most of the material in this thesis presents new algorithms that can be used in solving SVMs with massive amounts of data. We therefore present here a brief review on other algorithms that have been developed.

Since an SVM is simply an optimization problem stated as a quadratic program, the simplest approach in solving it is to use a quadratic or nonlinear programming solver. A number of tools are available for doing so, such as CPLEX [47], MINOS [86], and LOQO [115]. This technique works reasonably for small problems, on the order of hundreds or thousands of data points. Larger problems can require exorbitant amounts of memory to solve, and can take prohibitively long. As a result a number of algorithms have been proposed that are are more efficient, as they take advantage of the structure of the SVM problem.

Osuna, Freund, and Girosi [91] propose a decomposition method. This algorithm repeatedly selects small *working sets*, or "chunks" of constraints from the original problem, and uses a standard quadratic programming solver on each of these chunks. The QP solver can find a solution for each chunk quickly due to its small size. Moreover, only a relatively small amount of memory is needed at a time, since optimization takes place over a small set of constraints. The speed at which such an algorithm converges depends largely on the strategy used to select the working sets. To that end, the SVM^{light} algorithm [49] uses the decomposition ideas mentioned above coupled with techniques for appropriately choosing the working set. SVM^{light} solves an independent optimization problem at each iteration to find a direction of descent, while limiting the number of nonzero elements in the descent direction. The nonzero elements define the working set.

The SMO algorithm [96] can be considered to be an extreme version of decomposition where the working set always consists of only two constraints. This yields the advantage that the solution to each optimization problem can be found analytically and evaluated via a straightforward formula, i.e. a quadratic programming solver is not necessary. SMO has become quite popular in the SVM community, due to its relatively quick convergence speeds. As a result, further optimizations to SMO have been made that result in even further improvements in its speed [26, 51].

We now begin our look at new SVM algorithms that provide improvements in speed and/or scalability over the ideas presented above.

Chapter 3

Successive Overrelaxation for Support Vector Machines

3.1 Introduction

Successive overrelaxation (SOR), originally developed for the solution of large systems of linear equations [89, 90] has been successfully applied to mathematical programming problems [22, 54, 60, 61, 65, 93], some with as many 9.4 million variables [25]. By taking the dual of the quadratic program associated with a support vector machine [18, 117] for which the margin (distance between bounding separating planes) has been maximized with respect to *both* the normal to the planes as well as their location, we obtain a very simple convex quadratic program with bound constraints *only*. This problem is equivalent to a symmetric mixed linear complementarity problem (i.e. with upper and lower bounds on its variables [29]) to which SOR can be directly applied. This corresponds to solving the SVM dual convex quadratic program for one variable at a time, that is computing one multiplier of a potential support vector at a time.

We note that in the Kernel Adatron Algorithm [34, 35], Friess, Cristianini and Campbell propose a similar algorithm which updates multipliers of support vectors one at a time. They also maximize the margin with respect to both the normal to the separating planes as well as their location (bias). However, because they minimize the 2-norm of the constraint violation y of equation (3.1) and we minimize the 1-norm of y, our dual variables are bounded above in (3.5) whereas theirs are not. Boser, Guyon and Vapnik [6] also maximize the margin with respect to both the normal to the separating planes as well as their location using a strategy from [116].

In Section 3.2 we state our discrimination problem as a classical support vector machine problem (3.1) and introduce our variant of the problem (3.3) that allows us to state its dual (3.5) as an SOR-solvable convex quadratic program with bounds. We show in Proposition 3.1 that both problems yield the same answer under fairly broad conditions. In Section 3.3 we state our SOR algorithm and establish its linear convergence using a powerful result of Luo and Tseng [57, Proposition 3.5]. In Section 3.4 we give numerical results for problems with datasets with as many as 10 million points.

3.2 The Support Vector Machine and its Variant

We consider the problem of discriminating between m points in the n dimensional real space R^n, represented by the $m \times n$ matrix A, according to membership of each point A_i in the classes 1 or -1 as specified by a given $m \times m$ diagonal matrix D with ones or minus ones along its diagonal. For this problem the standard support vector machine with a linear kernel AA' [18, 117] is given by the following for some $\nu > 0$, as seen in (2.8):

$$\begin{array}{rrcl} \min\limits_{w,\gamma,y} & \nu e'y + \frac{1}{2}w'w & & \\ \text{s.t.} & D(Aw - e\gamma) + y & \geq & e \\ & y & \geq & 0. \end{array} \tag{3.1}$$

Here w is the normal to the bounding planes:

$$\begin{array}{rcl} x'w - \gamma &=& +1 \\ x'w - \gamma &=& -1. \end{array} \tag{3.2}$$

The one-norm of the slack variable y is minimized with weight ν in (3.1). The quadratic term in (3.1), which is twice the reciprocal of the square of the 2-norm distance $\frac{2}{\|w\|_2}$ between the two planes of (3.2) in the n-dimensional space of $w \in R^n$ for a *fixed* γ, maximizes that distance. In our approach here, which is similar to that of [6, 34, 35], we measure the distance between the planes in the $(n+1)$-dimensional space of $[w; \gamma] \in R^{n+1}$ which is $\frac{2}{\|[w;\gamma]\|_2}$. Thus using twice its reciprocal squared instead, yields our variant of the SVM problem as follows:

$$\begin{array}{ll} \min\limits_{w,\gamma,y} & \nu e'y + \frac{1}{2}(w'w + \gamma^2) \\ \text{s.t.} & D(Aw - e\gamma) + y \ \geq \ e \\ & \qquad\qquad\qquad y \ \geq \ 0. \end{array} \tag{3.3}$$

The Wolfe duals [59, Section 8.2] to the quadratic programs (3.1) and (3.3) are as follows.

$$\begin{array}{c} \max\limits_{u} \ -\dfrac{1}{2}u'DAA'Du + e'u, \ \text{s.t. } e'Du = 0, \ 0 \leq u \leq \nu e \\ (w = A'Du). \end{array} \tag{3.4}$$

$$\begin{array}{c} \max\limits_{u} \ -\dfrac{1}{2}u'DAA'Du - \dfrac{1}{2}u'Dee'Du + e'u, \ \text{s.t. } 0 \leq u \leq \nu e \\ \\ (w = A'Du, \ \gamma = -e'Du, \ y = (e - D(Aw - e\gamma))_+). \end{array} \tag{3.5}$$

We note immediately that the variables (w, γ, y) of the primal problem (3.3) can be directly computed from the solution u of its dual (3.5) as indicated. However, *only* w of the primal problem (3.1) variables can be directly computed from the solution u of its dual (3.4) as indicated. The remaining variables (γ, y) of (3.1) can be computed by setting $w = A'Du$ in (3.1), where u is a solution of its dual (3.4), and solving the resulting linear program for (γ, y). Alternatively, γ can be determined by minimizing the expression for $e'y = e'(e - D(Aw - e\gamma))_+$ as a function of the single variable γ after w has been expressed as a function of the dual solution u as indicated in (3.4), that is:

$$\min_{\gamma \in R} e'(e - D(AA'Du - e\gamma))_+). \tag{3.6}$$

We note that the formulation (3.5) can be extended to a general nonlinear kernel $K(A, A')$ by replacing AA' by the kernel $K(A, A')$ and the SOR approach can similarly be extended to a general nonlinear kernel. This is fully described in Chapter 4.

It is interesting to note that very frequently the standard SVM problem (3.1) and our variant (3.3) give the same w. For $1,000$ randomly generated such problems with $A \in R^{40\times 5}$ and the same ν, only 34 cases had solution vectors w that differed by more than 0.001 in their 2-norm. In fact we can state the following result which gives sufficient conditions that ensure that every solution of (3.3) is also a solution of (3.1) for a possibly larger ν.

Proposition 3.1 *Each solution $(\tilde{w}, \tilde{\gamma}, \tilde{y})$ of (3.3) is a solution of (3.1) for a possibly larger value of ν in (3.1) whenever the following linear system has a solution $\tilde{v}$:*

$$A'Dv = 0,\ e'Dv = \tilde{\gamma},\ v \geq 0, \tag{3.7}$$

such that

$$e'\tilde{v}(e'\tilde{y} - 1) \leq \tilde{\gamma}^2. \tag{3.8}$$

We note immediately that condition (3.8) is automatically satisfied if $e'\tilde{y} \leq 1$. We skip the straightforward proof of this proposition which consists of writing the Karush-Kuhn-Tucker (KKT) conditions [59] for the two problems (3.1) and (3.3) and showing that if

$(\tilde{w}, \tilde{\gamma}, \tilde{y}, \tilde{u})$ is a KKT point for (3.3), then $(\tilde{w}, \tilde{\gamma}, \tilde{y}, (\tilde{u}+\tilde{v}))$ is a KKT point for (3.1) with ν replaced by $\nu + e'\tilde{v}$.

We turn now to the principal computational aspect of this chapter.

3.3 Successive Overrelaxation for Support Vector Machines

The main reason for introducing our variant (3.3) of the SVM is that its dual (3.5) does not contain an equality constraint as does the dual (3.4) of (3.1). This enables us to apply in a straightforward manner the effective matrix splitting methods such as those of [57, 60, 61] that process one constraint of (3.3) at a time through its dual variable, without the complication of having to enforce an equality constraint at each step on the dual variable u. This permits us to process massive data without bringing it all into fast memory.

If we define

$$H = D[A \quad -e], \;\; L + E + L' = HH', \tag{3.9}$$

where the nonzero elements of $L \in R^{m\times m}$ constitute the strictly lower triangular part of the symmetric matrix HH', and the nonzero elements of $E \in R^{m\times m}$ constitute the diagonal of HH', then the dual problem (3.5) becomes the following *minimization* problem after its objective function has been replaced by its negative:

$$\min_u \frac{1}{2}\|H'u\|^2 - e'u, \text{ s.t. } u \in S = \{u \mid 0 \le u \le \nu e\}. \tag{3.10}$$

A necessary and sufficient optimality condition for (3.10) is the following gradient projection optimality condition [57, 97]:

$$u = (u - \omega E^{-1}(HH'u - e))_{\#},\ \omega > 0, \tag{3.11}$$

where $(\cdot)_{\#}$ denotes the 2-norm projection on the feasible region S of (3.10), that is:

$$((u)_{\#})_i = \left\{ \begin{array}{l} 0 \ \text{ if } u_i \le 0 \\ u_i \ \text{ if } 0 < u_i < \nu \\ \nu \ \text{ if } u_i \ge \nu \end{array} \right\},\ i = 1, \ldots, m. \tag{3.12}$$

Our SOR method, which is a matrix splitting method that converges linearly to a point $\bar{u}$ satisfying (3.11), consists of splitting the matrix HH' into the sum of two matrices as follows:

$$HH' = \omega^{-1}E(B + C),\ \text{s.t. } B - C \text{ is positive definite.} \tag{3.13}$$

For our specific problem we take:

$$B = (I + \omega E^{-1}L),\ C = ((\omega - 1)I + \omega E^{-1}L'),\ 0 < \omega < 2. \tag{3.14}$$

This leads to the following linearly convergent [57, Equation (3.14)] matrix splitting algorithm:

$$u^{i+1} = (u^{i+1} - Bu^{i+1} - Cu^i + \omega E^{-1}e)_{\#}, \tag{3.15}$$

for which

$$B + C = \omega E^{-1}HH',\ B - C = (2 - \omega)I + \omega E^{-1}(L - L'). \tag{3.16}$$

Note that for $0 < \omega < 2$, the matrix $B + C$ is positive semidefinite and matrix $B - C$ is positive definite. The matrix splitting algorithm (3.15) results in the following easily implementable SOR algorithm once the values of B and C given in (3.14) are substituted in (3.15).

Algorithm 3.1 (SOR Algorithm) *Choose $\omega \in (0,2)$. Start with any $u^0 \in R^m$. Having u^i compute u^{i+1} as follows:*

$$u^{i+1} = (u^i - \omega E^{-1}(HH'u^i - e + L(u^{i+1} - u^i)))_{\#}, \tag{3.17}$$

until $\|u^{i+1} - u^i\|$ is less than some prescribed tolerance.

Remark 3.1 *The components of u^{i+1} are computed in order of increasing component index. Thus the SOR iteration (3.17) consists of computing u_j^{i+1} using $(u_1^{i+1}, \ldots, u_{j-1}^{i+1}, u_j^i, \ldots, u_m^i)$. That is, the latest computed components of u are used in the computation of u_j^{i+1}. The strictly lower triangular matrix L in (3.17) can be thought of as a substitution operator, substituting $(u_1^{i+1}, \ldots, u_{j-1}^{i+1})$ for $(u_1^i, \ldots, u_{j-1}^i)$. Thus, SOR can be interpreted as using each new component value of u immediately after it is computed, thereby achieving improvement over other iterative methods such as gradient methods where all components of u are updated at once.*

We have immediately from [57, Proposition 3.5] the following linear convergence result.

Theorem 3.1 (SOR Linear Convergence) *The iterates $\{u^i\}$ of the SOR Algorithm 3.1 converge R-linearly to a solution of $\bar{u}$ of the dual problem (3.5), and the objective function values $\{f(u^i)\}$ of (3.5) converge Q-linearly to $f(\bar{u})$. That is for $i \geq \bar{i}$ for some $\bar{i}$:*

$$\begin{aligned} \|u^i - \bar{u}\| &\leq \mu\delta^i, \text{ for some } \mu > 0,\ \delta \in (0,1), \\ f(u^{i+1}) - f(\bar{u}) &\leq \tau(f(u^i) - f(\bar{u})), \text{ for some } \tau \in (0,1). \end{aligned} \tag{3.18}$$

Remark 3.2 *Even though our SOR iteration (3.17) is written in terms of the full $m \times m$ matrix HH', it can easily be implemented one row at a time without bringing all of the* *data into memory as follows for $j = 1, \ldots, m$:*

$$u_j^{i+1} = (u_j^i - \omega E_{jj}^{-1}(H_j(\sum_{\ell=1,\, j>1}^{j-1} H_\ell u_\ell^{i+1} + \sum_{\ell=j}^{m} H_l u_\ell^i - 1)))_{\#}. \tag{3.19}$$

A simple interpretation of this step is that one component of the multiplier u_j is updated at a time by bringing one constraint of (3.3) at a time.

3.4 Numerical Testing

3.4.1 Implementation Details

We implemented the SOR algorithm in C++, utilizing some heuristics to speed up convergence. After initializing the u variables to zero, the first iteration consists of a sweep through all the data points. Since we assume that data points generally reside on disk and are expensive to retrieve, we retain in memory all support vectors, that is, constraints of (3.3) corresponding to nonzero components of u. We only utilize these support vectors for subsequent evaluations, until we can no longer make progress in improving the objective. We then do another sweep through all data points. This large sweep through all data points typically results in larger jumps in objective values than sweeping through the support vectors only, though it takes significantly longer. Moving back and forth between all data points and support vectors works quite well, as indicated by Platt's results on the SMO algorithm [96].

Another large gain in performance is obtained by sorting the support vectors in memory by their u values before sweeping through them to apply the SOR algorithm. Interestingly enough, sorting in either ascending or descending order gives significant improvement over no sorting at all. The experiments described below were all implemented using sorting in descending order when sweeping through all constraints, and sorting in ascending order when sweeping through just the support vectors. This combination yielded the best performance on the University of California at Irvine (UCI) Adult dataset [85].

All calculations were coded to consider three types of data structures: non-sparse, sparse, and binary. The calculations were all optimized to take advantage of the particular structure of the input data.

Finally, the SOR algorithm requires that parameters $\omega \in (0, 2)$ and $\nu > 0$ be set in advance. All the experiments here utilize $\omega = 1.0$ and $\nu = 0.05$. These values showed good performance on the Adult dataset; some of the experimental results presented here could conceivably be improved further by experimenting more with these parameters. Additional experimentation may lead to other values of ω and ν that achieve faster convergence.

3.4.2 Experimental Methodology

In order to evaluate the effectiveness of the SOR algorithm, we conducted two types of experiments. One set of experiments demonstrates the performance of the SOR algorithm in comparison with Platt's SMO algorithm [96] and Joachims' SVM^{light} algorithm [49]. We report below on results from running the SOR and SVM^{light} algorithms on the UCI Adult dataset along with the published results from SMO. The SMO experiments are reported to have been carried out on a 266 MHz Pentium II processor running Windows NT 4, using Microsoft's Visual C++ 5.0 compiler. We have run our comparable SOR experiments on a 200 MHz Pentium Pro processor with 64 megabytes of RAM, also running Windows NT 4 and Visual C++ 5.0. We ran our SVM^{light} experiments on the same hardware, but under the Solaris 5.6 operating system using the code available from

Joachims' web site [48].

The other set of experiments is directed towards evaluating the efficacy of SOR on much larger datasets. These experiments were conducted under Solaris 5.6, using the GNU EGCS 1.0.3 C++ compiler, and run on the University of Wisconsin Computer Sciences Department Ironsides cluster which utilizes 250 MHz UltraSPARC II processors with a maximum of 8 Gigabytes of memory available.

We first look at the effect of varying degrees of separability on the performance of the SOR algorithm for a dataset of 50,000 data points. We do this by varying the fraction of misclassified points in our generated data, and measure the corresponding performance of SOR. A tuning set of 0.1% is held out so that generalization can be measured as well. We use this tuning set to determine when the SOR algorithm has achieved 95% of the true separability of the dataset. Note that the expression "true separability", in this context, refers to the fraction of points which we classified correctly in the construction of the synthetic data. This is technically only a lower bound on the actual separability of the dataset, as a different separating surface from the one used in the construction algorithm might yield slightly better accuracy.

For the SMO experiments, the datasets are small enough in size so that the entire dataset can be stored in memory. These differ significantly from larger datasets, however, which must be maintained on disk. A disk-based dataset results in significantly larger convergence times, due to the slow speed of I/O access as compared to direct memory access. The C++ code is therefore designed to easily handle datasets stored either in memory or on disk. Our experiments with the UCI Adult dataset were conducted by storing all points in memory. For all other experiments, we kept the dataset on disk and stored only support vectors in memory.

We next ran SOR on the same dataset of 1 million points in R^{32} which was used in evaluating the LPC (Linear Programming Chunking) algorithm [8]. The previous LPC work measured how long it took the algorithm to converge to a stable objective. We consider this here as well; we also monitor training and tuning set accuracy. In order to do this, we remove 1% of the data (10,000 data points) to serve as a tuning set. This tuning set is not used at all in the SOR algorithm. Instead, we use it to monitor the generalizability of the separating plane which we produce. We chose 1% in order to balance the fact that we want the tuning set to be a significant fraction of the training set, yet it must be entirely stored in memory in order to be utilized efficiently.

Finally, we continue in a similar fashion and evaluate the success of SOR on a dataset one order of magnitude larger, namely consisting of 10 million points in R^{32}. We created the data by generating 10 million uniformly distributed random non-sparse data points, and a random separating hyperplane to split the data into two distinct classes. In order to make the data not linearly separable, we intentionally mislabeled a fraction of the points on each side of the plane in order to obtain a training set with a minimum prescribed separability of the data.

We note that a problem of this magnitude might be solvable by using a sample of the data. Preliminary experiments have shown that sampling may be less effective on datasets which are less separable. A proper experiment on the effect of sampling would require running SOR to convergence on various subsets of a massive dataset, and comparing test set accuracies.

3.4.3 Experimental Results

Table 3.1 shows the effect of training set separability on the speed of convergence of the SOR algorithm on a 50,000-point dataset in R^{32} to 95% correctness on a 5,000-point tuning set. We note that it takes about four times as long for SOR to converge for a training set that is 80% separable as it does for a set that is 99.5% separable. These results are also depicted in Figure 3.1. We note that there are many real world datasets that are 95%-99% separable [17, 85].

Table 3.2 illustrates running times, numbers of support vectors, and iterations for the SOR, SMO, and SVM^{light} algorithms using the UCI Adult dataset. Testing set accuracies are also included for SOR and SVM^{light}. We quote the SMO numbers from [96], and adopt the similar convention of showing both bound-constrained support vectors (those where $u_i = \nu$) and non-bound-constrained support vectors (those where $0 < u_i < \nu$). Note that the "iterations" column takes on very different meanings for the three algorithms. The SMO work defines an iteration as a single optimization process taking place between two data points. SVM^{light} defines an iteration as solving an optimization problem over a a small number of constraints. For SOR we define an iteration to be a complete sweep through the data. By "data" we mean either all the data points on disk, or all the support vectors in memory, depending on the type of iteration. We observe that although we ran our experiments on a **slower** processor, the larger datasets converged almost twice as fast under SOR as they did under SMO. These results are seen in Figure 3.2. Finally, we see that SOR converged much more quickly than SVM^{light} while still producing similar test set accuracies.

We note the underlying optimization problems (3.1) and (3.3) for SMO and SOR respectively are both strictly convex in the variable w. Hence this variable, which determines the orientation of the separating planes, is unique in both the SMO and SOR solutions. However these solutions may be different unless conditions of Proposition 3.1 hold. Even when the solution to these two problems are the same theoretically, different implementation details of the two different algorithms can lead to different approximate solutions. Similar comments apply when comparing SOR with SVM^{light}.

The SOR algorithm performed quite well on the 1 million point dataset of [8]. Previous experiments with the LPC algorithm converged to a solution in 231.3 hours [8]. SOR converged in only 9.7 hours. This vast improvement may be partially due to the fact that we used a less stringent criterion for convergence in the SOR algorithm than in LPC; like SMO, we required that at least one u value change by more than 10^{-3} in order to proceed with another iteration. Furthermore (as in SMO), an attempt at updating a dual variable is made only if the associated primal constraint violates the KKT conditions by more than a particular tolerance (also 10^{-3}). In order to verify the quality of the solution that we obtained, we looked at training set and tuning set accuracy as described above. Both training set and tuning set accuracy achieved levels of 99.7% after 18 minutes. Though the algorithm continued to run for many hours, no further improvement in training and tuning accuracy was seen. We note that LPC did not review training or tuning set accuracy, and could possibly use a similar criterion for early stopping. These results are summarized in Table 3.3.

Finally, Table 3.4 shows SOR performance on a 99.9% linearly separable dataset of 10 million data points. After 14.3 hours, 95% of the true separability of the dataset was achieved.

True Separability*	CPU Minutes to: 95% of True Separability Final Convergence	Iterations to: 95% of True Separability Final Convergence
80.0%	303.0 498.4	301,500 559,898
85.0%	83.3 334.6	76,000 445,213
90.0%	51.9 260.9	63,000 443,868
95.0%	39.2 230.6	62,500 351,266
98.5%	15.2 178.3	24,500 228,788
99.5%	4.5 127.8	9,000 129,535

Table 3.1: Effect of separability on SOR performance for a 50,000-point dataset in R^{32}.
*See section 3.4.2 for the definition of "true separability."

SOR can be used to solve SVM problems with nonlinear kernels as well. We consider this in the next chapter.

