OPTIMIZATION-BASED MACHINE LEARNING AND DATA MINING

by

Edward W. Wild III

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

Doctor of Philosophy

(Computer Sciences)

at the

UNIVERSITY OF WISCONSIN–MADISON

2008

Dedicated to Mom, Dad, Bill, Amanda and Julianna.

ACKNOWLEDGMENTS

I wish to thank my adviser, Olvi Mangasarian, and the other members of the committee: Michael Ferris, Robert Meyer, Jude Shavlik, Grace Wahba, and Jerry Zhu. Also, I wish to thank my coauthors: Glenn Fung, Rich Maclin, Lisa Torrey, and Trevor Walker. Hector Corrada Bravo, Greg Quinn, and Nicholas LeRoy provided assistance with Condor, David Musicant provided helpful feedback on the material covered in Chapter 2, and Soumya Ray provided access to datasets used in Chapter 8. I also wish to thank my former office mate, Michael Thompson, the AI Reading Group, the research groups of Grace Wahba and Sündüz Keleş, Jim Doherty and the members of the UW Big Band, and everyone else in the UW community that I have had the fortune to interact with these past six years. Finally, I want to thank Ray Mooney of the University of Texas for getting me started.

DISCARD THIS PAGE

TABLE OF CONTENTS

Page

Page

DISCARD THIS PAGE

LIST OF TABLES

DISCARD THIS PAGE

LIST OF FIGURES

Figure Page

Figure Page

Figure Page

ABSTRACT

Novel approaches for six important problems in machine learning, and two methods for solving linear programs, are introduced. Each machine learning problem is addressed by formulating it as an optimization problem. By using results based on theorems of the alternative for linear or convex functions, we are able to incorporate prior knowledge into function approximations or classifiers generated by linear combinations of linear or nonlinear kernels. We will consider prior knowledge consisting of linear inequalities to be satisfied over multiple polyhedral regions, *nonlinear* inequalities to be satisfied over *arbitrary* regions, and nonlinear *equalities* to be satisfied over arbitrary regions. Each kind of prior knowledge leads to different formulations, each with certain advantages.

In privacy-preserving classification, data is divided into groups belonging to different entities unwilling to share their privately-held data. By using a *completely random* matrix, we are able to construct public classifiers that do not reveal the privately held data, but have accuracy comparable to that of an ordinary support vector machine classifier based on the entire data.

To address the problem of feature selection in clustering, we propose a modification of the objective function of a standard clustering algorithm which allows features to be eliminated. For feature selection in nonlinear kernel classification, we propose a mixed-integer algorithm which alternates between optimizing the continuous variables of an ordinary nonlinear support vector machine and optimizing integer variables which correspond to selecting or removing features from the classifier.

We also propose a classifier based on proximity to two planes which are not required to be parallel. Finally, we tackle the multiple instance classification problem by formulating the problem as the minimization of a linear function subject to linear and bilinear constraints.

In several of these problems, the solution we propose involves solving linear programs. We consider sufficient conditions which allow us to determine whether a solution of a linear program obtained by either of two algorithms is exact. Both of these two algorithms can be implemented using only a solver for linear systems of equations.

Chapter 1

Introduction

We introduce novel approaches that address six important problems in machine learning, and also consider methods for solving linear programs, an important class of mathematical programs which arise in several of our solutions.

Our first problem is that of knowledge-based function approximation and classification. We tackle this problem by incorporating prior knowledge into a function approximation or classification generated by a linear combination of linear or nonlinear kernels. In addition, the approximation or classifier needs to satisfy conventional conditions such as given exact or inexact function values or class labels at certain points. First, we shall introduce prior knowledge in the form of linear inequalities that a function approximation should satisfy over multiple polyhedral regions. This approach leads to a linear programming formulation, which can be solved efficiently. By using nonlinear kernels and mapping the prior polyhedral knowledge in the input space to one defined by the kernels, the prior knowledge translates into nonlinear inequalities in the original input space. Through a number of computational examples, including a real-world breast cancer prognosis dataset and an application to reinforcement learning, it is shown that prior knowledge can significantly improve function approximation.

Although nonlinear kernels lead to good computational results for prior knowledge expressed as linear inequalities over polyhedral regions, the interpretation of the prior knowledge in the original input space is lost when mapping to the kernel space. To increase the generality of prior knowledge added to a nonlinear classifier or function approximation,

we consider prior knowledge in the form of nonlinear inequalities that need to be satisfied over arbitrary sets. A sufficient condition for prior knowledge incorporation is given which requires *no assumptions whatsoever* on the knowledge. Adding this sufficient condition to a linear programming formulation for nonlinear kernel classification or approximation leads to a semi-infinite linear program, that is, a linear program with a finite number of variables but an infinite number of constraints. Discretizing over these constraints results in a finite linear program. Computational examples on several datasets, including a real-world breast cancer prognosis dataset demonstrate that incorporating prior knowledge can significantly improve function approximation or classification. We will also see that very similar prior knowledge, involving arbitrary nonlinear *equalities* over arbitrary regions, can be incorporated as linear *equality* constraints. These constraints can be added to a proximal support vector machine, requiring only the solution of a linear system of equations. Computational examples show that this formulation has similar accuracy to the linear programming formulation, while being approximately an order of magnitude faster to solve.

Our second problem is privacy-preserving classification. We propose a privacy-preserving SVM classifier in which the data matrix A is divided into groups belonging to different entities. In the first case, each entity owns a different group of the input feature columns of A, and A is said to be *vertically partitioned*. In the second case, each entity owns a different group of individual rows of A, and A is said to be *horizontally partitioned*. In both cases, the entities are unwilling to share their portion of A with the other entities. Our classifiers for both kinds of data are based on the concept of a reduced kernel $K(A, B')$ where B' is the transpose of a completely random matrix B. The classifiers, which are public but do not reveal the privately-held data, have accuracy comparable to that of an ordinary SVM classifier based on the entire data.

Our third problem is that of selecting features in clustering unlabeled data. We propose a modification of the objective function of the standard k-median clustering algorithm. The modification consists of perturbing the objective function by a term that drives the medians of each of the k clusters toward the (shifted) global median of zero for the entire dataset. As

the perturbation parameter is increased, more and more features are driven automatically toward the global zero median and are eliminated from the problem. An error curve for unlabeled data clustering as a function of the number of features used gives reduced-feature clustering error relative to the "gold standard" of the full-feature clustering. This clustering error curve parallels a classification error curve based on real data labels. This fact justifies the utility of the former error curve for unlabeled data as a means of choosing an appropriate number of reduced features in order to achieve a correctness comparable to that obtained by the original, full set of features. For example, on the three-class Wine dataset, clustering with four selected input space features is comparable to within 4% to clustering using the original thirteen features of the problem.

Our fourth problem is that of selecting features in nonlinear kernel classifiers. We propose an easily implementable mixed-integer algorithm is that generates a nonlinear kernel support vector machine (SVM) classifier with reduced input space features. A single parameter controls the reduction. On one publicly available dataset, the algorithm obtains 92.4% accuracy with 34.7% of the features compared to 94.1% accuracy with all features. On a synthetic dataset with 1000 features, 900 of which are irrelevant, our approach improves the accuracy of a full-feature classifier by over 30%. The proposed algorithm introduces a diagonal matrix E with ones for features present in the classifier and zeros for removed features. By alternating between optimizing the continuous variables of an ordinary nonlinear SVM and the integer variables on the diagonal of E, a decreasing sequence of objective function values is obtained. This sequence converges to a local solution minimizing the usual data fit error and solution complexity while *also* minimizing the number of features used.

Our fifth problem is that of binary data classification without the parallelism restriction of data-bounding or proximal planes. We propose here a new approach to support vector machine (SVM) classification wherein each of two datasets are proximal to one of two distinct planes that are *not parallel* to each other. Each plane is generated such that it is closest to

one of the two datasets and as far as possible from the other dataset. Each of the two non-parallel proximal planes is obtained by a single MATLAB [94] command as the eigenvector corresponding to a smallest eigenvalue of a generalized eigenvalue problem. Classification by proximity to two distinct nonlinear surfaces generated by a nonlinear kernel also leads to two simple generalized eigenvalue problems. Effectiveness of the proposed method is demonstrated by tests on simple examples as well as on a number of public datasets. These examples show advantages of the proposed approach in both computation time and test set correctness.

Our sixth problem is that of multiple-instance classification [16, 2, 62] for which a novel formulation is proposed, using a linear or nonlinear kernel, as the minimization of a linear function in a finite dimensional (noninteger) real space subject to linear and bilinear constraints. In multiple-instance classification, the goal is to separate positive and negative *bags*, each containing multiple points, such that for each positive bag *at least one* point in the bag is on the positive side of the decision surface while for each negative bag *all* points in the bag are on the negative side of the decision surface. A linearization algorithm is proposed that solves a succession of fast linear programs that converges in a few iterations to a local solution. Computational results on a number of datasets indicate that the proposed algorithm is competitive with the considerably more complex integer programming and other formulations.

In several of the problems described above, our solution involves solving linear programs. We give sufficient conditions for a classical dual exterior penalty function of a linear program to be independent of its penalty parameter. This ensures that an exact solution to the primal linear program can be obtained by minimizing the dual exterior penalty function. We use the sufficient conditions to give a precise value to such a penalty parameter introduced in [75], and also in a fast iterative algorithm that solves a sequence of linear equations. Both algorithms can be implemented using only a solver for linear systems of equations, which are more readily available than linear programming packages. Computational results on linear

programs with up to one million variables or constraints compare favorably to CPLEX 9.0 [48] and validate the proposed approach.

1.1 Notation

All vectors will be column vectors unless transposed to a row vector by a prime $'$. The scalar (inner) product of two vectors x and y in the n-dimensional real space R^n will be denoted by $x'y$. A vector of ones in a real space of arbitrary dimension will be denoted by e. Thus for $e \in R^m$ and $y \in R^m$ the notation $e'y$ will denote the sum of the components of y. A vector of zeros in a real space of arbitrary dimension will be denoted by 0. For $x \in R^n$ and $1 \leq p < \infty$, the p-norm and the ∞-norm are defined as follows:

$$\|x\|_p = \left(\sum_{j=1}^{n} |x_j|^p\right)^{\frac{1}{p}}, \ \|x\|_\infty = \max_{1 \leq j \leq n} |x_j|.$$

For simplicity, in some specific instances $\|x\|$ will stand for the 2-norm of x. The plus function $(x)_+$ is defined as $max\{0, x\}$, while $sign(x)$ denotes a vector with components of ± 1 for positive and negative components of x respectively and 0 for zero components of x.

The notation $A \in R^{m \times n}$ will signify a real $m \times n$ matrix. For such a matrix, A' will denote the transpose of A, A_i will denote the i-th row of A and $A_{\cdot j}$ the j-th column of A. The identity matrix of arbitrary dimension will be denoted by I. For $A \in R^{m \times n}$ and $B \in R^{n \times k}$, a *kernel* $K(A, B)$ is an arbitrary function that maps $R^{m \times n} \times R^{n \times k}$ into $R^{m \times k}$. In particular, if x and y are column vectors in R^n then $K(x', y)$ is a real number, $K(x', A')$ is a row vector in R^m and $K(A, A')$ is an $m \times m$ matrix. The base of the natural logarithm will be denoted by ε. A frequently used kernel in nonlinear classification is the Gaussian kernel [118, 11, 72] whose ij-th element, $i = 1 \ldots, m, \ j = 1 \ldots, k$, is given by: $(K(A, B))_{ij} = \varepsilon^{-\mu \|A_i' - B_{\cdot j}\|^2}$, where $A \in R^{m \times n}$, $B \in R^{n \times k}$ and μ is a positive constant.

A *separating plane*, with respect to two given point sets $\mathcal{A}$ and $\mathcal{B}$ in R^n, is a plane that attempts to separate R^n into two halfspaces such that each open halfspace contains points mostly of $\mathcal{A}$ or $\mathcal{B}$. A *bounding plane* to the set $\mathcal{A}$ is a plane that places $\mathcal{A}$ in one of the two closed halfspaces that the plane generates.

For a convex function $f : R^n \longrightarrow R^1$ that is nondifferentiable, such as $\|x\|_1$, a *subgradient* $\partial f(x) \in R^n$ exists [106, Theorem 23.4], also [102], with the property that:

$$f(y) - f(x) \geq \partial f(x)'(y - x),\ \forall x, y \in R^n. \tag{1.1}$$

Thus for $\|x\|_1$, $x \in R^n$ and $i = 1, \ldots, n$, a subgradient $\partial\|x\|_1$ satisfies:

$$(\partial\|x\|_1)_i = \begin{cases} -1 & \text{if } x_i < 0 \\ \in [-1, 1] & \text{if } x_i = 0 \\ +1 & \text{if } x_i > 0 \end{cases} \tag{1.2}$$

The subgradient plays the role of a gradient for differentiable convex functions, except that it is not unique. Thus a necessary and sufficient condition for x to be a minimizer of $f(x)$ on an open set is that

$$\partial f(x) = 0. \tag{1.3}$$

For a concave function f on R^n, the inequality (1.1) is reversed and $\partial f(x)$ is called a *supergradient* of f at x and (1.3) becomes a necessary and sufficient condition for x to be a maximizer of $f(x)$ on an open set.

For a piecewise-quadratic function, such as $f(x) = \frac{1}{2}||(Ax - b)_+||^2 + \frac{1}{2}x'Px$, where $A \in R^{m\times n}$, $P \in R^{n\times n}$, $P = P'$, P positive semidefinite and $b \in R^m$, the ordinary Hessian does not exist because its gradient, the $n \times 1$ vector $\nabla f(x) = A'(Ax-b)_+ + Px$, is not differentiable but is Lipschitz continuous with a Lipschitz constant of $\|A'\|\,\|A\| + \|P\|$. However, one can define its **generalized Hessian** [43, 22, 74,] which is the $n \times n$ symmetric positive semidefinite matrix:

$$\partial^2 f(x) = A' diag\Big(sign\big((Ax - b)_+\big)\Big)A + P,$$

where $diag\Big(sign\big((Ax - b)_+\big)\Big)$ denotes an $m \times m$ diagonal matrix with diagonal elements $sign\big((A_i x - b_i)_+\big)$, $i = 1, \ldots, m$. The generalized Hessian has many of the properties of the regular Hessian [43, 22, 74,] in relation to $f(x)$. If the smallest eigenvalue of $\partial^2 f(x)$ is greater than some positive constant for all $x \in R^n$, then $f(x)$ is a strongly convex piecewise-quadratic function on R^n.

Approximate equality is denoted by $\approx$. The symbol $\wedge$ denotes the logical "and" while $\vee$ denotes the logical "or". For a countable set S, $card(S)$ denotes the cardinality of S, that is the number of elements in S. The abbreviation "s.t." stands for "subject to."

1.2 Support vector machines

In several of the following chapters, we utilize a support vector machine (SVM) for both classification and approximation. We consider a dataset in R^n represented by the m rows of the matrix $A \in R^{m \times n}$, with associated labels represented by the m-dimensional vector d. For approximation problems, d will be real valued, that is $d \in R^m$. For classification problems, d will have values $+1$ or -1 according to the class of each example, that is $d \in \{-1, +1\}^m$. The learned function f from R^n to R is defined as follows:

$$f(x) = K(x', B')u - \gamma \tag{1.4}$$

where $B \in R^{k \times n}$ is an arbitrary basis matrix and $K(x', B') : R^{1 \times n} \times R^{n \times k} \longrightarrow R^{1 \times k}$ is an arbitrary kernel function. In an approximation setting, we will use $f(x)$ directly to obtain a predicted value at $x \in R^n$, while in a classification setting we will use the value of the sign of $f(x)$ to obtain the predicted class of x. In general, the matrix B is set equal to A [72]. However, in reduced support vector machines [55, 47] $B = \bar{A}$, where $\bar{A}$ is a submatrix of A whose rows are a small subset of the rows of A. In fact, B can be an arbitrary matrix in $R^{k \times n}$ [81, 92, 90]. The variables $u \in R^k$ and $\gamma \in R$ are parameters determined by the following optimization problem:

$$\min_{u,\gamma} \ \nu L(u, \gamma, A, d) + J(u, \gamma), \tag{1.5}$$

where L is an arbitrary loss function, J is a penalty on the model complexity, and ν is a positive parameter that controls the weight in the optimization problem of data fitting versus complexity reduction. In approximation problems, the loss function L will measure the difference between the actual values of the given data d and the predicted values given by f. For classification problems, the loss function L will be related to misclassification rate.

We will choose J to be either the 1-norm of u to promote solution sparsity [8, 129], or the 2-norm of $\binom{u}{\gamma}$ to ensure that the objective is strongly convex. Below, we explicitly state three formulations obtained from (1.5) that we will encounter frequently.

1.2.1 1-norm support vector machine approximation

In the approximation setting, we wish to approximate a function, f, given exact or approximate function values of a dataset of points represented by the rows of the matrix $A \in R^{m\times n}$. Thus, for each point A_i we are given an exact or inexact value of f, denoted by a real number d_i, $i = 1, \ldots, m$. We therefore desire the parameters (u, γ) of (1.4) to be determined such that:

$$K(A, B')u - e\gamma - d \approx 0. \tag{1.6}$$

An appropriate choice of L to enforce the above condition is:

$$L(u, \gamma, A, d) = \|K(A, B')u - e\gamma - d\|_1. \tag{1.7}$$

This loss function is the sum of the absolute values of the differences between the predicted values $f(A_i)$ and the given values d_i, $i = 1, \ldots, m$. In order to incorporate this loss function into the optimization problem (1.5) we introduce a vector $s \in R^m$ defined by:

$$-s \le K(A, B')u - e\gamma - d \le s. \tag{1.8}$$

We can then incorporate the loss defined by (1.7) into (1.5), along with setting $J(u, \gamma) = \|u\|_1$ as follows:

$$\begin{array}{ll} \min\limits_{u,\gamma,s} & \nu\|s\|_1 + \|u\|_1 \\ \text{s.t.} & -s \le K(A, B')u - e\gamma - d \le s. \end{array} \tag{1.9}$$

Note that at the solution of (1.9), $\|s\|_1 = \|K(A, B')u - e\gamma - d\|_1$. Note further that (1.9) is a linear program.

1.2.2 1-norm support vector machine classification

The classification problem consists of classifying the points represented by the rows of the matrix $A \in R^{m\times n}$ into the two classes $+1$ and -1 according to their labels given as

$d \in \{-1, +1\}^m$. Thus, for each point A_i we are given its label $d_i \in \{-1, +1\}$, $i = 1, \ldots, m$, and we seek to find a function f of the form in (1.4) that satisfies, to the extent possible, the separation condition:

$$D(K(A, B')u - e\gamma) \geq 0, \tag{1.10}$$

where $D \in R^{m \times m}$ is the diagonal matrix with diagonal d, that is, $D_{ii} = d_i$, $i = 1, \ldots, m$. Note that this condition is satisfied if and only if d_i and $f(A_i)$ both have the same sign. The *separating surface* of the classifier is given by:

$$f(x) = K(x', B')u - \gamma = 0. \tag{1.11}$$

Support vector machines attempt to impose a stronger condition, $D(K(A, B')u - e\gamma) \geq e$, using the hinge loss:

$$L(u, \gamma, A, d) = \|(e - D(K(A, B')u - e\gamma))_+\|_1. \tag{1.12}$$

Note that the hinge loss involves the plus function, $(x)_+ = max\{x, 0\}$, introduced in Section 1.1. This loss function can be added to the optimization problem (1.5) by introducing a nonnegative slack variable $s \in R^m$ as follows:

$$D(K(A, B')u - e\gamma) + s \geq e, s \geq 0. \tag{1.13}$$

The loss (1.12) is incorporated into the optimization problem (1.5), along with $J(u, \gamma) = \|u\|_1$ as follows:

$$\begin{array}{ll} \min\limits_{u,\gamma,s} & \nu\|s\|_1 + \|u\|_1 \\ \text{s.t.} & D(K(A, B')u - e\gamma) + s \geq 0, \\ & s \geq 0. \end{array} \tag{1.14}$$

We note that at the solution, $\|s\|_1 = \|(e - D(K(A, B')u - e\gamma))_+\|_1$. Note further that (1.14) is a linear programming problem.