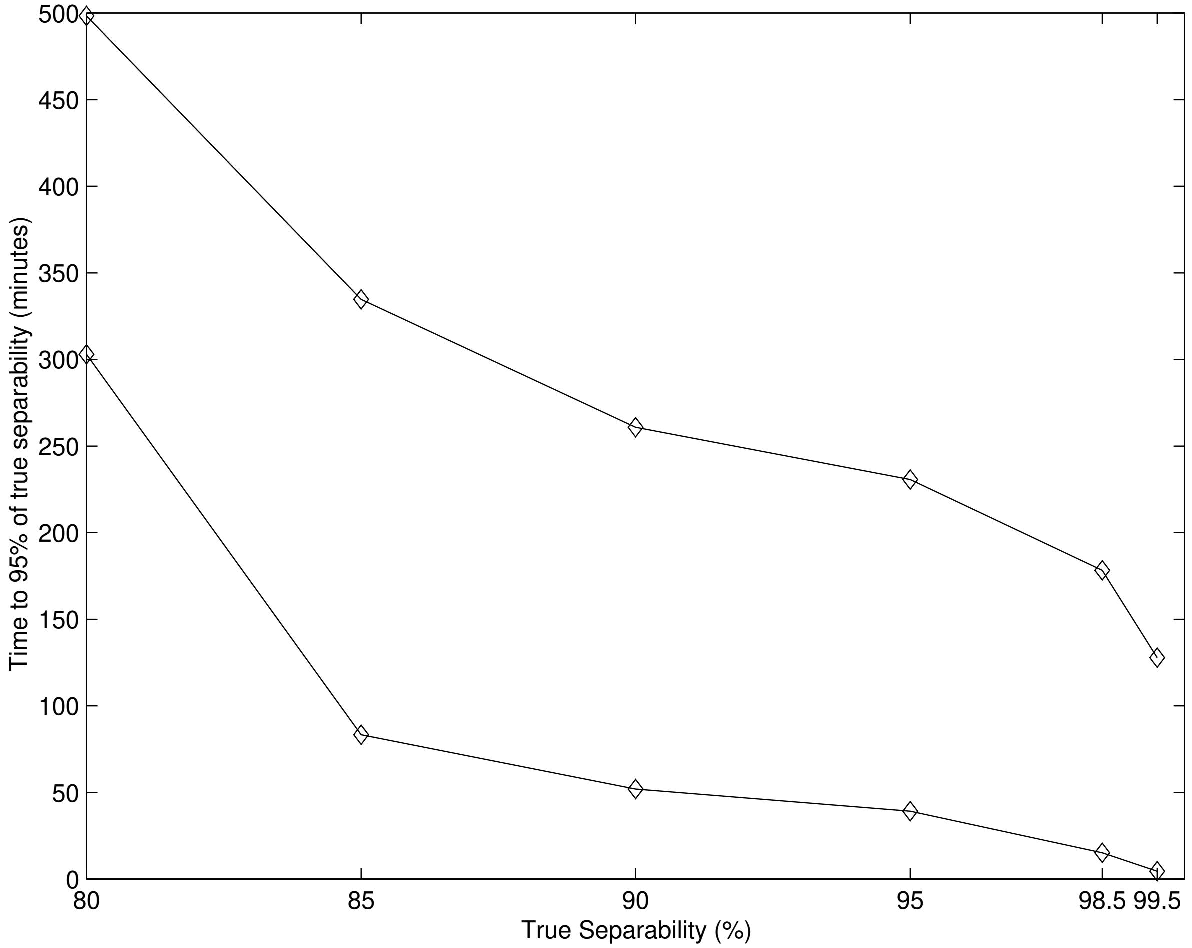

Figure 3.1: Effect of dataset separability on SOR performance on a 50,000-point dataset in R^{32}: Time to convergence (top curve) and time to 95% of true separability (bottom curve) versus true training set separability

Training Set Size	CPU Sec	Non-Bound SVs	Bound SVs	Iter*	Test Set Accuracy
	SOR	SOR	SOR	SOR	SOR
	SMO	SMO	SMO	SMO	SMO**
	SVM^l	SVM^l	SVM^l	SVM^l	SVM^l
1605	0.3	49	635	924	84.06
	0.4	42	633	3230	
	5.4	44	634	292	84.25
2265	1.2	51	930	1142	84.24
	0.9	47	930	4635	
	10.8	49	929	383	84.43
3185	1.4	52	1221	962	84.23
	1.8	57	1210	6950	
	21.0	62	1208	732	84.40
4781	1.6	65	1794	1819	84.28
	3.6	63	1791	9847	
	43.2	67	1788	865	84.47
6414	4.1	64	2379	1799	84.30
	5.5	61	2370	10669	
	87.6	63	2370	956	84.43
11221	18.8	81	4085	2642	84.37
	17.0	79	4079	17128	
	306.6	85	4078	1625	84.68
16101	24.8	86	5852	2995	84.62
	35.3	67	5854	22770	
	667.2	77	5849	2234	84.83
22697	31.3	103	8209	6061	85.06
	85.7	88	8209	35822	
	1425.6	109	8199	3960	85.17
32561	83.9	201	11550	4702	84.96
	163.6	149	11558	44774	
	2184.0	166	11554	5075	85.05

Table 3.2: SOR, SMO [96], and SVM^{light} [49] comparisons on the Adult dataset in R^{123}. *SOR, SMO, and SVM^{light} iterations are not directly comparable. See Section 3.4.3. **SMO test set accuracies on the Adult dataset have not been published. Oral communications with John Platt have suggested that SMO shows similar accuracies to those shown here for SOR and SVM^{light}.

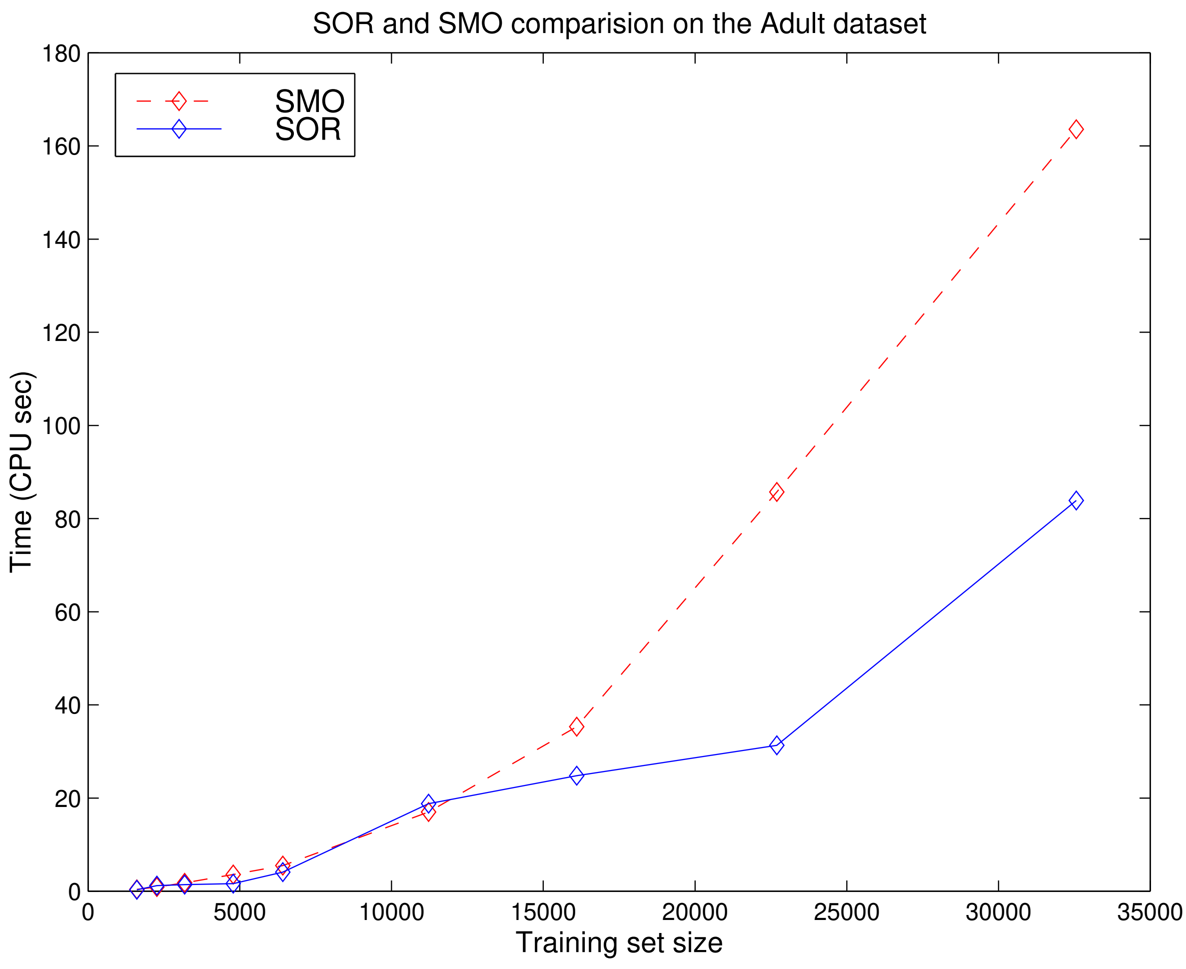

Figure 3.2: SOR and SMO comparison on the Adult dataset in R^{123}

Training Set Size	CPU hrs SOR LPC	CPU hrs to 99.7% Accuracy SOR LPC
1,000,000	9.7 231.1	0.3 —

Table 3.3: SOR and LPC comparison on 1 million point dataset in R^{32}

Training Set Size	Time to 95% separability	Iterations
10,000,000	14.3 hours	10,000

Table 3.4: SOR applied to 10 million point dataset in R^{32}

Chapter 4

Data Discrimination via Nonlinear Generalized Support Vector Machines

4.1 Introduction

The previous chapter showed how to apply successive overrelaxation (SOR) to massive linear SVM problems. In this chapter we show how to obtain a *nonlinear* separating surface for more complex datasets by using SOR, thus opening the possibility of solving large scale nonlinear separation problems. The nonlinear surface is induced by a possibly indefinite kernel. We also use linear programming as another approach for nonlinear kernel separation, where the finite simplex algorithm [23] is used. The SVM literature generally requires that nonlinear kernels be of a certain type, namely kernels that satisfy Mercer's positive definiteness condition [16]. In [64], it is shown theoretically and by means of a small example that linear programming based SVMs do not require Mercer's condition. In this work, we demonstrate the effectiveness and power of this approach using different nonlinear kernels on five publicly available datasets.

In Section 4.2 we first state our linear discrimination problem as a conventional SVM quadratic program (4.1) and its dual (4.3). We then introduce our modification of the

SVM quadratic program (4.1), which is given in problem (4.2) and its dual in (4.4). This modification is an SOR-solvable convex quadratic program with bound constraints only and a positive semidefinite kernel. The dual problems (4.3) and (4.4) for a linear separator are generalized to a kernel-induced nonlinear separator in (4.5) and (4.8). Another version of our generalized nonlinear support vector machine is also given in (4.9) below, in which the kernel is completely arbitrary but the problem is convex and hence also SOR-solvable. In Section 4.3 we state our SOR algorithm for nonlinear kernels and establish its linear convergence. In Section 4.4 we give linear programming formulations of the generalized support vector machine problem, including a novel one that includes two or more arbitrary kernels. In Section 4.5 we give numerical results for various datasets all of which show the improvement obtained in using the generalized support vector machine with a nonlinear kernel over a linear one.

4.2 The Support Vector Machine and its Generalization: Quadratic Formulation

We consider the problem of discriminating between m points in the n dimensional real space R^n, represented by the $m \times n$ matrix A, according to membership of each point A_i in the classes 1 or -1 as specified by a given $m \times m$ diagonal matrix D with ones or minus ones along its diagonal. For this problem the standard linear support vector machine with a linear kernel AA' (2.8) is given by the following for some $\nu > 0$:

$$\begin{array}{lrcl} \min\limits_{w,\gamma,y} & \nu e'y + \frac{1}{2}w'w & & \\ \text{s.t.} & D(Aw - e\gamma) + y & \geq & e \\ & y & \geq & 0. \end{array} \tag{4.1}$$

As in Chapter 3, we consider the following modification to this problem:

$$\begin{array}{rrcl} \displaystyle\min_{w,\gamma,y} & \nu e'y + \frac{1}{2}(w'w+\gamma^2) & & \\ \text{s.t.} & D(Aw-e\gamma)+y & \geq & e \\ & y & \geq & 0. \end{array} \tag{4.2}$$

The Wolfe duals to the quadratic programs (4.1) and (4.2) are as follows.

$$\begin{array}{c} \displaystyle\max_{u} \; -\frac{1}{2}u'DAA'Du + e'u, \text{ s.t. } e'Du = 0, \; 0 \leq u \leq \nu e \\ (w = A'Du). \end{array} \tag{4.3}$$

$$\begin{array}{c} \displaystyle\max_{u} \; -\frac{1}{2}u'DAA'Du - \frac{1}{2}u'Dee'Du + e'u, \text{ s.t. } 0 \leq u \leq \nu e \\ \\ (w = A'Du, \; \gamma = -e'Du, \; y = (e - D(Aw - e\gamma))_+). \end{array} \tag{4.4}$$

Formulations (4.3) and (4.4) can be extended to a general nonlinear kernel $K(A, A') : R^{m\times n} \times R^{n\times m} \rightarrow R^{m\times m}$ by replacing AA' by the kernel $K(A, A')$ and the SOR approach can similarly be extended to a general nonlinear kernel [64]. Thus the conventional support vector machine with a positive semidefinite kernel $K(A, A')$ [18, 117] is given by the following dual convex quadratic program for some $\nu > 0$ which is obtained from (4.3) by replacing AA' by $K(A, A')$:

$$\begin{array}{rrcl} \displaystyle\min_{u\in R^m} & \frac{1}{2}u'DK(A,A')Du - e'u & & \\ \text{s.t.} & e'Du & = & 0 \\ & 0 \leq u & \leq & \nu e. \end{array} \tag{4.5}$$

A solution u of this quadratic program leads to the nonlinear separating surface

$$K(x', A')Du = \gamma, \tag{4.6}$$

where γ is defined as in (3.6) by a solution of:

$$\min_{\gamma \in R} e'(e - D(K(A, A')Du - e\gamma))_+. \tag{4.7}$$

Note that the nonlinear surface (4.6) in R^n obviates the need for computing w, since w is defined only in the higher dimensional space in which the nonlinear surface (4.6) is mapped into a plane.

By a similar approach we obtain the following quadratic dual with bound constraints only and a *nonlinear* kernel $K(A, A')$ by replacing AA' in (4.4) by $K(A, A')$:

$$\begin{gathered} \min_u \frac{1}{2}u'D[K(A, A') + ee']Du - e'u, \text{ s.t. } 0 \leq u \leq \nu e \\ (\gamma = -e'Du), \end{gathered} \tag{4.8}$$

with an explicit expression for γ and the same separating surface (4.6). The formulation (4.8) allows a direct use of the SOR algorithm to solve very large problems.

We mention in passing another possible change in (4.5) allows the use of possibly indefinite kernels. One particular formulation motivated by [64, Equation (10)] is the following one which, as in (4.8), incorporates the equality $e'Du = 0$ into the objective function:

$$\begin{gathered} \min_u \frac{1}{2}u'D[K(A, A')K(A, A')' + ee']Du - e'u, \text{ s.t. } 0 \leq u \leq \nu e \\ (\gamma = -e'Du), \end{gathered} \tag{4.9}$$

with a separating surface [64, Equations (1) & (10)] different from (4.6):

$$K(x', A')K(A, A')'Du = \gamma. \tag{4.10}$$

Note that the kernel $K(A, A')$ in the formulation (4.9) is completely arbitrary and need not satisfy any positive semidefiniteness condition in order for the objective function of (4.9) to be convex. This makes the separating surface (4.10) quite general.

4.3 Successive Overrelaxation for Nonlinear GSVM

Our goal is now to present an algorithm which can be used to efficiently solve the support vector machines presented in the previous section. Traditional quadratic programming techniques do not exploit the structure of this problem, and run exceedingly slowly. Moreover, such techniques expect that all the data is available in memory. This criterion results in significant limitations in scaling these algorithms. Successive overrelaxation was used effectively in Chapter 3 for linear support vector machines in order to overcome these difficulties. We therefore apply it here to the nonlinear SVMs (4.8) and (4.9). Both of these problems can be stated as:

$$\min_{u} \frac{1}{2}u'Mu - e'u, \text{ s.t. } u \in S = \{u \mid 0 \leq u \leq \nu e\}, \tag{4.11}$$

with the symmetric matrix M defined as

$$D(K(A, A') + ee')D$$

and

$$D(K(A, A')K(A, A')' + ee')D$$

respectively for these two problems. Thus M will be positive semidefinite if we assume that $K(A, A')$ is positive semidefinite in the former case and under no assumptions in the latter case. If we decompose M as follows:

$$M = L + E + L', \tag{4.12}$$

where $L \in R^{m\times m}$ is the strictly lower triangular part of the symmetric matrix M, and $E \in R^{m\times m}$ is the positive diagonal of M, then a necessary and sufficient optimality condition for (4.11) for positive semidefinite M is the following gradient projection optimality condition [57, 97]:

$$u = (u - \omega E^{-1}(Mu - e))_{\#},\ \omega > 0, \tag{4.13}$$

where $(\cdot)_{\#}$ denotes the 2-norm projection on the feasible region S of (4.11), as defined in (3.12). We now proceed in a similar fashion as we did in Chapter 3, developing SOR as a matrix splitting method that converges linearly to a point $\bar{u}$ satisfying (4.13). We split M into two matrices as follows:

$$M = \omega^{-1}E(B + C),\ \text{s.t. } B - C \text{ is positive definite.} \tag{4.14}$$

The derivation is then identical to that presented in (3.13)–(3.16), with M substituted for HH'. This yields:

Algorithm 4.1 (SOR Algorithm) *Choose $\omega \in (0, 2)$. Start with any $u^0 \in R^m$. Having u^i compute u^{i+1} as follows:*

$$u^{i+1} = (u^i - \omega E^{-1}(Mu^i - e + L(u^{i+1} - u^i)))_{\#}, \tag{4.15}$$

until $\|u^{i+1} - u^i\|$ is less than some prescribed tolerance.

The linear convergence results in Theorem 3.1 apply here as well. We now turn our attention to linear programming formulations of the nonlinear GSVM.

4.4 The Nonlinear GSVM as a Linear Program

We briefly describe now how the nonlinear separating surface (4.6) can also be generated by a linear program. Some of the material here has been shown in more detail in [64], though we include it here for completeness.

We begin with the standard linear support vector machine (4.1). Motivated by the Wolfe dual (4.3), we make the substitution $w = A'Du$ into the primal and obtain:

$$\begin{array}{rrcl} \min\limits_{u,\gamma,y} & \nu e'y + \frac{1}{2}uDAA'Du & & \\ \text{s.t.} & D(AA'Du - e\gamma) + y & \geq & e \\ & y & \geq & 0. \end{array} \tag{4.16}$$

As in the SOR formulation, we substitute a nonlinear kernel for the AA' term. Additionally, we "simplify" the regularization term in the objective to depend only on the variable u in a linear fashion. We achieve this by introducing a variable s, which takes on the value of the absolute value of u. The substitution surrogates support vector suppression for maximizing the distance between separating bounding planes, which has a similar regularizing effect. This results in the linear program [64, Equation (11)]:

$$\begin{array}{c} \min\limits_{u,\gamma,y,s} \ \nu e'y + e's \\ \text{s.t.} \quad D(K(A,A')Du - e\gamma) + y \ \geq \ e \\ s \ \geq \ u \ \geq \ -s \\ y \ \geq \ 0. \end{array} \tag{4.17}$$

A solution (u, γ, y, s) to this linear program for a completely arbitrary kernel $K(A, A')$ will provide a separating surface given by (4.6). We note that if the linear kernel (i.e. polynomial kernel of degree 1) $K(A, A') = AA'$ is used in the linear program above and w is introduced and set equal to $A'Du$ we obtain the high-performing 1-norm linear SVM proposed in [11] and utilized successfully in [4, 7, 10].

Since the linear programming formulation does not impose restrictions on the kernel employed, we are free to use more than one kernel in a given formulation. For example we can use **both** a linear kernel and a discontinuous neural network kernel in a single linear programming formulation. More generally if we let $K^1(A, A')$ and $K^2(A, A')$ be two arbitrary kernels, then the corresponding linear programming formulation becomes:

$$
\begin{array}{rrcl}
\displaystyle\min_{u^1,u^2,\gamma,y,s^1,s^2} & \nu e'y + e's^1 + e's^2 & & \\
\text{s.t.} \quad D(K^1(A,A')Du^1 + K^2(A,A')Du^2 - e\gamma) + y & \geq & e & \\
s^1 \;\geq\; u^1 & \geq & -s^1 & \\
s^2 \;\geq\; u^2 & \geq & -s^2 & \\
y & \geq & 0. &
\end{array}
\tag{4.18}
$$

The corresponding separating surface to a solution $(u^1, u^2, \gamma, y, s^1, s^2)$ of this linear program is given by:

$$K^1(x', A')Du^1 + K^2(x', A')Du^2 = \gamma \tag{4.19}$$

For example we can use a linear and a discontinuous neural network kernel for $K^1(A, A')$ and $K^2(A, A')$ as follows:

$$K^1(A', A') = (AA' - ee');\; K^2(A', A') = (AA' - ee')_*, \tag{4.20}$$

where, as defined in Section 1.4, the subscript $*$ denotes replacement of each component of the matrix AA' by $1,\ 0$, or -1, depending on whether it is positive, zero or negative, respectively. Note that in general this neural network kernel is not positive semidefinite and hence standard support vector techniques cannot make use of it. We also note that the linear programming formulation (4.18) can be extended to more than two kernels in an obvious manner.

4.5 Numerical Testing

We tested nonlinear kernels versus linear ones by using different formulations of the generalized SVM, namely the quadratic programming formulations as given in (4.8) and

(4.9), and the linear programming formulation as given in (4.17). Most of our code development was in the MATLAB environment [76], though the actual solutions to the SVMs were handled externally. We coded the SOR algorithm in C, and accessed it via the MATLAB "mex" interface which allows one to execute external C code as a native MATLAB function [77]. The linear programs were solved with the MINOS solver [86]. All experiments were run on the University of Wisconsin Computer Sciences Department Ironsides cluster. This cluster of four Sun Enterprise E6000 machines consists of 16 UltraSPARC II 250 MHz processors and 2 gigabytes of RAM on each node resulting in a total of 64 processors and 8 gigabytes of RAM.

The SVMs were tested on the following datasets from the University of California at Irvine (UCI) repository [85]:

- The *liver-disorders* dataset contains 345 points, each consisting of six features. Class 1 contains 145 points, and class -1 contains 200 points.

- The *letter-recognition* dataset is used for recognizing letters of the alphabet. Traditionally, this dataset is used for multi-category classification. We used it here in a two class situation by taking a subset of those points which correspond to the letter "A," and a subset of those points which correspond to the letter "B." This resulted in a dataset of 600 points with 6 features, where each class contained 300 points.

- The *mushroom* dataset is a two class dataset which contains a number of categorical attributes. We transformed each categorical attribute into a series of binary attributes, one attribute for each distinct value. For example, this dataset contains an attribute called "cap surface," which can take one one of four categories, namely "fibrous," "grooves," "scaly," or "smooth." We represented this as four attributes.

A 1 is assigned to the attribute that corresponds to the actual category, and a 0 to the rest. Thus each instance of this specific categorical attribute corresponds to one row of a 4 x 4 diagonal matrix of ones. A subset of the entire mushroom dataset was used, so as to shorten the running times of our experiments. The final dataset we used contained 22 features with 200 points in class 1 and 300 points in class -1.

- The *tic-tac-toe* dataset is a two class dataset that contains legal complete tic-tac-toe games. All those games where “X” has a “win” (three in a row) end up in category 1, and all other games end up in category -1. We have intentionally represented this problem with a poor representation scheme to show the power of non-linear kernels in overcoming this difficulty. Each tic-tac-toe game is represented by 9 attributes, where each attribute corresponds to a spot on the tic-tac-toe board. An “X” is represented by 1, an “O” is represented by 0, and a blank is represented by -1. We used a subset of this dataset with 9 attributes, 200 points in class 1, and 100 points in class -1.

We additionally used some synthetic datasets, namely the checkerboard dataset [43, 50] and also a synthetic dataset of Gaussian distribution generated from our NDC data generator [87]. The Gaussian synthetic dataset, suggested by Usama Fayyad of Microsoft Research, was created by constructing 20 random centers, then generating “clouds” of points around these centers in a Gaussian fashion based on random covariance matrices. An arbitrary hyperplane was used to separate the centers into two classes. Each center was assigned to the class corresponding to the side of the hyperplane it fell on; furthermore, each randomly generated point was assigned to the same class as the center from which it came. We then used a scaling parameter to size the clouds so that the data was not linearly separable.

All experiments were conducted using tenfold cross validation, which is is a standard technique used for evaluating machine learning algorithms. We randomly partitioned each dataset into ten equal parts, and ran the algorithm ten times on 90% of this dataset. Each time, a different one of the ten segments was held out to serve as a test set. We used this test set to measure the generalization capability of the learning method. Kernels and parameters were chosen via the use of a *tuning* set, an artificially created test set surrogate taken completely from the training set. We ran our experiments for a variety of parameters and kernels on the training data with the tuning set removed, and found those parameters that performed best on the tuning set. We then used these settings to retrain on the training set as a whole (tuning set included) to report our generalization results on the test set. Tuning set accuracy is usually a good predictor of test accuracy, as shown in Figure 4.1.

Table 4.1 shows results on a number of datasets, using the SOR Algorithm 4.1 with both linear and nonlinear kernels. In each of the cases shown, the chosen nonlinear kernel generalizes better than the linear kernel. This is particularly noticeable for the tic-tac-toe and Gaussian datasets. In general, the kernels we used are as follows:

Kernel 4.1 *(***Polynomial***)* $((\frac{A}{\lambda} - \rho)(\frac{A}{\lambda} - \rho)' - \mu)^d_\bullet$

Kernel 4.2 *(***Negative Polynomial***)* $-((\frac{A}{\lambda} - \rho)(\frac{A}{\lambda} - \rho)' - \mu)^d_\bullet$

The variables λ, ρ, μ are scalar parameters and d is a positive integer. When a parameter is added or subtracted from a matrix or vector, it means componentwise addition or subtraction of the parameter in conformity with MATLAB notation. The "negative squared cubic" kernel listed for the mushroom dataset is a negative polynomial

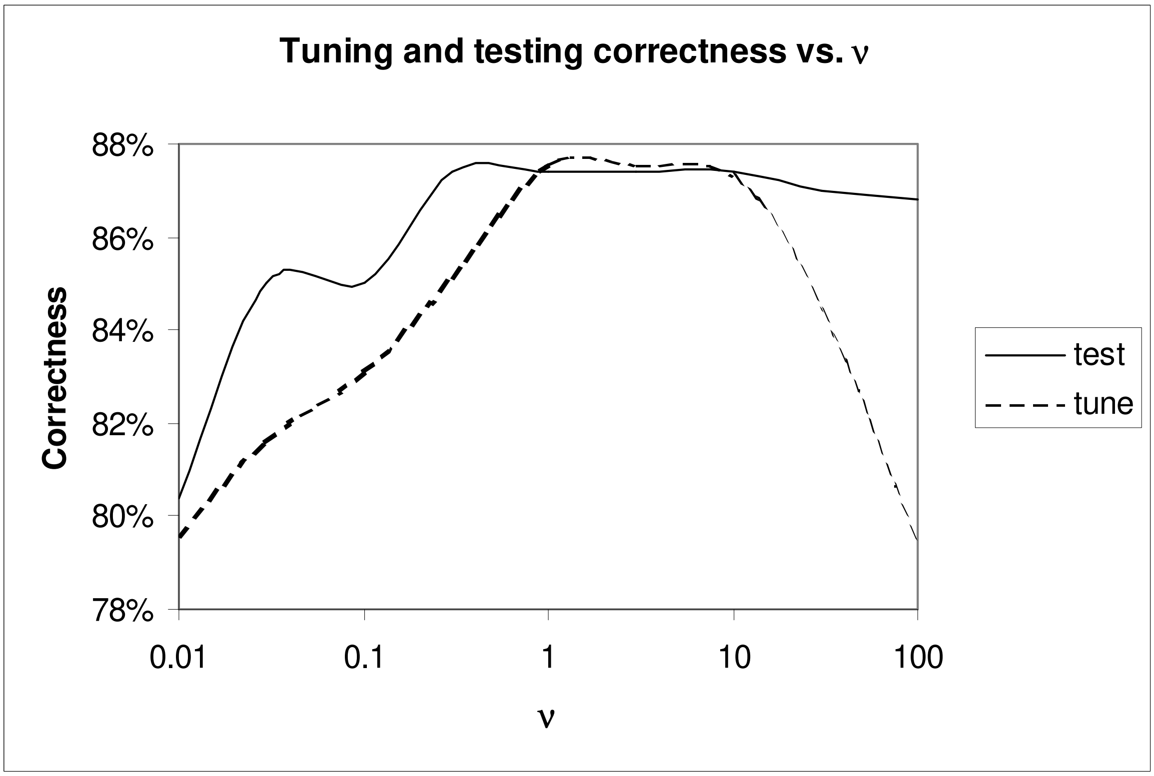

Figure 4.1: Tuning and test set accuracy for SOR with a linear kernel on the mushroom dataset.

Dataset	Kernel Type	Training Correctness	Testing Correctness	Time (secs)
Letter Recognition	Linear	85.00%	84.83%	11.1
	6th degree polynomial	92.65%	86.33%	11.0
Mushroom	Linear	87.40%	87.40%	6.6
	Negative squared cubic	89.38%	88.80%	6.7
Tic Tac Toe	Linear	60.37%	56.33%	1.4
	Quadratic	100.00%	83.33%	1.4
Gaussian	Linear	83.20%	81.70%	43.0
	Quadratic	97.68%	93.41%	43.0

Table 4.1: SOR training and test set correctness for linear and quadratic kernels

Dataset	Kernel Type	Training Correctness	Testing Correctness	Time (secs)
Liver Disorders	Linear	71.21%	68.70%	23.5
	Quadratic	78.33%	73.37%	46.8
Letter Recognition	Linear	85.43%	84.33%	62.5
	Negative Cubic	97.15%	87.67%	553.7
Tic Tac Toe	Linear	66.67%	66.67%	17.1
	Quadratic	98.78%	90.67%	21.4
Mushroom	Linear	87.40%	87.40%	45.5
	Negative 7th degree polynomial	89.56%	88.80%	35.1
Checkerboard	Linear	51.12%	48.60%	206.8
	6th degree polynomial	99.97%	98.50%	289.0

Table 4.2: Linear programming training and test set correctness for linear and nonlinear support vector machines

kernel of degree 3, with the problem formulation as in (4.9). Other kernels use the problem formulation shown in (4.8).

Table 4.2 shows training and test set correctness, using the linear programming formulation (4.17) with various kernels, under tenfold cross validation for the above mentioned datasets. For these experiments, we again implemented a linear kernel and various polynomial kernels. Note that these polynomial kernels are indefinite in general, and therefore do not satisfy Mercer's condition.

As in the SOR case above, the nonlinear kernels gave improved test set results over linear kernels. The most striking results are those for the checkerboard dataset of Figure

4.2. An initial attempt using a k-nearest-neighbor algorithm yields the very coarse separation shown in Figure 4.3. We instead used a nonlinear SVM, and obtained a tenfold test set correctness of over 97.70% for our nonlinear kernel. This is reflected in the very realistic rendering in Figure 4.4 depicting the resulting separation of the original dataset. Our experiments were inspired by those of Kaufman in [50] where eighth degree polynomial and radial basis kernels were used in a standard support vector machine formulation to separate the checkerboard dataset. However, our separation depicted in Figure 4.4 is considerably sharper than those of [50].

The next two chapters of this thesis present a further modification of the traditional SVM, above and beyond the one given in (4.2), which allow us to develop even stronger algorithms for solving SVMs without sacrificing generalization capability.

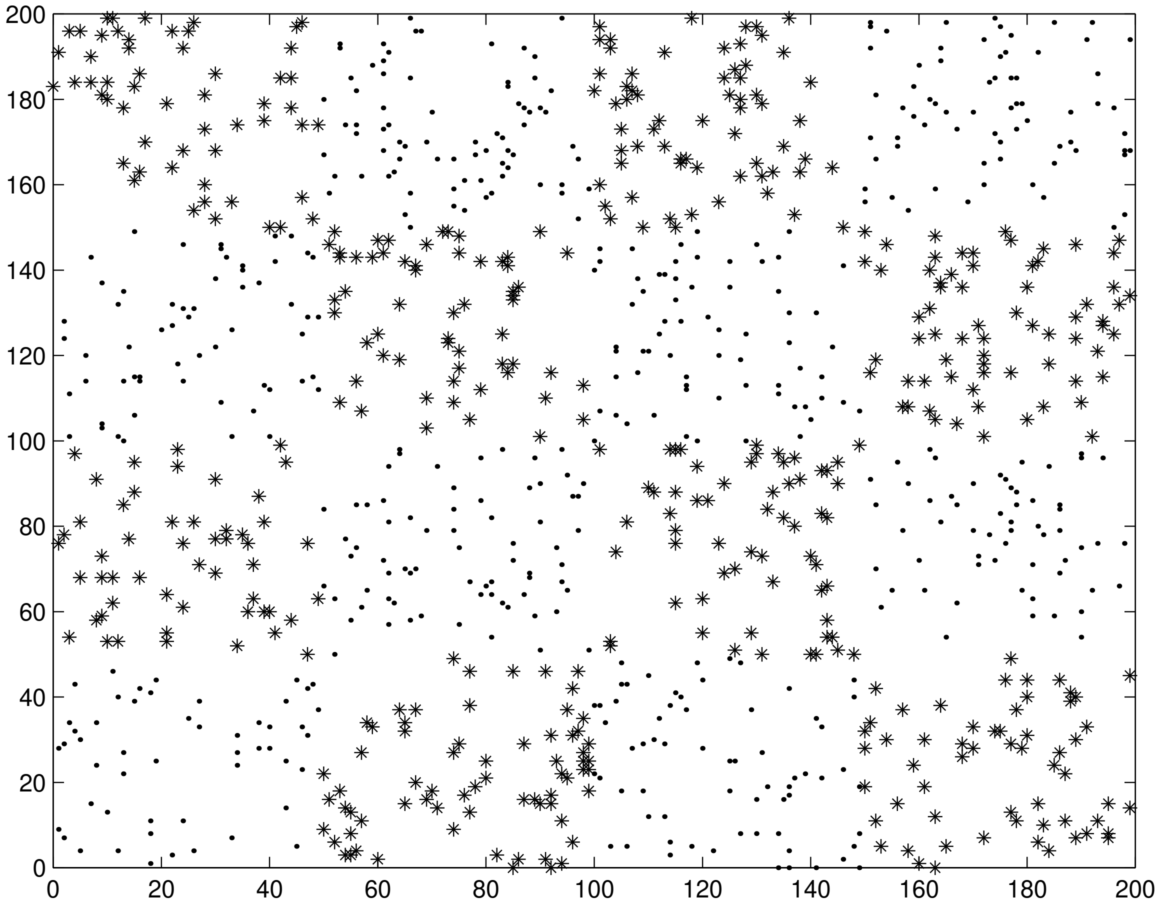

Figure 4.2: Checkerboard training dataset

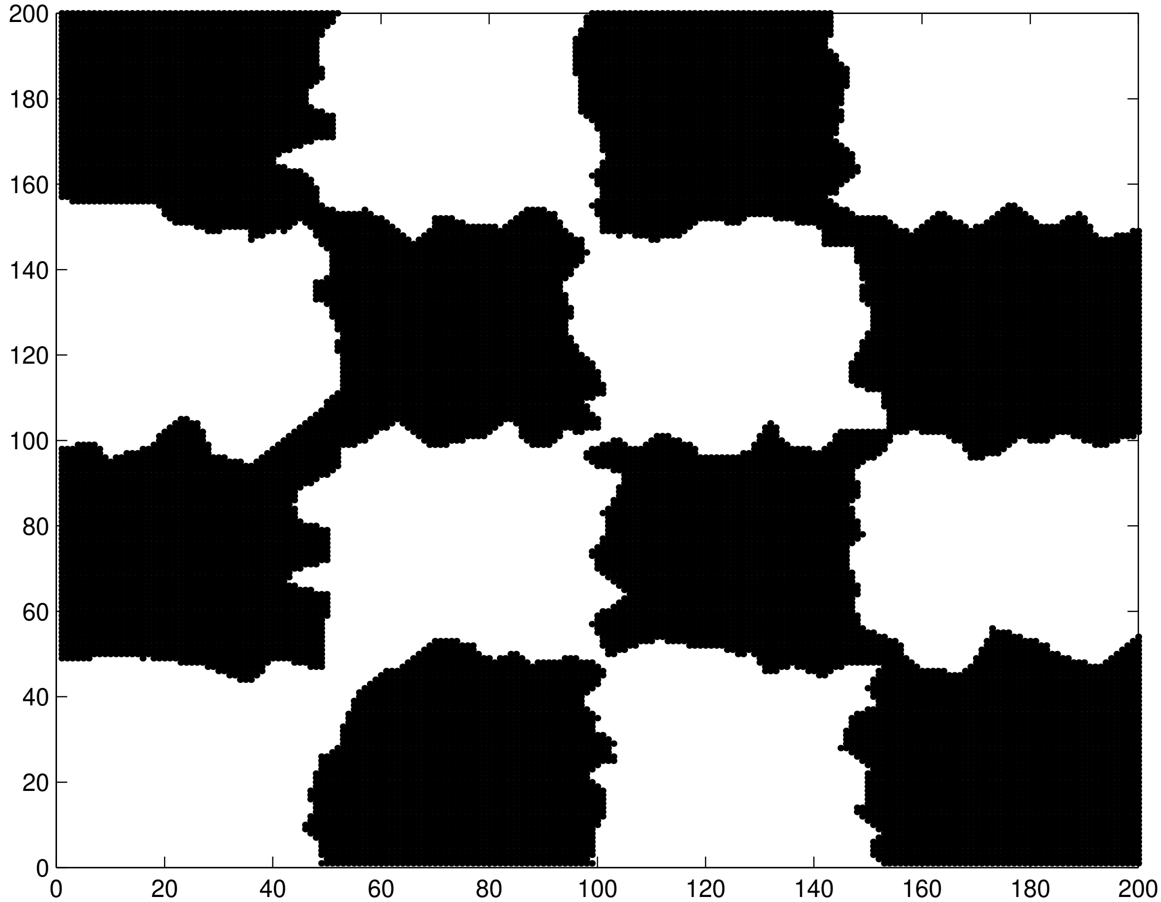

Figure 4.3: k-nearest-neighbor separation of the checkerboard dataset (k=1, 2-norm)

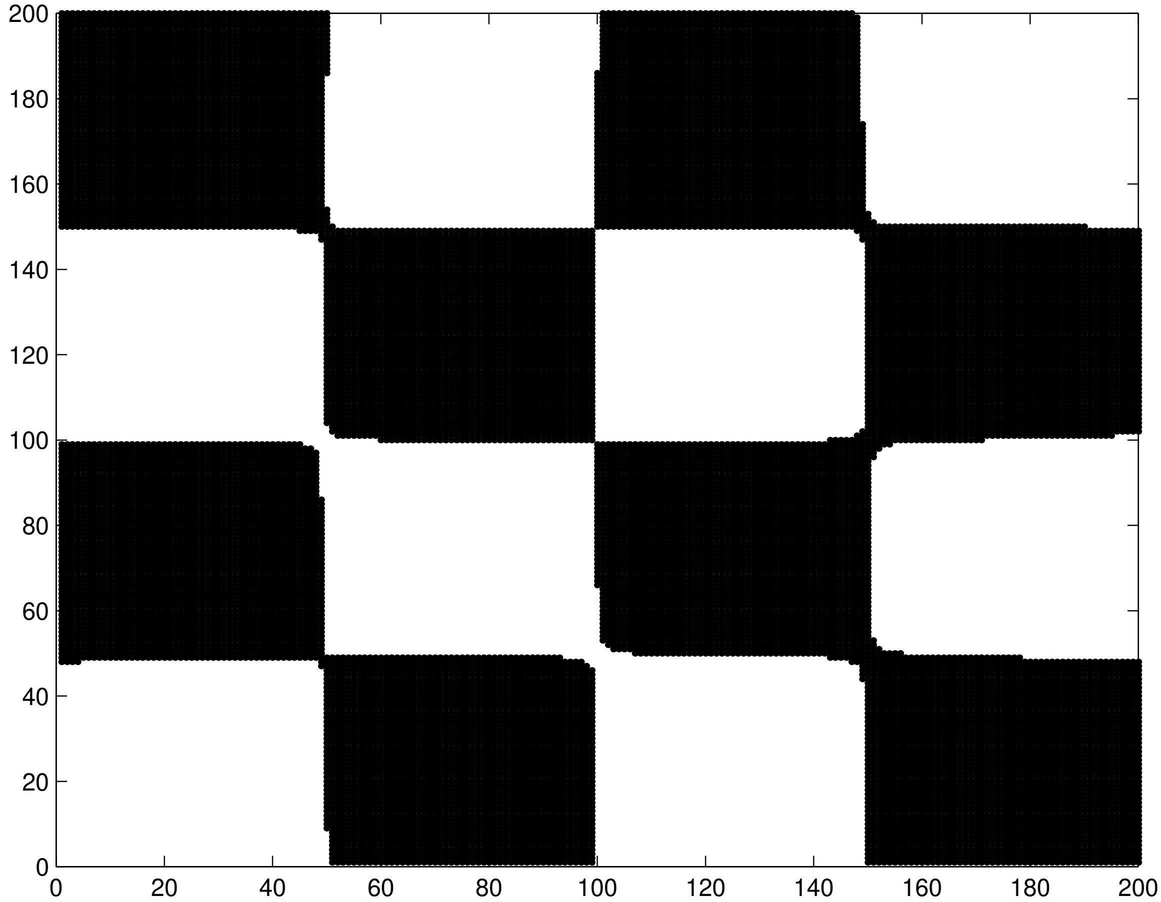

Figure 4.4: Indefinite sixth degree polynomial kernel separation of the checkerboard dataset ($\nu = 10,000,\ \lambda = 100,\ \rho = 1,\ d = 6,\ \mu = 0.5$)

Chapter 5

Active Support Vector Machines

5.1 Introduction

In this chapter, we propose a very fast simple algorithm for solving SVMs based on an active set strategy for solving quadratic programs with bounds [82]. The algorithm is capable of accurately solving problems with millions of points and requires nothing more complicated than a commonly available linear equation solver [1, 20, 76] for a typically small (100) dimensional input space of the problem.

Key to our approach are the following two changes to the standard linear SVM:

1. Maximize the margin (distance) between the parallel separating planes with respect to both orientation (w) as well as location relative to the origin (γ). See equation (5.6) below. Such an approach was also successfully utilized in the successive over-relaxation (SOR) approach in Chapters 3 and 4 as well as the smooth support vector machine (SSVM) approach of [53].

2. The error in the soft margin (y) is minimized using the 2-norm squared instead of the conventional 1-norm. See equation (5.6).

These simple, but fundamental changes, lead to a considerably simpler positive definite dual problem with nonnegativity constraints only. See equation (5.7).

In Section 5.2 we begin with the standard SVM formulation and its dual and then give our formulation and its simpler dual. We immediately note here, and corroborate this with solid computational evidence, that our simpler formulation does not compromise on generalization ability as evidenced by numerical tests in Section 5.4 on 6 public datasets. See Table 5.1. Section 5.3 gives our active support vector machine (ASVM) Algorithm 5.1 which consists of solving a system of linear equations in m dual variables with a positive definite matrix. By invoking the Sherman-Morrison-Woodbury (SMW) formula (5.1) we need only invert an $(n+1) \times (n+1)$ matrix where n is the dimensionality of the input space. This is a key feature of our approach that allows us to solve problems with millions of points by merely inverting much smaller matrices of the order of n. In concurrent work [33] Ferris and Munson also use the SMW formula but in conjunction with an interior point approach to solve massive problems based on our formulation (5.7) as well as the conventional formulation (5.5). Section 5.4 describes our numerical results which indicate that the ASVM formulation has a tenfold testing correctness that is as good as the ordinary SVM on smaller problems and has the capability of accurately solving massive problems with millions of points that cannot be attacked by standard methods for ordinary SVMs.

We note that a special case of the Sherman-Morrison-Woodbury (SMW) formula [40] will be utilized:

$$(\frac{I}{\nu} + HH')^{-1} = \nu(I - H(\frac{I}{\nu} + H'H)^{-1}H'), \tag{5.1}$$

where ν is a positive number and H is an arbitrary $m \times k$ matrix. This formula, which can be easily verified by premultiplying both sides by $(\frac{I}{\nu} + HH')$, enables us to invert a large $m \times m$ matrix by merely inverting a smaller $k \times k$ matrix.

5.2 The Linear Support Vector Machine

We consider the problem of classifying m points in the n-dimensional real space R^n, represented by the $m \times n$ matrix A, according to membership of each point A_i in the class $A+$ or $A-$ as specified by a given $m \times m$ diagonal matrix D with plus ones or minus ones along its diagonal. For this problem the standard support vector machine with a linear kernel [18, 117] is given by the following quadratic program with parameter $\nu > 0$:

$$\begin{array}{rrcl} \displaystyle\min_{(w,\gamma,y)\in R^{n+1+m}} & \nu e'y + \frac{1}{2}w'w & & \\ \text{s.t.} & D(Aw - e\gamma) + y & \geq & e \\ & y & \geq & 0. \end{array} \tag{5.2}$$

Here w is the normal to the separating surface

$$x'w = \gamma, \tag{5.3}$$

with bounding planes:

$$\begin{array}{rcl} x'w & = & \gamma + 1 \\ x'w & = & \gamma - 1, \end{array} \tag{5.4}$$

Traditionally the 1-norm of the error variable y is minimized parametrically with weight ν in (5.2) resulting in an approximate "best" separation. The dual to the standard quadratic linear SVM (5.2)

$$\begin{array}{cc} \displaystyle\min_{u\in R^m} \frac{1}{2}u'DAA'Du - e'u & \\ \text{s.t.} \qquad e'Du = 0 & \\ 0 \leq u \leq \nu e. & \end{array} \tag{5.5}$$

The variables (w, γ) of the primal problem which determine the separating surface (5.3) can be obtained from the solution of the dual problem above (Chapter 3).

We note immediately that the matrix $DAA'D$ appearing in the dual objective function (5.5) is not positive definite in general because typically $m >> n$. Also, there is an equality constraint present, in addition to bound constraints, which for large problems necessitates special computational procedures such as SMO [95]. Furthermore, a one-dimensional optimization problem (Chapter 3) must be solved in order to determine the locator γ of the separating surface (5.3). In order to overcome all these difficulties as well as that of dealing with the necessity of having to essentially invert a very large matrix of the order of $m \times m$, we propose the following simple but critical modification of the standard SVM formulation (5.2). We change the 1-norm of y to a 2-norm squared which makes the constraint $y \geq 0$ redundant. We also append the term γ^2 to $w'w$. This in effect maximizes the margin between the parallel separating planes (5.4) with respect to both w and γ (Chapter 3), that is with respect to both orientation and location of the planes, rather that just with respect to w which merely determines the orientation of the plane. This leads to the following reformulation of the SVM:

$$\begin{array}{lrcl} \displaystyle\min_{(w,\gamma,y)\in R^{n+1+m}} & \nu\frac{y'y}{2} + \frac{1}{2}(w'w+\gamma^2) & & \\ \text{s.t.} & D(Aw-e\gamma)+y & \geq & e. \end{array} \tag{5.6}$$

the dual of of this problem is [62]:

$$\min_{0\leq u\in R^m} \frac{1}{2}(\frac{I}{\nu} + D(AA'+ee')D)u - e'u. \tag{5.7}$$

The variables (w, γ) of the primal problem which determine the separating surface (5.3) are recovered directly from the solution of the dual (5.7) above by the relations:

$$w = A'Du, \;\; y = \frac{u}{\nu}, \;\; \gamma = -e'Du. \tag{5.8}$$

We immediately note that the matrix appearing in the dual objective function is positive definite and that there is no equality constraint and no upper bound on the dual variable

u. The only constraint present is a simple nonnegativity one. These facts lead us to our simple finite active set algorithm which requires nothing more sophisticated than inverting an $(n+1)\times(n+1)$ matrix at each iteration in order to solve the dual problem (5.7).

5.3 ASVM (Active Support Vector Machine) Algorithm

The algorithm consists of determining a partition of the dual variable u into nonbasic and basic variables. The nonbasic variables are those which are set to zero. The values of the basic variables are determined by finding the gradient of the objective function of (5.7) with respect to these variables, setting this gradient equal to zero, and solving the resulting linear equations for the basic variables. If any basic variable takes on a negative value after solving the linear equations, it is set to zero and becomes nonbasic. **This is the essence of the algorithm.** In order to make the algorithm converge and terminate, a few additional safeguards need to be put in place in order to allow us to invoke the Moré-Toraldo finite termination result [82]. The other key feature of the algorithm is a computational one and makes use of the SMW formula. This feature allows us to invert an $(n+1)\times(n+1)$ matrix at each step instead of a much bigger matrix of order $m\times m$.

Before stating our algorithm we define two matrices to simplify notation as follows:

$$H = D[A \quad -e], \qquad Q = \frac{I}{\nu} + HH'. \tag{5.9}$$

With these definitions the dual problem (5.7) becomes

$$\min_{0\le u\in R^m} f(u) := \frac{1}{2}u'Qu - eu. \tag{5.10}$$

It will be understood that within the ASVM Algorithm, Q^{-1} will always be evaluated using the SMW formula and hence only an $(n+1)\times(n+1)$ matrix is inverted. We state our algorithm now. Note that commented (%) parts of the algorithm are not needed in general and were rarely used in our numerical results presented in Section 5.4. The essence of the algorithm is displayed in the two boxes below.

Algorithm 5.1 (Active SVM (ASVM) Algorithm for (5.7))

(0) *Start with* $u^0 := (Q^{-1}e)_+$. *For* $i = 0, 1, 2, \ldots,$ *having* u^i *compute* u^{i+1} *as follows.*

(1) *Define* $B^i := \{j \mid u^i_j > 0\},\ N^i := \{j \mid u^i_j = 0\}$.

(2) *Determine*

$$u^{i+1}_{B^i} := (Q^{-1}_{B^iB^i}e_{B^i})_+,\ u^{i+1}_{N^i} := 0.$$

Stop if u^{i+1} *is the global solution, that is if* $0 \le u^{i+1} \perp Qu^{i+1} - e \ge 0$.

(2a) *% If* $f(u^{i+1}) \ge f(u^i)$, *then go to (4a).*

(2b) *% If* $0 \le u^{i+1}_{B^{i+1}} \perp Q_{B^{i+1}B^{i+1}}u^{i+1}_{B^{i+1}} - e_{B^{i+1}} \ge 0$, *then* u^{i+1} *is a global solution on the face of active constraints:* $u_{N^i} = 0$. *Set* $u^i := u^{i+1}$ *and go to (4b).*

(3) *Set* $i := i + 1$ *and go to (1).*

(4a) *% Move in the direction of the global minimum on the face of active constraints,* $u_{N^i} = 0$. *Set* $\tilde{u}^{i+1}_{B^i} := Q^{-1}_{B^iB^i}e_{B^i}$ *and* $u^{i+1}_{B^i} := argmin_{0\le\lambda\le 1}\{f(u^i_{B^i} + \lambda(\tilde{u}^{i+1}_{B^i} - u^i_{B^i})) \mid u^i_{B^i} + \lambda(\tilde{u}^{i+1}_{B^i} - u^i_{B^i}) \ge 0\}$. *If* $u^{i+1}_j = 0$ *for some* $j \in B^i$, *set* $i := i + 1$ *and go to (1). Otherwise* u^{i+1} *is a global minimum on the face* $u_{N^i} = 0$, *and go to (4b).*

(4b) *% Iterate a gradient projection step. Set* $k := 0$ *and* $\tilde{u}^k := u^i$. *Iterate* $\tilde{u}^{k+1} := argmin_{0\le\lambda\le 1}\ f(\tilde{u}^k - \lambda(\tilde{u}^k - (Q\tilde{u}^k - e))_+),\ k := k + 1$ *until* $f(\tilde{u}^k) < f(u^i)$. *Set*

$u^{i+1} := \tilde{u}^k$. *Set* $i := i + 1$ *and go to (1).*

Remark 5.1 *All commented (%) parts of the algorithm are not usually performed unless the algorithm gets stuck, which it rarely did on our examples. Hence our algorithm is particularly simple and usually requires only steps (0), (1),(2) and (3). The commented parts were inserted in order to comply with the active set strategy of Moré-Toraldo result [82] for which they give finite termination.*

Remark 5.2 *The iteration in step (4b) is a gradient projection step which is guaranteed to converge to the global solution of (5.7) [5, pp 223-225] and is placed here to ensure that the strict inequality* $f(\tilde{u}^k) < f(u^i)$ *eventually holds as required in [82]. Similarly, the step in (4a) ensures that the function value does not increase when it remains on the same face, in compliance with [82, Algorithm BCQP(b)].*

5.4 Numerical Implementation and Comparisons

We implemented ASVM in Visual C++ 6.0 under Windows NT 4.0. The experiments were run on the UW-Madison Data Mining Institute Locop2 machine, which utilizes a 400 MHz Pentium II Xeon Processor and a maximum of 2 Gigabytes of memory available per process. We wrote all the code ourselves except for the linear equation solver, for which we used CLAPACK [1, 20]. Our stopping criterion for ASVM is triggered when the error bound residual [73] $\|u - (u - Qu + e)_+\|$, which is zero at the solution of (5.10), goes below 0.1.