1.2.3 Proximal support vector machine classification

We now formulate the classification problem (1.10) using a proximal support vector machine approach [27, 114, 20]. The error in a proximal formulation is measured by closeness

to the two following proximal surfaces that are parallel to the classifier (1.11) in the u-space:

$$\begin{aligned} K(x', B')u - \gamma &= +1 \\ K(x', B')u - \gamma &= -1 \end{aligned} \tag{1.15}$$

In contrast to an ordinary support vector machine, the two parallel surfaces in (1.15) are chosen so that the first surface is close to the training points in class $+1$ and the second surface is close to points in class -1, and the two surfaces are as far apart as possible. Thus, we measure the error in satisfying (1.10) by how close the points in class $+1$ and class -1 are to the first and second surface of (1.15), respectively. This error can be succinctly written as:

$$L(u, \gamma, A, d) = \|D(K(A, B')u - e\gamma) - e\|^2 \tag{1.16}$$

Note that this loss function measures the sum of the squared differences of each point from the appropriate proximal surface. Adding this loss function to (1.5), along with setting $J(u, \gamma) = \binom{u}{\gamma}$ gives:

$$\min_{(u,\gamma)} \quad \frac{\nu}{2}\|D(K(A, B')u - e\gamma) - e\|^2 + \frac{1}{2}\|u\|^2 + \frac{1}{2}\gamma^2. \tag{1.17}$$

This strongly convex unconstrained quadratic optimization problem can be uniquely solved by setting its gradient, a system of linear equations in (u, γ), equal to zero.

Before proceeding, we note that in some cases the approaches we discuss in the following chapters can be reasonably applied to formulations of (1.5) with other L and J than those we specify, such as the standard 2-norm support vector machine [109, 12] and logistic regression [120]. However, this would have the effect of replacing a linear program or system of linear equations with a more complicated quadratic program or convex program.

Chapter 2

Knowledge-Based Kernel Approximation

Support vector machines (SVMs) have been used extensively for function approximation [17, 111, 21, 78]. However, in some cases the given data alone may not be sufficient to generate a good approximation. Prior knowledge has been incorporated into SVM classifiers in [108, 34, 32]. In this chapter, prior knowledge is added to the SVM function approximation in the form of linear inequalities to be satisfied by the function on polyhedral regions of the input space as was implemented by Mangasarian et al. in [79].

2.1 Prior knowledge for a linear kernel approximation

We begin with a linear kernel model and show how to introduce prior knowledge into such an approximation. We consider an unknown function f from R^n to R for which approximate or exact function values are given on a dataset of m points in R^n denoted by the matrix $A \in R^{m \times n}$. Thus, corresponding to each point A_i we are given an exact or inexact value of f, denoted by a real number y_i, $i = 1, \ldots, m$. We wish to approximate f by some linear or nonlinear function of the matrix A with unknown linear parameters. We first consider the simple linear approximation:

$$f(x) \approx w'x + b, \tag{2.1}$$

for some unknown weight vector $w \in R^n$ and constant $b \in R$ which is determined by minimizing some error criterion that leads to:

$$Aw + be - y \approx 0. \tag{2.2}$$

If we consider w to be a linear combination of the rows of A, i.e. $w = A'\alpha$, $\alpha \in R^m$, which is similar to the dual representation in a linear support vector machine for the weight w [72, 109], we then have:

$$AA'\alpha + be - y \approx 0. \tag{2.3}$$

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 is nonlinear in A but linear in α:

$$K(A, A')\alpha + be - y \approx 0. \tag{2.4}$$

We will measure the error in (2.4) componentwise by a vector $s \in R^m$ defined by:

$$-s \leq K(A, A')\alpha + be - y \leq s. \tag{2.5}$$

We now drive this error down 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 that determines the relative weight of exact data fitting to complexity reduction:

$$\begin{array}{rccc} & \min\limits_{(\alpha,b,s)} & \|\alpha\|_1 + C\|s\|_1 & \\ \text{s.t.} \quad -s & \leq & K(A, A')\alpha + be - y & \leq \quad s, \end{array} \tag{2.6}$$

which can be represented as the following linear program:

$$\begin{array}{rccc} & \min\limits_{(\alpha,b,s,a)} & e'a + Ce's & \\ \text{s.t.} \quad -s & \leq & K(A, A')\alpha + be - y & \leq \quad s, \\ -a & \leq & \alpha & \leq \quad a. \end{array} \tag{2.7}$$

We note that the 1-norm formulation employed here leads to a linear programming formulation without regard to whether the kernel $K(A, A')$ is positive semidefinite or not. This would not be the case if we used a kernel-induced norm on α that would lead to a quadratic program. This quadratic program would be more difficult to solve than our linear

program especially when it is nonconvex, which would be an NP-hard problem [97], as is the case when the kernel employed is not positive semidefinite.

We now introduce prior knowledge for a linear kernel as follows. Suppose that it is known that the function f represented by (2.1) satisfies the following condition. For all points $x \in R^n$, not necessarily in the training set but lying in the nonempty polyhedral set determined by the linear inequalities:

$$Bx \leq d, \tag{2.8}$$

for some $B \in R^{k \times n}$, the function f, and hence its linear approximation $w'x + b$, must dominate a given linear function $h'x + \beta$, for some user-provided $(h, \beta) \in R^{n+1}$. That is, for a *fixed* (w, b) we have the implication:

$$Bx \leq d \implies w'x + b \geq h'x + \beta, \tag{2.9}$$

or equivalently in terms of α, where $w = A'\alpha$:

$$Bx \leq d \implies \alpha' Ax + b \geq h'x + \beta. \tag{2.10}$$

Thus, the implication (2.10) needs to be added to the constraints of the linear program (2.7). To do that we make use of the following equivalence relationship that converts the implication (2.10) to a set of linear constraints that can be appended to the linear program (2.7). A similar technique was used in [34, Proposition 2.1] to incorporate prior knowledge into linear classifiers.

Proposition 2.1.1 Prior Knowledge Equivalence. Let the set $\{x \mid Bx \leq d\}$ be nonempty. Then for a fixed (α, b, h, β), the implication (2.10) is equivalent to the following system of linear inequalities having a solution $u \in R^k$:

$$B'u + A'\alpha - h = 0,\ -d'u + b - \beta \geq 0, u \geq 0. \tag{2.11}$$

Proof The implication (2.10) is equivalent to the following system having no solution $(x, \zeta) \in R^{n+1}$:

$$Bx - d\zeta \leq 0,\ (\alpha' A - h')x + (b - \beta)\zeta < 0,\ -\zeta < 0. \tag{2.12}$$

By the Motzkin theorem of the alternative [69, Theorem 2.4.2] we have that (2.12) is equivalent to the following system of inequalities having a solution (u, η, τ):

$$B'u + (A'\alpha - h)\eta = 0,\ -d'u + (b - \beta)\eta - \tau = 0,\ u \geq 0,\ 0 \neq (\eta, \tau) \geq 0. \tag{2.13}$$

If $\eta = 0$ in (2.13), then we contradict the nonemptiness of the knowledge set $\{x \mid Bx \leq d\}$. Because, for $x \in \{x \mid Bx \leq d\}$ and (u, τ) that solve (2.13) with $\eta = 0$, we obtain the contradiction:

$$0 \geq u'(Bx - d) = x'B'u - d'u = -d'u = \tau > 0. \tag{2.14}$$

Hence $\eta > 0$ in (2.13). Dividing (2.13) by η and redefining (u, α, τ) as $(\frac{u}{\eta}, \frac{\alpha}{\eta}, \frac{\tau}{\eta})$ we obtain (2.11). □

Adding the constraints (2.11) to the linear programming formulation (2.7) with a linear kernel $K(A, A') = AA'$, we obtain our desired linear program that incorporates the prior knowledge implication (2.10) into our approximation problem:

$$\begin{array}{rccccl} & & & \min\limits_{(\alpha,b,s,a,u\geq 0)} & e'a + Ce's & \\ \text{s.t.} & -s & \leq & AA'\alpha + be - y & \leq & s, \\ & -a & \leq & \alpha & \leq & a, \\ & & & A'\alpha + B'u & = & h, \\ & & & -d'u & \geq & \beta - b. \end{array} \tag{2.15}$$

Note that in this linear programming formulation with a linear kernel approximation, both the approximation $w'x + b = \alpha'Ax + b$ to the unknown function f as well as the prior knowledge are linear in the input data A of the problem. This is somewhat restrictive, and therefore we turn now to the incorporation of prior knowledge into a *nonlinear* kernel approximation.

2.2 Knowledge-based nonlinear kernel approximation

In this chapter we will incorporate prior knowledge by using a nonlinear kernel in *both* the linear programming formulation (2.7) as well as in the prior knowledge implication (2.10).

We begin with the latter, the linear prior knowledge implication (2.10). If we again consider x as a linear combination of the rows of A, that is:

$$x = A't, \tag{2.16}$$

then the implication (2.10) becomes:

$$BA't \leq d \implies \alpha' AA't + b \geq h'A't + \beta, \tag{2.17}$$

for a given fixed (α, b). The assumption (2.16) is not restrictive for the many problems where a sufficiently large number of training data points are available so that any vector in input space can be represented as a linear combination of the training data points.

If we now "kernelize" the various matrix products in the above implication, we have the implication:

$$K(B, A')t \leq d \implies \alpha' K(A, A')t + b \geq h'A't + \beta. \tag{2.18}$$

We note that the two kernels appearing in (2.18) need not be the same and neither needs to satisfy Mercer's positive semidefiniteness condition. In particular, the first kernel of (2.18) could be a linear kernel which renders the left side of the implication of (2.18) the same as that of (2.17). We note that for a nonlinear kernel, implication (2.18) is nonlinear in the input space data, but is linear in the implication variable t. We have thus mapped the polyhedral implication (2.9) into a nonlinear one (2.18) in the input space data. Assuming for simplicity that the kernel K is symmetric, that is $K(B, A')' = K(A, B')$, it follows directly by Proposition 2.1.1 that the following equivalence relation holds for implication (2.18).

Proposition 2.2.1 Nonlinear Kernel Prior Knowledge Equivalence. Let the set $\{t \mid K(B, A')t \leq d\}$ be nonempty. Then for a given (α, b, h, β), the implication (2.18) is equivalent to the following system of linear inequalities having a solution $u \in R^k$:

$$K(A, B')u + K(A, A')\alpha - Ah = 0,\ -d'u + b - \beta \geq 0, u \geq 0. \tag{2.19}$$

We now append the constraints (2.19), which are equivalent to the nonlinear kernel implication (2.18), to the linear programming formulation (2.7). This gives the following

linear program for approximating a given function with prior knowledge using a nonlinear kernel:

$$\begin{array}{rcccl} & & \min\limits_{(\alpha,b,s,a,u\geq 0)} & e'a + Ce's & \\ \text{s.t.} \quad -s & \leq & K(A,A')\alpha + be - y & \leq & s, \\ -a & \leq & \alpha & \leq & a, \\ & & K(A,B')u + K(A,A')\alpha & = & Ah, \\ & & -d'u & \geq & \beta - b. \end{array} \tag{2.20}$$

Since we are not certain that the prior knowledge implication (2.18) is satisfiable, and since we wish to balance the influence of prior knowledge with that of fitting conventional data points, we need to introduce error variables z and ζ associated with the last two constraints of the linear program (2.20). These error variables are then driven down by a modified objective function as follows:

$$\begin{array}{rcccl} & & \min\limits_{(\alpha,b,s,a,z,(u,\zeta)\geq 0)} & e'a + Ce's + \mu_1 e'z + \mu_2\zeta & \\ \text{s.t.} \quad -s & \leq & K(A,A')\alpha + be - y & \leq & s, \\ -a & \leq & \alpha & \leq & a, \\ -z & \leq & K(A,B')u + K(A,A')\alpha - Ah & \leq & z, \\ & & -d'u + \zeta & \geq & \beta - b, \end{array} \tag{2.21}$$

where (μ_1, μ_2) are some positive parameters. This is our final linear program for a single prior knowledge implication. If we have more than one such implication, then the last two sets of constraints are repeated for each implication. For the sake of simplicity we omit these details. The values of the parameters C, μ_1, and μ_2 are chosen so as to balance fitting conventional numerical data versus the given prior knowledge. One way to choose these parameters is to set aside a "tuning set" of data points and then choose the parameters so as to give a best fit of the tuning set. We also note that all three kernels appearing in (2.21) could possibly be distinct kernels from each other and none needs to be positive semidefinite. In fact, the kernel $K(A, B')$ could be the linear kernel AB' which was actually tried in some of our numerical experiments without a noticeable change from using a Gaussian kernel.

We now turn to our numerical experiments.

2.3 Numerical experiments

In order to illustrate the power of the formulation proposed in this chapter, we tested our algorithm on three synthetic examples and two real-world examples with and without prior knowledge. Two of the synthetic examples are based on the "sinc" function which has been extensively used for kernel approximation testing [119, 3], while the third synthetic example is a two-dimensional hyperboloid. All our results indicate significant improvement due to prior knowledge. The parameters for the synthetic examples were selected using a combination of exhaustive search and a simple variation on the Nelder-Mead simplex algorithm [99] that uses only reflection, with average error as the criterion. The chosen parameter values are given in the captions of relevant figures.

2.3.1 One-dimensional sinc function

We consider the one-dimensional sinc function:

$$f(x) = sinc(x) = \frac{sin\pi x}{\pi x} \tag{2.22}$$

Given data for the sinc function includes approximate function values for 52 points on the intervals $-3 \leq x \leq -1.4303$ and $1.4303 \leq x \leq 3$. The endpoints ± 1.4303 are approximate local minima of the sinc function. The given approximate function values for $sinc(x)$ are normally perturbed around the true values, with mean 0 and standard deviation 0.5. In addition, there are also three given values at $x = 0$. One of these values is 1, which is the actual limit of the sinc function at 0. The other values at $x = 0$ are 0 and -1 which are intended to be misleading to the approximation.

Figure 2.1(a) depicts $sinc(x)$ by a dashed curve and its approximation *without* prior knowledge by a solid curve based on the 55 points shown by diamonds. The nine solid diamonds depict "support" points, that is rows A_i of A for which $\alpha_i \neq 0$ in the solution of the nonlinear Gaussian kernel approximation of (2.7) for $f(x)$:

$$f(x) \approx K(x', A')\alpha + b. \tag{2.23}$$

The approximation in Figure 2.1(a) has an average error of 0.3113. This error is computed by averaging over a grid of 100 equally spaced points in the interval $[-3, 3]$.

Figure 2.1(b) depicts $sinc(x)$ by a dashed curve and its much better approximation *with* prior knowledge by a solid curve based on the 55 points shown, which are the same as those of Figure 2.1(a). The seven solid diamond points are "support" points, that is rows A_i of A for which $\alpha_i \neq 0$ in the solution of the nonlinear Gaussian kernel approximation (2.23) of (2.21) for $f(x)$. The approximation in Figure 2.1(b) has an average error of 0.0901 computed over a grid of 100 equally spaced points on $[-3, 3]$. The prior knowledge used to approximate the one-dimensional sinc function is $-\frac{1}{4} \leq x \leq \frac{1}{4} \Rightarrow f(x) \geq \frac{sin(\pi/4)}{\pi/4}$. The value $\frac{sin(\pi/4)}{\pi/4}$ is the minimum of $sinc(x)$ on the knowledge interval $[-\frac{1}{4}, \frac{1}{4}]$. This prior knowledge is implemented by replacing $f(x)$ by its nonlinear kernel approximation (2.23) and then using the implication (2.18) as follows:

$$K(I, A')t \leq \frac{1}{4} \ \wedge \ K(-I, A')t \leq \frac{1}{4} \Longrightarrow \alpha' K(A, A')t + b \geq \frac{sin(\pi/4)}{\pi/4}. \tag{2.24}$$

2.3.2 Two-dimensional sinc function

Our second example is the two-dimensional $sinc(x)$ function for $x \in R^2$:

$$f(x_1, x_2) = sinc(x_1)sinc(x_2) = \frac{sin\pi x_1}{\pi x_1}\frac{sin\pi x_2}{\pi x_2}. \tag{2.25}$$

The given data for the two-dimensional sinc function includes 210 points in the region $\{(x_1, x_2) | (-3 \leq x_1 \leq -1.4303 \vee 1.4303 \leq x_1 \leq 3) \wedge (-3 \leq x_2 \leq -1 \vee 1 \leq x_2 \leq 3)\}$. This region excludes the largest bump in the function centered at $(x_1, x_2) = (0, 0)$. The given values are exact function values. There are also three values given at $(x_1, x_2) = (0, 0)$, similar to the previous example with the one dimensional sinc. The first value is the actual limit of the function at $(0, 0)$, which is 1. The other two values are 0 and -1. These last two values are intended to mislead the approximation.

Figure 2.2(a) depicts the two-dimensional sinc function of (2.25). Figure 2.2(b) depicts an approximation of $sinc(x_1)sinc(x_2)$ *without* prior knowledge by a surface based on the

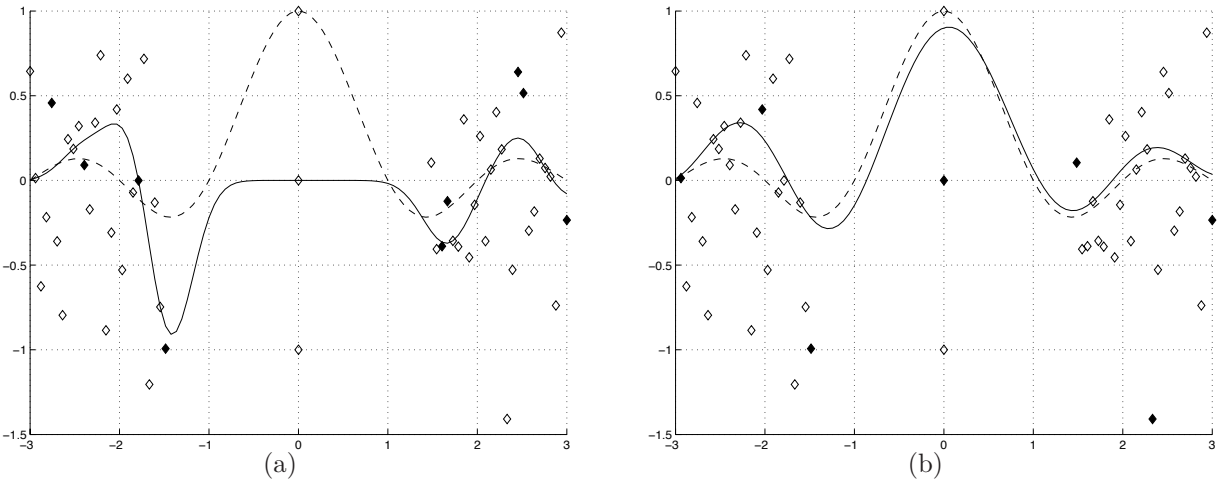
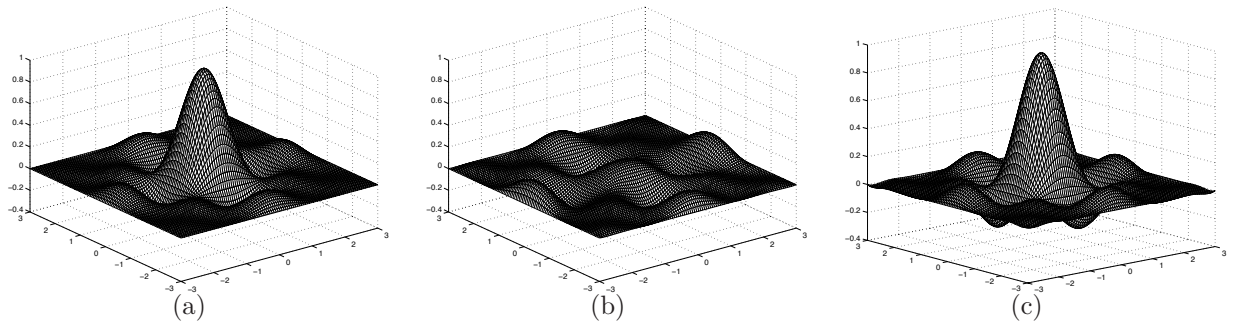


Figure 2.1 The one-dimensional sinc function $sinc(x) = \frac{sin\pi x}{\pi x}$ depicted by a dashed curve, and its Gaussian kernel approximation (a) *without* prior knowledge and (b) *with* prior knowledge depicted by a solid curve based on the 55 points shown by diamonds. The solid diamonds depict the "support" points used by the nonlinear Gaussian kernel in generating the approximation of $sinc(x)$. That is, they are the rows A_i of A for which $\alpha_i \neq 0$ in the solution of the nonlinear Gaussian kernel approximation of (2.7) for $f(x)$: $f(x) \approx K(x', A')\alpha + b$. The prior knowledge used in (b) consists of the implication $-\frac{1}{4} \leq x \leq \frac{1}{4} \Rightarrow f(x) \geq \frac{\sin(\pi/4)}{\pi/4}$, which is implemented by replacing $f(x)$ by its nonlinear kernel approximation (2.23). The approximation in (a) *without knowledge* has an average error of 0.3113 over a grid of 100 points in the interval $[-3, 3]$, while the approximation in (b) *with knowledge* has an average error of 0.0901, which is less than $\frac{1}{3.4}$ times the error in (a). Parameter values used: (a) $\mu = 7, C = 5$; (b) $\mu = 1, C = 13, \mu_1 = 5, \mu_2 = 450$.



(a) (b) (c)

Figure 2.2 The (a) exact product sinc function $f(x_1, x_2) = \frac{sin\pi x_1}{\pi x_1}\frac{sin\pi x_2}{\pi x_2}$, and its Gaussian kernel approximation based on 211 exact function values plus two incorrect function values, (b) *without* prior knowledge, and (c) *with* prior knowledge consisting of $(x_1, x_2) \in \{[-0.1, 0.1] \times [-0.1, 0.1]\}\} \Rightarrow f(x_1, x_2) \geq (\frac{sin(\pi/10)}{\pi/10})^2$. Over a grid of 2500 points in the set $\{[-3, 3] \times [-3, 3]\}$, the approximation without prior knowledge in (b) has an average error of 0.0501, while the approximation in (c) with prior knowledge has an average error of 0.0045, which is less than $\frac{1}{11.1}$ times the error in (b). Parameter values used: (b) $\mu = 0.2, C = 10^6$; (c) $\mu = 1, C = 16000, \mu_1 = 15000, \mu_2 = 5 \cdot 10^6$.

213 points described above. The approximation in Figure 2.2(b) has an average error of 0.0501. This value is computed by averaging over a grid of 2500 equally spaced points in $\{[-3, 3] \times [-3, 3]\}$.

Figure 2.2(c) depicts a much better approximation of $sinc(x_1)sinc(x_2)$ *with* prior knowledge by a surface based on the same 213 points. The approximation in Figure 2.2(c) has an average error of 0.0045. This value is computed by averaging over 2500 equally spaced points in $\{[-3, 3] \times [-3, 3]\}$. The prior knowledge consists of the implication $(x_1, x_2) \in \{[-0.1, 0.1] \times [-0.1, 0.1]\} \Rightarrow f(x_1, x_2) \geq (\frac{sin(\pi/10)}{\pi/10})^2$. The value $(\frac{sin(\pi/10)}{\pi/10})^2$ is equal to the minimum value of $sinc(x_1)sinc(x_2)$ on the knowledge set $\{[-0.1, 0.1] \times [-0.1, 0.1]\}$. This prior knowledge is implemented by replacing $f(x_1, x_2)$ by its nonlinear kernel approximation (2.23) and then using the implication (2.18).

2.3.3 Two-dimensional hyperboloid function

Our third example is the two-dimensional hyperboloid function:

$$f(x_1, x_2) = x_1 x_2. \tag{2.26}$$

For the two-dimensional hyperboloid function, the given data consists of 11 points along the line $x_2 = x_1, x_1 \in \{-5, -4, \ldots, 4, 5\}$. The given values at these points are the actual function values.

Figure 2.3(a) depicts the two-dimensional hyperboloid function of (2.26). Figure 2.3(b) depicts an approximation of the hyperboloid function, *without* prior knowledge, by a surface based on the 11 points described above. The approximation in Figure 2.3(b) has an average error of 4.8351 computed over a grid of 2500 equally spaced points in $\{[-5, 5] \times [-5, 5]\}$.

Figure 2.3(c) depicts a much better approximation of the hyperboloid function by a nonlinear surface based on the same 11 points above *plus* prior knowledge. The approximation in Figure 2.3(c) has an average error of 0.2023 computed over a grid of 2500 equally spaced points in $\{[-5, 5] \times [-5, 5]\}$. The prior knowledge consists of the following two implications:

$$(x_1, x_2) \in \{(x_1, x_2) | -\frac{1}{3}x_1 \leq x_2 \leq -\frac{2}{3}x_1\} \Rightarrow f(x_1, x_2) \leq 10x_1 \tag{2.27}$$

and

$$(x_1, x_2) \in \{(x_1, x_2) | -\frac{2}{3}x_1 \leq x_2 \leq -\frac{1}{3}x_1\} \Rightarrow f(x_1, x_2) \leq 10x_2. \tag{2.28}$$

These implications are implemented by replacing $f(x_1, x_2)$ by its nonlinear kernel approximation (2.23) and then using the implication (2.18). The regions on which the knowledge is given are cones on which $x_1 x_2$ is negative. Since the two implications are analogous, we explain (2.27) only. This implication is justified on the basis that $x_1 x_2 \leq 10x_1$ over the knowledge cone $\{(x_1, x_2) | -\frac{1}{3}x_1 \leq x_2 \leq -\frac{2}{3}x_1\}$ for sufficiently large x_2, that is $x_2 \geq 10$. This is intended to capture coarsely the global shape of $f(x_1, x_2)$ and succeeds in generating a more accurate overall approximation of the function.

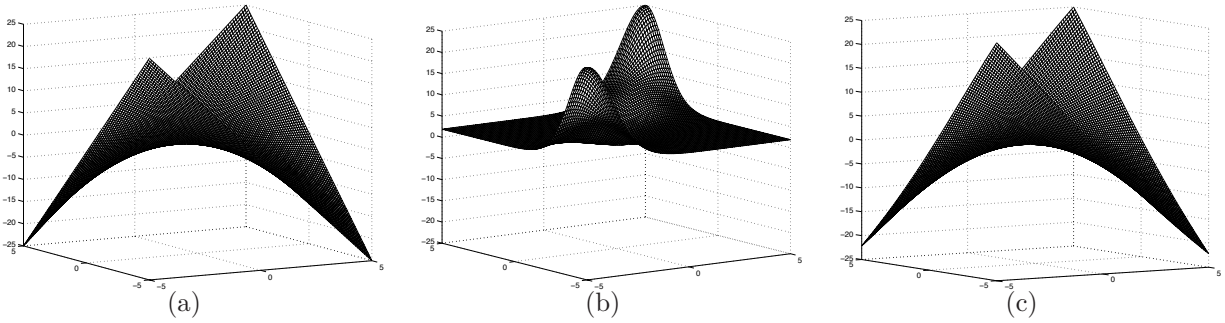


Figure 2.3 The (a) exact hyperboloid function $f(x_1, x_2) = x_1 x_2$, and its Gaussian kernel approximation based on 11 exact function values along the line $x_2 = x_1, x_1 \in \{-5, -4, \ldots, 4, 5\}$, (b) *without* prior knowledge and (c) *with* prior knowledge consisting of the implications (2.27) and (2.28). Over a grid of 2500 points in the set $\{[-5, 5] \times [-5, 5]\}$, the approximation without prior knowledge in (b) has average error 4.8351, while the approximation with prior knowledge in (c) has average error 0.2023, which is less than $\frac{1}{23.9}$ times the error of (b). Parameter values used: (b) $\mu = 0.361, C = 145110$; (c) $\mu = 0.0052, C = 5356, \mu_1 = 685, \mu_2 = 670613$.

2.3.4 Predicting lymph node metastasis

For our first example involving a real-world dataset we describe a potentially useful application of knowledge-based approximation to breast cancer prognosis [80, 123, 57]. An important prognostic indicator for breast cancer recurrence is the number of metastasized lymph nodes under a patient's armpit, which could be as many as 30. To determine this number, a patient must undergo optional surgery in addition to the removal of the breast tumor. If the predicted number of metastasized lymph nodes is sufficiently small, then the oncological surgeon may decide not to perform the additional surgery. Thus, it is useful to approximate the number of metastasized lymph nodes as a function of thirty available cytological features and one histological feature. The cytological features are obtained from a fine needle aspirate during the diagnostic procedure while the histological feature is obtained during surgery. Our proposed knowledge-based approximation can be used to improve the determination of such a function, $f : R^{31} \longrightarrow R$, that predicts the number of metastasized lymph nodes. For example, in certain polyhedral regions of R^{31}, past training data indicate the existence of a substantial number of metastasized lymph nodes, whereas certain other regions indicate the unlikely presence of any metastasis. This knowledge can be applied to obtain a hopefully more accurate lymph node function f than that based on numerical function approximation alone.

We have performed preliminary experiments with the Wisconsin Prognostic Breast Cancer (WPBC) data available from [96]. In our experiments we reduced R^{31} to R^4 and predicted the number of metastasized lymph nodes based on three cytological features: mean cell texture, worst cell smoothness, and worst cell area, as well as the histological feature tumor size. The tumor size is an obvious histological feature to include, while the three other cytological features were the same as those selected for breast cancer diagnosis in [73]. Thus, we are approximating a function $f : R^4 \longrightarrow R$. Note that the online version of the WPBC data contains four entries with no lymph node information which were removed for our experiments. After removing these entries, we were left with 194 examples in our dataset.

To simulate the procedure of an expert obtaining prior knowledge from past data we used the following procedure. First we took a random 20% of the dataset to analyze as "past data." Inspecting this past data, we choose the following background knowledge:

$$x_1 \geq 22.4 \ \wedge \ x_2 \geq 0.1 \ \wedge \ x_3 \geq 1458.9 \ \wedge \ x_4 \geq 3.1 \implies f(x_1, x_2, x_3, x_4) \geq 1, \qquad (2.29)$$

where x_1, x_2, x_3, and x_4 denote mean texture, worst smoothness, worst area, and tumor size respectively. This prior knowledge is based on a typical oncological surgeon's advice that larger values of the variables are likely to result in more metastasized lymph nodes. The constants in (2.29) were chosen by taking the average values of $x_1, \ldots, x_4$ for the entries in the past data with at least one metastasized lymph node.

We used ten-fold cross validation to compare the average absolute error between an approximation without prior knowledge and an approximation with the prior knowledge of Equation (2.29) on the 80% of the data that was not used as "past data" to generate the constants in (2.29). Parameters in (2.21) using a Gaussian kernel were chosen using the Nelder-Mead algorithm on a tuning set taken from the training data for each fold. The average absolute error of the function approximation with no prior knowledge was 3.75 while the average absolute error with prior knowledge was 3.35, a 10.5% reduction. The mean function value of the data used in the ten-fold cross validation experiments is 3.30, so neither approximation is accurate. However, these results indicate that adding prior knowledge does indeed improve the function approximation substantially. Hopefully more sophisticated prior knowledge, based on a more detailed analysis of the data and consultation with domain experts, will further reduce the error.

2.3.5 Reinforcement learning

Our second example using a real-world dataset is an application to a reinforcement learning task [113], where the goal is to predict the value of taking an action at a given state. We model this task by learning a function for each possible action. The domain of these functions is the set of states. In particular, we use the *BreakAway* subtask of the soccer game developed in [64]. In *BreakAway* the objective is to cooperate with teammates to score

a goal within ten seconds, without allowing the opposing team to take control of the ball or allowing the ball to go out of bounds. The state description includes measurements such as the distance to each of the opposing players, the distance to the soccer ball, the distances to the edges of the field, etc. Actions include shooting the ball at the goal, holding the ball, and attempting a pass to a teammate. It has been demonstrated that providing prior knowledge can improve the choice of actions significantly [52, 63]. One example of advice (that is, prior knowledge) that has been successfully used in this domain is the simple advice that "if at least fifteen meters from the goal and a teammate is closer to the goal and at least three meters from the goalie, then it is a good idea to pass to that teammate." In our approach we approximate the value function of passing, v_p, as a function of states. Advice can be stated as the following implication, assuming one teammate:

$$d_1 \geq 15 \ \wedge \ d_2 \leq d_1 \ \wedge \ d_3 \geq 3 \implies v_p \geq c, \tag{2.30}$$

where d_1 denotes the distance to the goal, d_2 the distance of the teammate to the goal, d_3 the distance of the teammate to the goalie, v_p the predicted value of passing, and c is some constant.

Figure 2.4 shows a comparison of three function approximations to the *BreakAway* task. One approximation, labeled "No Advice," used no prior knowledge while KBKR and Pref-KBKR both incorporated prior knowledge using the approach described in this chapter. In KBKR only one value function may appear on the right-hand side of the implication (e. g., $v_p \geq c$), while in Pref-KBKR two value functions may appear on the right-hand side (e. g., $v_p \geq v_s$, where v_s denotes the predicted value of shooting). Thus, in Pref-KBKR preference for one action over another can be expressed as prior knowledge. For example, a preference for passing instead of shooting in some states can be expressed in Pref-KBKR, but not KBKR. The approximations that incorporated prior knowledge performed better than the formulation with no prior knowledge. Details of the data and prior knowledge used for the function approximations can be found in [64].

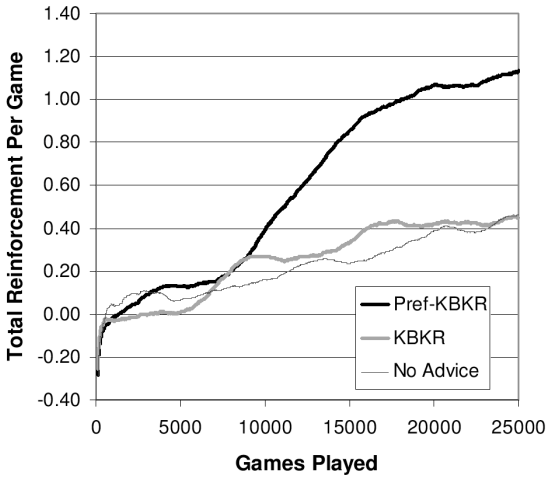


Figure 2.4 Reinforcement (reflecting successful action) as a function of games played for three function approximations. Higher reinforcement is better. "No Advice" denotes an approximation with no prior knowledge, while KBKR denotes knowledge-based kernel regression and Pref-KBKR denotes preference KBKR. Both KBKR and Pref-KBKR incorporate prior knowledge.

Chapter 3

Nonlinear Knowledge in Kernel Machines

Prior knowledge has been used effectively in improving classification both for linear [34] and nonlinear [33] kernel classifiers as well as for nonlinear kernel approximation as described in Chapter 2. In all these applications prior knowledge was converted to linear inequalities that were imposed on a linear program. The linear program generated a linear or nonlinear classifier, or a linear or nonlinear function approximation, all of which were more accurate than the corresponding results that did not utilize prior knowledge. However, whenever a nonlinear kernel was utilized in these applications, as in Proposition 2.2.1, kernelization of the prior knowledge was not a transparent procedure that could be easily related to the original sets over which prior knowledge was given. In contrast, prior knowledge over arbitrary general sets has been recently incorporated *without* kernelization of the prior knowledge sets into nonlinear kernel approximation [89] and nonlinear kernel classification [86]. Here, we present a unified formulation of both approaches introduced in [87], which are made possible through the use of a fundamental theorem of the alternative for convex functions that we describe in Section 3.1. An interesting, novel approach to knowledge-based support vector machines that modifies the hypothesis space rather than the optimization problem is given in [53]. In another recent approach, prior knowledge is incorporated by adding additional points labeled based on the prior knowledge to the dataset [65].

We also consider adding nonlinear prior knowledge as linear *equalities* to proximal nonlinear classification, as implemented in [91]. Proximal nonlinear classifiers [27, 31, 29]

require only the solution of a system of linear equations. Solvers for such systems are both computationally cheaper and more available than linear programming solvers.

3.1 General linear programming formulation

We wish to impart knowledge to a function learned on a dataset in R^n represented by the m rows of the matrix $A \in R^{m\times n}$ with associated labels represented by the m-dimensional vector d. For approximation problems, d will be real valued, that is $d \in R^m$. For classification problems, d will have values $+1$ or -1 according to the class of each example, that is $d \in \{-1, +1\}^m$. The learned function f from R^n to R is defined in (1.4) as $f(x) = K(x', B')u - \gamma$, where $B \in R^{k\times n}$ is an arbitrary basis matrix and $K(x', B') : R^{1\times n} \times R^{n\times k} \longrightarrow R^{1\times k}$ is an arbitrary kernel function. The variables $u \in R^k$ and $\gamma \in R$ are parameters determined by the following optimization problem, which is the same as (1.5) with $J(u, \gamma) = \|u\|_1$:

$$\min_{u,\gamma} \ \nu L(u, \gamma, A, d) + \|u\|_1, \tag{3.1}$$

where L is an arbitrary loss function, and ν is a positive parameter that controls the weight in the optimization problem of data fitting versus complexity reduction.

We now impose prior knowledge on the construction of our learned function $f(x) = K(x', B')u - \gamma$ through the following implication:

$$g(x) \leq 0 \Longrightarrow K(x', B')u - \gamma \geq \phi(x), \ \forall x \in \Gamma. \tag{3.2}$$

Here, $g(x) : \Gamma \subset R^n \longrightarrow R^p$ is a p-dimensional function defined on a subset Γ of R^n that determines the region in the input space where prior knowledge requires that $K(x', B')u - \gamma$ be larger than some known function $\phi(x) : \Gamma \subset R^n \longrightarrow R$. In Chapter 2, and in previous work [79, 35, 33], prior knowledge implications such as (3.2) could not be handled directly as we shall do here by using Proposition 3.1.1 below. Instead, the inequality $g(x) \leq 0$ was kernelized. This led to an inequality, such as (2.18), that could not be easily related to the original constraint $g(x) \leq 0$. In addition, in [79, 35, 33] could only handle linear $g(x)$ and

$\phi(x)$. The implication (3.2) can be written in the following equivalent logical form:

$$\begin{gathered} g(x) \leq 0,\ K(x', B')u - \gamma - \phi(x) < 0, \\ \text{has no solution } x \in \Gamma. \end{gathered} \tag{3.3}$$

It is precisely implication (3.2) that we shall convert to a system which is linear in the parameters of f, (u, γ), by means of the following theorem of the alternative for convex functions. The alternatives here are that either the negation of (3.3) holds, or (3.4) below holds, but *never both*.