The first set of experiments are designed to show that our reformulation (5.7) of the SVM (5.6) and its associated algorithm ASVM yield similar performance to the standard SVM (5.2), referred to here as SVM-QP. For six datasets available from the UCI Machine

Learning Repository [85], we performed tenfold cross validation in order to compare test set accuracies between ASVM and SVM-QP. We implemented SVM-QP using the high-performing CPLEX barrier quadratic programming solver [47], and utilized a tuning set for both algorithms to find the optimal value of the parameter ν, using the default stopping criterion of CPLEX. Altering the CPLEX default stopping criterion to match that of ASVM did not result in significant change in timing relative to ASVM, but did reduce test set correctness for SVM-QP. Some algorithms for SVMs, such as SVM^{light} [48], recommend normalizing the data before optimizing. We show the results for both normalized data (by subtracting the mean and dividing by the standard deviation) and for unnormalized data in Table 5.1. ASVM performs comparably to SVM with respect to generalizability, but is considerably faster (and much faster than SVM^{light}).

		Normalized			Not Normalized		
Dataset m x n	Algorithm	Training Correctness	Testing Correctness	Time (CPU sec)	Training Correctness	Testing Correctness	Time (CPU sec)
Liver Disorders 345 x 6	SVM–QP	70.69%	66.95%	9.89	70.69%	68.41%	8.00
	ASVM	70.11%	68.39%	0.03	69.37%	67.55%	0.03
Cleveland Heart 297 x 13	SVM–QP	87.84%	85.20%	5.06	87.84%	85.20%	4.76
	ASVM	87.36%	84.86%	0.06	87.39%	84.86%	0.06
Pima Diabetes 768 x 8	SVM–QP	77.91%	77.21%	166.29	77.91%	77.21%	138.40
	ASVM	78.07%	77.99%	0.10	78.08%	77.99%	0.10
Ionosphere 351 x 34	SVM–QP	90.25%	87.18%	1.23	90.25%	87.18%	1.16
	ASVM	90.00%	87.46%	0.16	90.00%	87.46%	0.15
Tic Tac Toe 958 x 9	SVM–QP	65.34%	65.34%	178.63	65.34%	65.34%	173.71
	ASVM	70.22%	69.72%	0.05	70.22%	69.72%	0.04
Votes 435 x 16	SVM–QP	97.55%	95.85%	59.02	97.55%	95.85%	46.02
	ASVM	97.34%	96.31%	0.11	97.34%	96.31%	0.09

Table 5.1: Comparison of ASVM and the conventional SVM-QP on UCI datasets. ASVM test correctness is comparable to that of SVM-QP, but much faster.

# of Points	Iterations	Training Correctness	Testing Correctness	Time (CPU min)
4 million	5	86.09%	86.06%	38.04
7 million	5	86.10%	86.28%	95.57

Table 5.2: Performance of ASVM on NDC generated datasets in R^{32}. ($\nu = 0.01$)

The second set of experiments show that ASVM performs well on massive datasets. We created synthetic data of Gaussian distribution by using our own NDC Data Generator [87] as suggested by Usama Fayyad. A more detailed description of this data generation method can be found in Section 4.5. The results of our experiments are shown in Table 5.2. Note that for these experiments, all the data was brought into memory. As such, the running time reported consists of the time used to actually solve the problem to termination excluding I/O time. This is consistent with the measurement techniques used by other popular approaches [48, 95]. Putting all the data in memory is simpler to code and results in faster running times. However, it is not a fundamental requirement of our algorithm — block matrix multiplications, incremental evaluations of Q^{-1} using another application of the SMW formula, and indices on the dataset can be used to create an efficient disk based version of ASVM.

Our code is available free for academic and research use at [88], and we encourage others in the community to use it.

As shown above, ASVM produces extremely fast solution times for SVMs, though the algorithm involves some "special case" considerations in order to guarantee convergence. The next chapter will present an even simpler algorithm with guaranteed convergence properties, and fast solution times as well.

Chapter 6

Lagrangian Support Vector Machines

6.1 Introduction

In this chapter we propose a very fast simple algorithm, based on an an implicit Lagrangian formulation [75] of the dual of a simple reformulation of the standard quadratic program of a linear support vector machine. This leads to the minimization of an **unconstrained** differentiable convex function in an m-dimensional space where m is the number of points to be classified in a given n dimensional input space. The necessary optimality condition for this unconstrained minimization problem can be transformed into a very simple symmetric positive definite complementarity problem (6.6). A linearly convergent iterative Lagrangian support vector machine (LSVM) Algorithm 6.1 is given for solving (6.6). LSVM requires the solution, once, of a system of linear equations defined by a positive definite matrix of the order of the dimensionality of the original input space plus one: $(n + 1)$. The algorithm can accurately solve problems with millions of points and requires only standard native MATLAB commands without any optimization tools such as as linear or quadratic programming solvers.

As was the case in Chapter 5, the following two simple changes were made to the standard linear SVM: (i) The margin (distance) between the parallel bounding planes was maximized with respect to both orientation (w) as well as location relative to the origin (γ). Such a change was also carried out in the successive overrelaxation (SOR) approach of Chapter 3 as well as in the smooth support vector machine (SSVM) approach of [53]. See equation (6.1). (ii) The error in the soft margin (y) was minimized using the 2-norm squared instead of the conventional 1-norm. See equation (6.1). These straightforward, but important changes, lead to a considerably simpler positive definite dual problem with nonnegativity constraints only. See equation (6.2).

In Section 6.2 we begin with the standard SVM formulation and its dual and then give our formulation and its dual. We note that computational evidence given in Table 5.1 shows that this alternative formulation does not compromise on generalization ability. Section 6.3 gives our simple iterative Lagrangian support vector machine (LSVM) Algorithm 6.1 and establishes its global linear convergence. LSVM, stated in 11 lines of MATLAB Code 6.1 below, solves **once** at the outset a single system of $n + 1$ equations in $n + 1$ variables given by a symmetric positive definite matrix. It then uses a linearly convergent iterative method to solve the problem. In Section 6.4 LSVM is extended to positive semidefinite kernels. Section 6.5 describes our numerical results which show that LSVM gives comparable or better testing results than those of other SVM algorithms, and in some cases is dramatically faster than a standard quadratic programming SVM solver.

6.2 The Linear Support Vector Machine

As in Chapter 5, we start with our reformulated SVM (5.6):

$$\begin{array}{rl} \displaystyle\min_{(w,\gamma,y)\in R^{n+1+m}} & \nu\frac{y'y}{2}+\frac{1}{2}(w'w+\gamma^2) \\ \text{s.t.} & D(Aw-e\gamma)+y \ \geq \ e. \end{array} \tag{6.1}$$

The dual of this problem is [62]:

$$\min_{0\leq u\in R^m}\frac{1}{2}u'(\frac{I}{\nu}+D(AA'+ee')D)u-e'u. \tag{6.2}$$

The variables (w,γ) of the primal problem which determine the separating surface are recovered directly from the solution of the dual (6.2) above by the relations:

$$w=A'Du,\ \ y=\frac{u}{\nu},\ \ \gamma=-e'Du. \tag{6.3}$$

We immediately note that the matrix appearing in the dual objective function is positive definite and that there is no equality constraint and no upper bound on the dual variable u. The only constraint present is a nonnegativity one. These facts lead us to our simple iterative Lagrangian SVM Algorithm which requires the solution of a single system of linear equations, determined by an $(n+1)\times(n+1)$ matrix, at the beginning of the algorithm followed by a straightforward linearly convergent iterative scheme that requires no optimization package.

6.3 LSVM (Lagrangian Support Vector Machine) Algorithm

Before stating our algorithm we define two matrices to simplify notation as follows:

$$H=D[A\quad -e],\qquad Q=\frac{I}{\nu}+HH'. \tag{6.4}$$

With these definitions the dual problem (6.2) becomes

$$\min_{0 \le u \in R^m} f(u) := \frac{1}{2} u' Q u - e' u. \tag{6.5}$$

It will be understood that within the LSVM Algorithm, the single time that Q^{-1} is computed at the outset of the algorithm, the SMW identity (5.1) will be used. Hence only an $(n+1) \times (n+1)$ matrix is inverted.

The LSVM Algorithm is based directly on the Karush-Kuhn-Tucker necessary and sufficient optimality conditions [62, KTP 7.2.4, page 94] for the dual problem (6.5):

$$0 \le u \ \perp \ Qu - e \ge 0. \tag{6.6}$$

By using the easily established identity between any two real numbers (or vectors) a and b:

$$0 \le a \ \perp \ b \ge 0 \iff a = (a - \alpha b)_+, \ \alpha > 0, \tag{6.7}$$

the optimality condition (6.6) can be written in the following equivalent form for any positive α:

$$Qu - e = ((Qu - e) - \alpha u)_+. \tag{6.8}$$

These optimality conditions lead to the following very simple iterative scheme which constitutes our LSVM Algorithm:

$$u^{i+1} = Q^{-1}(e + ((Qu^i - e) - \alpha u^i)_+), \ i = 0, 1, \ldots, \tag{6.9}$$

for which we will establish global linear convergence from any starting point under the easily satisfiable condition:

$$0 < \alpha < \frac{2}{\nu}. \tag{6.10}$$

We impose this condition as $\alpha = 1.9/\nu$ in all our experiments, where ν is the parameter of our SVM formulation (6.1). It turns out, and this is the way that led us to this iterative scheme, that the optimality condition (6.8), is also the necessary and sufficient condition for the unconstrained minimum of the implicit Lagrangian [75] associated with the dual problem (6.5):

$$\min_{u \in R^m} L(u, \alpha) = \min_{u \in R^m} \frac{1}{2}u'Qu - e'u + \frac{1}{2\alpha}(\|(-\alpha u + Qu - e)_+\|^2 - \|Qu - e\|^2). \quad (6.11)$$

Setting the gradient with respect to u of this convex and differentiable Lagrangian to zero gives

$$(Qu - e) + \frac{1}{\alpha}(Q - \alpha I)((Q - \alpha I)u - e)_+ - \frac{1}{\alpha}Q(Qu - e) = 0, \quad (6.12)$$

or equivalently:

$$(\alpha I - Q)((Qu - e) - ((Q - \alpha I)u - e)_+) = 0, \quad (6.13)$$

which is equivalent to the optimality condition (6.8) under the assumption that α is positive and not an eigenvalue of Q.

We establish now the global linear convergence of the iteration (6.9) under condition (6.10).

Algorithm 6.1 (LSVM Algorithm & Its Global Convergence) *Let $Q \in R^{m \times m}$ be the symmetric positive definite matrix defined by (6.4) and let (6.10) hold. Starting with an arbitrary $u^0 \in R^m$, the iterates u^i of (6.9) converge to the unique solution $\bar{u}$ of (6.5) at the linear rate:*

$$\|Qu^{i+1} - Q\bar{u}\| \leq \|I - \alpha Q^{-1}\| \cdot \|Qu^i - Q\bar{u}\|. \quad (6.14)$$

Proof Because $\bar{u}$ is the solution of (6.5), it must satisfy the optimality condition (6.8) for any $\alpha > 0$. Subtracting that equation with $u = \bar{u}$ from the iteration (6.9) premultiplied by Q and taking norms gives:

$$\|Qu^{i+1} - Q\bar{u}\| = \|(Qu^i - e - \alpha u^i)_+ - (Q\bar{u} - e - \alpha\bar{u})_+\| \tag{6.15}$$

Using the Projection Theorem [5, Proposition 2.1.3] which states that the distance between any two points in R^m is not less than the distance between their projections on any convex set (the nonnegative orthant here) in R^m, the above equation gives:

$$\begin{array}{rcl} \|Qu^{i+1} - Q\bar{u}\| & \leq & \|(Q - \alpha I)(u^i - \bar{u})\| \\ & \leq & \|I - \alpha Q^{-1}\| \cdot \|Q(u^i - \bar{u})\|. \end{array} \tag{6.16}$$

All we need to show now is that $\|I - \alpha Q^{-1}\| < 1$. This follows from (6.10) as follows. Noting the definition (6.4) of Q and letting λ_i, $i = 1, \ldots, m$, denote the nonnegative eigenvalues of HH', all we need is:

$$-1 < 1 - \alpha(\frac{1}{\nu} + \lambda_i)^{-1} < 1, \tag{6.17}$$

or equivalently:

$$2 > \alpha(\frac{1}{\nu} + \lambda_i)^{-1} > 0, \tag{6.18}$$

which is satisfied under the assumption (6.10).$\Box$

We give now a complete MATLAB [76] code of LSVM which is capable of solving problems with millions of points using only native MATLAB commands. The input parameters, besides A, D and ν of (6.4), which define the problem, are: itmax, the maximum number of iterations and tol, the tolerated nonzero error in $\|u^{i+1} - u^i\|$ at termination. The quantity $\|u^{i+1} - u^i\|$ bounds from above:

$$\|Q\|^{-1} \cdot \|Qu^i - e - ((Qu^i - e) - \alpha u^i)_+\|, \tag{6.19}$$

which measures the violation of the optimality criterion (6.8). It follows by [73] that $\|u^{i+1} - u^i\|$ also bounds $\|u^i - \bar{u}\|$, and by (6.3) it also bounds $\|w^i - \bar{w}\|$ and $\|\gamma^i - \bar{\gamma}\|$, where $(\bar{w}, \bar{\gamma}, \bar{y})$ is the unique solution of the primal SVM (6.1).

Code 6.1 (LSVM MATLAB Code)

```
function [it, opt, w, gamma] = svml(A,D,nu,itmax,tol)
% lsvm with SMW for min 1/2*u'*Q*u-e'*u s.t. u=>0,
% Q=I/nu+H*H', H=D[A -e]
% Input: A, D, nu, itmax, tol; Output: it, opt, w, gamma
% [it, opt, w, gamma] = svml(A,D,nu,itmax,tol);
  [m,n]=size(A);alpha=1.9/nu;e=ones(m,1);H=D*[A -e];it=0;
  S=H*inv((speye(n+1)/nu+H'*H));
  u=nu*(1-S*(H'*e));oldu=u+1;
  while it<itmax & norm(oldu-u)>tol
    z=(1+pl(((u/nu+H*(H'*u))-alpha*u)-1));
    oldu=u;
    u=nu*(z-S*(H'*z));
    it=it+1;
  end;
  opt=norm(u-oldu);w=A'*D*u;gamma=-e'*D*u;

function pl = pl(x); pl = (abs(x)+x)/2;
```

6.4 LSVM for Nonlinear Kernels

In this section we show how LSVM can be used to solve classification problems with positive semidefinite nonlinear kernels. Algorithm 6.1 and its convergence can be extended for such nonlinear kernels as we show below. The only price paid for this extension is that problems with large datasets can be handled using the Sherman-Morrison-Woodbury (SMW) identity (5.1) only if the inner product terms of the kernel [64, Equation (3)] are explicitly known, which in general they are not. Nevertheless LSVM may be a useful tool for classification with nonlinear kernels because of its extreme simplicity as we demonstrate below with the simple MATLAB code which does not make use of the Sherman-Morrison-Woodbury identity nor any optimization package.

We shall use the notation of [64]. For $A \in R^{m\times n}$ and $B \in R^{n\times \ell}$, the **kernel** $K(A, B)$

maps $R^{m\times n}\times R^{n\times \ell}$ into $R^{m\times \ell}$. A typical kernel is the Gaussian kernel

$$[K(A,B)]_{i,j} = \exp(-\mu\|A_i - B_j\|^2),\ i,j = 1,\dots,m,\ \ell = m,$$

while a linear kernel is $K(A,B) = AB$. For a column vector x in R^n, $K(x', A')$ is a row vector in R^m, and the linear separating surface (5.3) is replaced by the nonlinear surface

$$K([x' \quad -1], \begin{bmatrix} A' \\ -e' \end{bmatrix})Du = 0, \tag{6.20}$$

where u is the solution of the dual problem (6.5) with Q re-defined for a general nonlinear kernel as follows:

$$G = [A \quad -e], \qquad Q = \frac{I}{\nu} + DK(G,G')D. \tag{6.21}$$

Note that the nonlinear separating surface (6.20) degenerates to the linear one if we let $K(G,G') = GG'$ and make use of (6.3).

To justify the nonlinear kernel formulation (6.21) we refer the reader to [64, Equation (8.9)] for a similar result and give a brief derivation here. If we rewrite our dual problem for a linear kernel (6.2) in the equivalent form:

$$\min_{0\le u\in R^m} \frac{1}{2}u'(\frac{I}{\nu} + DGG'D)u - e'u, \tag{6.22}$$

and replace the linear kernel GG' by a general nonlinear positive semidefinite symmetric kernel $K(G,G')$ we obtain:

$$\min_{0\le u\in R^m} \frac{1}{2}u'(\frac{I}{\nu} + DK(G,G')D)u - e'u. \tag{6.23}$$

This is the formulation given above in (6.21). We note that the Karush-Kuhn-Tucker necessary and sufficient optimality conditions for this problem are:

$$0 \le u \quad \perp \quad (\frac{I}{\nu} + DK([A \quad -e], \begin{bmatrix} A' \\ -e' \end{bmatrix})u - e \ge 0, \tag{6.24}$$

which underly LSVM for a nonlinear positive semidefinite kernel $K(G, G')$. The positive semidefiniteness of the nonlinear kernel $K(G, G')$ is needed in order to ensure the existence of a solution to both (6.23) and (6.24).

All the results of the previous section remain valid, with Q re-defined as above for any positive semidefinite kernel K. This includes the iterative scheme (6.9) and the convergence result given under the Algorithm 6.1. However, because we do not make use of the Sherman-Morrison-Woodbury identity for a nonlinear kernel, the LSVM MATLAB Code 6.1 is somewhat different and is as follows:

Code 6.2 (LSVM MATLAB Code for Nonlinear Kernel $K(\cdot, \cdot)$)

```
function [it, opt,u] = svmlk(nu,itmax,tol,D,KM)
% lsvm with nonlinear kernel for min 1/2*u'*Q*u-e'*u s.t. u=>0
% Q=I/nu+DK(G,G')D, G=[A -e]
% Input: nu, itmax, tol, D,  KM=K(G,G')
% [it, opt,u] = svmlk(nu,itmax,tol,D,KM);
  m=size(KM,1);alpha=1.9/nu;e=ones(m,1);I=speye(m);it=0;
  Q=I/nu+D*KM*D;P=inv(Q);
  u=P*e;oldu=u+1;
  while it<itmax & norm(oldu-u)>tol
    oldu=u;
    u=P*(1+pl(Q*u-1-alpha*u));
    it=it+1;
  end;
  opt=norm(u-oldu);[it opt]

function pl = pl(x); pl = (abs(x)+x)/2;
```

6.5 Numerical Implementation and Comparisons

The implementation of LSVM is straightforward, as shown in the previous sections. Our first "dry run" experiments were on randomly generated problems just to test the speed and effectiveness of LSVM on large problems. We first used a Pentium III 500 MHz notebook with 384 megabytes of memory (and additional swap space) on 2 million randomly generated points in R^{10} with $\nu = \frac{1}{m}$ and $\alpha = \frac{1.9}{\nu}$. LSVM solved the problem in 6 iterations in 81.52 minutes to an optimality criterion of $9.398 \cdot 10^{-5}$ on a 2-norm violation of (6.8). The same problem was solved in the same number of iterations and to the same accuracy in 6.74 minutes on a 250 MHz UltraSPARC II processor with 2 gigabytes of memory. After these preliminary encouraging tests we proceeded to more systematic numerical tests as follows.

Most of the rest of our experiments were run on the UW-Madison Data Mining Institute Locop2 machine, which utilizes a 400 MHz Pentium II Xeon processor and a maximum of 2 Gigabytes of memory available for each process. This computer runs Windows NT Server 4.0, with MATLAB 5.3.1. Since SVM^{light} is easier to install on a UNIX machine, the one set of experiments comparing LSVM to SVM^{light} were run on the UW-Madison Ironsides cluster, which utilizes 250 MHz UltraSPARC II processors with a maximum of 8 Gigabytes of memory available which in general is shared with up to a dozen other users. Ironsides runs MATLAB 5.3.0 under an experimental version of Solaris 5.6.

The first results we present are designed to show that our reformulation of the SVM (6.1) and its associated algorithm LSVM yield similar performance to the standard SVM (5.5), referred to here as SVM-QP. Results are also shown for our active set SVM (ASVM) algorithm which we described in Chapter 5. For six datasets available from the UCI Machine Learning Repository [85], we performed tenfold cross validation in order to compare test set accuracies between the methodologies. Moreover, we utilized a tuning set for each algorithm to find the optimal value of the parameter ν. For both LSVM and

ASVM, we used an optimality tolerance of 0.001 to determine when to terminate. SVM-QP was implemented using the high-performing CPLEX barrier quadratic programming solver [47] with its default stopping criterion. Altering the CPLEX default stopping criterion to match that of LSVM did not result in significant change in timing relative to LSVM, but did reduce test set correctness for SVM-QP. Some algorithms for SVMs, such as SVM^{light} [48], recommend normalizing the data before optimizing. We show the results for both normalized data (by subtracting the mean and dividing by the standard deviation) and for unnormalized data in Table 6.1.

The careful reader may observe that the numbers in Table 6.1 are different from those presented in Table 5.1. This is due to two factors. The experiments shown in Tables 5.1 and 6.1 were run with different seeds for the random cross-validation process. Additionally, the stopping criterion for ASVM is different in the two tables. For Table 5.1, an optimality tolerance of 0.1 was used. This was to demonstrate that good generalization performance could be achieved while rarely invoking the commented portions of Algorithm 5.1. The experiments in Table 6.1 were less lenient, as they used an optimality tolerance of 0.001 for both ASVM and LSVM.

The results demonstrate that LSVM performs comparably to SVM-QP with respect to generalizability but is considerably faster. LSVM and ASVM show nearly identical generalization performance, as they solve the same optimization problem. Any differences in generalization observed between the two are only due to different approximate solutions being found at termination. LSVM and ASVM usually run in roughly the same amount of time, though there are some cases where ASVM is noticeably faster than LSVM. Nevertheless, this is impressive performance on the part of LSVM, which is a dramatically simpler algorithm than both ASVM and the one used to solve SVM-QP.

We next compare LSVM with SVM^{light} on the Adult dataset [85], which is commonly used to compare SVM algorithms [96, Chapter 3 in this thesis]. These results, shown in Table 6.2, demonstrate that for the largest training sets LSVM performs faster than SVM^{light} with similar test set accuracies. Note that the "iterations" column takes on very different meanings for the two algorithms. SVM^{light} defines an iteration as solving an optimization problem over a a small number, or "chunk," of constraints. LSVM, on the other hand, defines an iteration as a matrix calculation which updates all the dual variables simultaneously. These two numbers are not directly comparable, and are included here only for purposes of monitoring scalability.

Table 6.3 shows results from running LSVM on a massively sized dataset. This dataset was created using our own NDC Data Generator [87] as suggested by Usama Fayyad. The results show that LSVM can be used to solve massive problems quickly, which is particularly intriguing given the simplicity of the LSVM Algorithm. Note that for these experiments, all the data was brought into memory. As such, the running time reported consists of the time used to actually solve the problem to termination excluding I/O time. This is consistent with the measurement techniques used by other popular approaches [48, 95]. Putting all the data in memory is simpler to code and results in faster running times. However, it is not a fundamental requirement of our algorithm — block matrix multiplications, incremental evaluations of Q^{-1} using another application of the Sherman-Morrison-Woodbury identity, and indices on the dataset can be used to create an efficient disk based version of LSVM.

Dataset m x n	Algorithm	Normalized			Not Normalized		
		Training Correctness	Testing Correctness	Time (CPU sec)	Training Correctness	Testing Correctness	Time (CPU sec)
Liver Disorders 345 x 6	SVM−QP	70.69%	68.41%	*8.45	70.69%	68.41%	*8.87
	ASVM	70.21%	68.68%	*0.03	70.24%	67.24%	*0.03
	LSVM	70.21%	68.68%	0.06	70.24%	67.24%	0.09
Cleveland Heart 297 x 13	SVM−QP	*87.39%	85.23%	*5.03	87.24%	86.55%	*4.94
	ASVM	86.12%	85.91%	*0.03	87.06%	85.20%	*0.05
	LSVM	86.12%	85.91%	0.003	87.06%	85.20%	0.14
Pima Diabetes 768 x 8	SVM−QP	77.94%	78.00%	*171.6	*76.95%	76.69%	*198.76
	ASVM	78.02%	77.99%	*0.08	78.05%	77.99%	*0.08
	LSVM	78.02%	77.99%	0.24	78.05%	77.99%	0.40
Ionosphere 351 x 34	SVM−QP	*92.72%	88.03%	*1.33	*92.72%	88.03%	*1.33
	ASVM	94.05%	89.17%	0.98	94.05%	89.17%	0.96
	LSVM	94.02%	89.17%	0.93	94.02%	89.17%	0.93
Tic Tac Toe 958 x 9	SVM−QP	*65.34%	65.34%	*208.55	*65.34%	65.34%	*224.85
	ASVM	70.32%	69.72%	*0.05	70.17%	69.61%	*0.05
	LSVM	70.32%	69.72%	0.02	70.17%	69.61%	0.01
Votes 435 x 16	SVM−QP	95.30%	95.16%	*26.90	95.27%	95.16%	*26.35
	ASVM	95.30%	94.70%	*0.08	95.27%	95.16%	*0.08
	LSVM	95.30%	94.70%	0.06	95.27%	95.16%	0.06

Table 6.1: Comparison of LSVM with SVM-QP and ASVM on UCI datasets. LSVM test correctness is comparable to that of SVM-QP, but much faster, e.g. four orders of magnitude faster than SVM-QP on non-normalized Tic Tac Toe. An asterisk indicates that the results are significantly different from LSVM at significance level $\alpha = 0.05$.