Proposition 3.1.1 Prior Knowledge as System of Linear Inequalities For a fixed $u \in R^k, \gamma \in R$, consider the following statements:

(i) The implication (3.2) or equivalently (3.3) holds.

(ii) There exists $v \in R^p$, $v \geq 0$ such that:

$$K(x', B')u - \gamma - \phi(x) + v'g(x) \geq 0,\ \forall x \in \Gamma. \tag{3.4}$$

Then (ii)$\Longrightarrow$(i), with **no additional assumptions**. If, in addition, $g(x)$ and $K(x', B')$ are convex on Γ, $\phi(x)$ is concave on Γ, Γ is a convex subset of R^n, $u \geq 0$ and that $g(x) < 0$ for some $x \in \Gamma$, then (i)$\Longrightarrow$(ii).

Proof (i)$\Longrightarrow$(ii): This follows from [67, Corollary 4.2.2], the fact that the functions $g(x)$ and $K(x', B')u - \gamma - \phi(x)$ of (3.3) are convex on Γ and that $g(x) < 0$ for some $x \in \Gamma$.

(i)$\Longleftarrow$(ii): If (i) did not hold then there exists an $x \in \Gamma$ such that $g(x) \leq 0,\ K(x', B')u - \gamma - \phi(x) < 0$, which would result in the contradiction:

$$0 > K(x', B')u - \gamma - \phi(x) + v'g(x) \geq 0. \tag{3.5}$$

□

We note immediately that in the proposed application of converting prior knowledge to linear inequalities in the parameters (u, γ) all we need is the implication (i)$\Longleftarrow$(ii), which **requires no assumptions whatsoever** on the functions $g(x)$, $K(x', B')$, $\phi(x)$ or on

the parameter u. We further note that the implication (3.2) can represent fairly complex knowledge such as $K(x', B')u - \gamma$ being equal to any desired function whenever $g(x) \leq 0$.

We note that Proposition 3.1.1 can also be invoked on the following prior knowledge implication, which is similar to (3.2):

$$h(x) \leq 0 \Longrightarrow K(x', B')u - \gamma \leq -\psi(x), \ \forall x \in \Lambda. \tag{3.6}$$

We now incorporate the prior knowledge contained in implications (3.2) and (3.6) into the optimization problem (3.1) as follows:

$$\begin{array}{rl} \min\limits_{u,\gamma,z_1,\ldots,z_\ell,q_1,\ldots,q_t} & \nu L(u,\gamma,A,d) + \|u\|_1 + \sigma(\sum\limits_{i=1}^{\ell} z_i + \sum\limits_{j=1}^{t} q_j) \\ \text{s.t.} & K(x^{i\prime}, B')u - \gamma - \phi(x^i) + v'g(x^i) + z_i \geq 0, \\ & z_i \geq 0, \ i = 1, \ldots, \ell, \\ & v \geq 0, \\ & -K(x^{j\prime}, B')u + \gamma - \psi(x^j) + r'h(x^j) + q_j \geq 0, \\ & q_j \geq 0, \ j = 1, \ldots, t, \\ & r \geq 0. \end{array} \tag{3.7}$$

We note that we have discretized the variable $x \in \Gamma$ and $x \in \Lambda$ in the constraints above to the finite meshes of points $\{x^1, x^2, \ldots, x^\ell\}$ and $\{x^1, x^2, \ldots, x^t\}$ in order to convert a semi-infinite program [37] with an infinite number of constraints into a finite mathematical program. We have also added nonnegative slack variables $z_1, z_2, \ldots, z_\ell$ and $q_1, q_2, \ldots, q_t$ to allow small deviations in the prior knowledge. The sum of these nonnegative slack variables for the prior knowledge inequalities is minimized with weight $\sigma > 0$ in the objective function in order to drive them to zero to the extent possible. Thus, the magnitude of the parameter σ enforces prior knowledge while the magnitude of ν enforces data fitting.

Using the Motzkin theorem of the alternative [69, Theorem 2.4.2], the discretized version of the knowledge constraint can be shown, under mild assumptions, to be equivalent to an average imposition of the knowledge (3.2) at the discretized points $\{x^1, \ldots, x^\ell\}$:

$$\exists (t^1, \ldots, t^\ell) \geq 0 : t^{1\prime} g(x^1) + \ldots + t^{\ell\prime} g(x^\ell) \leq 0 \Longrightarrow \sum_{i=0}^{\ell} \big(K(x^{i\prime}, B')u - \gamma\big)t^i \geq \sum_{i=0}^{\ell} \phi(x^i)t^i. \tag{3.8}$$

We turn now to specific formulations for knowledge-based kernel approximation and knowledge-based kernel classification.

3.2 Knowledge-based kernel approximation

In the approximation setting, we wish to approximate a function, f, given exact or approximate function values of a dataset of points represented by the rows of the matrix $A \in R^{m\times n}$. Thus, for each point A_i we are given an exact or inexact value of f, denoted by a real number d_i, $i = 1, \ldots, m$. We therefore desire the parameters (u, γ) of (1.4), $f(x) = K(x', B')u - \gamma$, to be determined such that (1.6), $K(A, B')u - e\gamma \approx 0$, is satisfied. As discussed in Section 1.2.1, an appropriate loss function to achieve this is (1.7). Incorporating this loss function into the optimization problem (3.7) gives:

$$\begin{aligned} \min_{u,\gamma,z_1,\ldots,z_\ell,q_1,\ldots,q_t,s} \quad & \nu\|s\|_1 + \|u\|_1 + \sigma(\sum_{i=1}^{\ell} z_i + \sum_{j=1}^{t} q_j) \\ \text{s.t.} \quad & K(x^{i\prime}, B')u - \gamma - \phi(x^i) + v'g(x^i) + z_i \geq 0, \\ & z_i \geq 0,\ i = 1, \ldots, \ell, \\ & v \geq 0, \\ & -K(x^{j\prime}, B')u + \gamma - \psi(x^j) + r'h(x^j) + q_j \geq 0, \\ & q_j \geq 0,\ j = 1, \ldots, t, \\ & r \geq 0, \\ & -s \leq K(A, B')u - e\gamma - d \leq s. \end{aligned} \tag{3.9}$$

Note that at the solution of (3.9), $\|s\|_1 = \|K(A, B')u - e\gamma - d\|_1$. Note further that (3.9) is the same as [89, Equation 10], except that here we have explicitly included implication (3.6) in addition to (3.2).

3.3 Knowledge-based kernel classification

The classification problem consists of classifying the points represented by the rows of the matrix $A \in R^{m\times n}$ into the two classes $+1$ and -1 according to their labels given as $d \in \{-1, +1\}^m$. Thus, for each point A_i we are given its label $d_i \in \{-1, +1\}$, $i = 1, \ldots, m$

and we seek to find a function f of the form in (1.4), $f(x) = K(x', B')u - \gamma$, that satisfies, to the extent possible, the separation condition (1.10), $D(K(A, B')u - e\gamma) \geq 0$. As discussed in Section 1.2.2, an appropriate loss function to achieve this condition is the hinge loss given in (1.12). Incorporating this loss function into the optimization problem (3.7) gives:

$$\begin{aligned} \min_{u,\gamma,z_1,\ldots,z_\ell,q_1,\ldots,q_t,s} \quad & \nu\|s\|_1 + \|u\|_1 + \sigma(\sum_{i=1}^{\ell} z_i + \sum_{j=1}^{t} q_j) \\ \text{s.t.} \quad & K(x^{i\prime}, B')u - \gamma - \phi(x^i) + v'g(x^i) + z_i \geq 0, \\ & z_i \geq 0,\ i = 1, \ldots, \ell, \\ & v \geq 0, \\ & -K(x^{j\prime}, B')u + \gamma - \psi(x^j) + r'h(x^j) + q_j \geq 0, \\ & q_j \geq 0,\ j = 1, \ldots, t, \\ & r \geq 0, \\ & D(K(A, B')u - e\gamma) + s \geq 0, \\ & s \geq 0. \end{aligned} \tag{3.10}$$

We note that at the solution, $\|s\|_1 = \|(e - D(K(A, B')u - e\gamma))_+\|_1$. Note further that (3.10) is the same as the optimization problem in [86] with $\phi(x) = \psi(x) = \alpha,\ \forall x$.

3.4 Proximal knowledge-based classification

In contrast to the linear programming SVM formulation used above, proximal SVM classifiers [27, 31, 29] require only the solution of a system of linear equations. Here, we give a formulation for adding prior knowledge in a way that preserves this computational advantage. We note that each prior knowledge implication given in (3.2) leads to a system of linear inequalities when added to an optimization problem through the discretization of the sufficient condition given by Proposition 3.1.1. Therefore, we here give a method for converting prior knowledge into a system of linear *equalities*, which can be added to a nonlinear proximal classifier and still require only the solution of a linear system of equations.

We now impose prior knowledge on the construction of our classifier function $K(x', B')u - \gamma$ to ensure that a certain set of points lies on the $+1$ side of the classifier (1.4). We achieve

this through the following implication:

$$g(x) \leq 0 \implies K(x', B')u - \gamma = 1, \quad \forall x \in \Gamma_1. \tag{3.11}$$

Here, $g(x) : \Gamma_1 \subset R^n \longrightarrow R^r$ is an r-dimensional function defined on a subset Γ_1 of R^n that determines the region $\{x|g(x) \leq 0\}$. The prior knowledge requires that for any x in this region, the classifier function $K(x', B')u - \gamma$ return a value $+1$. Such points would thus be classified in class $+1$. Prior knowledge about the class -1 can be added through a similar implication.

It is interesting to compare the implication (3.11) to the similar implication (3.2) treated in Section 3.1. This implication has the same interpretation as (3.11), but uses an inequality constraint to impose the condition that any x for which $g(x) \leq 0$ is classified in class $+1$. While implication (3.2) can be added as a set of linear inequality constraints to the linear programming formulation of an SVM, here we shall preserve the computational advantages of the proximal SVM by adding implication (3.11) as a set of linear *equality* constraints as follows.

The implication (3.11) can be written in the following equivalent form:

$$g(x)_+ = 0 \implies K(x', B')u - \gamma = 1, \quad \forall x \in \Gamma_1, \tag{3.12}$$

where $g(x)_+ = \max\{g(x), 0\}$. The use of $g(x)_+ = 0$ in (3.12) in place of $g(x) \leq 0$ is a key observation which allows us to convert the implication to a linear equality via Proposition 3.4.1 with a multiplier that is not required to be nonnegative. We can then add this linear equality to a proximal nonlinear classifier *without* adding nonnegativity constraints. Motivated by Proposition 3.1.1, Proposition 3.4.1 ensures implication (3.12) is satisfied once a certain linear equality is satisfied.

Proposition 3.4.1 Prior Knowledge as a Linear Equality The implication (3.12), or equivalently the implication (3.11), is satisfied if $\exists v \in R^r$ such the following linear equality in v is satisfied:

$$K(x', B')u - \gamma - 1 + v'g(x)_+ = 0, \forall x \in \Gamma_1. \tag{3.13}$$

Proof If the implication (3.12) does not hold then for some $x \in \Gamma_1$ such that $g(x)_+ = 0$ it follows that $K(x', B')u - \gamma \neq 1$. However this leads to the following contradiction for that x:

$$0 = K(x', B')u - \gamma - 1 + v'g(x)_+ = K(x', B')u - \gamma - 1 \neq 0, \tag{3.14}$$

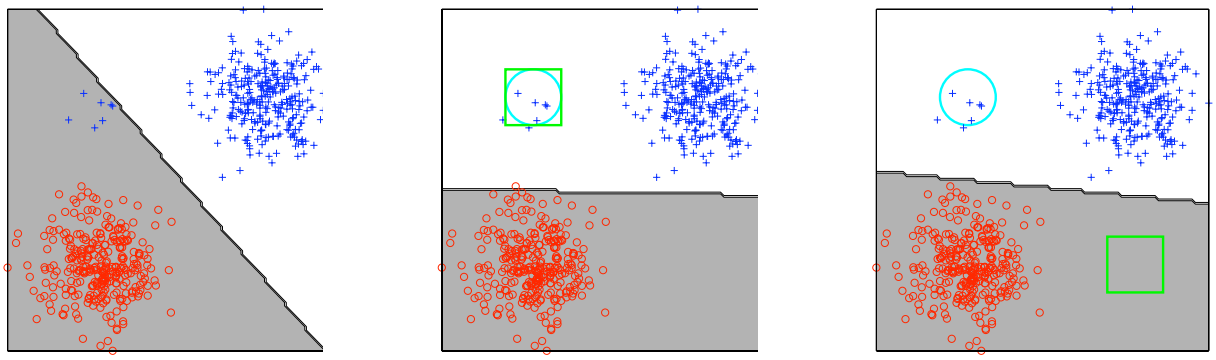
where the first equality follows from (3.13), the second equality from $g(x)_+ = 0$ and the inequality follows from $K(x', B')u - \gamma \neq 1$. □

We now use Proposition 3.4.1 to formulate proximal knowledge-based nonlinear kernel classification. Prior knowledge in the form of the implication (3.12) may be incorporated into the proximal support vector machine (1.17) by means of the linear equality (3.13), with weight $\sigma/2$, to be satisfied in a least square sense at ℓ discrete points x^i, $i = 1, \ldots, \ell$ in the set Γ_1 as follows:

$$\begin{array}{ll} \min\limits_{(u,\gamma,v)} & \frac{\nu}{2}\|D(K(A, B')u - e\gamma) - e\|^2 + \\ & \frac{\sigma}{2}\sum\limits_{i=1}^{\ell}(K(x^{i'}, B')u - \gamma - 1 + \\ & v'g(x^i)_+)^2 + \\ & \frac{1}{2}\|u\|^2 + \frac{1}{2}\gamma^2 + \frac{1}{2}\|v\|^2. \end{array} \tag{3.15}$$

In the above equation, we have also added an additional regularization term, $\|v\|^2$. This term ensures that the optimization problem is strongly convex, and trades off the size of v with the fit to the data, the extent to which (3.13) is satisfied, and the solution complexity. Figure 3.1 and its caption describe an example of the effect of v.

Remark 3.4.2 One important consequence of Proposition 3.4.1 is that the multiplier v causes the linear equality (3.13) to be sufficient for the implication (3.12) to hold **even if** Γ_1 **does not closely match** $\{x|g(x)_+ = 0\}$. In terms of (3.15), the discretization points x^i, $i = 1, \ldots, \ell$ need not be in the set $\{x|g(x)_+ = 0\}$. This sets our approach apart from those such as [65] where knowledge is imparted through discrete points in $\Gamma_1 = \{x|g(x)_+ = 0\}$. In the extreme example shown in Figure 3.1, our formulation (3.15) gives similar results whether Γ_1 is disjoint from $\{x|g(x)+ = 0\}$ or not. In Section 3.6.1 we do not ensure that every point in the discretization satisfies the left-hand side of the prior knowledge implication.



(a) (b) (c)

Figure 3.1 An example showing that the set Γ_1 discretized in (3.15) need not contain the region $\{x|g(x)_+ = 0\}$ in which the left-hand side of the implication (3.12) is satisfied. Each of the figures (a), (b) and (c) depict 600 points denoted by "+" and "o" that are obtained from three bivariate normal distributions. Another 400 points from the same three distributions in the same proportions were used for training and tuning each of the three classifiers of figures (a), (b) and (c). For simplicity, we use the linear kernel $K(A, B') = AB'$ to obtain three different linear classifiers to discriminate between the +'s and o's. A linear classifier without prior knowledge is shown in (a). Note that some + points from the rarely sampled distribution are misclassified. Figures (b) and (c) show classifiers using the same data *and* the prior knowledge $(\|x - \binom{-3}{3}\| - 1)_+ = 0 \Longrightarrow K(x', B')u - \gamma = 1$. The left hand side of this implication is true in the region enclosed by the circle in (b) and (c) and contains most of the + points from the rarely sampled distribution. The prior knowledge is imposed over a single grid of 100 evenly spaced points in the square shown and the parameter σ of (3.15) was set to 1. In (b), the square contains the set $\{x|(\|x - \binom{-3}{3}\| - 1)_+ = 0\}$, but the square of (c) is highly disjoint from the set $\{x|(\|x - \binom{-3}{3}\| - 1)_+ = 0\}$. Nevertheless, the classifier of (c) is nearly indistinguishable from that in (b). Techniques such as [65], which incorporate prior knowledge by adding points which conform to the knowledge as "pseudo" points to the training set, will not make use of a discretization such as that of (c). Our approach is able to handle points in the discretization that are not in $\{x|g(x)_+ = 0\}$ partly because of the multiplier v in (3.13). At the solution shown in (c), $v'g(x)_+ > 1$ in the discretized region. For such x, (3.13) implies that x should have class -1. Thus, v can select which side of the separating surface (1.11) points with a relatively large $g(x)_+$ should lie on. In (3.15), the size of v is traded off against the fit to the data, the extent to which (3.13) is satisfied, and the solution complexity. This extreme example illustrates an important property of our approach: Proposition 2.1 does not require that Γ_1 match the set $\{x|g(x)_+ = 0\}$ closely.

Since the objective function of (3.15) is strongly convex with a symmetric positive definite Hessian matrix, it has a unique solution obtained by setting its gradient equal to zero. This gives a system of $k + 1 + r$ *nonsingular linear equations* in as many unknowns (u, γ, v). We note that additional prior knowledge about the class -1 can be added to (3.15) in an analogous way. Finally, the prior knowledge can easily be imposed at many points by using the incremental technique of [29] since the size of the system of linear equations does not depend on ℓ. We note that in the linear programming approach of [86], both the number of constraints and the number of variables in the linear program grow linearly with ℓ. The fact that the size of the system of equations does not depend on ℓ allows our approach to potentially handle knowledge which is imposed at a huge number of points, which would be difficult to solve using the linear programming formulation.

3.5 Numerical experience

The effectiveness of our proposed linear programming formulation has been illustrated on three approximation tasks [89] and two classification tasks [86]. We describe these experiments and their results here. It is important to point out that the present formulation is very different in nature from that presented in [33] and [79]. Our primary concern here is to incorporate prior knowledge in an explicit and transparent manner without having to kernelize it as was done in [33] and [79]. In particular, we are able to directly incorporate general implications involving nonlinear inequalities in a linear program by utilizing Proposition 3.1.1. Synthetic examples were used to show how our approach uses nonlinear prior knowledge to obtain approximations or classifiers that are much better than those obtained without prior knowledge. Although the given prior knowledge for the synthetic example is strong, the example illustrates the simplicity and effectiveness of our approach to incorporate prior knowledge into nonlinear support vector classification and approximation. The Wisconsin Prognostic Breast Cancer (WPBC) dataset was used to demonstrate situations in which prior knowledge and data are combined to obtain a better approximation or classifier than by using only prior knowledge or data alone.

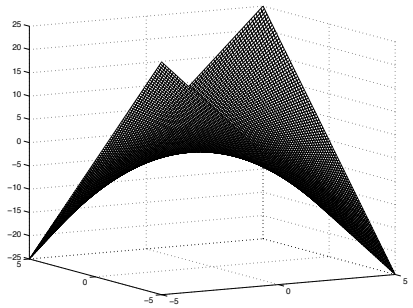


Figure 3.2 The exact hyperboloid function $\eta(x_1, x_2) = x_1 x_2$.

3.5.1 Approximation datasets

The effectiveness of our proposed approximation formulation (3.9) has been illustrated on a synthetic dataset and the Wisconsin Prognostic Breast Cancer (WPBC) database, available from [96].

3.5.1.1 Two-dimensional hyperboloid function

The first approximation example is the two-dimensional hyperboloid function:

$$\eta(x_1, x_2) = x_1 x_2. \tag{3.16}$$

This function was studied in [79]. The given data consists of eleven points along the line $x_1 = x_2$, $x_1 \in \{-5, -4, \ldots, 4, 5\}$. The given values at these points are the actual function values.

Figure 3.2 depicts the two-dimensional hyperboloid function of (3.16). Figure 3.3 depicts the approximation of the hyperboloid function by a surface based on the eleven points described above *without* prior knowledge. Figures 3.2 and 3.3 are taken from [79].

Figure 3.4 depicts a much better approximation than that of Figure 3.3 of the hyperboloid function by a nonlinear surface based on the same eleven points above *plus* prior knowledge. The prior knowledge consisted of the implication:

$$x_1 x_2 \leq 1 \Longrightarrow f(x_1, x_2) \leq x_1 x_2, \tag{3.17}$$

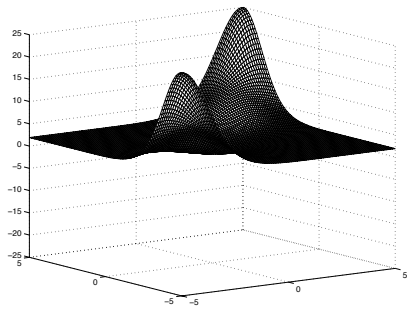


Figure 3.3 Approximation of the hyperboloid function $\eta(x_1, x_2) = x_1 x_2$ based on eleven exact function values along the line $x_2 = x_1, x_1 \in \{-5, -4, \ldots, 4, 5\}$, but *without* prior knowledge.

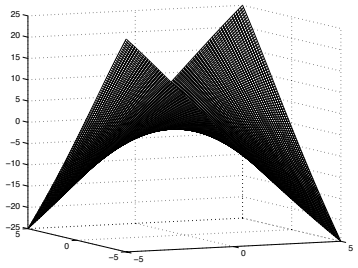


Figure 3.4 Approximation of the hyperboloid function $\eta(x_1, x_2) = x_1 x_2$ based on the same eleven function values as Figure 3.3 *plus* prior knowledge consisting of the implication (3.17).

which, because of the nonlinearity of x_1x_2, cannot be handled by [79]. Note that even though the prior knowledge implication (3.17) provides only partial information regarding the hyperboloid (3.16) being approximated, applying it is sufficient to improve our kernel approximation substantially as depicted in Figure 3.4. The prior knowledge implication (3.17) was applied in its equivalent inequality (3.4) form, at discrete points as stated in the inequality constraints of (3.9). In this example, the knowledge was applied at eleven points along the line $x_1 = -x_2$, $x_1 \in \{-5, -4, \ldots, 4, 5\}$.

It is instructive to compare (3.17) with the prior knowledge used in Chapter 2 to obtain a visually similar improvement. In that case, the prior knowledge given in (2.27), (2.28) was used:

$$(x_1, x_2) \in \{(x_1, x_2) | -\tfrac{1}{3}x_1 \leq x_2 \leq -\tfrac{2}{3}x_1\} \Rightarrow$$
$$f(x_1, x_2) \leq 10x_1$$

and

$$(x_1, x_2) \in \{(x_1, x_2) | -\tfrac{2}{3}x_1 \leq x_2 \leq -\tfrac{1}{3}x_1\} \Rightarrow$$
$$f(x_1, x_2) \leq 10x_2.$$

These implications were incorporated into a linear program with no discretization required using Proposition 2.2.1. However, these implications are not correct everywhere, but are merely intended to coarsely model the global shape of $\eta(x_1, x_2)$. This inexactness arises because of the limitation that knowledge be linear in the input space, and because the use of the nonlinear kernel to map knowledge in the input space to higher dimensions is difficult to interpret. In contrast, the prior knowledge of implication (3.17) is always correct and exactly captures the shape of the function. Thus, this example illustrates that there is a significant gain in usability due to the fact that the knowledge may be nonlinear in input space features.

3.5.1.2 Predicting lymph node metastasis

To demonstrate the effectiveness of our approximation formulation (3.9) on a real-world dataset, a potentially useful application of knowledge-based approximation to breast cancer prognosis [80, 123, 58] was considered. An important prognostic indicator for breast cancer

recurrence is the number of metastasized lymph nodes under a patient's armpit which could be as many as 30. To obtain this number, a patient must optionally undergo a potentially debilitating surgery in addition to the removal of the breast tumor. Thus, it is useful to approximate the number of metastasized lymph nodes using available information. The Wisconsin Prognostic Breast Cancer (WPBC) data, in which the primary task is to determine time to recurrence [96], contains information on the number of metastasized lymph nodes for 194 breast cancer patients, as well as thirty cytological features obtained by a fine needle aspirate and one histological feature, tumor size, obtained during surgery. Mangasarian et al. demonstrated in [79] that a function that approximated the number of metastasized lymph nodes using four of these features could be improved using prior knowledge. The formulation developed here has been used to approximate the number of metastasized lymph nodes using only the tumor size.

In order to simulate the situation where an expert provides prior knowledge regarding the number of metastasized lymph nodes based on tumor size, the following procedure was used. First, 20% of the data was randomly selected as "past data." This past data was used to develop prior knowledge, while the remaining 80% of the data, the "present data," was used for evaluation. The goal is to simulate the situation in which an expert can provide prior knowledge, but no more data is available. To generate such prior knowledge, kernel approximation was used to find a function $\phi(x) = K(x', B')u - \gamma$, where B is the matrix containing the past data and K is the Gaussian kernel defined in Section 1.1. This function was then used as the basis for our prior knowledge. Since this function was not believed to be accurate for areas where there was little data in the past data set, this knowledge was imposed only on the region $p(x) \geq 0.1$, where $p(x)$ was the density function for the tumor sizes in B estimated with the `ksdensity` routine, available in the MATLAB statistics toolbox [94]. The following prior knowledge implication was considered:

$$p(x) \geq 0.1 \Longrightarrow f(x) \geq \phi(x) - 0.01. \tag{3.18}$$

That is, the number of metastasized lymph nodes was greater than the predicted value on the past data, with a tolerance of 0.01. This implication incorporates a typical oncological

Approximation	RMSE
Without knowledge	5.92
With knowledge	**5.04**
Improvement due to knowledge	14.8%

Table 3.1 Leave-one-out root-mean-squared-error (RMSE) of approximations with and without knowledge on the present WPBC data. Best result is in bold.

surgeon's advice that the number of metastasized lymph nodes increases with tumor size. In order to accurately simulate the desired conditions, this knowledge was formed by observing only the past data. No aspect of the prior knowledge was changed after testing began on the present data.

Table 3.1 illustrates the improvement resulting from the use of prior knowledge. The first two entries compare the leave-one-out error of function approximations without and with prior knowledge. When training functions on each training set, ten points of the training set were selected as a tuning set. This set was used to choose the value of C from the set $\{2^i | i = -7, \ldots, 7\}$. The kernel parameter was set to 2^{-7}, which gave a smooth curve on the past data set. This value was fixed before testing on the present data. For the approximation *with* knowledge, the parameter ν was set to 10^6, which ensured that the prior knowledge would be taken into account by the approximation. Implication (3.18) was imposed as prior knowledge, and the discretization for the prior knowledge was 400 equally spaced points on the interval $[1, 5]$. This interval approximately covered the region on which $p(x) \geq 0.1$. We note that the use of prior knowledge led to a 14.8% improvement. In our experience, such an improvement is difficult to obtain in medical tasks, and indicates that the approximation with prior knowledge is more potentially useful than the approximation without prior knowledge.

In order to further illustrate the effectiveness of using prior knowledge, two other experiments were performed. First, the root-mean-squared-error (RMSE) of the function

ϕ was calculated on the present data, which was not used to create ϕ. The resulting RMSE was 6.12, which indicates that this function does not, by itself, do a good job predicting the present data. The leave-one-out error on the present data of an approximation that included the present data *and* the past data, but *without* prior knowledge was also calculated. This approach led to less than one percent improvement over the approximation without knowledge shown in Table 3.1, which indicates that the prior knowledge *in the form of the implication (3.18)* contains more useful information than the *raw* past data alone. These results indicate that the inclusion of the prior knowledge with the present data is responsible for the 14.8% improvement.

3.5.2 Classification datasets

The effectiveness of our proposed classification formulation (3.10) has been illustrated on two publicly available datasets: The Checkerboard dataset [44], and the Wisconsin Prognostic Breast Cancer (WPBC) dataset [96].

3.5.2.1 Checkerboard problem

The first classification example was based on the frequently utilized checkerboard dataset [44, 50, 77, 55, 33]. This synthetic dataset contains two-dimensional points in $[-1, 1] \times [-1, 1]$ labeled so that they form a checkerboard. For this example, a dataset consisting of only the sixteen points at the center of each square in the checkerboard was used to generate a classifier without knowledge. The rows of both matrices A and B of (3.10) were set equal to the coordinates of the sixteen points, which are the standard values. Figure 3.5 shows a classifier trained on these sixteen points *without* any additional prior knowledge.

Figure 3.6 shows a much more accurate classifier trained on the same sixteen points as used in Figure 3.5, *plus* prior knowledge representing only the leftmost two squares in the bottom row of the checkerboard. This knowledge was imposed via the following implications:

$$\begin{aligned} -1 \leq x_1 \leq -0.5 \wedge -1 \leq x_2 \leq -0.5 &\Longrightarrow f(x_1, x_2) \geq 0, \\ -0.5 \leq x_1 \leq 0 \wedge -1 \leq x_2 \leq -0.5 &\Longrightarrow f(x_1, x_2) \leq 0. \end{aligned} \tag{3.19}$$

The implication on the first line was imposed at 100 uniformly spaced points in $[-1, -0.5] \times [-1, -0.5]$, and the implication on the second line were imposed at 100 uniformly spaced points in $[-0.5, 0] \times [-1, -0.5]$. No prior knowledge was given for the remaining squares of the checkerboard. We note that this knowledge is very similar to that used in [33], although our classifier is more accurate here. This example demonstrates that knowledge of the form used in [33] can be easily applied with our proposed approach without kernelizing the prior knowledge.

3.5.2.2 Predicting breast cancer survival time

We conclude our experimental results with a potentially useful application of the Wisconsin Prognostic Breast Cancer (WPBC) dataset [96, 56]. This dataset contains thirty cytological features obtained from a fine needle aspirate and two histological features, tumor size and the number of metastasized lymph nodes, obtained during surgery for breast cancer patients. The dataset also contains the amount of time before each patient experienced a recurrence of the cancer, if any. Here, the task of predicting whether a patient will remain cancer free for at least 24 months is considered. Past experience with this dataset has shown that an accurate classifier for this task is difficult to obtain. In this dataset, 81.9% of patients are cancer free after 24 months. To our knowledge, the best result on this dataset prior to [86] is 86.3% correctness obtained by Bennett in [4]. It is possible that incorporating expert information about this task is necessary to obtain higher accuracy on this dataset. In [86] it was demonstrated that with sufficient prior knowledge, our approach can achieve 91.0% correctness.

To obtain prior knowledge for this dataset, the number of metastasized lymph nodes was plotted against the tumor size, along with the class label, for each patient. An oncological surgeon's advice was then simulated by selecting regions containing patients who experienced a recurrence within 24 months. In a typical machine learning task, not all of the class labels would be available. However, the purpose here is to demonstrate that if an expert is able to provide useful prior knowledge, our approach can effectively apply that knowledge to learn

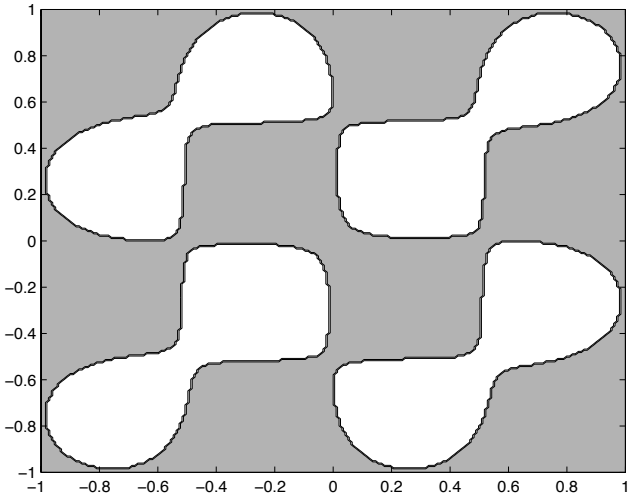


Figure 3.5 A classifier for the checkerboard dataset trained using only the sixteen points at the center of each square *without* prior knowledge. The white regions denote areas where the classifier returns a value greater than zero, and the gray regions denote areas where the classifier returns a value less than zero. A uniform grid consisting of $40,000$ points was used to create the plot utilizing the obtained classifier.

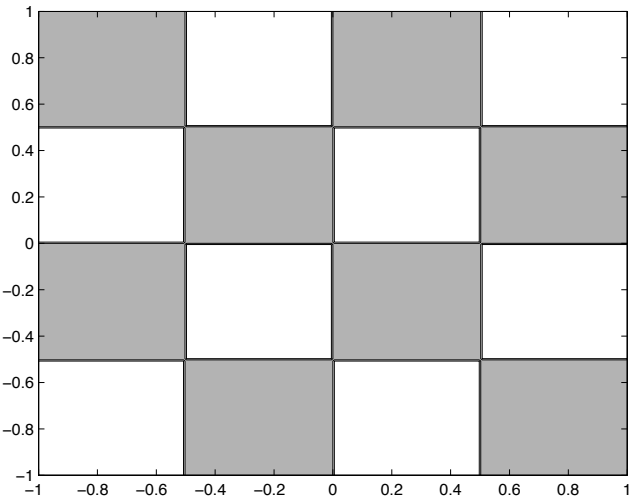


Figure 3.6 A classifier for the checkerboard dataset trained using the sixteen points at the center of each square *with* prior knowledge representing the two leftmost squares in the bottom row given in (3.19). The white regions denote areas where the classifier returns a value greater than zero, and the gray regions denote areas where the classifier returns a value less than zero. A uniform grid consisting of $40,000$ points was used to create the plot utilizing the knowledge-based classifier.

a more accurate classifier. We leave studies on this dataset in which an expert provides knowledge without all of the labels available to future work. In such studies, the expert would be given information regarding the class of only data points in a training set that is a subset of all the data, and then give advice on the class of points in the entire dataset. The prior knowledge constructed for this dataset is depicted in Figure 3.7 and consists of the following three implications:

$$\begin{array}{rl} \left\| \binom{(5.5)x_1}{x_2} - \binom{(5.5)7}{9} \right\| + \left\| \binom{(5.5)x_1}{x_2} - \binom{(5.5)4.5}{27} \right\| - 23.0509 & \leq 0 \Longrightarrow f(x) \geq 1 \\ \begin{pmatrix} -x_2 + 5.7143x_1 - 5.75 \\ x_2 - 2.8571x_1 - 4.25 \\ -x_2 + 6.75 \end{pmatrix} & \leq 0 \Longrightarrow f(x) \geq 1 \\ \frac{1}{2}(x_1 - 3.35)^2 + (x_2 - 4)^2 - 1 & \leq 0 \Longrightarrow f(x) \geq 1. \end{array} \tag{3.20}$$

The class +1 represents patients who experienced a recurrence in less than 24 months. Here, x_1 is the tumor size, and x_2 is the number of metastasized lymph nodes. Each implication is enforced at the points in the dataset for which the left-hand side of the implication is true. These regions are shown in Figure 3.7. The first implication corresponds to the region closest to the upper right-hand corner. The triangular region corresponds to the second implication, and the small elliptical region closest to the x_1 axis corresponds to the third implication. Although these implications seem complicated, it would not be difficult to construct a more intuitive interface similar to standard graphics programs to allow a user to create arbitrary regions. Applying these regions with our approach would be straightforward.

In order to evaluate our proposed approach, the misclassification rates of two classifiers on this dataset were compared. One classifier is learned without prior knowledge, while the second classifier is learned using the prior knowledge given in (3.20). For both cases the rows of the matrices A and B of (3.10) were set to the usual values, that is to the coordinates of the points of the training set. The misclassification rates are computed using leave-one-out cross validation. For each fold, the parameter ν and the kernel parameter μ were chosen from the set $\{2^i | i \in \{-7, \ldots, 7\}\}$ by using ten-fold cross validation on the training set of the fold. In the classifier with prior knowledge, the parameter σ was set to 10^6, which

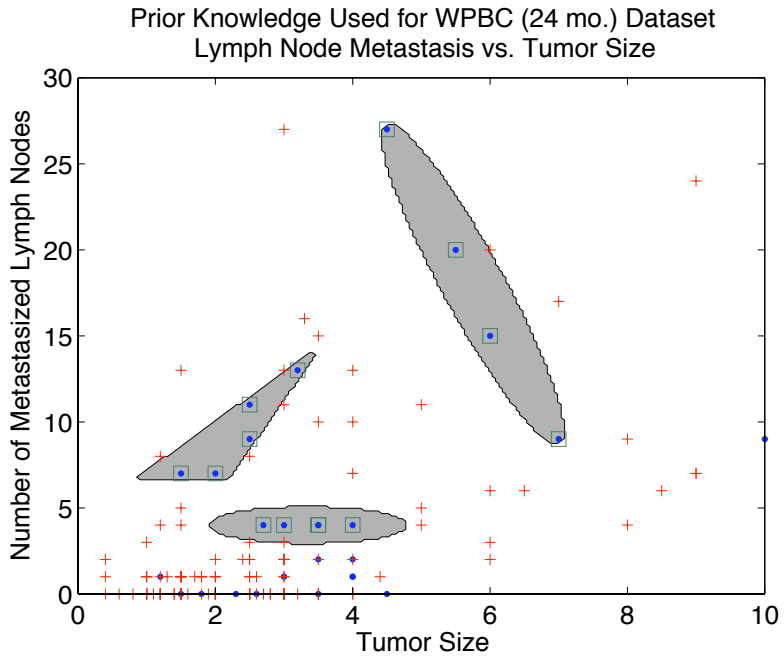


Figure 3.7 Number of metastasized lymph nodes versus tumor size for the WPBC (24 mo.) dataset. The solid dots represent patients who experienced a recurrence within 24 months of surgery, while the crosses represent the cancer free patients. The shaded regions which correspond to the areas in which the left-hand side of one of the three implications in Equation (3.20) is true simulate an oncological surgeon's prior knowledge regarding patients that are likely to have a recurrence. Prior knowledge was enforced at the points enclosed in squares.

Classifier	Misclassification Rate
Without knowledge	0.1806
With knowledge	**0.0903**
Improvement due to knowledge	50.0%

Table 3.2 Leave-one-out misclassification rate of classifiers with and without knowledge on the WPBC (24 mo.) dataset. Best result is in bold.

corresponds to very strict adherence to the prior knowledge. The results are summarized in Table 3.2. The reduction in misclassification rate indicates that our classification approach can use appropriate prior knowledge to obtain a classifier on this difficult dataset with 50% improvement.