Training Set Size		CPU Sec	Iterations	Test Set Accuracy
		SVM^{light}	SVM^{light}	SVM^{light}
		LSVM	LSVM	LSVM
1605	SVM^{light}	2.0	149	84.05%
	LSVM	3.3	38	84.27%
2265	SVM^{light}	3.6	299	84.38%
	LSVM	5.0	40	84.66%
3185	SVM^{light}	5.1	401	84.22%
	LSVM	8.1	43	84.55%
4781	SVM^{light}	7.2	574	84.33%
	LSVM	13.4	46	84.55%
6414	SVM^{light}	13.5	729	84.47%
	LSVM	18.8	47	84.68%
11221	SVM^{light}	25.5	1154	84.58%
	LSVM	38.9	50	84.84%
16101	SVM^{light}	49.2	1831	84.81%
	LSVM	60.5	52	85.01%
22697	SVM^{light}	95.6	2622	85.16%
	LSVM	92.0	54	85.35%
32562	SVM^{light}	178.3	4497	85.02%
	LSVM	140.9	55	85.05%

Table 6.2: Comparison of LSVM with SVM^{light} on the UCI adult dataset. LSVM test correctness is comparable to that of SVM^{light}, but is faster on large datasets. ($\nu = 0.03$)

We now demonstrate the effectiveness of LSVM in solving nonlinear classification problems through the use of kernel functions. One highly nonlinearly separable but simple example is the "tried and true" checkerboard dataset [43], which has often been used to show the effectiveness of nonlinear kernel methods on a dataset for which a linear separation clearly fails. The checkerboard dataset contains 1000 points randomly sampled from a checkerboard. These points are used as a training set in LSVM to try to reproduce an accurate rendering of a checkerboard. For this problem, we used the

# of Points	# of Attributes	Iterations	Training Correctness	Testing Correctness	Time (CPU min)
2 million	10	81	69.80%	69.44%	33.6

Table 6.3: Performance of LSVM on NDC generated dataset. ($\nu = 0.1$)

following Gaussian kernel:

$$K(G, G') = \exp(-2 \cdot 10^{-4} \|G_i' - G_{\cdot j}\|_2^2), i, j = 1, \ldots, m \tag{6.25}$$

Our results, shown in Figure 6.1, demonstrate that LSVM shows similar or better generalization capability on this dataset when compared to other methods [50, 53, Figure 4.4 in this thesis]. Total time for the checkerboard problem using LSVM with a Gaussian kernel was 2.85 hours on the Locop2 Pentium II Xeon machine on the 1000-point training set in R^2 after 100,000 iterations. Test set accuracy was 97.0% on a 39,000-point test set. However, within 58 seconds and after 100 iterations, a quite reasonable checkerboard was obtained with a 95.9% accuracy on the same test set. This is shown in Figure 6.2.

The final set of experiments, shown in Table 6.4, demonstrates the effectiveness of nonlinear kernel functions with LSVM on UCI datasets [85] specialized as follows for purposes of our tests:

- The *liver-disorders* dataset contains 345 points, each consisting of six features. Class 1 contains 145 points, and class -1 contains 200 points.

- The *letter-recognition* dataset is used for recognizing letters of the alphabet. Traditionally, this dataset is used for multi-category classification. We used it here in a two class situation by taking a subset of those points which correspond to the letter "A," and a subset of those points which correspond to the letter "B." This resulted in a dataset of 600 points with 6 features, where each class contained 300 points.

- The *mushroom* dataset is a two class dataset which contains eight categorical attributes. We transformed each categorical attribute into a series of binary attributes, one attribute for each distinct value. For example, this dataset contains

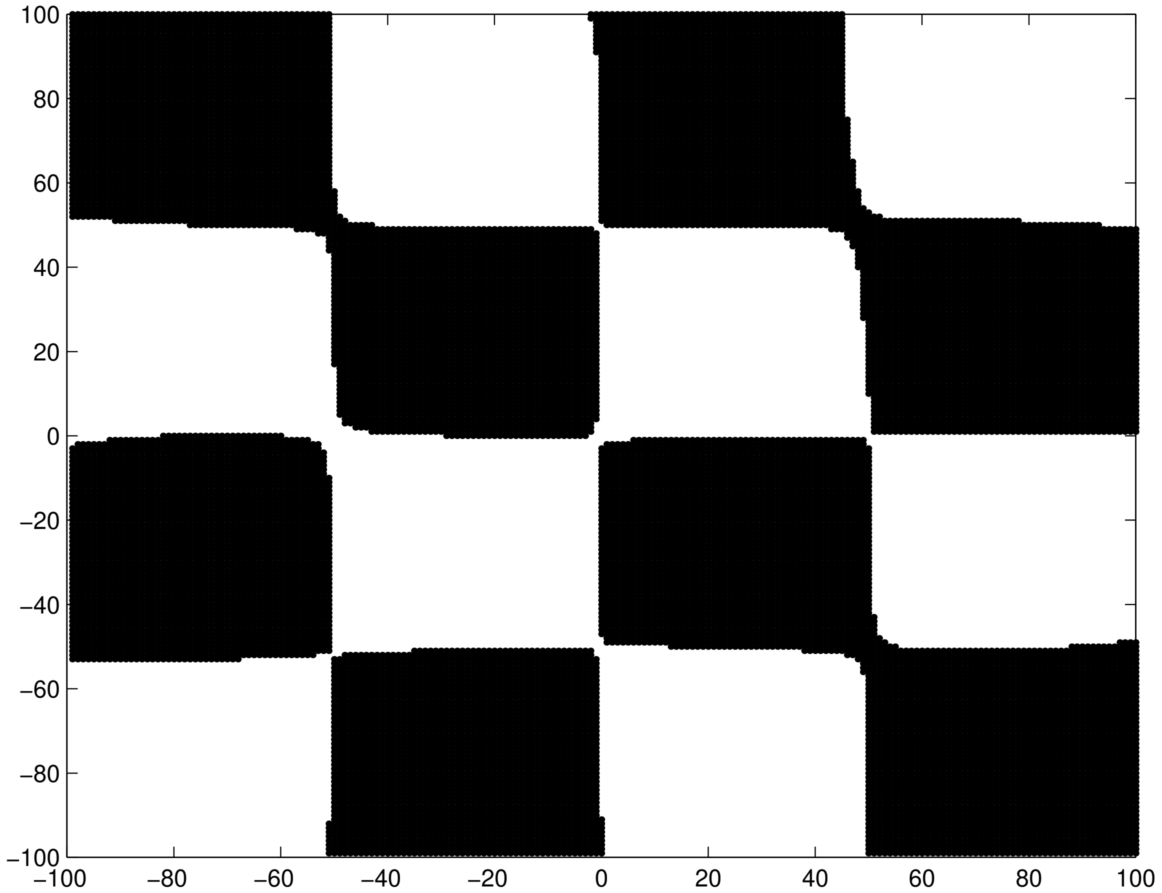

Figure 6.1: Gaussian kernel LSVM performance on checkerboard training dataset. ($\nu = 10^5$)

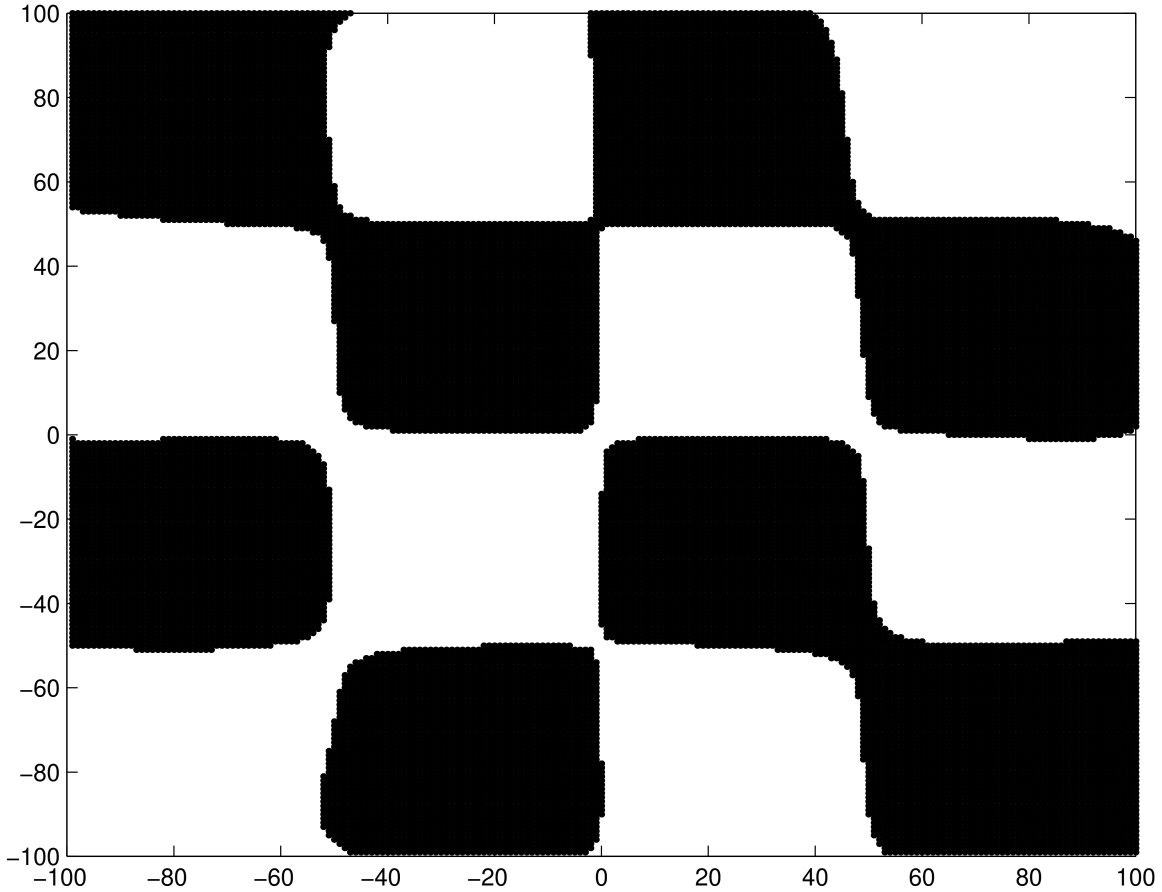

Figure 6.2: Gaussian kernel LSVM performance with early stopping on checkerboard training dataset. ($\nu = 10^5$)

an attribute called "cap surface," which can take one one of four categories, namely "fibrous," "grooves," "scaly," or "smooth." We represented this as four binary attributes. A 1 is assigned to the attribute that corresponds to the actual category, and a 0 to the rest. Thus the categorical value "fibrous" would be represented by [1 0 0 0], while the value "smooth" would be represented by [0 0 0 1]. A subset of the entire mushroom dataset was used, so as to shorten the running times of our experiments. The final dataset we used contained 22 features with 200 points in class 1 and 300 points in class -1.

- The *tic-tac-toe* dataset is a two class dataset that contains legal complete tic-tac-toe games. All those games where "X" has a "win" (three in a row) end up in category 1, and all other games end up in category -1. We have intentionally represented this problem with a poor representation scheme to show the power of non-linear kernels in overcoming this difficulty. Each tic-tac-toe game is represented by 9 attributes, where each attribute corresponds to a spot on the tic-tac-toe board. An "X" is represented by 1, an "O" is represented by 0, and a blank is represented by -1. We used a subset of this dataset with 9 attributes, 200 points in class 1, and 100 points in class -1.

We chose which kernels to use and appropriate values of ν via tuning sets and tenfold cross-validation techniques. The results show that nonlinear kernels produce test set accuracies that improve on those obtained with linear kernels.

The previous four chapters have presented a number of algorithms for dealing with SVM classification problems. We now leave behind the classification problem, and consider the regression problem for the remainder of this thesis.

Dataset m x n	Kernel Type	Training Correctness	Testing Correctness	Time (secs)
Liver Disorders	Linear	70.15%	67.53%	0.08
345 x 6	Quadratic	76.36%	71.63%	0.97
Tic–Tac–Toe	Linear	66.85%	63.67%	0.08
300 x 9	Quadratic	100.00%	93.33%	4.48
Letter Recognition	Linear	83.46%	82.33%	0.38
600 x 6	Cubic	89.69%	88.00%	7.45
Mushroom	Linear	87.29%	87.20%	0.25
500 x 22	Cubic	89.84%	89.20%	3.08

Table 6.4: LSVM performance with linear, quadratic, and cubic kernels.

Chapter 7

Massive Support Vector Regression

7.1 Introduction

We now shift our attention to the regression problem. As described in Chapter 1, regression is concerned with predicting some continuous numerical value associated with each point. This is different from the classification problem which we have considered so far, which was only concerned with assigning a discrete binary class assignment to each point. Regression can be formulated by a support vector machine, using much of the same ideas we have seen in classification. Tolerating a small error in fitting a given set of data, i.e. disregarding errors that fall within some positive ε, can improve test set correctness over a standard zero tolerance error fit [113]. Schölkopf et al [100, 101] use quadratic-programming-based support vector machines to automatically generate an ε-insensitive "tube" around the data within which errors are discarded. In [110], Smola et al use a linear-programming-based support vector machine approach for ε-insensitive approximation. In this work we simplify the linear programming formulation of [110] by using a fewer number of variables. This simplification leads to considerably shorter computing times as well as a more efficient implementation of a generalization of the linear programming chunking algorithm of [8] for very large datasets. This generalization consists of chunking *both* data points and problem variables (row and column chunking) which allows us to solve linear programs with more than a billion nonzero matrix elements.

We briefly outline the contents of this chapter. In Section 7.2 we derive and give the theoretical justification of our linear programming formulation (7.7) based on the loss function of Figure 7.1 and show (Lemma 7.1) that it is equivalent to the linear program (7.9) of [110] but with roughly 75% of the number of variables. Proposition 7.1 shows that the tolerated error interval length 2ε varies directly with the size of the parameter $\mu \in [0, 1]$ of our linear program (7.7), where $\mu = 0$ corresponds to a least 1-norm fit of the data (7.8) while the value $\mu = 1$ disregards the data and is therefore meaningless. It turns out, that depending on the amount of noise in the data, some positive value of $\mu < 1$ gives a best fit as indicated by the computational results summarized in Table 7.1. Section 7.3 implements and tests our linear programming formulation (7.7) and compares it with the linear programming formulation (7.9) of Smola et al [110]. Our formulation yielded an average computational time reduction as high as 26.0% on average, with a maximal time reduction of 79.7%. Our chunking implementation solved problems with as many as 10,000 observations. By contrast the largest dataset attempted in [110] contained 506 points.

7.2 The Support Vector Regression Problem

We consider a given dataset of m points in n dimensional real space R^n represented by the matrix $A \in R^{m\times n}$. Associated with each point A_i is a given observation, a real number y_i, $i = 1, \ldots, m$. We wish to approximate $y \in R^m$ by some linear or nonlinear function of the matrix A with linear parameters, such as the simple linear approximation:

$$Aw + be \approx y, \tag{7.1}$$

where $w \in R^n$ and $b \in R^1$ are parameters to be determined by minimizing some error criterion. If we consider w to be a linear combination of the rows of A, i.e. let $w = A'\alpha$, $\alpha \in R^m$, then we have:

$$AA'\alpha + be \approx y. \tag{7.2}$$

This immediately suggests the much more general idea of replacing the linear kernel AA' by some arbitrary nonlinear kernel $K(A, A') : R^{m\times n} \times R^{n\times m} \longrightarrow R^{m\times m}$ that leads to the following approximation, which may be nonlinear in A but linear in α:

$$K(A, A')\alpha + be \approx y. \tag{7.3}$$

We will measure the error in (7.3) by a vector $s \in R^m$ defined by:

$$-s \le K(A, A')\alpha + be - y \le s, \tag{7.4}$$

which we modify by tolerating a small fixed positive error ε, possibly to overlook errors in the observations y, as follows:

$$\begin{array}{rcccl} -s & \le & K(A, A')\alpha + be - y & \le & s \\ & & e\varepsilon & \le & s \end{array} \tag{7.5}$$

We now drive the error s no lower than the fixed (for now) tolerance ε by minimizing the 1-norm of the error s together with the 1-norm of α for complexity reduction or stabilization. This leads to the following constrained optimization problem with positive parameter C:

$$\begin{array}{lrcccl} & & \displaystyle\min_{(\alpha,b,s)} & \displaystyle\frac{1}{m}\|\alpha\|_1 + \frac{C}{m}\|s\|_1 & & \\ \text{s.t.} & -s & \le & K(A, A')\alpha + be - y & \le & s \\ & & & e\varepsilon & \le & s, \end{array} \tag{7.6}$$

which can be represented as a linear program. For a linear kernel and a *fixed* tolerance ε, this is essentially the model proposed in [113], which utilizes a 2-norm instead of a 1-norm. We now allow ε to be a *variable* in the optimization problem above that will be driven to some positive tolerance error determined by the size of the parameter μ. By making use of linear programming perturbation theory [66] we parametrically maximize ε in the objective function with a positive parameter μ to obtain our basic linear programming formulation of the nonlinear support vector regression (SVR) problem:

$$\begin{array}{rrcll} & & \displaystyle\min_{(\alpha,b,s,\varepsilon,a)} \frac{1}{m}e'a + \frac{C}{m}e's - C\mu\varepsilon & & \\ \text{s.t.} & -s \leq & K(A,A')\alpha + be - y & \leq & s \\ & & e\varepsilon & \leq & s, \\ & -a \leq & \alpha & \leq & a. \end{array} \tag{7.7}$$

For $\mu = 0$ this problem is equivalent to the classical stabilized least 1-norm error minimization of:

$$\min_{(\alpha,b)} \|\alpha\|_1 + C\|K(A,A')\alpha + be - y\|_1. \tag{7.8}$$

For positive values of μ, the zero-tolerance ε of the stabilized least 1-norm problem is increased monotonically to $\varepsilon = \|y\|_\infty$ when $\mu = 1$. (See Proposition 7.1 below.) Components of the error vector s are not driven below this increasing value of the tolerated error ε. Thus, as μ increases from 0 to 1, ε increases correspondingly over the interval $[0, \|y\|_\infty]$. Correspondingly, an increasing number of error components $(K(A,A')\alpha + be - y)_i$, $i = 1, \ldots, m$, fall within the interval $[-\varepsilon, \varepsilon]$ until all of them do so when $\mu = 1$ and $\varepsilon = \|y\|_\infty$. (See Proposition 7.1 below.)

It turns out that our linear program (7.7) is equivalent to the one proposed by Smola et al [110]:

$$\begin{array}{llcll} \min\limits_{(\alpha^1,\alpha^2,b,\xi^1,\xi^2,\varepsilon)} & & \frac{1}{m}e'(\alpha^1+\alpha^2)+\frac{C}{m}e'(\xi^1+\xi^2)+C(1-\mu)\varepsilon & & \\ \text{s.t.} & -\xi^2-e\varepsilon \;\leq & K(A,A')(\alpha^1-\alpha^2)+be-y & \leq & \xi^1+e\varepsilon \\ & & \alpha^1,\ \alpha^2,\ \xi^1,\ \xi^2 & \geq & 0, \end{array} \tag{7.9}$$

with $4m+2$ variables compared to our $3m+2$ variables. We now show that the two problems are equivalent. Note first that at optimality of (7.9) we have that ${\alpha^1}'\alpha^2 = 0$ and hence for $\alpha = \alpha^1 - \alpha^2$ it follows that $|\alpha| = \alpha^1 + \alpha^2$. We thus have the following result.

Lemma 7.1 (LP Equivalence) *The linear programming formulations (7.7) and (7.9) are equivalent.*

Proof Define the deviation error:

$$d = K(A,A')\alpha + be - y. \tag{7.10}$$

By noting that at an optimal solution of (7.9), $\xi^1 = \max\{d - e\varepsilon, 0\} = (d-e\varepsilon)_+$ and $\xi^2 = \max\{-d - e\varepsilon, 0\} = (-d-e\varepsilon)_+$, the linear program (7.9) can be written as:

$$\min_{(\alpha,b,\varepsilon,d)} \frac{1}{m}\|\alpha\|_1 + \frac{C}{m}(e'(d-e\varepsilon)_+ + e'(-d-e\varepsilon)_+) + C(1-\mu)\varepsilon, \tag{7.11}$$

or equivalently

$$\min_{(\alpha,b,\varepsilon,d)} \frac{1}{m}\|\alpha\|_1 + \frac{C}{m}e'(|d|-e\varepsilon)_+ + C(1-\mu)\varepsilon. \tag{7.12}$$

On the other hand, note that at optimality the variable s of the linear program (7.7) is given by:

$$s = \max\{|d|, e\varepsilon\} = e\varepsilon + \max\{|d|-e\varepsilon, 0\} = e\varepsilon + (|d|-e\varepsilon)_+. \tag{7.13}$$

Thus the linear program (7.7) becomes, upon noting that $\frac{e'e}{m} = 1$:

$$\min_{(\alpha,b,\varepsilon,d)} \quad \frac{1}{m}\|\alpha\|_1 + \frac{C}{m}e'(|d| - e\varepsilon)_+ + C(1-\mu)\varepsilon, \tag{7.14}$$

which is identical to (7.12).□

Note that (7.14) is equivalent to:

$$\min_{(\alpha,b,\varepsilon,d)} \quad \frac{1}{m}\|\alpha\|_1 + \frac{C}{m}(\|(|d| - e\varepsilon)_+\|_1 + m(1-\mu)\varepsilon), \tag{7.15}$$

and Figure 7.1 depicts the loss function $(\|(|d| - e\varepsilon)_+\|_1 + m(1-\mu)\varepsilon)$ of (7.15) as a function of a one dimensional error d with $m = 1$. The loss function is being minimized by the equivalent linear programs (7.7) and (7.9). The linear program (7.7) generates a tolerance $\varepsilon \in [0, \|y\|_\infty]$ corresponding to the parameter $\mu \in [0, 1]$. Within the increasing-size interval $[-\varepsilon, \varepsilon]$, depicted in Figure 7.1, an increasing number of errors corresponding to the constraints of (7.7) fall, i.e.:

$$-\varepsilon \leq (K(A, A')\alpha + be - y)_i \leq \varepsilon, \quad i \in I \subset \{1, \ldots, m\}.$$

By using linear programming perturbation theory we can give a useful interpretation of the role of the parameter $(1-\mu)$ appearing in the formulations above. We note first that the parameter μ lies in the interval $[0, 1]$. This is so because for $\mu > 1$ the linear program (7.9) is unbounded below (let ε go to ∞), while for negative μ the linear program (7.7) is also unbounded below (let ε go to $-\infty$).

Proposition 7.1 (Perturbation parameter interpretation)

(i) For $\mu = 0$ *the linear program (7.7) is equivalent to the classical stabilized least* 1*-norm approximation problem (7.8).*

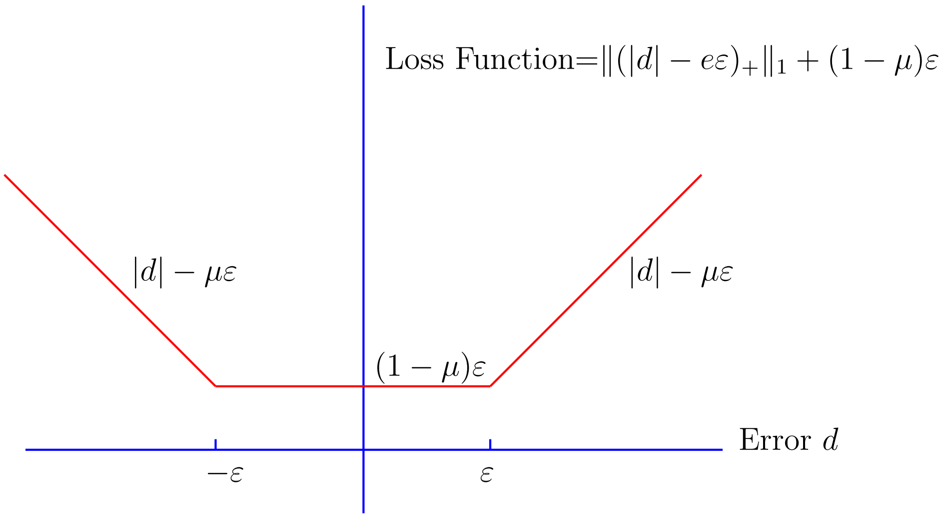

Figure 7.1: One dimensional loss function minimized

(ii) *For all* $\mu \in [0, \bar{\mu}]$ *for some fixed* $\bar{\mu} \in (0, 1]$*, we obtain a fixed solution of the stabilized least* 1*-norm approximation (7.8) as in (i) with the* ADDITIONAL *property of maximizing* ε*, the least error component. Hence* ε *is fixed over the interval* $\mu \in [0, \bar{\mu}]$.