3.6 Comparing linear programming and proximal knowledge-based classification

Here, we compare the knowledge-based nonlinear proximal classification formulation (3.15) to the linear programming formulation (3.10) on three publicly available datasets: the Checkerboard dataset [44], the Spiral dataset [122], and the Wisconsin Prognostic Breast Cancer (WPBC) dataset [96]. Our results show that our proximal formulation can obtain solutions with accuracy similar to the linear programming formulation, while being much faster to solve.

The frequently used Checkerboard [44, 33] and Spiral [122, 27] datasets are synthetic datasets for which prior knowledge can be easily constructed [86, 33]. The checkerboard dataset consists of points with labels "black" and "white" arranged in the shape of a checkerboard, while the spiral dataset consists of points from two concentric spirals. Table 3.3 shows both the accuracy and CPU time needed to run the experiments of [86] using both the linear programming formulation originally used in [86] and our proposed proximal formulation. On both datasets, 225 optimization problems were solved. In the checkerboard

experiment, the matrix B has 16 rows, and the prior knowledge is imposed at 200 points. In the spiral dataset, the matrix B has 194 rows, and the knowledge is imposed at 194 points. For the checkerboard experiment, the knowledge consists only of the two leftmost squares of the bottom row, while for the spiral dataset, the knowledge was constructed by inspecting the source code used to generate the spiral. The experiments were performed using MATLAB 7.2 [94] under CentOS Linux 4.4 on an Intel Pentium IV 3 GHz processor with 1 gigabyte of RAM. The running times were calculated by the MATLAB profiler, and represent the total time during the experiment consumed in setting up and solving the optimization problem. Linear programs were solved using CPLEX 9.0 [48], and the linear systems of equations were solved using the `chol` routine of MATLAB. For the spiral dataset, flush-to-zero mode was enabled to speed the multiplication of numbers with very small magnitude. This change had a significant impact on the running time of our proximal formulation, and negligible impact on the linear programming formulation. Note that the proximal formulation has similar accuracy to the linear programming formulation, while being *approximately an order of magnitude faster* to solve. We further note that considering only the time taken to solve the linear program or linear system of equations gives a similar result, thus we do not believe that the difference in computation time can be attributed to the setup procedure.

We have also tested our proposed proximal formulation on the WPBC dataset, using the setup and prior knowledge described above in Section 3.5.2.2. The proximal formulation achieved the same results as the linear programming formulation, given in Table 3.2. Even though they generate slightly different separating surfaces, both the linear programming and proximal formulations misclassify the same number of examples on this dataset. The reduction in misclassification rate indicates that our proximal approach can achieve the same 50% improvement in classification accuracy using prior knowledge as the linear programming formulation.

Table 3.3 Accuracy and CPU time in seconds for the linear programming formulation [86] and the proposed proximal formulation. Each running time result is the total time needed to set up and solve the optimization problem, either as a linear program or a linear system of equations, 225 times. The time ratio is the time for the linear programming formulation divided by the time for the proximal formulation.

Dataset	Linear Programming SVM [86] Accuracy CPU Time in Seconds	Proximal SVM Accuracy CPU Time in Seconds	Time Ratio
Checkerboard without Knowledge	89.2% 2.3	94.2% 0.2	11.5
Checkerboard with Knowledge	100% 26.4	98.0% 3.2	8.3
Spiral without Knowledge	79.9% 21.3	80.4% 4.3	5.0
Spiral with Knowledge	100% 300.2	100% 19.0	15.8

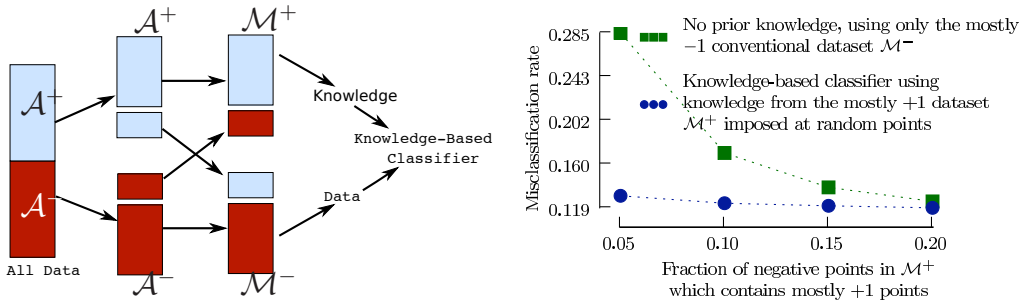
3.6.1 Generating prior knowledge from ordinary classification datasets

In order to further demonstrate the effectiveness of our proposed formulation, we generate prior knowledge from a subset of an ordinary classification dataset. In these experiments, we will ensure that the prior knowledge is generated without knowing the true distribution of the data, or inspecting the full data set. By using ordinary datasets, we are also able to easily demonstrate our proposed formulation on datasets with more than two features. In order for prior knowledge to improve classification accuracy when combined with ordinary data, the prior knowledge and the data must contain different information about the "true" dataset. Thus, we simulate a situation where a knowledge-based classifier using both prior knowledge and data will be superior to a classifier that uses either the data or prior knowledge alone. In our scenario, the set $\mathcal{M}^+$ will consist mostly of points from the class $+1$ and will be used only to generate prior knowledge, while the set $\mathcal{M}^-$ will consist mostly of points from the class -1 and will be used only as ordinary data. We varied the percentage of negative points in $\mathcal{M}^+$. As this percentage approaches fifty percent one expects that $\mathcal{M}^+$ and $\mathcal{M}^-$ will contain the same information, and the gain due to incorporating prior knowledge will be minimal. Construction of the sets $\mathcal{M}^+$ and $\mathcal{M}^-$ is illustrated in Figure 3.8 (a). The motivation for this scenario is a situation in which prior knowledge is available about data in the set $\mathcal{M}^+$, while only the set $\mathcal{M}^-$ is available as ordinary data. Thus, the learning algorithm will need to incorporate both prior knowledge about $\mathcal{M}^+$ and the conventional data in $\mathcal{M}^-$ in order to generalize well to new points.

One can imagine many methods of automatically generating prior knowledge from $\mathcal{M}^+$, such as [13]. However, we used the simple approach of learning a proximal support vector machine on the points in $\mathcal{M}^+$. The knowledge we used was the following:

$$(-\phi(x))_+ = 0 \Longrightarrow K(x', B')u - \gamma = 1, \ \forall x \in \Gamma_1, \tag{3.21}$$

where $\phi(x)$ is the classifier function (1.4) learned on the set $\mathcal{M}^+$. This knowledge simply states that if the proximal support vector machine represented by $\phi(x)$ labels the point



(a) (b)

Figure 3.8 (a) Generation of prior knowledge from a standard dataset. The dataset is first separated into the datasets $\mathcal{A}^+$ which consists of all +1 points, and $\mathcal{A}^-$ which consists of all −1 points. Then the mostly +1 dataset $\mathcal{M}^+$ is formed by replacing a small fraction of +1 points in $\mathcal{A}^+$ with an equal number of −1 points from $\mathcal{A}^-$. The mostly −1 dataset $\mathcal{M}^-$ is formed from the points not used in $\mathcal{M}^+$. We use $\mathcal{M}^+$ to produce prior knowledge, and $\mathcal{M}^-$ as ordinary data. Combining the knowledge from $\mathcal{M}^+$ and the data from $\mathcal{M}^-$ leads to a knowledge-based classifier which is superior to a classifier formed using either $\mathcal{M}^+$ as pure knowledge or $\mathcal{M}^-$ as pure data alone. (b) Prior knowledge experiment on the NDCC dataset: 300 points in R^{50}.

as +1, then the point should be labeled +1 by the classifier which combines both data and knowledge. We impose the prior knowledge of (3.21) at a random sample from a multivariate normal distribution fit to the points in $\mathcal{M}^+$. We chose the multivariate normal distribution for simplicity, but one can easily imagine using a more sophisticated distribution. Investigation of different methods of generating prior knowledge is left to future research. Since $\phi(x)$ is learned with few negative examples, it will likely not be accurate over the entire dataset. In fact, in our experiments $\phi(x)$ alone always had similar accuracy on the test set to the classifier built using only the data in $\mathcal{M}^-$. Intuitively, since $\phi(x)$ is a kernel classifier, the knowledge we imposed states that points which are "close" to the +1 points in $\mathcal{M}^+$ used to construct $\phi(x)$ should be labeled +1. However, enforcement of this knowledge is balanced against fitting the ordinary data points in $\mathcal{M}^-$. Thus, the combination of data and prior knowledge is necessary to obtain high accuracy.

Figure 3.8 (b) shows the result of applying the above procedure to Thompson's Normally Distributed Clusters on Cubes (NDCC) dataset [115]. This dataset generates points

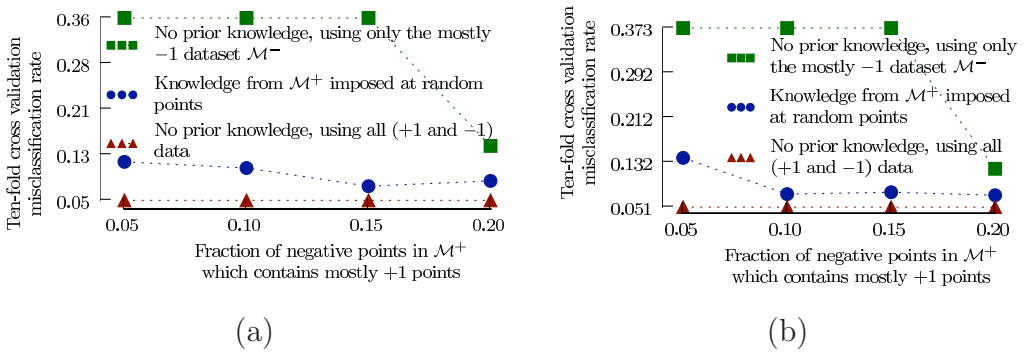


Figure 3.9 Prior knowledge experiment on (a) the WDBC dataset: 569 points in R^{30}, with 212 malignant tumors labeled +1; and (b) the Ionosphere dataset: 351 points in R^{34}, with 126 bad radar returns labeled +1.

according to multivariate normal distributions centered at the vertices of two concentric 1-norm cubes. Points are mostly labeled according to the cube they were generated from, with some specified fraction of noisy labels. We generated a dataset of 20000 points in R^{50}, with ten percent label noise. We used 300 points as a training set, 2000 separate points to choose parameters from the set $\{2^i | i = -7, \ldots, 7\}$, and the remaining 17700 points to evaluate the classifiers. In Figure 3.8 (b), we compare an approach using only the data in the set $\mathcal{M}^-$ and no prior knowledge to an approach based on the same data *plus* prior knowledge obtained from the points in $\mathcal{M}^+$, with σ equal to ν. The knowledge was imposed on $|\mathcal{M}^+|$ randomly sampled points as described above, where $|\mathcal{M}^+|$ is the cardinality of $\mathcal{M}^+$. In our experience on this dataset, reducing the number of sampled points to less than half $|\mathcal{M}^+|$ had very little impact on accuracy. Determining the appropriate number of points to sample for a given dataset is left to future work. The approach using prior knowledge is able to approach ten percent misclassification error even when relatively few points in $\mathcal{M}^-$ have label +1.

Figure 3.9 shows the result of applying the above procedure to the publicly available Wisconsin Diagnostic Breast Cancer (WDBC) and Ionosphere datasets [96]. In the WDBC dataset, the task is to classify tumors as either malignant or benign based on the 30 features given. To simulate the scenario in which most information about malignant tumors is

available only through prior knowledge, while information about benign tumors is more readily gathered, we label malignant tumors +1. In the Ionosphere dataset, the task is to classify radar returns as either good or bad based on the 34 features given. We chose to label bad radar returns +1. To asses the generalization performance of our approach, we computed ten-fold cross validation misclassification rates. We chose all parameters from the set $\{2^i | i = -7, \ldots, 7\}$ using ten-fold cross validation on the training set. When using prior knowledge, we set σ equal to ν. In carrying out the cross validation experiment, $\mathcal{M}^+$ and $\mathcal{M}^-$ were formed from the training set for each fold. In Figure 3.9, three different approaches are compared. In the first approach, represented by squares, the classifier is learned using only the data in $\mathcal{M}^-$ with *no* prior knowledge. This classifier performs poorly until a sufficient number of +1 points are present in $\mathcal{M}^-$. The second approach, represented by circles, learns a classifier using the data in $\mathcal{M}^-$ *plus* the prior knowledge from $\mathcal{M}^+$ described by (3.21). The knowledge was imposed at $|\mathcal{M}^+|$ randomly generated points as described above. Note that the use of prior knowledge results in considerable improvement, especially when there are few points in $\mathcal{M}^+$ with class -1. For reference, we include an approach represented by triangles which uses no prior knowledge, but *all* the data. Note that this classifier has the same misclassification rate regardless of the fraction of negative points in $\mathcal{M}^+$. Including this approach illustrates that our approach is able to use the prior knowledge generated from $\mathcal{M}^+$ to recover most of the information in $\mathcal{M}^+$. Recall that we are simulating a situation in which $\mathcal{M}^+$ is only available as prior knowledge.

Chapter 4

Privacy-Preserving Classification via Random Kernels

Recently there has been wide interest in privacy-preserving support vector machine (SVM) classifiers. Basically the problem revolves around generating a classifier based on data, parts of which are held by private entities who for various reasons are unwilling to make them public. When each entity holds its own group of input feature values for all individuals while other entities hold other groups of feature values for the same individuals, the data is referred to as *vertically partitioned*. This is so because feature values are represented by columns of a data matrix while individuals are represented by rows of the data matrix. Privacy-preserving associations for vertically partitioned data were first proposed in [117]. When each entity holds all the feature values for its own group of individuals while other entities hold similar data for other groups of individuals, the data is referred to as *horizontally partitioned*. In [124, 125], *horizontally partitioned* privacy-preserving SVMs and induction tree classifiers were obtained for data where different entities hold the same input features for different groups of individuals. In [60] multiplicative data perturbation was utilized for privacy-preserving data mining. Other privacy preserving classifying techniques include cryptographically private SVMs [107], wavelet-based distortion [61] and rotation perturbation [10]. A privacy-preserving decision tree for multiple entities is described in [112].

We consider highly efficient privacy-preserving SVM (PPSVM) classifiers for either horizontally or vertically partitioned data, first implemented in [92] and [90], based on the use of a *completely random* matrix. That is, for a given data matrix $A \in R^{m \times n}$, instead

of the usual kernel function $K(A, A') : R^{m\times n} \times R^{n\times m} \longrightarrow R^{m\times m}$, we use a *reduced* kernel [55, 54] $K(A, B') : R^{m\times n} \times R^{n\times \bar{m}} \longrightarrow R^{m\times \bar{m}}$, where B is a completely random matrix, instead of a submatrix of the rows of A. This idea will allow us to describe algorithms for vertically and horizontally partitioned data which protect the privacy of each partition while generating an SVM classifier which has tenfold correctness comparable to that of an ordinary SVM classifier. Before describing these algorithms, we show computational results which demonstrate that a random kernel $K(A, B')$ achieves comparable accuracy to the usual kernel $K(A, A')$ or the reduced kernel $K(A, \bar{A}')$.

4.1 Comparison of a random kernel to full and reduced kernels

To justify the use of a random kernel in our proposed PPSVM algorithms, we compare the performance of a 1-norm SVM using a random kernel with both an ordinary 1-norm SVM using a full kernel matrix and a 1-norm SVM using a reduced kernel matrix (RSVM) [55]. Figure 4.1 shows scatterplots comparing the error rates of a random kernel with an ordinary full kernel, and with a reduced kernel, using linear kernels. Note that points close to the 45 degree line represent datasets for which the classifiers being compared have similar error rates. All of the error rates were obtained using datasets used are those in Table 4.1. Figure 4.2 shows similar results using the Gaussian kernel. All of the error rates were obtained using the datasets in Table 4.1. For both linear and nonlinear kernels, $\bar{A}$ consisted of ten percent of the rows of A randomly selected, while B was a completely random matrix with the same size as $\bar{A}$. Each entry of B was selected from a normal distribution with mean zero and standard deviation one. All datasets were normalized so that each feature had mean zero and standard deviation one. For the linear classifiers, the regularization parameter ν was selected from $\{10^i | i = -7, \ldots, 7\}$ for each dataset using a ten percent of the training data as a tuning set. To save time for the nonlinear classifiers, we used the tuning strategy described in [46]. In this Nested Uniform Design approach, rather than evaluating a classifier at each point of a grid in the parameter space, the classifier is evaluated only at a set of points which is designed to "cover" the original grid to the extent possible. The point from this

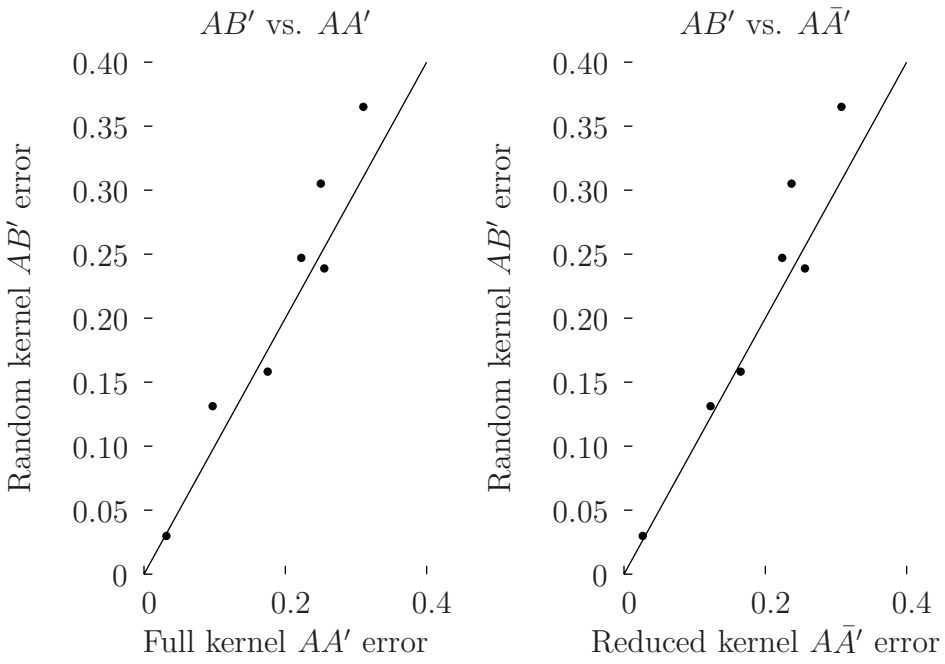


Figure 4.1 Error rate comparison of 1-norm linear SVMs for random kernel versus full and reduced kernels. For points below the diagonal, the random kernel has a lower error rate. The diagonal line in each plot marks equal error rates. One result is given for each dataset in Table 4.1.

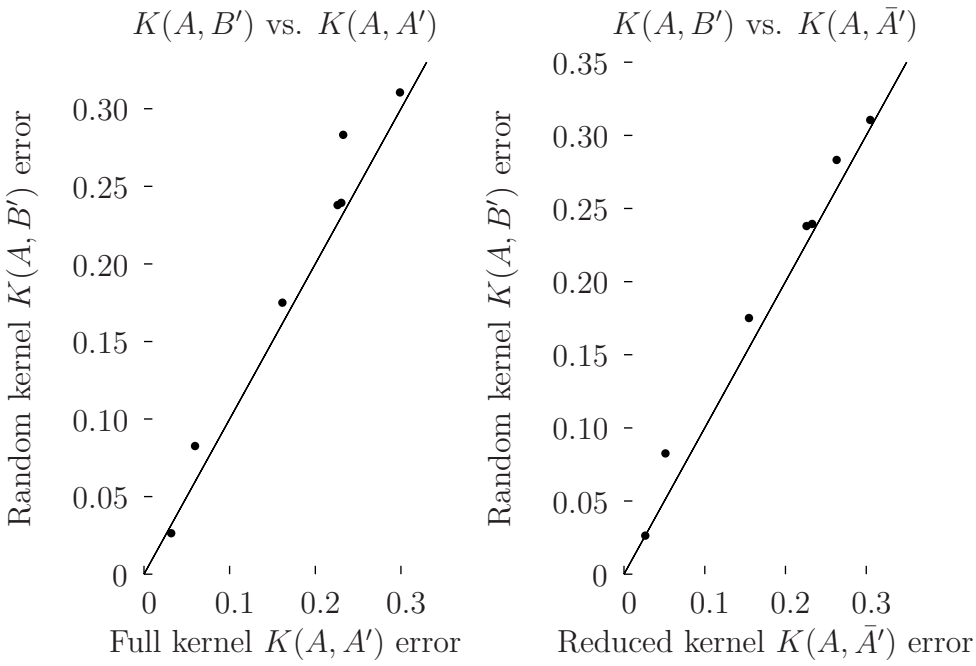


Figure 4.2 Error rate comparison of 1-norm nonlinear SVM for random kernel versus full and reduced kernels. For points below the diagonal, the random kernel has a lower error rate. The diagonal line in each plot marks equal error rates. One result is given for each dataset in Table 4.2.

smaller set on which the classifier does best is then made the center of a grid which covers a smaller range of parameter space, and the process is repeated. Huang et al. [46] demonstrate empirically that this approach finds classifiers with similar misclassification error as a brute-force search through the entire grid. We set the initial range of $\log_{10}\nu$ to $[-7, 7]$, and the initial range of $\log_{10}\mu$ as described in [46]. We used a Uniform Design with thirty runs from `http://www.math.hkbu.edu.hk/UniformDesign` for both nestings, and used five-fold cross validation on the training set to evaluate each (ν, μ) pair. We note that the misclassification error using a random kernel, as in the following proposed PPSVM approaches, is comparable to that of 1-norm SVM and RSVM using both linear and Gaussian kernels.

Before describing our PPSVM algorithms, it is helpful to consider the linear kernel classifier that the algorithms will learn for a given data matrix A. This classifier will be a separating plane in R^n:

$$x'w - \gamma = x'B'u - \gamma = 0, \tag{4.1}$$

which classifies a given point x according to the sign of $x'w - \gamma$. Here, $w = B'u$, $w \in R^n$ is the normal to the plane $x'w - \gamma = 0$, $\gamma \in R$ determines the distance of the plane from the origin in R^n and B is a random matrix in $R^{k\times n}$. The change of variables $w = B'u$ is employed in order to kernelize the data and is motivated by the fact that when $B = A$ and hence $w = A'u$, the variable u is the dual variable for a 2-norm SVM [72]. One justification for the above results can be given for the case when $\bar{m} \geq n$ and the rank of the $m \times n$ matrix B is n. For such a case, when B is replaced by A in (4.1), this results in a regular linear SVM formulation with a solution, say $v \in R^m$. In this case , the reduced SVM formulation (4.1) can match the regular SVM term $AA'v$ by the term $AB'u$, since $B'u = A'v$ has a solution u for any v because B' has rank n.

We now turn to our PPSVM algorithm for vertically partitioned data.

4.2 Privacy-preserving linear classifier for vertically partitioned data

The dataset that we wish to obtain a classifier for consists of m points in R^n represented by the m rows of the matrix $A \in R^{m \times n}$. The matrix A is divided into p vertical blocks of $n_1, n_2, \ldots$ and n_p columns with $n_1 + n_2 + \ldots + n_p = n$. Each block of columns is "owned" by an entity that is unwilling to make it public or share it with the other entities. Furthermore, each row of A is labeled as belonging to the class $+1$ or -1 by a corresponding diagonal matrix $D \in R^{m \times m}$ of ± 1's. We shall now partition the random matrix B into p column blocks with each column block belonging to one of the p entities held privately by it and never made public. Thus, we have:

$$B = [B_{\cdot 1} \;\; B_{\cdot 2} \ldots B_{\cdot p}]. \tag{4.2}$$

We are ready to state our algorithm which will provide a classifier for the data without revealing privately held data blocks $[A_{\cdot 1} \quad A_{\cdot 2} \ldots A_{\cdot p}]$. The accuracy of this algorithm will be comparable to that of an SVM using a publicly available A instead of merely $K(A_{\cdot 1}, B_{\cdot 1}'), K(A_{\cdot 2}, B_{\cdot 2}'), \ldots, K(A_{\cdot p}, B_{\cdot p}')$, as is the case here. We note that we require the nonlinear kernel to be separable in the following sense:

$$K([E \;\; F], [G \;\; H]') = K(E, G') \oplus K(F, H'), \tag{4.3}$$

where $E \in R^{m \times n_1}$, $F \in R^{m \times n_2}$, $G \in R^{k \times n_1}$ and $H \in R^{k \times n_2}$. It is easy to verify that this definition of separability is satisfied by a linear kernel using matrix addition, and by a Gaussian kernel using the Hadamard component-wise product [45].

Algorithm 4.2.1 PPSVM Algorithm for Vertically Partitioned Data

(I) All p entities agree on the *same* labels for each data point, that $D_{ii} = \pm 1$, $i = 1, \ldots, p$ and on the magnitude of $\bar{m}$, the number of rows of the random matrix B. (If an agreement on D is not possible, we can use semisupervised learning to handle such data points [5, 28].)

(II) Each entity generates its own privately held random matrix $B_{\cdot j} \in R^{\bar{m} \times n_j}$, $j = 1, \ldots, p$, where n_j is the number of input features held by entity j.

(III) Each entity j makes public its nonlinear kernel $K(A_{\cdot j}, B_{\cdot j}{}')$. This does not reveal $A_{\cdot j}$ but allows the public computation of the full nonlinear kernel:

$$K(A, B') = K(A_{\cdot 1}, B_{\cdot 1}{}') \odot K(A_{\cdot 2}, B_{\cdot 2}{}') \odot \ldots \odot K(A_{\cdot p}, B_{\cdot p}{}'), \tag{4.4}$$

where the symbol $\cdot$ denotes the element-wise, or Hadamard, product.

(IV) A publicly calculated linear classifier $K(x', B)u - \gamma = 0$ is computed by some standard method such as 1-norm SVM [72, 8]:

$$\begin{array}{rll} \min\limits_{(u,\gamma,y)} & \nu\|y\|_1 + \|u\|_1 & \\ \text{s.t.} & D(K(A, B')u - e\gamma) + y \;\geq\; e, & \\ & y \;\geq\; 0. & \end{array} \tag{4.5}$$

(V) For each *new* $x \in R^n$, each entity makes public $K(x_j{}', B_{\cdot j}{}')$ from which a public nonlinear classifier is computed as follows:

$$K(x', B')u - \gamma = (K(x_1{}', B_{\cdot 1}{}') \odot K(x_2{}', B_{\cdot 2}{}') \odot \ldots \odot K(x_p{}', B_{\cdot p}{}'))u - \gamma = 0, \tag{4.6}$$

which classifies the given x according to the sign of $K(x', B')u - \gamma$.

Remark 4.2.2 Note that in the above algorithm no entity j reveals its dataset $A_{\cdot j}$ nor its components of a new data point x_j. This is so because it is impossible to compute the mn_j numbers constituting $A_{\cdot j} \in R^{m \times n_j}$ given only the $m\bar{m}$ numbers constituting $K(A_{\cdot j}, B_{\cdot j}{}') \in R^{m \times \bar{m}}$ and not even knowing $B_{\cdot j} \in R^{\bar{m} \times n_j}$. Similarly it is impossible to compute $x_j \in R^{n_j}$ from $K(x_j{}', B_{\cdot j}{}') \in R$ without even knowing $B_{\cdot j}$. Hence, all entities share the publicly computed nonlinear classifier (4.11) using $K(A, B')$ and $K(x', B')$ without revealing either the individual datasets or new point components.

Before turning to our computational results, it is important to note that Algorithm 4.2.1 can be used easily with other kernel classification algorithms instead of the 1-norm SVM,

including the ordinary 2-norm SVM [109], the proximal SVM [27], and logistic regression [120].

It is instructive to compare our proposed privacy preserving SVM (PPSVM) given in Algorithm 4.2.1 to other recent work on privacy preserving classification for vertically-partitioned data which also makes use of random matrices. Du et al. [18] propose a method by which two parties can compute a privacy-preserving linear kernel classifier by securely computing the matrix $A'A$ with the use of random matrices. Yu et al. [126] use random matrices to securely compute the full kernel matrix, $K(A, A')$. Our approach is motivated by the observation that the accuracy of an SVM using a *random kernel*, $K(A, B')$, where B is a completely random matrix, is comparable to the accuracy of an SVM using the full kernel $K(A, A')$. We provide experimental support for this observation in Section 4.3. By using $K(A, B')$ instead of $K(A, A')$ we are able to obtain an accurate classifier with only very simple, asynchronous communication required among the entities. That is, each entity j needs only to broadcast $K(A_{\cdot j}, B'_{\cdot j})$ to and receive the corresponding message from each of the other entities. In [18, 126], synchronized communication steps are needed to securely compute $A'A$ or $K(A, A')$. For example, in [126], each entity must wait for a message to be passed through each of the other $p-1$ entities sequentially to compute $K(A, A')$. The primary benefit of our approach is simplicity. For example, it is conceptually easy to add or remove entities at any time with our approach. All that is required to add an entity at any point is for the new entity j to make $K(A_{\cdot j}, B'_{\cdot j})$ available to the other entities, and receive the existing kernel. If an entity needs to be removed, then the remaining entities can recompute the kernel matrix without that entity's contribution. This flexibility is possible because entity j sends $K(A_{\cdot j}, B'_{\cdot j})$ directly to all the other entities. Another potential benefit is that it seems reasonable to implement our approach in practice without specialized software to coordinate the communication steps even with large numbers of entities. Furthermore, our approach can be implemented in "parallel" with each entity receiving $p-1$ datasets of size $m \times \bar{m}$. In contrast, the process in [126] is inherently "serial" because the $m \times \bar{m}$

perturbed dataset of each entity j must be processed sequentially by the other $p-1$ entities and returned to entity j before that entity can utilize it in its classifier.

We have found that the parallelism in our approach allows entities to compute the kernel matrix faster than the approach of [126] in some circumstances. To test this result, we implemented both approaches to compute the kernel matrix for one entity and tested them on machines in the Computer Systems Lab at the University of Wisconsin-Madison. Each entity was run on a randomly selected computer, with no computer running more than one entity. We ran each algorithm with between three and ten entities, using matrices with 1000, 10000, and 100000 elements. Each algorithm was run ten times for each level of entities and matrix size, using ten different randomly selected sets of machines. On average, our approach was more than twice as fast as the approach in [126]. In some cases, particularly with smaller matrices, our approach was over 5 times as fast, while for larger matrices our approach tended to be between 0.5 and 1.5 times as fast as [126], and most often our approach was faster.

4.3 Computational results for vertically partitioned data

We illustrate the effectiveness of our proposed privacy preserving SVM (PPSVM) for vertically partitioned data by demonstrating that our approach can obtain classifiers with lower misclassification error than classifiers obtained using only the input features of each entity alone. In all of our results, $\bar{A}$ consisted of ten percent of the rows of A randomly selected, while B was a completely random matrix of the same size as $\bar{A}$. Each entry of B was selected from a normal distribution with mean zero and standard deviation one. All datasets were normalized so that each feature had mean zero and standard deviation one. Note that this normalization is carried out for each feature independently, and does not require cooperation among the entities.

We investigate the benefit of using our PPSVM approach instead of using only the input features available to each entity using seven datasets from the UCI repository [96]. To simulate a situation in which each entity has only a subset of the features for each data point, we randomly distribute the features among the entities such that each entity receives

about the same number of features. We chose arbitrarily to perform experiments using five entities for each dataset, and also to perform experiments using whatever number of entities was needed so that each entity received about three features. We also investigate our approach as the number of entities increases on the Ionosphere dataset described below and in Figures 4.5 and 4.6.

Figure 4.3 shows results comparing the ten-fold cross validation misclassification error of our linear kernel PPSVM with the average misclassification error of the 1-norm SVM classifiers learned using only the unshared input features available to each entity. Figure 4.3 also compares our linear kernel PPSVM with the misclassification error of the majority vote of the 1-norm SVM classifiers learned using only the input features available to each entity. We use circles to show how our approach compares to the average error of the entities and triangles to show how our approach compares to the majority vote of the entities. Points below the 45 degree line represent experiments in which our PPSVM has lower error rate than the average error rate of the classifiers learned with only each entity's subset of the features. This indicates that the entities can expect improved performance using PPSVM instead of going it alone. Note that each dataset is represented by four points: one circle and one triangle for the experiment using five entities, and one circle and one triangle for the experiment using a sufficient number of entities so that each entity receives about three features. The results shown in Figure 4.3 are detailed in Table 4.1. We note that PPSVM obtains classifiers with lower error than the average of the classifiers using only each entity's features in eleven of fourteen experiments, and lower error than the majority vote of classifiers using only each entity's features in thirteen of fourteen experiments. The parameter ν was selected from $\{10^i | i = -7, \ldots, 7\}$ for each dataset using a random ten percent of each training set as a tuning set.

Figure 4.4 shows results for similar experiments using Gaussian kernels, with details in Table 4.2. We used the same datasets as for the experiments described above. To save time, we used the tuning strategy from [46], as described in Section 4.1.

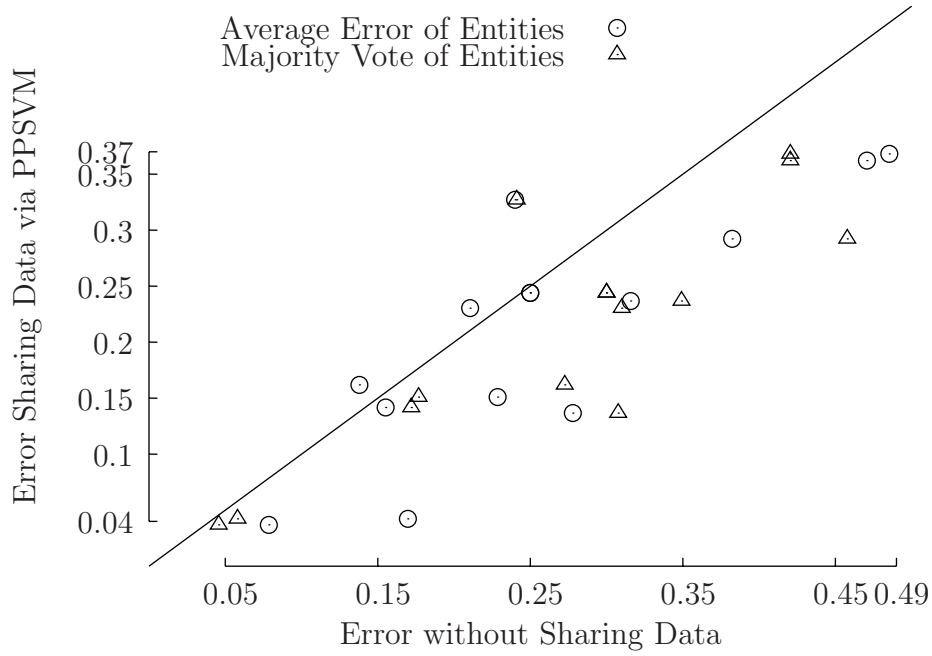


Figure 4.3 Error rate comparison of a 1-norm linear SVM sharing $A_{\cdot j}B'_{\cdot j}$ data for each entity versus a 1-norm linear SVM using just the input features $A_{\cdot j}$ of each entity. We compare to both the average error rate of the entities using just the input features, and to a classifier which combines the labels of all the entities by majority vote. Points below the diagonal represent situations in which the error rate for sharing is lower than the error rate for not sharing. Results are given for each dataset with features randomly distributed evenly among 5 entities, and with features randomly distributed so that each entity has about 3 features. Seven datasets given in Table 4.1 were used to generate four points each.

Dataset Examples × Input Features	No. of Entities	No Sharing	Majority Vote	Sharing
Cleveland Heart	5	0.1379	0.2725	0.1620
297 × 13	4	0.1552	0.1720	0.1416
Ionosphere	5	0.2286	0.1767	0.1510
351 × 34	11	0.2779	0.3076	0.1367
WDBC	5	0.0786	0.0457	0.0369
569 × 30	10	0.1696	0.0581	0.0423
Arrhythmia	5	0.2400	0.2411	0.3272
452 × 279	93	0.3823	0.4578	0.2922
Pima Indians	5	0.3158	0.3491	0.2368
768 × 8	2	0.2105	0.3100	0.2303
Bupa Liver	5	0.4706	0.4204	0.3622
345 × 6	2	0.4853	0.4204	0.3681
German Credit	5	0.2500	0.3000	0.2440
1000 × 24	8	0.2500	0.3000	0.2440

Table 4.1 Comparison of error rates for entities not sharing and sharing their datasets using a 1-norm linear SVM.

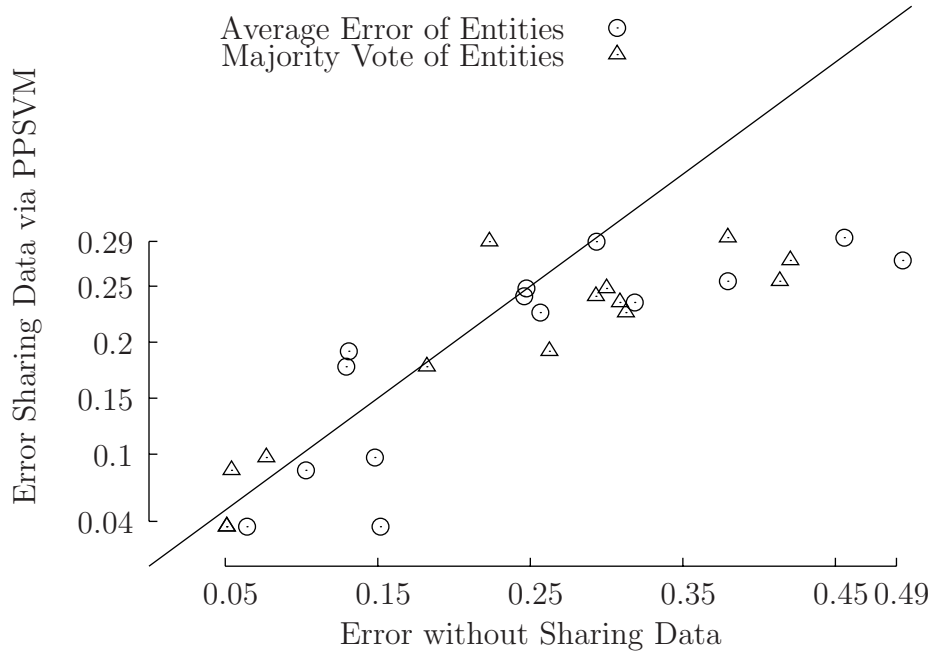


Figure 4.4 Error rate comparison of a 1-norm nonlinear SVM sharing $K(A_{\cdot j}, B'_{\cdot j})$ data for each entity versus a 1-norm nonlinear SVM using just the input features $A_{\cdot j}$ of each entity. We compare to both the average error rate of the entities using just the input features, and to a classifier which combines the labels of all the entities by majority vote. Points below the diagonal represent situations in which the error rate for sharing is lower than the error rate for not sharing. Results are given for each dataset with features randomly distributed evenly among 5 entities, and with features randomly distributed so that each entity has about 3 features. Seven datasets given in Table 4.2 were used to generate four points each.