(iii) *For* $\mu \in [\bar{\mu}, 1]$ *we get a monotonically nondecreasing least error component* ε *dependent on* μ *until the value* $\mu = 1$ *is reached whereat all error components have been trapped within the interval* $[-\varepsilon, \varepsilon]$ *with the trivial solution:* $\alpha = 0$, $b = 0$, $\varepsilon = \|y\|_\infty$, $s = e\varepsilon$.

(iv) *For all* $\mu \in [\hat{\mu}, 1]$ *for some fixed* $\hat{\mu}$ *in [0,1), we obtain a fixed solution of*

$$\min_{(\alpha,b,\varepsilon,d)} \frac{1}{m}\|\alpha\|_1 + \frac{C}{m}e'(|d| - e\varepsilon)_+ \tag{7.16}$$

with the ADDITIONAL *property of minimizing* ε*, the least error component. Hence* ε *is fixed over the interval* $\mu \in [\hat{\mu}, 1]$.

Proof

(i) Because $a \geq 0$, $s \geq 0$ and by hypothesis ε has a coefficient of zero in the objective function of (7.7), ε can be set to zero and the resulting problem is that of minimizing the stabilized classical least 1-norm approximation (7.8).

(ii) We first note that the linear program (7.7) is solvable for all $\mu \in [0, 1]$ because its objective function is bounded below by zero on its nonempty feasible region as follows:

$$\begin{aligned} \tfrac{1}{m}e'a + \tfrac{C}{m}e's - C\mu\varepsilon &= \tfrac{1}{m}e'a + \tfrac{C}{m}e'(s - \mu e\varepsilon) \\ &\geq \tfrac{1}{m}e'a + \tfrac{C}{m}e'(s - e\varepsilon) \geq 0. \end{aligned} \tag{7.17}$$

The desired result is then a direct consequence of [66, Theorem 1] which states that for all sufficiently small perturbations of a linear programming objective function by another possibly nonlinear function, a solution of the linear program is obtained that in addition optimizes the perturbation function as well. This translates here to obtaining a solution of the stabilized least 1-norm approximation problem (7.8) that also maximizes ε.

(iii) Let $\bar{\mu} \leq \mu^1 < \mu^2 \leq 1$. Let the two points $(\alpha^1, b^1, s^1, \varepsilon^1, a^1)$ and $(\alpha^2, b^2, s^2, \varepsilon^2, a^2)$ be corresponding solutions of (7.7) for the values of μ of μ^1 and μ^2 respectively. Because both of the two points are feasible for (7.7), without regard to the value of μ which appears in the objective function only, it follows by the optimality of each point that:

$$\frac{1}{m}e'a^1 + \frac{C}{m}e's^1 - C\mu^1\varepsilon^1 \leq \frac{1}{m}e'a^2 + \frac{C}{m}e's^2 - C\mu^1\varepsilon^2,$$

$$\frac{1}{m}e'a^2 + \frac{C}{m}e's^2 - C\mu^2\varepsilon^2 \leq \frac{1}{m}e'a^1 + \frac{C}{m}e's^1 - C\mu^2\varepsilon^1.$$

Adding these two inequalities gives:

$$C(\mu^2 - \mu^1)\varepsilon^1 \leq C(\mu^2 - \mu^1)\varepsilon^2.$$

Dividing the last inequality by the positive number $C(\mu^2 - \mu^1)$, gives $\varepsilon^1 \leq \varepsilon^2$.

To establish the last statement of (iii) we note from (7.17) that the objective function of (7.7) is nonnegative. Hence for $\mu = 1$, the point $(\alpha = a = 0,\ b = 0,\ \varepsilon = \|y\|_\infty,\ s = e\varepsilon)$ is feasible and renders the nonnegative objective function zero, and hence must be optimal. Thus $\varepsilon = \|y\|_\infty$.

(iv) We observe that the linear program (7.7) is equivalent to the linear program (7.12), as shown in Lemma 7.1. We therefore obtain result (iv) by a similar application of linear programming perturbation theory as in (ii). □

We turn now to computational implementation of our SVR linear program (7.7) and its numerical testing.

7.3 Numerical Testing

We conducted two different kinds of experiments to illustrate the effectiveness of our formulation of the support vector regression problem. We first show the effectiveness of our linear programming formulation (7.7) when compared to the linear programming formulation (7.9) by using both methods on some large datasets. We then implement our method with a chunking methodology in order to demonstrate regression on massive datasets.

All experiments were run on Locop2, which is one of the machines associated with the University of Wisconsin Computer Sciences Department Data Mining Institute. Locop2 is a Dell PowerEdge 6300 server powered with four 400 MHz Pentium II Xeon processors, four gigabytes of memory, 36 gigabytes of disk space, and the Windows NT Server 4.0 operating system. All linear programs were solved with the state-of-the-art CPLEX solver [47].

We used the Gaussian radial basis kernel [18, 117] in our experiments, namely

$$K(A, A') = \exp(-\gamma \|A_i - A_j\|_2^2), i, j = 1, \ldots, m, \tag{7.18}$$

where γ is a small positive number specified in Table 7.1.

7.3.1 Comparison of Methods

Our linear programming formulation (7.7) for handling support vector regression will be referred to hereafter as the MM method. The experiments in this section compare MM to the linear programming formulation (7.9) given by Smola, Schölkopf, and Rätsch in [110], which will be referred to as the SSR method. We implemented the bookkeeping for both these methods in the MATLAB environment [76]. The actual linear programs were solved by the CPLEX [47] solver, using a homegrown MATLAB executable "mex" interface file [77] to CPLEX. This interface allows us to call CPLEX from MATLAB as if it were a native MATLAB function.

Three datasets were used for testing the methods. The first dataset, Census, is a version of the US Census Bureau "Adult" dataset, which is publicly available from Silicon Graphics' website [15]. This dataset contains nearly 300,000 data points with 11 numeric attributes, and is used for predicting income levels based on census attributes. The second dataset, Comp-Activ, was obtained from the Delve website [27]. This dataset contains 8192 data points and 25 numeric attributes. We implemented the "cpuSmall prototask", which involves using twelve of these attributes to predict what fraction of a CPU's processing time is devoted to a specific mode ("user mode"). The third dataset, Boston Housing, is a fairly standard dataset used for testing regression problems. It contains 506 data points with 12 numeric attributes, and one binary categorical attribute.

The goal is to determine median home values, based on various census attributes. This dataset is available at the UCI repository [85].

We present test set results from tenfold cross-validation on the above datasets. For the Boston Housing dataset, all points were used. For the Census and Comp-Activ datasets, a randomly selected subset of 2000 points was utilized for these experiments. The parameter C of the linear programs (7.7) and (7.9), and the parameter γ in the kernel (7.18) were chosen by experimentation to yield best performance. The parameter μ of (7.7) and (7.9) was varied as described below. We also observed in our experiments that tolerant training yields stronger improvements over standard least 1-norm fitting when there is a significant amount of noise in the training set. We added Gaussian noise of mean zero and standard deviation σ to all training sets. Our parameter settings are shown in Table 7.1 below along with the results.

For each dataset, we began by setting $\mu = 0$ and doing the regression. This corresponds to doing a standard least 1-norm fit. Increasing μ for values near $\mu = 0$ typically increases fitting accuracy. We therefore raised μ at increments of 0.1 until we saw a degradation in test set accuracy. The "total time" column in Table 7.1 reflects for a single fold the total time required to solve all problems we posed, from $\mu = 0$ up to our stopping point for μ. The test set errors shown are averages over tenfold cross-validation, and are measured as percentage deviations of the predicted values from the actual vales. In other words, for a given dataset the error of the predicted values $\hat{y}$ relative to the actual test values y is measured as $\frac{\|\hat{y}-y\|_1}{\|y\|_1}$.

Table 7.1 summarizes the results for this group of numerical tests. We make the following points:

(i) The MM method is faster on all problems, by as much as 26.0% on average and

79.7% maximally, than the SSR method.

(ii) Test set error bottoms out at an intermediate positive value of the tolerated error ε. The size of the tolerated error ε is directly proportional to the parameter μ of the linear programs (7.7) and (7.9).

Dataset	Parameters		Train size / Test size		μ								Total Time (sec)	Time Improvement
					0	0.1	0.2	0.3	0.4	0.5	0.6	0.7		
Census	C	100,000	2000	Test set error	5.10%	4.74%	4.52%	4.24%	4.16%	3.99%	3.79%	3.91%		Max
	γ	0.01	1000	ε	0.00	0.02	0.05	0.08	0.11	0.14	0.17	0.21		79.7%
	σ	0.2		SSR time (sec)	980	935	814	623	540	504	403	287	5086	Avg
				MM time (sec)	199	294	351	392	480	573	710	766	3765	26.0%
Comp-Activ	C	100	2000	Test set error	6.60%	6.32%	6.16%	6.09%	6.01%	5.88%	5.60%	5.79%		Max
	γ	0.01	200	ε	0.00	3.09	7.06	11.20	15.43	19.87	24.90	30.65		65.7%
	σ	30		SSR time (sec)	1364	1286	1015	1045	917	747	601	629	7604	Avg
				MM time (sec)	468	660	664	859	815	983	1043	1040	6533	14.1%
Boston Housing	C	1,000,000	481	Test set error	14.69%	14.62%	14.23%	13.71%	13.59%	13.77%				Max
	γ	0.0001	25	ε	0.00	0.42	1.37	2.31	3.23	4.20				52.0%
	σ	6		SSR time (sec)	36	34	28	25	24	23			170	Avg
				MM time (sec)	17	23	24	23	26	27			140	17.6%

C, μ: Parameters of the linear programs (7.7) and (7.9)

γ: Parameter of the Gaussian kernel (7.18)

σ: Standard deviation of introduced noise

ε: Variable in linear programs (7.7) and (7.9) which indicates tolerated error (see Figure 7.1). Its size is directly proportional to the parameter μ.

Empty columns under μ indicate experiments were not needed due to degradation of test set accuracy.

Table 7.1: Tenfold cross-validation results for MM and SSR methods.

Dataset	$\bar{\mu}$	$\hat{\mu}$
Census	0.051	5×10^{-7}
Comp-Activ	0.01	1×10^{-5}
Boston Housing	0.005	1×10^{-7}

Table 7.2: Experimental values for $\bar{\mu}$ and $\hat{\mu}$.

Finally, we ran some experiments to determine the values of $\bar{\mu}$ and $\hat{\mu}$ as in Proposition 7.1. We perturbed μ from 0 and from 1, and observed when the appropriate objective function value began to change. Table 7.2 shows the results for a single fold from each of the data sets under consideration. Note that Table 7.1 shows that the optimal values for μ are typically in the 0.4-0.6 range, and therefore are not included in either the intervals $[0, \bar{\mu}]$ or $[\hat{\mu}, 1]$. This yields the interesting result that the best solution to linear program (7.7) is a Pareto optimal solution that optimizes a weighted sum of the terms in the objective function. In other words, the solution is not subject to interpretations (ii) or (iii) of Proposition 7.1.

7.3.2 Massive Datasets via Chunking

The Linear Programming Chunking method (LPC) has proven to be a useful approach in applying support vector machines to massive datasets by chunking the constraints that correspond to data points [8]. We describe below how to chunk both the constraints and variables, which allows the solution of considerably larger problems than those considered previously. We first present a brief review of how the basic LPC is implemented. We then discuss the optimizations and improvements that we made.

The basic chunking approach is to select a subset of the constraints of the linear program (7.7) under consideration. When optimization on this first chunk of constraints is complete, a second chunk is created by combining all active constraints (those with positive Lagrange multipliers) from the first chunk and adding to it new unseen constraints. This process is repeated, looping a finite number of times through all constraints of the linear program until the objective function remains constant. This approach yields a non-decreasing objective value that will eventually terminate at a solution of the original linear program [8].

To make our chunking implementation simpler, we apply the variable substitutions $s = t + e\varepsilon$ and $\alpha = \alpha^1 - \alpha^2, \alpha^1 \geq 0, \alpha^2 \geq 0$, which results in a linear program that is solved faster:

$$\begin{array}{llcll} & \min\limits_{(\alpha^1,\alpha^2,b,t,\varepsilon)} & & \frac{1}{m}e'(\alpha^1+\alpha^2) + \frac{C}{m}e't + C(1-\mu)\varepsilon & \\ \text{s.t.} & -t - e\varepsilon & \leq & K(A,A')(\alpha^1-\alpha^2) + be - y & \leq \ t + e\varepsilon \\ & & & \alpha^1,\ \alpha^2,\ t & \geq \ 0, \end{array} \tag{7.19}$$

Formulation (7.19) looks similar to (7.9), but it has an important distinction. The two variables ξ^1 and ξ^2 have been replaced by the single variable t. This version of our problem is simpler for chunking than (7.7), since we do not need to be concerned with chunking the $e\varepsilon \leq s$ constraint.

One difference between our approach and the LPC in [8] is that each support vector has associated with it two constraints, namely the first two inequalities in (7.19). For a given support vector, only one of the two constraints associated with it has a nonzero multiplier (except possibly for a trivial case involving equalities). Our code tracks both constraints for a given support vector, and only retains from chunk to chunk those constraints with positive multipliers.

We make a further computational improvement involving upper bounds of the multipliers. This is motivated by looking at the dual problem of (7.19), which can be written as

$$
\begin{array}{rrcc}
 & \max\limits_{(u,v)} \quad y'(u-v) & & \\
\text{s.t.} & K(u-v) & \leq & \frac{e}{m} \\
 & K(-u+v) & \leq & \frac{e}{m} \\
 & e'(u-v) & = & 0 \\
 & u+v & \leq & \frac{C}{m}e \\
 & e'(u+v) & \leq & C(1-\mu) \\
 & u,\ v & \geq & 0,
\end{array}
\tag{7.20}
$$

where u and v are the multipliers associated with the constraints that we use for chunking. The constraint $u+v \leq \frac{C}{m}e$ indicates that these multipliers have an upper bound of $\frac{C}{m}e$. Experimentation has shown us that constraints with multipliers u_i or v_i at this upper bound have a strong likelihood of remaining at that upper bound in future chunks. We therefore use this knowledge to our advantage. When optimization over a chunk of constraints is complete, we find all active constraints in that chunk that have multipliers at this upper bound. In the next chunk, we constrain the multiplier to be at this bound. This way the constraint remains active, but CPLEX does not have to spend resources determining this multiplier value. This frees up memory to add more new constraints to the chunk. This constraint on the multiplier is removed when the constraint selection procedure loops again through the entire constraint list, and tries to re-add this constraint to the problem. Our software actually represents the primal version of the linear program (7.19), as experimentation has shown that this solves faster than the dual. We therefore fix multipliers at this upper bound value by taking all associated constraints and adding them together to create a single collapsed constraint.

One difficulty in using the approach presented thus far is that as the number of data points increases, both the number of constraints and the number of variables increases. This is due to the kernel $K(A, A')$. The vectors α^1 and α^2, which are variables in the optimization problem, grow as data points are added to the problem. This means that the number of points that can fit in memory *decreases* as the problem size increases overall. This somewhat limits the success of chunking. A sufficiently large problem has so many variables that even a small number of support vectors cannot be represented in memory.

We therefore present a new approach for dealing with this issue: we chunk not only on the rows of the linear program, i.e. the constraints, but also on the columns of the linear program, i.e. the variables. We have observed that at optimality, most values of α^1 and α^2 are zero. For a given set of constraints, we choose a small subset of the α^1 and α^2 values to actually participate in the the linear program. All others are fixed at zero. This drastically reduces the size of the chunk, as the chunk width is no longer determined by the number of points in the entire problem, but merely by the number of nonzero α^1 and α^2. We solve this small problem, retain all α^1 and α^2 values which are nonzero, and proceed to chunk on another set of columns. For a fixed set of constraints, we loop repeatedly over the columns in the linear program until we terminate for that particular set of rows. Then another chunk of rows is selected as described above, and the process begins again. The column chunking is similar to the column generation method of Gilmore-Gomory and Dantzig-Wolfe [24, 39].

We used this chunking approach on a subset of the Census dataset consisting of 16,000 data points. This results in a linear program with 32,000 rows and over 32,000 non-sparse columns, thereby containing over *one billion* nonzero values. Values for C and γ were chosen the same as shown in Table 7.1. We also chose $\sigma = 1$ and $\mu = 0.9$, and picked the initial chunk size to be 5000 rows and 2500 columns.

This massive problem was solved in 18.8 days, terminating with 1621 support vectors and 33 nonzero components of α^1 and α^2. Approximately 90 row chunks, or 10 sweeps through the dataset, were required to reach termination. Figures 7.2 and 7.3 depict the objective function value and tuning set error for each row-column chunk.

The final tuning set error achieved by this fit was 9.8% on a tuning set of 1000 points. We note that at termination of the first chunk, the tuning set error was at 16.2%. This indicates that a large sample of points can prove useful in getting better solutions.

In the next chapter, we will look at using quadratic programming to do support vector regression for the Huber M-estimator loss function [45], which is quadratic for small errors and linear for large errors.

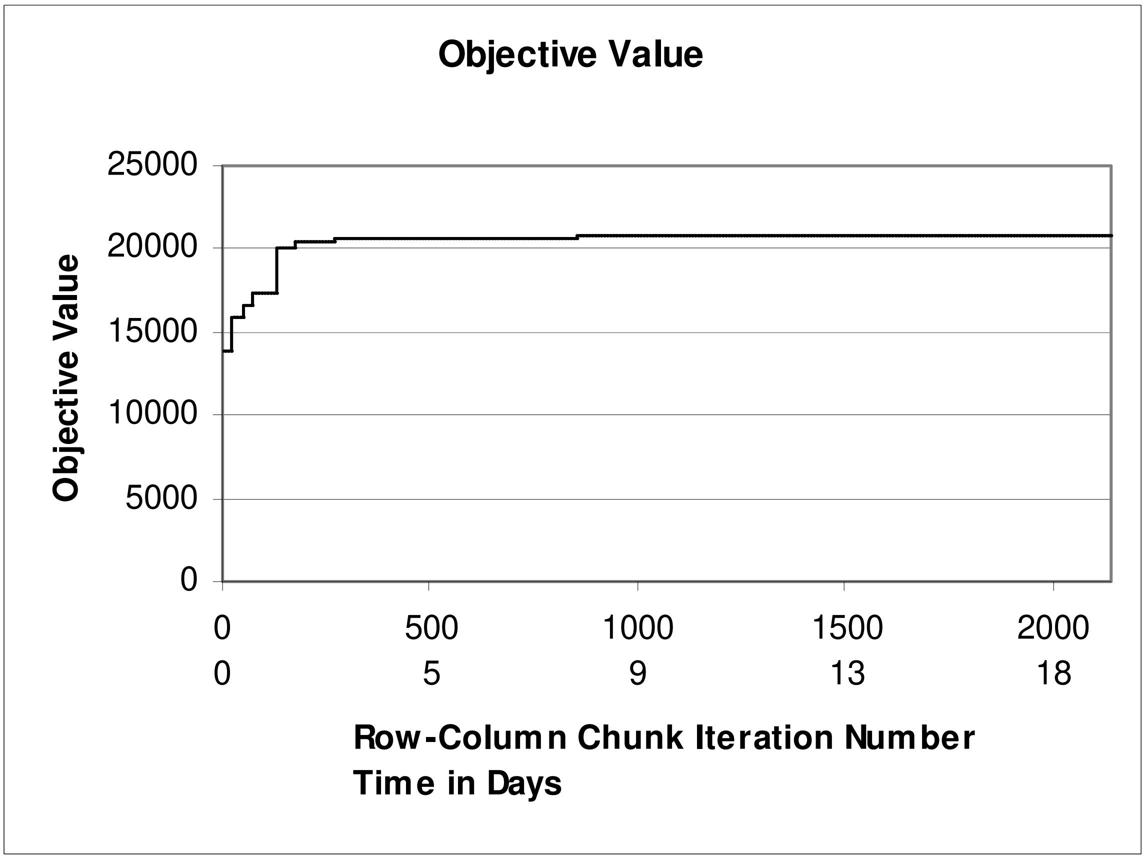

Figure 7.2: Objective function value versus row-column iteration and time in days.

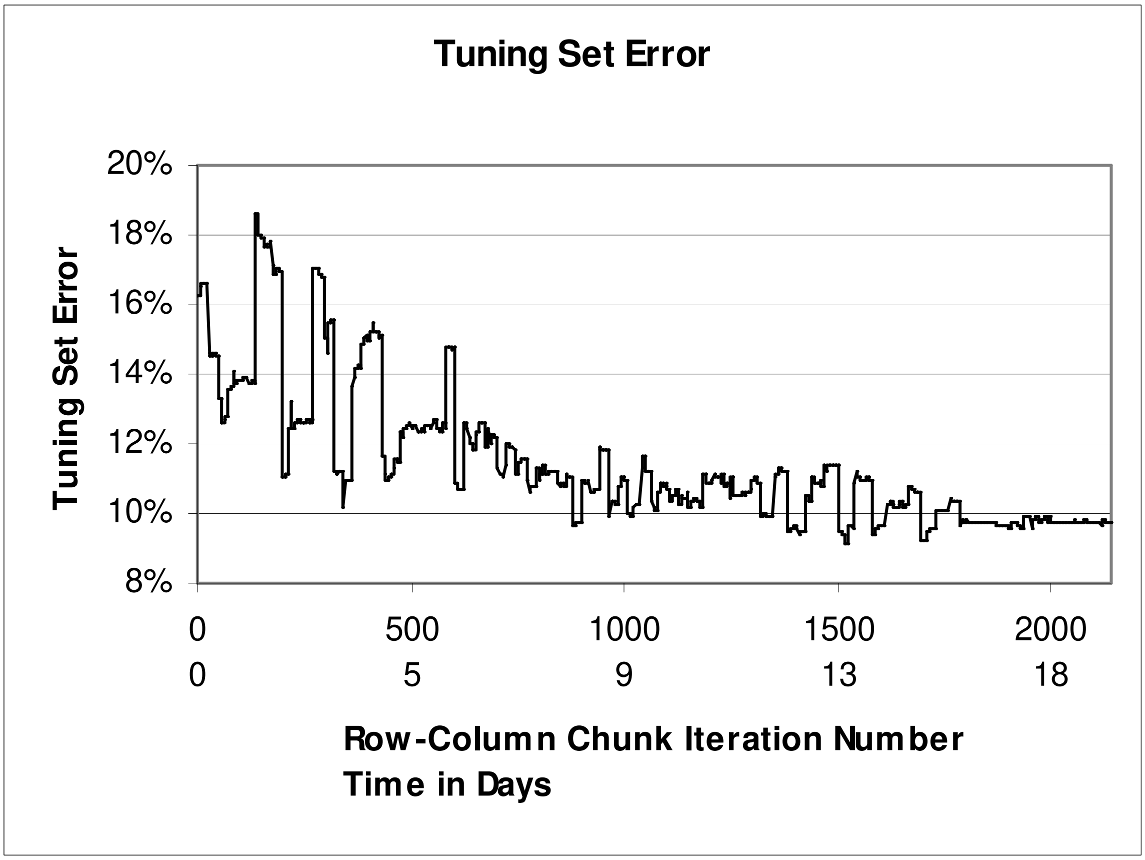

Figure 7.3: Tuning set error versus row-column iteration and time in days.

Chapter 8

Robust Linear and Support Vector Regression

8.1 Introduction

We again consider the generally unsolvable system of linear equations

$$Ax = b, \tag{8.1}$$

where A is a given $m \times n$ real matrix of m observations and b is a real $m \times 1$ vector of corresponding values, all taken from a given dataset. To obtain an approximate solution of (8.1), one usually minimizes an error residual:

$$\min_x \sum_{i=1}^{m} \rho((Ax - b)_i), \tag{8.2}$$

where typically ρ is the absolute value function or its square. In order to de-emphasize outliers and avoid the nondifferentiability of the robust absolute value error residual, a popular residual is the Huber M-estimator cost function [45]:

$$\rho(t) = \begin{cases} \frac{1}{2}t^2, & \text{if } |t| \leq \gamma \\ \gamma|t| - \frac{1}{2}\gamma^2, & \text{if } |t| > \gamma, \end{cases} \tag{8.3}$$

where γ is some positive number. Because the function switches at $|t| = \gamma$ from being quadratic to linear, special methods have been developed for the linear error minimization problem (8.2) with the Huber M-estimator. A Newton method was proposed in [46], which is possibly nonconvergent because the objective function of (8.2) is not twice differentiable. Other fairly complex Newton type methods were considered in [31, 55, 58, 104]. In [121] a minmax formulation leading to a mixed complementarity problem is proposed. Relations between the ℓ_1 and Huber estimators as well as iterative methods are proposed in [56].

We outline the content of this chapter now. In Section 8.2 we first give a simple quadratic programming formulation of the Huber M-estimator cost function in one dimension (Lemma 8.1) and then set up the corresponding convex quadratic program (8.9) for the linear estimator (8.2) with ρ given by (8.3) (Proposition 8.1). This explicit convex quadratic programming formulation in the original primal space of the problem is rather simple and easily interpretable but does not seem to have been given previously in other works that have considered dual formulations of this problem [56, 107, 111]. By using parametric perturbation results of linear programming [66] we show that, for all values of the parameter $\gamma \geq \bar{\gamma}$ for some $\bar{\gamma}$, a Huber linear estimator is just an ordinary least squares estimate (Proposition 8.2). On the other hand for all sufficiently small values of γ the Huber estimates depend linearly on γ and converge to a least 1-norm solution (Proposition 8.4). Li and Swetits [56] have studied the dual (8.20) of the Huber M-estimator quadratic program (8.9). They show that (8.20) is a least 2-norm formulation of the dual of the least 1-norm estimator. In addition they give perturbation results for the solution of (8.20). Smola [107] also presents a dual formulation of the Huber M-estimator quadratic program [111]. In Section 8.3 we set up a convex quadratic program (8.26) for a nonlinear generalized support vector machine [64, 117] which extend the Huber loss function to large classes of nonlinear regression problems. Related but different loss functions were studied in [100, 101, 110, Chapter 7 in this thesis] in conjunction with support vector regression. Numerical test results presented in Section 8.4 show that our direct convex quadratic formulation is considerably faster than earlier proposed methods such as Huber's Gauss-Seidel method [55], Smola's dual formulation [107], Madsen and Nielsen's Newton type method [58], and Li's conjugate gradient method [55].

8.2 Robust Linear Regression as a Convex Quadratic Program

We begin with a very simple lemma that generates the Huber cost function in one dimension as the unconstrained minimum value of a simple convex quadratic function.

Lemma 8.1 (Huber cost as a minimum of a convex quadratic function) *The Huber cost function $\rho(t)$ of (8.3) is given by:*

$$\rho(t) = \min_{z \in R^1} \ \frac{1}{2}z^2 + \gamma|t - z|, \quad t \in R^1. \tag{8.4}$$

Proof Let $\theta(z,t) = \frac{1}{2}z^2 + \gamma|t - z|$. A subgradient [97] of this convex objective function of (8.4) with respect to z is given by:

$$\partial\theta(z,t) = \begin{cases} z - \gamma, & \text{when } \ t > z \\ z + \lambda\gamma, \quad \lambda \in [-1,1], & \text{when } \ t = z \\ z + \gamma, & \text{when } \ t < z \end{cases} \tag{8.5}$$

This subgradient is zero when:

$$\begin{cases} z = \gamma, & \text{for } \ t > z = \gamma \\ z = -\lambda\gamma, & \text{for } \ t = z = -\lambda\gamma, \quad \lambda \in [-1,1] \\ z = -\gamma, & \text{for } \ t < z = -\gamma \end{cases} . \tag{8.6}$$

The zero-subgradient necessary and sufficient optimality condition [97] gives values of z at which the convex objective function of (8.4) attains a minimum for each $t \in R^1$. Evaluating this objective function at these values of z gives:

$$\begin{cases} \frac{1}{2}\gamma^2 + \gamma(t-\gamma) = \gamma t - \frac{1}{2}\gamma^2, & \text{when } \ t > \gamma \\ \frac{1}{2}t^2, & \text{when } \ t \in [-\gamma, \gamma] \\ \frac{1}{2}\gamma^2 + \gamma(-t-\gamma) = -\gamma t - \frac{1}{2}\gamma^2, & \text{when } \ t < -\gamma \end{cases}, \tag{8.7}$$

which is equivalent to the definition (8.3) of $\rho(t)$.□

This lemma allows us immediately to set up the linear estimation problem (8.2) with the Huber loss function (8.3) as the following problem:

$$\min_{x \in R^n, z \in R^m} \ \frac{1}{2}\|z\|_2^2 + \gamma\|Ax - b - z\|_1. \tag{8.8}$$

This problem in turn can be reduced to a simple convex quadratic program as follows.

Proposition 8.1 (Huber linear estimator as a convex QP) *A Huber linear estimator $x(\gamma)$ that solves (8.2) with ρ defined by (8.3) is given by any solution $(x(\gamma), z(\gamma), t(\gamma))$ of the following convex quadratic program:*

$$\begin{array}{rl} \min\limits_{x \in R^n, z \in R^m, t \in R^m} & \frac{1}{2}\|z\|_2^2 + \gamma e't \\ s.t. & -t \leq Ax - b - z \leq t. \end{array} \tag{8.9}$$

Using this formulation, an immediate consequence of a perturbation result of linear programming [66] is the intuitively plausible result when all errors fall within γ, that for each linear estimation problem (8.1) there exists a positive $\bar{\gamma}$ such that every Huber estimator for all $\gamma \geq \bar{\gamma}$ is identical to the classical least 2-norm estimator. Hence for all γ larger than this threshold a classical least squares solution also solves (8.1). We state this result as follows.

Proposition 8.2 (Huber M-estimator is a least squares estimator for all large γ) *For a given $A \in R^{m\times n}$ and $b \in R^{m\times 1}$, there exists a $\bar{\gamma}$ such that for all $\gamma \geq \bar{\gamma}$ a Huber M-estimator obtained by solving either of the equivalent problems (8.8) or (8.9) is equivalent to a classical least squares estimator that solves*

$$\min_{x\in R^n} \ \|Ax-b\|_2^2 \ . \tag{8.10}$$

Proof Theorem 1 of [66] says that under certain easily satisfiable technical assumptions (which are satisfied here), the solution of the perturbation of any solvable linear program is also a solution of the original linear program itself, as long as the perturbation is sufficiently small. More formally, there exists $\bar{\gamma}$ such that for each $\gamma \geq \bar{\gamma}$, each corresponding Huber estimate solution $(x(\gamma), z(\gamma))$ of (8.8) is a solution of:

$$\min_{x\in R^n, z\in R^m} \ \|Ax-b-z\|_1, \tag{8.11}$$

that is: $z(\gamma) = Ax(\gamma) - b$, which also solves:

$$\min_{z\in R^m} \ \|z\|_2^2, \tag{8.12}$$

that is $x(\gamma)$ solves (8.10).$\Box$

In fact, we can quantify the value of $\bar{\gamma}$ as given in the following proposition.

Proposition 8.3 *[80] Let $\bar{x}$ be a solution to*

$$\min_{x\in R^n} \ \|Ax-b\|_2^2 \ . \tag{8.13}$$

Let $\bar{z} = A\bar{x} - b$. Then $\bar{\gamma} = \|z\|_\infty$.

Proof We need to demonstrate that $x = \bar{x}$, $z = \bar{z}$, $t = 0$ is a solution to (8.9) for all $\gamma \geq \bar{\gamma}$. We first observe that since $\bar{x}$ is a solution to (8.13), the normal equations

$A'A\bar{x} = A'b$ and therefore $A'\bar{z} = 0$ hold. We then add the constraint $t \geq 0$ to (8.9) (which clearly does not change the problem), and write down the Karush-Kuhn-Tucker optimality conditions [62]:

$$\begin{aligned} -A'(r-s) &= 0 \\ z+r-s &= 0 \\ r+s+v &= \gamma e \\ 0 \leq r,\ \ Ax-z+t-b &\geq 0, \\ 0 \leq s,\ \ -Ax+z+t+b &\geq 0, \\ 0 \leq v,\ \ t &\geq 0 \\ r'(Ax-z+t-b)=0,\ \ s'(-Ax+z+t+b) &= 0,\ \ vt=0. \end{aligned} \tag{8.14}$$

Then $x = \bar{x}$, $z = \bar{z}$, $t = 0$, and $\gamma \geq \|z\|_\infty$ satisfy (8.14) with the following multiplier choices:

$$\begin{aligned} r &= (-\bar{z})_+ \\ s &= (\bar{z})_+ \\ v &= \gamma e - r - s. \qquad \square \end{aligned} \tag{8.15}$$

Perturbing γ in the other direction towards zero is more interesting. In fact one can show that Huber M-estimators depend linearly on the parameter as the latter converges to zero as shown in Proposition 8.4 below. However, an example [56, Example 3.8] shows that Huber M-estimators need not be least 1-norm estimators even for arbitrarily small values of the parameter γ. This is in contrast with Proposition 8.2 which states that for all sufficiently large γ Huber M-estimators are least 2-norm estimators.

Proposition 8.4 (Huber M-estimators converge linearly to a least 1-norm solution) *For every sequence $\{\gamma^i\}$ converging to zero, a corresponding subsequence of Huber M-estimators $\{x^{i_j}\}$ such that $\{x^{i_j}, z^{i_j}, t^{i_j}\}$ solve (8.9) for $\gamma = \gamma^{i_j}$ depends linearly on*

$\{\gamma^{i_j}\}$, *that is for some* p *and* q *in* R^n*:*

$$x^{i_j} = p + q\gamma^{i_j}, \quad \{\gamma^{i_j}\} \to 0. \tag{8.16}$$

Proof For $\gamma = \gamma^i$, the Karush-Kuhn-Tucker optimality conditions [62] for (8.9) are:

$$\begin{array}{rcl} -A'(r-s) & = & 0 \\ z + r - s & = & 0 \\ r + s & = & \gamma^i e \\ 0 \le r, \quad Ax - z + t - b & \ge & 0, \\ 0 \le s, \quad -Ax + z + t + b & \ge & 0, \\ r'(Ax - z + t - b) = 0, \quad s'(-Ax + z + t + b) & = & 0. \ (*) \end{array} \tag{8.17}$$

For each γ^i, pick a basic solution [74, Lemma 2.1], [38, Theorem 2.11] which also satisfies the complementarity condition (*). Since there are a finite number of linearly independent columns in the linear system of (8.17), one set of linearly independent columns is used infinitely often by a basic solution corresponding to the sequence $\{\gamma^i\}$. Choose a subsequence $\{\gamma^{i_j}\}$ corresponding to this repeated basic solution. Hence there exists a matrix M with $n + 2m + m_1 + m_2$ rows $(m_1, m_2 \le m)$ and linearly independent columns corresponding to basic components y_B (typically nonzero) of $y := (x,\ z,\ t,\ r,\ s)$, such that:

$$My_B = \begin{bmatrix} 0 \\ 0 \\ \gamma^{i_j} e \\ b \\ -b \end{bmatrix}. \tag{8.18}$$

It follows then that:

$$y_B = (M'M)^{-1}M' \begin{bmatrix} 0 \\ 0 \\ \gamma^{i_j}e \\ b \\ -b \end{bmatrix} \tag{8.19}$$

from which (8.16) follows. $\square$

It is interesting to note that Li and Swetits [56] have established a related result, Lipschitz continuity, for the dual of the quadratic Huber M-estimator problem (8.9) which turns out to be:

$$\min_u \{\frac{\gamma}{2}\|u\|_2^2 + b'u \mid A'u = 0,\ -e \leq u \leq e\}. \tag{8.20}$$

They show that the solution $u(\gamma)$ of this dual problem, which is a least 2-norm solution of the dual of the least 1-norm problem $\min_x \|Ax - b\|_1$, is Lipschitzian with respect to γ.

Another interesting observation is that if the square of the 2-norm in (8.20) is replaced by the 1-norm as follows:

$$\min_u \{\gamma\|u\|_1 + b'u \mid A'u = 0,\ -e \leq u \leq e\}, \tag{8.21}$$

then the linear programming dual of (8.21) is:

$$-\min_{x,z} \{\|Ax - b - z\|_1 \mid \|z\|_\infty \leq \gamma\}. \tag{8.22}$$

The loss function corresponding to this problem is essentially that of [110, Chapter 4], which is zero in the interval $[-\gamma, \gamma]$, and otherwise linear. Thus, this loss function replaces the quadratic part of the Huber loss function by zero. Such a loss function was also studied in [113].

Finally, we also note that Smola [107, 111] uses rather different techniques to derive a dual formulation of the quadratic Huber M-estimator problem (8.9). Smola's formulation can be stated as follows, after removing a suppression term:

$$\begin{array}{ll} \min\limits_{u,v} & \frac{\gamma}{2}(\|u\|_2^2 + \|v\|_2^2) + b'(u-v) \\ \text{s.t.} & A'(u-v) = 0 \\ & 0 \leq u, v \leq e, \end{array} \tag{8.23}$$

which is equivalent to the Li-Swetits dual formulation (8.20) above. It is this formulation of Smola's that we use in our comparative experiments.

8.3 Robust Nonlinear Support Vector Regression as a Convex Quadratic Program

In order to apply the Huber loss function to nonlinear regression problems using kernel functions [64, 117] we follow an approach similar to that of [100, 101, Chapter 7 in this thesis] where loss functions other than the Huber one were utilized. The idea is to implicitly transform the data of the problem from the given *input space* into a higher dimensional *feature space*, and perform linear regression in that space. This then corresponds to a nonlinear regression surface in the original input space. For that purpose we make a variable transformation in our system of linear equations (8.1):

$$x = A'\alpha, \quad \alpha \in R^m. \tag{8.24}$$

Corresponding to this transformation our convex quadratic program of the Huber linear estimator becomes:

$$\begin{aligned} \min_{\alpha\in R^m, z\in R^m, t\in R^m} \quad & \tfrac{1}{2}\|z\|_2^2 + \gamma e't \\ \text{s.t.} \quad & -t \leq AA'\alpha - b - z \leq t. \end{aligned} \tag{8.25}$$

We observe that problem (8.25) depends only on a matrix consisting of scalar products of different rows in A, i.e. AA'. This immediately leads to the idea of replacing the kernel AA' for a linear estimator by a much more general kernel $K(A, A') : R^{m\times n} \times R^{n\times m} \to R^{m\times m}$. Under certain conditions, this kernel is a surrogate for mapping the data into a higher dimensional space and performing the scalar products there. The use of a kernel function, therefore, allows an implicit mapping into a higher dimensional space while saving significant computational costs. This concept is described in much more detail in some of the support vector machine literature $[21, 64, 117]$. Substituting AA' for $K(A, A')$ leads to the following convex quadratic program for a Huber nonlinear estimator:

$$\begin{aligned} \min_{\alpha\in R^m, z\in R^m, t\in R^m} \quad & \tfrac{1}{2}\|z\|_2^2 + \gamma e't \\ \text{s.t.} \quad & -t \leq K(A, A')\alpha - b - z \leq t. \end{aligned} \tag{8.26}$$

We determine the predicted value for a new data point p by the calculation $K(p', A')\alpha$. As such, the regression surface depends strongly on the training matrix A in precisely the same fashion as a classification or regression surface does in traditional support vector machines. We do point out that the matrix $K(A, A')$ has m by m nonzero elements. If the dataset is large, i.e. m is large, than this problem is significantly larger than the linear formulation which has m by n nonzero elements in the matrix A. For a large dataset, the linear problem (8.9) will therefore solve dramatically faster than the nonlinear formulation (8.26). However, our experimental results in Section 8.4 show that nonlinear kernels can yield higher test set accuracies. Kernels have also been used in

Smola's dual QP method. We note that our description of kernels here follows directly from our formulation (8.9), and is noticeably different from the approach taken in [111].

8.4 Numerical Tests and Results

We first show the effectiveness of our formulation (8.9) of the Huber linear estimator as a QP by comparing it to four other algorithms to solve the same problem, namely:

- Smola's dual QP method [107]
- Huber's Gauss-Seidel method [55]
- Madsen and Nielsen's Newton type method [58]
- Li's conjugate gradient method [55]

In all cases except the last one, a termination tolerance of 10^{-5} was used. Li's conjugate gradient method showed extremely long termination times, so we reduced the tolerance for this algorithm to 10^{-3}. The primary purpose of these experiments is to examine differences in training times. Training and test set accuracies are not of importance in these experiments, since all these algorithms in theory converge to a solution of the same QP, thus leading to similar training and test set correctness. Inconsequential differences in accuracies arise in practice due to the fact that each algorithm yields a slightly different approximate solution. Additionally, it is possible that different algorithms could converge to different solutions if there are multiple solutions.

We note that in implementing our algorithm, we used the following equivalent variation of the QP given in (8.9):

$$\begin{array}{rl} \displaystyle\min_{x\in R^n, z\in R^m, r\in R^m, s\in R^m} & \frac{1}{2}\|z\|_2^2 + \gamma e'(r+s) \\ \text{s.t.} & Ax - b - z = r - s \\ & r, s \geq 0 \end{array} \tag{8.27}$$

Formulation (8.27) yielded faster running times than a straightforward implementation of (8.9).

All experiments were run on the University of Wisconsin Computer Sciences Department Ironsides cluster. This cluster of four Sun Enterprise E6000 machines consists of 16 UltraSPARC II 250 MHz processors and 2 gigabytes of RAM on each node resulting in a total of 64 processors and 8 gigabytes of RAM. We implemented data I/O, cross-validation procedures, and kernel calculations in the MATLAB environment [76], though all algorithms were implemented in C++ using a MATLAB executable "mex" file [77]. This methodology allows us to call the algorithms from MATLAB as if they were native functions. All QPs in our algorithm were solved with the state-of-the-art CPLEX solver [47].

To demonstrate the performance of these algorithms, we performed tenfold cross validation of these algorithms on two datasets. The first dataset, Census, is a version of the US Census Bureau "Adult" dataset, which is publicly available from Silicon Graphics' website [15]. This dataset contains nearly 300,000 data points with 11 numeric attributes, and is used for predicting income levels based on census attributes. We ran the algorithms on varying subsets of this dataset. The second dataset, Comp-Activ, was obtained from the Delve website [27]. This dataset contains 8192 data points and 25 numeric attributes. We implemented the "cpuSmall prototask", which involves using twelve of these attributes to predict what fraction of a CPU's processing time is devoted to a specific mode ("user mode").

We scaled the data by dividing by the following factor τ, suggested by Holland and Welsh in [44]:

$$\tau = 1.48 \cdot \text{med}_i\{|(A\hat{x} - b)_i - \text{med}_j\{(A\hat{x} - b)_j\}|\}, \tag{8.28}$$

where $\hat{x}$ is an estimate of the solution x. We determined $\hat{x}$ by taking a small sample of the data and doing a standard least-squares regression. The factor 1.48 provides an approximately unbiased estimate of scale when the error model is Gaussian [44].

We also varied the parameter γ as specified in (8.3). Holland and Welsh recommend $\gamma = 1.345$, which satisfies certain statistical properties [44]. We have also used $\gamma = 1$ and $\gamma = 0.1$. We note that these smaller values of γ provide a closer approximation to a traditional 1-norm regression problem.

Table 8.1 shows the comparison of the various algorithms. In nearly all cases, our algorithm shows faster running times than the other algorithms. The differences in running times between our algorithm and the others become more pronounced for the larger versions of the Census dataset. In comparison to the other algorithms, Smola's dual QP also performs reasonably. Huber's Gauss-Seidel method performs well for large γ, but deteriorates significantly for small γ. Madsen and Nielsen's method, while very low in the number of iterations, runs somewhat slowly. Li's conjugate gradient method appears somewhat in the middle, though it should be remembered that a less stringent termination tolerance was used.

Our second set of experiments is designed to show the effectiveness of using a kernel function in performing nonlinear robust regression. We used the Gaussian radial basis kernel [18, 117] in our experiments, namely

$$[K(A, A')]_{i,j} = \exp(-\nu \|A_i - A_j\|_2^2), i, j = 1, \ldots, m, \tag{8.29}$$

where ν is a small positive parameter. These experiments take significantly longer to run than the first set, as the kernel function enlarges the size of the problem. We therefore ran these experiments for a subset of the CPUSmall task containing only 500 data points, and for the Boston Housing dataset (containing 506 points) available at the UCI machine learning repository [85]. The training and testing set accuracies shown are measured as percentage deviations of the predicted values from the actual vales. In other words, for a given dataset the accuracy of the predicted values $\hat{y}$ relative to the actual values y is measured as $1 - \frac{\|\hat{y}-y\|_1}{\|y\|_1}$. The results of these experiments are shows in Table 8.2. In both cases, the Gaussian kernel resulted in improved test set accuracy over the linear one. This demonstrates that robust regression can be used for finding complex nonlinear regression surfaces.

		$\gamma = 0.1$		$\gamma = 1$		$\gamma = 1.345$	
Dataset	**Algorithm**	**Time (sec)**	**Iterations**	**Time (sec)**	**Iterations**	**Time (sec)**	**Iterations**
CPUSmall	Mangasarian/Musicant QP	**7.46**	12.4	**6.26**	10.1	6.37	10.0
	Smola Dual QP	10.63	18.8	7.08	11.9	6.98	11.7
	Huber Gauss-Seidel	50.89	1210.7	6.44	147.9	**5.24**	112.6
	Madsen/Nielsen Newton	160.08	6.2	44.90	4.3	18.31	3.5
	Li Conjugate Gradient	178.01	172.3	92.26	92.3	99.87	102.0
Census 1000	Mangasarian/Musicant QP	**0.58**	11.0	**0.48**	9.4	**0.54**	8.9
	Smola Dual QP	0.93	20.0	0.62	13.0	0.60	12.4
	Huber Gauss-Seidel	7.24	1838.5	1.46	279.8	0.97	217.0
	Madsen/Nielsen Newton	2.02	10.5	1.01	4.9	0.89	4.6
	Li Conjugate Gradient	2.52	24.9	11.68	115.3	5.71	57.1
Census 5000	Mangasarian/Musicant QP	**3.63**	12.9	**2.82**	9.9	**2.82**	9.7
	Smola Dual QP	6.90	22.9	4.46	14.5	4.36	14.1
	Huber Gauss-Seidel	37.74	1745.8	5.30	195.8	3.58	147.8
	Madsen/Nielsen Newton	34.93	9.3	18.96	5.5	14.86	5.2
	Li Conjugate Gradient	11.73	21.8	26.99	51.4	48.02	94.2
Census 10000	Mangasarian/Musicant QP	**7.99**	13.5	**6.42**	10.2	**6.16**	9.9
	Smola Dual QP	16.63	24.9	10.95	15.8	10.51	15.1
	Huber Gauss-Seidel	124.27	2522.6	10.89	215.7	8.24	158.8
	Madsen/Nielsen Newton	132.32	8.7	75.05	5.5	56.35	5.2
	Li Conjugate Gradient	36.19	34.2	92.74	86.7	113.56	105.2
Census 20000	Mangasarian/Musicant QP	**18.48**	14.0	**14.66**	10.5	**16.64**	10.2
	Smola Dual QP	39.21	28.6	26.13	17.7	25.23	16.8
	Huber Gauss-Seidel	305.79	2398.8	27.04	202.4	23.78	154.1
	Madsen/Nielsen Newton	516.76	8.0	318.45	5.4	254.78	5.2
	Li Conjugate Gradient	68.89	31.4	151.94	39.5	231.08	99.9

Table 8.1: Comparison of algorithms for robust linear regression. Best times, in bold type, were achieved by our simple quadratic programming formulation in all cases except one.

Dataset	Kernel	ν	Training Accuracy	Testing Accuracy	Time (cpu secs)
CPUSmall	Linear		94.50%	94.06%	0.21
	Gaussian	0.5	97.26%	95.90%	44.56
Boston Housing	Linear		85.60%	83.81%	0.18
	Gaussian	1	92.36%	88.15%	36.5

Table 8.2: Comparison of robust linear and nonlinear regression.

Chapter 9

Conclusion

9.1 Support Vector Machine Classification

Four new algorithms for implementing support vector machine classification have been presented. The first was obtained as a modification of the standard SVM problem (2.8) by maximizing the margin with respect to distances between the planes as well as location (3.3). This modification yields similar results to the standard SVM in practice, and identical results under certain conditions (Proposition 3.1). We therefore produced the SOR Algorithm 3.1, which is a matrix splitting method with linear convergence properties (3.18). By appropriately managing the data between memory and disk, as well as applying some heuristic techniques such as sorting constraints by dual variable values, we were able to apply SOR to massive problems with up to ten million data points in 32 dimensions. Experiments showed that SOR ran faster than SMO [96], and significantly faster than SVM^{light} [49], while maintaining similar generalization capability [70].

SOR can be extended for finding nonlinear classification surfaces by inserting a nonlinear kernel function into the dual formulation (4.5). This results in Algorithm 4.1, which is a generalization of Algorithm 3.1 with similar convergence properties. Experiments demonstrated use of this technique in successfully improving generalization accuracy over linear separators on a number of UCI datasets [85] and a synthetic dataset as well. Additionally, SOR has substantially fewer memory requirements than many other techniques in practice. A linear programming approach given in (4.17) is also shown to generalize well. The formulation, based on the generalized SVM [64], is interesting because it solves a linear program instead of a quadratic program [67]. It can successively handle nonlinear kernels, and accurately classifies the difficult "checkerboard" dataset as shown in Figure 4.4.

We proposed the ASVM Algorithm 5.1, which applies to a further extension of the standard SVM which additionally minimizes the square of the 2-norm of the soft margin error instead of the conventional 1-norm (5.6). The algorithm itself is an active set method which can solve massively sized linear SVM problems quickly, and requires nothing more complex than a commonly available linear equation solver [71]. In particular, we demonstrated that ASVM solved a problem of 7 million points and 32 features in only 96 minutes on a 400 MHz Pentium II (Table 5.2). The method should be extendible to larger datasets residing on disk, allowing even larger problems to be solved.

We proposed the LSVM Algorithm 6.1 which is considerably easier to code than SVM^{light} [48], SMO [96], SOR, and ASVM. It is capable of classifying datasets with millions of points, and can be implemented in a few lines of MATLAB code (Code 6.1, Code 6.2). For a linear kernel, LSVM is an iterative method which requires nothing more complex than the inversion of a single matrix of the order of the input space plus one and thus has the ability to handle massive problems (6.9). For a positive semidefinite nonlinear kernel, a single matrix inversion is required in the space of dimension equal to the number of points classified [72]. Hence for such nonlinear classifiers LSVM can handle only intermediate size problems.

ASVM and LSVM both obtain significant performance benefits by inverting a matrix whose size is on the order of the number of features of the problem. For most data mining problems, the number of features is significantly less than the number of points. It should be noted that some problems, such as those found in text data mining applications, can have tens or hundreds of thousands of features. SOR might be a more suitable algorithm for these kinds of problems. On the other hand, a practitioner might consider using a feature selection method [7, 9] to find an appropriate subset of the features to use. If the features can be reduced to a small subset, ASVM or LSVM may be appropriate.

9.2 Support Vector Machine Regression

Support vector regression provides a technique for solving regression problems while minimizing overfitting. We presented a new formulation of a linear program to perform support vector regression (7.7) which is considerably faster than other linear programming formulations of the same problem [110], yet it maintains identical test set generalization capability. Row-column chunking is a new approach presented which has been shown to successfully handle massive nonlinear regression problems with more than a billion entries [68].

Finally, a new methodology for solving the Huber M-estimator problem is presented which reduces the problem to a simple quadratic program (8.9). This formulation is shown to perform well when compared to other algorithms in the literature (Table 8.1). A modification of this quadratic program (8.26) is introduced to allow kernel functions to be used [69]. These kernel functions are used to find nonlinear regression surfaces which can yield better test set performances than a purely linear separating surface (Table 8.2).

9.3 Future Directions

There are many directions in which the ideas in this thesis can be taken. Many of the algorithms can be extended to run in parallel or distributed environments, taking advantage of additional computing power where available. There are a large number of heuristics which have been developed for older algorithms such as SMO and SVM^{light}. Some of these techniques, if applied to the algorithms contained herein, might provide additional performance gains.

In a broader sense, there are a number of directions in which SVM research may go. There is current work in reducing the number of support vectors used in the final separating surface [37, 108, 109]. This can provide a solution which takes up less space, and classifies test data more quickly. Some of these methodologies also can result in much faster training time. Another area of recent interest has been semi-supervised support vector learning, where training data of unknown classification can be used to improve generalization accuracy [3, 36]. As SVMs become more popular among practitioners, more work will be done in applying SVMs to different kinds of applications. This has been illustrated by applications in face detection [92] and the classification of microarray gene expression data [13, 84].

Support vector machines today are a growing area area of research. The number of SVM papers at `www.kernel-machines.org` have roughly doubled every year since 1995. Given the successes seen in this thesis and in other papers, it should be expected that SVM use will continue to grow among researchers and practitioners.

9.4 Final Wrapup

In this thesis, we have considered various new formulations of support vector machines for classification and regression. These new formulations contribute to the data mining community in that they are scalable and can therefore be applied to massive datasets. Some of the algorithms have been shown to successfully solve problems with millions of data points. Very few other approaches in the SVM literature have considered problems of this scale [8]. The techniques also offer significant speed increases over previous ones, and have been shown to exhibit strong performance on various real world datasets. It is our hope that the ideas presented in this thesis are ultimately used by practitioners, and help to promote the use of support vector machines among the data mining community.

Bibliography

[1] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen. *LAPACK Users' Guide*. Society for Industrial and Applied Mathematics, Philadelphia, PA, third edition, 1999.

[2] K. P. Bennett. Decision tree construction via linear programming. In M. Evans, editor, *Proceedings of the 4th Midwest Artificial Intelligence and Cognitive Science Society Conference*, pages 97–101, Utica, Illinois, 1992.

[3] K. P. Bennett and A. Demiriz. Semi-supervised support vector machines. In M. S. Kearns, S. A. Solla, and D. A. Cohn, editors, *Advances in Neural Information Processing Systems -10-*, pages 368–374, Cambridge, MA, 1998. MIT Press.

[4] K. P. Bennett, D. Hui, and L. Auslender. On support vector decision trees for database marketing. Department of Mathematical Sciences Math Report No. 98-100, Rensselaer Polytechnic Institute, Troy, NY 12180, March 1998. `http://www.math.rpi.edu/~bennek/`.

[5] D. P. Bertsekas. *Nonlinear Programming*. Athena Scientific, Belmont, MA, second edition, 1999.

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

[7] P. S. Bradley and O. L. Mangasarian. Feature selection via concave minimization and support vector machines. In J. Shavlik, editor, *Machine Learning Proceedings of the Fifteenth International Conference(ICML '98)*, pages 82–90, San Francisco, California, 1998. Morgan Kaufmann. `ftp://ftp.cs.wisc.edu/math-prog/tech-reports/98-03.ps`.

[8] P. S. Bradley and O. L. Mangasarian. Massive data discrimination via linear support vector machines. *Optimization Methods and Software*, 13:1–10, 2000. `ftp://ftp.cs.wisc.edu/math-prog/tech-reports/98-03.ps`.

[9] P. S. Bradley, O. L. Mangasarian, and W. N. Street. Feature selection via mathematical programming. *INFORMS Journal on Computing*, 10(2):209–217, 1998. `ftp://ftp.cs.wisc.edu/math-prog/tech-reports/95-21.ps`.

[10] E. J. Bredensteiner. *Optimization Methods in Data Mining and Machine Learning*. PhD thesis, Department of Mathematical Sciences, Rensselaer Polytechnic Institute, Troy, NY, 1997.

[11] E. J. Bredensteiner and K. P. Bennett. Feature minimization within decision trees. *Computational Optimizations and Applications*, 10:111–126, 1998.

[12] L. Breiman, J.H. Friedman, R.A. Olshen, and C.J. Stone. *Classification and Regression Trees*. Wadsworth Publishing Company, Belmont, CA, 1983.

[13] M.P.S. Brown, W.N. Grundy, D. Lin, N. Cristianini, C. Sugnet, M. Ares, and D. Haussler. Support vector machine classification of microarray gene expression data. Technical Report UCSC-CRL-99-09, University of California, Santa Cruz, 1999.

[14] W.L. Buntine. Graphical models for discovering knowledge. In *Advances in Knowledge Discovery and Data Mining*, pages 59–81. AAAI Press, Menlo Park, CA, 1996.

[15] US Census Bureau. Adult dataset. Publicly available from: `http://www.sgi.com/Technology/mlc/db/`.

[16] C. J. C. Burges. A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, 2(2):121–167, 1998.

[17] Chunhui Chen and O. L. Mangasarian. Hybrid misclassification minimization. *Advances in Computational Mathematics*, 5(2):127–136, 1996. `ftp://ftp.cs.wisc.edu/math-prog/tech-reports/95-05.ps`.

[18] V. Cherkassky and F. Mulier. *Learning from Data — Concepts, Theory and Methods*. John Wiley & Sons, New York, 1998.

[19] V. Chvatal. *Linear Programming*. W.H. Freeman and Company, 1983.

[20] CLAPACK. f2c'ed version of LAPACK. `http://www.netlib.org/clapack`.

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

[22] C. W. Cryer. The solution of a quadratic programming problem using systematic overrelaxation. *SIAM Journal on Control and Optimization*, 9:385–392, 1971.

[23] G. B. Dantzig. *Linear Programming and Extensions*. Princeton University Press, Princeton, New Jersey, 1963.

[24] G. B. Dantzig and P. Wolfe. Decomposition principle for linear programs. *Operations Research*, 8:101–111, 1960.

[25] R. De Leone and M. A. Tork Roth. Massively parallel solution of quadratic programs via successive overrelaxation. *Concurrency: Practice and Experience*, 5:623–634, 1993.

[26] D. DeCoste. Recent advances in SMO speed and accuracy. Talk given at NIPS99 Workshop on Learning with Support Vectors, December 1999.

[27] Delve. Data for evaluating learning in valid experiments. `http://www.cs.utoronto.ca/~delve/`.

[28] T. G. Dietterich. Approximate statistical tests for comparing supervised classification learning algorithms. *Neural Computation*, 1998. `http://www.cs.orst.edu/~tgd/cv/pubs.html`.

[29] S. P. Dirkse and M. C. Ferris. MCPLIB: A collection of nonlinear mixed complementarity problems. *Optimization Methods and Software*, 5:319–345, 1995. `ftp://ftp.cs.wisc.edu/tech-reports/reports/94/tr1215.ps`.

[30] N. R. Draper and H. Smith. *Applied Regression Analysis*. John Wiley and Sons, New York, 1966.

[31] H. Ekblom. A new algorithm for the Huber estimator in linear models. *BIT*, 28:123–132, 1988.

[32] U. Fayyad, G. Piatetsky-Shapiro, and P. Smyth. The KDD process for extracting useful knowledge from volumes of data. *Communications of the ACM*, 39:27–34, 1996.

[33] M. C. Ferris and T. S. Munson. Interior point methods for massive support vector machines. Technical report, Computer Sciences Department, University of Wisconsin, Madison, Wisconsin, May 2000. `ftp://ftp.cs.wisc.edu/pub/dmi/tech-reports/00-05.ps`.

[34] T.-T. Friess. Support vector neural networks: The kernel adatron with bias and soft margin. Technical report, Department of Automatic Control and Systems Engineering, University of Sheffield, Sheffield, England, 1998. Revised Version: `http://www.brunner-edv.com/friess/`.

[35] T.-T. Friess, N. Cristianini, and C. Campbell. The kernel-adatron algorithm: A fast and simple learning procedure for support vector machines. In Jude Shavlik, editor, *Machine Learning Proceedings of the Fifteenth International Conference (ICML'98)*, pages 188–196, San Francisco, 1998. Morgan Kaufmann. `http://svm.first.gmd.de/papers/FriCriCam98.ps.gz`.

[36] G. Fung and O. L. Mangasarian. Semi-supervised support vector machines for unlabeled data classification. Technical Report 99-06, Data Mining Institute, Computer Sciences Department, University of Wisconsin, Madison, Wisconsin, October 1999. `ftp://ftp.cs.wisc.edu/pub/dmi/tech-reports/99-05.ps`.

[37] G. Fung and O. L. Mangasarian. Data selection for support vector machine classifiers. Technical Report 00-02, Data Mining Institute, Computer Sciences Department, University of Wisconsin, Madison, Wisconsin, April 2000. `ftp://ftp.cs.wisc.edu/pub/dmi/tech-reports/00-02.ps`.

[38] D. Gale. *The Theory of Linear Economic Models.* McGraw-Hill Book Company, New York, 1960.

[39] P. C. Gilmore and R. E. Gomory. A linear programming approach to the cutting stock problem. *Operations Research*, 9:849–859, 1961.

[40] G. H. Golub and C. F. Van Loan. *Matrix Computations.* The John Hopkins University Press, Baltimore, Maryland, 3rd edition, 1996.

[41] D. Heckerman. Bayesian networks for knowledge discovery. In *Advances in Knowledge Discovery and Data Mining*, pages 273–305. AAAI Press, Menlo Park, CA, 1996.

[42] J. Hertz, A. Krogh, and R. G. Palmer. *Introduction to the Theory of Neural Computation*. Addison-Wesley, Redwood City, California, 1991.

[43] T.K. Ho and E.M. Kleinberg. Building projectable classifiers of arbitrary complexity. In *Proceedings of the 13th International Conference on Pattern Recognition*, pages 880–885, Vienna, Austria, 1996. `http://cm.bell-labs.com/who/tkh/pubs.html`. Checker dataset at: `ftp://ftp.cs.wisc.edu/math-prog/cpo-dataset/machine-learn/checker`.

[44] P.W. Holland and R.E. Welsch. Robust regression using iteratively reweighted least squares. *Communications in Statistics — Theory and Methods*, A6:813–827, 1977.

[45] P. J. Huber. *Robust Statistics*. John Wiley, New York, 1981.

[46] P. J. Huber and R. Dutter. Numerical solution of robust regression problems. In *COMPSTAT 1974, Proceedings of the Symposium on Computational Statistics, G. Brushmann (editor)*, pages 165–172, Berlin, 1974. Physike Verlag.

[47] ILOG CPLEX Division, Incline Village, Nevada. *ILOG CPLEX 6.5 Reference Manual*, 1999.

[48] T. Joachims. SVM^{light}, 1998. `http://www-ai.informatik.uni-dortmund.de/FORSCHUNG/VERFAHREN/SVM_LIGHT/svm_light.eng.html`.

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

[50] L. Kaufman. Solving the quadratic programming problem arising in support vector classification. In B. Schölkopf, C. J. C. Burges, and A. J. Smola, editors, *Advances in Kernel Methods — Support Vector Learning*, pages 147–167. MIT Press, 1999.

[51] S.S. Keerthi, S.K. Shevade, C. Bhattacharyya, and K.R.K. Murthy. Improvements to Platt's SMO algorithm for SVM classifier design. Technical Report CD-99-14,

National University of Singapore, Singapore, 1999. Available at `http://guppy.mpe.nus.edu.sg/~mpessk/smo_mod.ps.gz`.

[52] R. Kohavi and P. Foster. Editorial: Glossary of terms. *Machine Learning*, 30:271–274, 1998.

[53] Yuh-Jye Lee and O. L. Mangasarian. SSVM: A smooth support vector machine. Technical report. *Computational Optimization and Applications*, to appear. `ftp://ftp.cs.wisc.edu/pub/dmi/tech-reports/99-03.ps`.

[54] W. Li. Remarks on convergence of matrix splitting algorithm for the symmetric linear complementarity problem. *SIAM Journal on Optimization*, 3:155–163, 1993.

[55] W. Li. Numerical estimates for the Huber M-estimator problem. In C. K. Chui and L. L. Schumaker, editors, *Approximation Theory VIII*, pages 325–334, New York, 1995. World Scientific Publishing Co.

[56] W. Li and J. J. Swetits. The linear ℓ_1 estimator and the Huber M-estimator. *SIAM Journal on Optimization*, 8:457–475, 1998.

[57] Z.-Q. Luo and P. Tseng. Error bounds and convergence analysis of feasible descent methods: A general approach. *Annals of Operations Research*, 46:157–178, 1993.

[58] K. Madsen and H. B. Nielsen. Finite algorithms for robust linear regression. *BIT*, 30:682–699, 1990.

[59] O. L. Mangasarian. *Nonlinear Programming.* McGraw–Hill, New York, 1969. Reprint: SIAM Classic in Applied Mathematics 10, 1994, Philadelphia.

[60] O. L. Mangasarian. Solution of symmetric linear complementarity problems by iterative methods. *Journal of Optimization Theory and Applications*, 22(4):465–485, August 1977.

[61] O. L. Mangasarian. On the convergence of iterates of an inexact matrix splitting algorithm for the symmetric monotone linear complementarity problem. *SIAM Journal on Optimization*, 1:114–122, 1991.

[62] O. L. Mangasarian. *Nonlinear Programming.* SIAM, Philadelphia, PA, 1994.

[63] O. L. Mangasarian. Mathematical programming in data mining. *Data Mining and Knowledge Discovery*, 1(2):183–201, 1997. `ftp://ftp.cs.wisc.edu/math-prog/tech-reports/96-05.ps`.

[64] O. L. Mangasarian. Generalized support vector machines. In A. Smola, P. Bartlett, B. Schölkopf, and D. Schuurmans, editors, *Advances in Large Margin Classifiers*, pages 135–146, Cambridge, MA, 2000. MIT Press. `ftp://ftp.cs.wisc.edu/math-prog/tech-reports/98-14.ps`.

[65] O. L. Mangasarian and R. De Leone. Parallel gradient projection successive overrelaxation for symmetric linear complementarity problems. *Annals of Operations Research*, 14:41–59, 1988.

[66] O. L. Mangasarian and R. R. Meyer. Nonlinear perturbation of linear programs. *SIAM Journal on Control and Optimization*, 17(6):745–752, November 1979.

[67] O. L. Mangasarian and D. R. Musicant. Data discrimination via nonlinear generalized support vector machines. Technical Report 99-03, Computer Sciences Department, University of Wisconsin, Madison, Wisconsin, March 1999. To appear in: "Applications and Algorithms of Complementarity", M. C. Ferris, O. L. Mangasarian and J.-S. Pang, editors, Kluwer Academic Publishers,Boston 2000. `ftp://ftp.cs.wisc.edu/math-prog/tech-reports/99-03.ps`.

[68] O. L. Mangasarian and D. R. Musicant. Massive support vector regression. Technical Report 99-02, Data Mining Institute, Computer Sciences Department, University of Wisconsin, Madison, Wisconsin, July 1999. `ftp://ftp.cs.wisc.edu/pub/dmi/tech-reports/99-02.ps`.

[69] O. L. Mangasarian and D. R. Musicant. Robust linear and support vector regression. Technical Report 99-09, Data Mining Institute, Computer Sciences Department, University of Wisconsin, Madison, Wisconsin, Novemeber 1999. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, to appear. `ftp://ftp.cs.wisc.edu/pub/dmi/tech-reports/99-09.ps`.

[70] O. L. Mangasarian and D. R. Musicant. Successive overrelaxation for support vector machines. *IEEE Transactions on Neural Networks*, 10:1032–1037, 1999. `ftp://ftp.cs.wisc.edu/math-prog/tech-reports/98-18.ps`.

[71] O. L. Mangasarian and D. R. Musicant. Active support vector machine classification. Technical Report 00-04, Data Mining Institute, Computer Sciences Department, University of Wisconsin, Madison, Wisconsin, April 2000. `ftp://ftp.cs.wisc.edu/pub/dmi/tech-reports/00-04.ps`.

[72] O. L. Mangasarian and D. R. Musicant. Lagrangian support vector machines. Technical Report 00-06, Data Mining Institute, Computer Sciences Department, University of Wisconsin, Madison, Wisconsin, June 2000. `ftp://ftp.cs.wisc.edu/pub/dmi/tech-reports/00-06.ps`.

[73] O. L. Mangasarian and J. Ren. New improved error bounds for the linear complementarity problem. *Mathematical Programming*, 66:241–255, 1994.

[74] O. L. Mangasarian and T.-H. Shiau. Lipschitz continuity of solutions of linear inequalities, programs and complementarity problems. *SIAM Journal on Control and Optimization*, 25(3):583–595, May 1987.

[75] O. L. Mangasarian and M. V. Solodov. Nonlinear complementarity as unconstrained and constrained minimization. *Mathematical Programming, Series B*, 62:277–297, 1993.

[76] MATLAB. *User's Guide*. The MathWorks, Inc., Natick, MA 01760, 1992.

[77] MATLAB. *Application Program Interface Guide*. The MathWorks, Inc., Natick, MA 01760, 1997.

[78] J. L. McClelland and D. E. Rumelhart. *Explorations in Parallel Distributed Processing: A Handbook of Models, Programs, and Exercises*. MIT Press, Cambridge, Massachusetts, 1987.

[79] S. Menon and R. Sharda. Data mining update: New modes to pursue old objectives. *OR/MS Today*, pages 26–29, June 1999.

[80] R. Meyer. Private communication, July 2000.

[81] T. M. Mitchell. *Machine Learning*. McGraw-Hill, Boston, 1997.

[82] J. J. Moré and G. Toraldo. Algorithms for bound constrained quadratic programs. *Numerische Mathematik*, 55:377–400, 1989.

[83] B. Moxon. Defining data mining. *DBMS*, August 1996. Available at `http://www.dbmsmag.com/9608d53.html`.

[84] S. Mukherjee, P. Tamayo, J.P. Mesirov, D. Slonim, A. Verri, and T. Poggio. Support vector machine classification of microarray data. Technical Report 182, AI Memo 1676, CBCL, 1999.

[85] P. M. Murphy and D. W. Aha. UCI repository of machine learning databases, 1992. `http://www.ics.uci.edu/~mlearn/MLRepository.html.`

[86] B. A. Murtagh and M. A. Saunders. MINOS 5.0 user's guide. Technical Report SOL 83.20, Stanford University, December 1983. MINOS 5.4 Release Notes, December 1992.

[87] D. R. Musicant. NDC: Normally Distributed Clustered Datasets, 1998. `http://www.cs.wisc.edu/~musicant/data/ndc/`.

[88] D. R. Musicant. ASVM Software: Active Set Support Vector Machine Classification Software, 2000. `http://www.cs.wisc.edu/~musicant/data/asvm`.

[89] J. M. Ortega. *Numerical Analysis, A Second Course*. Academic Press, 1972.

[90] J. M. Ortega and W. C. Rheinboldt. *Iterative Solution of Nonlinear Equations in Several Variables*. Academic Press, 1970.

[91] 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.

[92] E. Osuna, R. Freund, and F. Girosi. Training support vector machines: An application to face detection. In *IEEE Conference on Computer Vision and Pattern Recognition, Puerto Rico, June 1997, 130-136*, 1997. `http://www.ai.mit.edu/people/girosi/home-page/svm.html`.

[93] J.-S. Pang. More results on the convergence of iterative methods for the symmetric linear complementarity problem. *Journal of Optimization Theory and Applications*, 49:107–134, 1986.

[94] R.L. Plackett. *Regression Analysis*. Clarendon Press, Oxford, UK, 1960.

[95] J. Platt. Sequential minimal optimization: A fast algorithm for training support vector machines. [102], pages 185–208. `http://www.research.microsoft.com/~jplatt/smo.html`.

[96] J. Platt. Sequential minimal optimization: A fast algorithm for training support vector machines. In B. Schölkopf, C. J. C. Burges, and A. J. Smola, editors, *Advances in Kernel Methods — Support Vector Learning*, pages 185–208. MIT Press, 1999. `http://www.research.microsoft.com/~jplatt/smo.html`.

[97] B. T. Polyak. *Introduction to Optimization*. Optimization Software, Inc., Publications Division, New York, 1987.

[98] J.R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers, San Mateo, CA, 1993.

[99] R. Ramakrishnan. *Database Management Systems*. WCB/McGraw-Hill, Boston, MA, 1998.

[100] B. Schölkopf, P. Bartlett, A. Smola, and R. Williamson. Support vector regression with automatic accuracy control. In L. Niklasson, M. Boden, and T. Ziemke, editors, *Proceedings of the 8th International Conference on Artificial Neural Networks*, pages 111–116, Berlin, 1998. Springer Verlag. Available at `http://www.kernel-machines.org/publications.html`.

[101] B. Schölkopf, P. Bartlett, A. Smola, and R. Williamson. Shrinking the tube: a new support vector regression algorithm. In M. S. Kearns, S. A. Solla, and D. A. Cohn, editors, *Advances in Neural Information Processing Systems 11*, pages 330–336, Cambridge, MA, 1999. MIT Press. Available at `http://www.kernel-machines.org/publications.html`.

[102] B. Schölkopf, C. Burges, and A. Smola (editors). *Advances in Kernel Methods — Support Vector Machines*. MIT Press, Cambridge, MA, 1998.

[103] G.A.F. Seber. *Linear Regression Analysis*. Wiley, New York, NY, 1977.

[104] D. F. Shanno and D. M. Rocke. Numerical methods for robust regression: Linear models. *SIAM Journal on Scientific and Statistical Computing*, 7:86–97, 1986.

[105] J. W. Shavlik and T. G. Dietterich (editors). *Readings in Machine Learning*. Morgan Kaufman, San Mateo, California, 1990.

[106] R. Simons. Why mathematical programming is useful. MP In Action: The Newsletter of Mathematical Programming in Industry and Commerce, August 1994. `http://www.exodus.com/action.html`.

[107] A. Smola. Regression estimation with support vector learning machines. Master's thesis, Technische Universität München, München, Germany, 1996.

[108] A. Smola, O. L. Mangasarian, and B. Schölkopf. Sparse kernel feature analysis. Technical Report 99-04, Data Mining Institute, Computer Sciences Department, University of Wisconsin, Madison, Wisconsin, October 1999. `ftp://ftp.cs.wisc.edu/pub/dmi/tech-reports/99-04.ps`.

[109] A. Smola and B. Schölkopf. Sparse greedy matrix approximation for machine learning. Technical report, 2000. International Conference on Machine Learning. Available at `http://www.kernel-machines.org/publications.html`.

[110] A. Smola, B. Schölkopf, and G. Rätsch. Linear programs for automatic accuracy control in regression. In *Ninth International Conference on Artificial Neural Networks*, Conference Publications No. 470, pages 575–580, London, 1999. IEE. Available at `http://www.kernel-machines.org/publications.html`.

[111] A. J. Smola. *Learning with Kernels*. PhD thesis, Technische Universität Berlin, Berlin, Germany, 1998.

[112] M. Stone. Cross-validatory choice and assessment of statistical predictions. *Journal of the Royal Statistical Society*, 36:111–147, 1974.

[113] W. N. Street and O. L. Mangasarian. Improved generalization via tolerant training. *Journal of Optimization Theory and Applications*, 96(2):259–279, February 1998. `ftp://ftp.cs.wisc.edu/math-prog/tech-reports/95-11.ps`.

[114] Y. Z. Tsypkin. *Foundations of the Theory of Learning Systems*. Academic Press, New York, 1973.

[115] R.J. Vanderbei. LOQO User's Manual - version 4.05. Technical Report ORFE-99-??, Princeton University, Princeton, New Jersey, July 1999. Available at http://www.sor.princeton.edu/~rvdb/ps/loqo/loqo405.pdf.

[116] V. N. Vapnik. *Estimation of Dependences Based on Empirical Data*. Springer, New York, 1982.

[117] V. N. Vapnik. *The Nature of Statistical Learning Theory*. Springer, New York, 1995.

[118] G. Wahba. *Spline Models for Observational Data*. SIAM, Philadelphia, PA, 1990.

[119] G. Wahba, Y. Lin, and H. Zhang. Generalized approximate cross validation for support vector machines, or, another way to look at margin-like quantities. In A. Smola, P. Bartlett, B. Schölkopf, and P. Schuurmans, editors, *Advances in Large Margin Classifiers*. MIT Press. To appear.

[120] G. Wahba, Y. Wang, C. Gu, R. Klein, and B. Klein. Structured machine learning for 'soft' classification with smoothing spline anova and stacked tuning, testing and evaluation. In *Advances in Neural Information Processing Systems*, volume 6, pages 415–422, San Mateo, CA, 1994. Morgan Kaufmann.

[121] S. J. Wright. Using complementarity and optimization methods in statistics. A talk given at the International Conference on Complementarity Problems, June 9-12, 1999. Madison, Wisconsin.