Dataset Examples × Input Features	No. of Entities	No Sharing	Majority Vote	Sharing
Cleveland Heart	5	0.1310	0.2625	0.1920
297 × 13	4	0.1293	0.1822	0.1780
Ionosphere	5	0.1029	0.0541	0.0856
351 × 34	11	0.1481	0.0769	0.0970
WDBC	5	0.0643	0.0510	0.0352
569 × 30	10	0.1518	0.0511	0.0352
Arrhythmia	5	0.2933	0.2232	0.2898
452 × 279	93	0.3795	0.4135	0.2544
Pima Indians	5	0.3184	0.3088	0.2355
768 × 8	2	0.2566	0.3127	0.2264
Bupa Liver	5	0.4941	0.4204	0.2730
345 × 6	2	0.4559	0.3795	0.2934
German Credit	5	0.2460	0.2930	0.2410
1000 × 24	8	0.2475	0.3000	0.2480

Table 4.2 Comparison of error rates for entities not sharing and sharing their datasets using a 1-norm nonlinear Gaussian SVM.

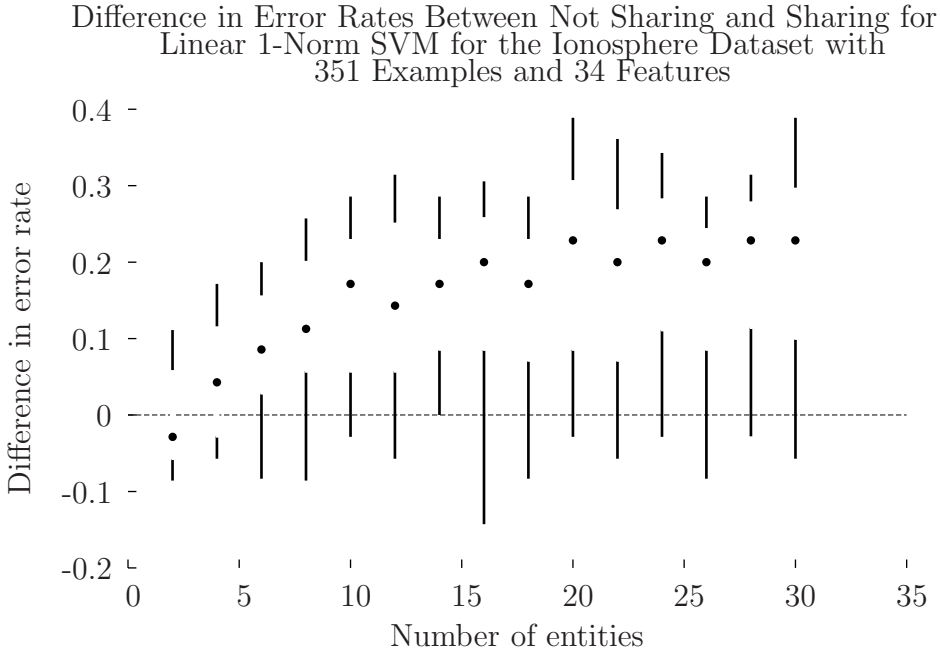


Figure 4.5 Box-and-whisker (median and interquartile) plot showing the improvement in error rate of *linear* kernel PPSVM as the number of entities increases from 2 to 30.

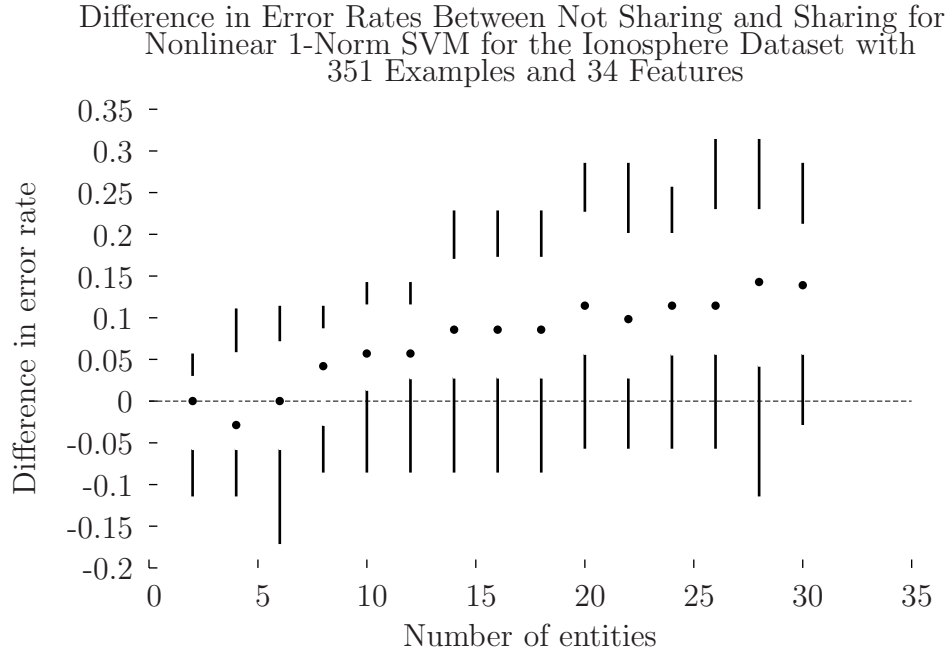


Figure 4.6 Box-and-whisker (median and interquartile) plot showing the improvement in error rate of *Gaussian* kernel PPSVM as the number of entities increases from 2 to 30.

We further explore the behavior of our approach as the number of entities changes on the Ionosphere dataset. Figure 4.5 shows the difference in misclassification error rates as the number of entities varies according to $\{2, 4, \ldots, 30\}$. For each number of entities, a box-and-whisker plot is given which shows the median (represented by a dot), interquartile range (the space between the dot and the vertical lines), and data range (the vertical lines) with outliers removed for each fold of ten-fold cross validation for all the entities. Note that as the number of entities increases, our PPSVM approach tends to have better median error rates, and that more of the observations favor PPSVM as more of the data lies above the $y = 0$ axis. The results shown in Figure 4.5 indicate that as each entity has fewer features, greater improvement due to using PPSVM would be expected, and also that some improvement is more likely to be observed. Figure 4.6 shows similar results using a Gaussian kernel. Each experiment was tuned according to the procedures for linear and nonlinear kernels described above.

We now consider privacy-preserving classification for horizontally partitioned data.

4.4 Privacy-preserving linear classifier for horizontally partitioned data

The dataset that we wish to obtain a classifier for consists of m points in R^n represented by the m rows of the matrix $A \in R^{m \times n}$. Each row contains values for n features associated with a specific individual, while each column contains m values of a specific feature associated with m different individuals. The matrix A is divided into q blocks of $m_1, m_2, \ldots m_q$ rows with $m_1 + m_2 + \ldots + m_q = m$, and each block of rows "owned" by an entity that is unwilling to make it public or share it with others. Furthermore, each row of A is labeled as belonging to the class $+1$ or -1 by a corresponding diagonal matrix $D \in R^{m \times m}$ of ± 1's. The linear kernel classifier to be generated based on this data will be a separating plane in R^n: We shall partition our data matrix A into q row blocks $A_1, A_2, \ldots, A_q$ with each row block belonging to one of the q entities and held privately by it and never made public. However, what is made public by each entity i is the matrix product $K(A_i, B')$ which allows the public

calculation of the kernel $K(A, B') \in R^{m \times \bar{m}}$ as follows:

$$\begin{bmatrix} K(A_1, B') \\ K(A_2, B') \\ \vdots \\ K(A_q, B') \end{bmatrix} \tag{4.7}$$

We are now ready to state our algorithm which will provide a classifier for the data without revealing the private blocks of the privately held data blocks $A_1, A_2, \ldots, A_p$. The accuracy of this algorithm will be comparable to that of an SVM using a publicly available kernel $K(A, A')$ instead of merely the blocks $K(A_1, B'), K(A_2, B'), \ldots, K(A_p, B')$ of (4.7) as is the case here. We note that we rely on the associative property of a nonlinear kernel:

$$K\left(\begin{bmatrix} E \\ F \end{bmatrix}, G'\right) = \begin{pmatrix} K(E, G') \\ K(F, G') \end{pmatrix}, \tag{4.8}$$

where $E \in R^{m_1 \times n}$, $F \in R^{m_2 \times n}$, $G \in R^{k \times n}$. It is straightforward to verify that (4.8) is satisfied by both a linear and a Gaussian kernel.

Algorithm 4.4.1 PPSVM Algorithm for Horizontally Partitioned Data

(I) All q entities agree on the same random matrix $B \in R^{\bar{m} \times n}$ with $\bar{m} < n$ for security reasons as justified in the explanation immediately following this algorithm.

(II) All entities make public the class matrix $D_{\ell\ell} = \pm 1$, $\ell = 1, \ldots, m$, for the data matrices A_i, $i = 1, \ldots, m$ that they all hold.

(III) Each entity i makes public its nonlinear kernel $K(A_i, B')$. This does not reveal A_i but allows the public computation of the full nonlinear kernel:

$$K(A, B') = K\left(\begin{bmatrix} A_1 \\ A_2 \\ \vdots \\ A_q \end{bmatrix}, B'\right) = \begin{bmatrix} K(A_1, B') \\ K(A_2, B') \\ \vdots \\ K(A_q, B') \end{bmatrix} \tag{4.9}$$

(IV) A publicly calculated nonlinear classifier $K(x', B)u - \gamma = 0$ is computed by some standard method such as a 1-norm SVM [72, 8]:

$$\begin{array}{rll} \min\limits_{(u,\gamma,y)} & \nu\|y\|_1 + \|u\|_1 & \\ \text{s.t.} & D(K(A, B')u - e\gamma) + y \geq e, & \\ & y \geq 0. & \end{array} \tag{4.10}$$

(V) For each *new* $x \in R^n$ obtained by an entity, that entity privately computes $K(x', B')$ from which a nonlinear classifier is computed as follows:

$$K(x', B')u - \gamma = 0 \tag{4.11}$$

Remark 4.4.2 Note that in the above algorithm no entity i reveals its dataset A_i. This is so because it is impossible to compute unique $m_i n$ numbers constituting the matrix $A_i \in R^{m_i \times n}$ given only the $m_i \bar{m}$ numbers constituting the revealed kernel matrix $K(A_i, B') \in R^{m_i \times \bar{m}}$ with $\bar{m} < n$. However, all entities share the publicly computed nonlinear classifier (4.11) without revealing their individual datasets A_i, $i = 1, \ldots, q$, or any new x that they obtain. Thus, for example, if we wish to compute the r-th row A_{ir} of entity i's data matrix A_i from the given matrix $P_i = K(A_i, B') \in R^{m_i \times \bar{m}}$, we need to solve the $\bar{m}$ *nonlinear* equations $K(B, {A_{ir}}') = {P_{ir}}'$ for the n components of $A_{ir} \in R^n$. Because $n > \bar{m}$, this would in general generate a nonlinear surface in R^n containing an infinite number of solutions which makes it impossible to determine A_{ir} uniquely. We make this statement more precise for a linear kernel $K(A, B') = AB'$ by first showing that at least an exponential number of matrices A_i satisfy $A_i B' = P_i$ for a given B and P_i when $\bar{m} < n$. We then show that the infinite number of matrices that lie in the affine hull of these matrices also satisfy $A_i B' = P_i$. This obviously precludes the possibility of determining the dataset A_i held by entity i given only $A_i B'$.

Proposition 4.4.3 Given the matrix product ${P_i}' = A_i B' \in R^{m_i \times \bar{m}}$ where $A_i \in R^{m_i \times n}$ is unknown and B is a known matrix in $R^{\bar{m} \times n}$ with $\bar{m} < n$, there are an infinite number of solutions, including:

$$\binom{n}{\bar{m}}^{m_i} = \left(\frac{n!}{(n-\bar{m})!\bar{m}!}\right)^{m_i}, \tag{4.12}$$

possible solutions $A_i \in R^{m_i \times n}$ to the equation $BA_i{}' = P_i$. Furthermore, the infinite number of matrices in the affine hull of these $\binom{n}{\bar{m}}^{m_i}$ matrices also satisfy $BA_i{}' = P_i$.

Proof Consider the problem of solving for row r of A_i, that is $A_{ir} \in R^n$, $r \in \{1, \ldots, m_i\}$, from the r-th equation of $BA_i{}' = P_i$:

$$BA_{ir}{}' = P_{ir}. \tag{4.13}$$

Since $\bar{m} < n$, and B is a random matrix, it follows by [23] that each of the $\binom{n}{\bar{m}}$ random $\bar{m} \times \bar{m}$ square submatrices of B are of full rank and hence nonsingular. Consequently, Equation (4.13) has $\binom{n}{\bar{m}}$ solutions for each row r of A_i, that is A_{ir}, $r \in \{1, \ldots, m_i\}$. Hence there are $\binom{n}{\bar{m}}^{m_i} = \left(\frac{n!}{(n-\bar{m})!\bar{m}!}\right)^{m_i}$ solutions for the m_i rows of A_i.

To prove the last statement of the proposition, we note that if each of k matrices $A_i^1, \ldots, A_i^k$ solve $BA_i{}' = P_i$ for a given B and P_i, then so does $A_i = \sum_{j=1}^{j=k} \lambda^j A_i^j$ for $\sum_{j=1}^{j=k} \lambda^j = 1$. Hence any matrix in the affine hull of $A_i^1, \ldots, A_i^k$, $\{H \mid H = \sum_{j=1}^{j=k} \lambda^j A_i^j, \sum_{j=1}^{j=k} \lambda^j = 1\}$ also satisfies (4.13). □

For the specific case of $\bar{m} = n - 1$, which is used for our numerical results, we have that:

$$\binom{n}{\bar{m}}^{m_i} = \left(\frac{n!}{(n-\bar{m})!\bar{m}!}\right)^{m_i} = (n)^{m_i}. \tag{4.14}$$

This translates to $(30)^{20}$ for the typical case of $n = 30$, $\bar{m} = 29$ and $m_i = 20$.

We turn now to our computational results for horizontally partitioned data.

4.5 Computational results for horizontally partitioned data

We illustrate the effectiveness of our proposed privacy preserving SVM (PPSVM) by demonstrating that entities using our approach can obtain classifiers with lower misclassification error than classifiers obtained using only the examples of each entity alone.

All experiments were run using both a linear kernel and the commonly used Gaussian kernel described in Section 1.1. In all of our results, $\bar{A}$ consisted of ten percent of the rows of A randomly selected, while B was a completely random matrix with the same number of columns as A. The number of rows of B was set to the minimum of $n-1$ and the number of rows of $\bar{A}$, where n is the number of features in the dataset. Thus, we ensure that the conditions discussed in the previous sections hold in order to guarantee the private data A_i cannot be recovered from $K(A_i, B')$. Each entry of B was selected independently from a uniform distribution on the interval $[0, 1]$. All datasets were normalized so that each feature was between zero and one. This normalization can be carried out if the entities disclose only the maximum and minimum of each feature in their datasets. When computing ten-fold cross validation, we first divided the data into folds and set up the training and testing sets in the usual way. Then each entity's dataset was formed from the training set of each fold. The accuracies of all classifiers were computed on the testing set.

We investigate the benefit of using our PPSVM approach instead of using only the examples available to each entity using seven datasets from the UCI repository [96]. To simulate a situation in which each entity has only a subset of the available examples, we randomly distribute the examples among the entities such that each entity receives about the same number of examples. We chose arbitrarily to perform experiments using however many entities were needed so that each entity received about 25 examples. To save time, we computed results only for three entities, though when sharing we assumed the entire dataset was shared.

Figure 4.7 shows results comparing the ten-fold cross validation misclassification error of our linear kernel PPSVM with the average misclassification error of the 1-norm SVM classifiers learned using only the examples available to each of three entities. Points below the 45 degree line represent experiments in which our PPSVM has lower error rate than the average error rate of the classifiers learned with only each entity's subset of the examples. This indicates that the entities can expect improved performance using PPSVM instead of going it alone. The results shown in Figure 4.7 are detailed in Table 4.3. We note

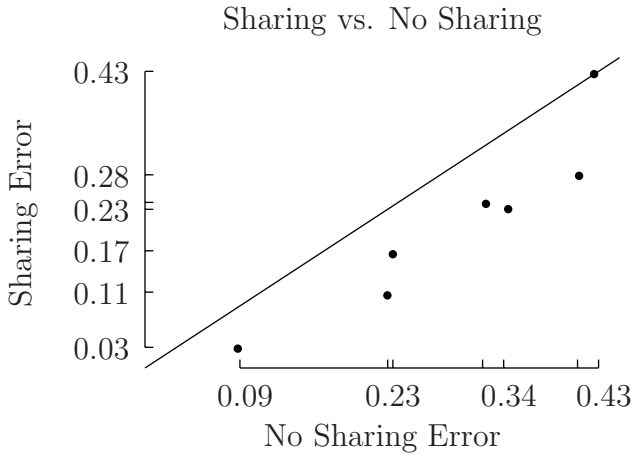


Figure 4.7 Error rate comparison for the seven datasets of Table 4.3 of a 1-norm linear SVM sharing A_iB' data for each entity versus a 1-norm linear SVM using only the examples A_i of each entity. Points below the diagonal represent situations in which the error rate for sharing is better than the error rate for not sharing. Results are given for each dataset with examples randomly distributed so that each entity has about 25 examples.

Dataset Examples × Input Features	No Sharing Error	Sharing Error
Cleveland Heart 297 × 13	0.2349	0.1652
Ionosphere 351 × 34	0.2298	0.1054
WDBC 569 × 30	0.0879	0.0281
Arrhythmia 452 × 279	0.4116	0.2788
Pima Indians 768 × 8	0.3443	0.2303
Bupa Liver 345 × 6	0.4259	0.4263
German Credit 1000 × 24	0.3233	0.2380

Table 4.3 Comparison of error rates for entities not sharing and sharing their datasets using a 1-norm linear SVM.

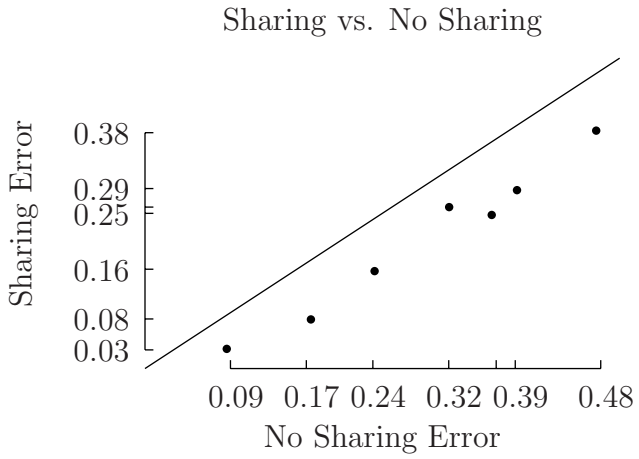


Figure 4.8 Error rate comparison for the seven datasets of Table 4.4 of a 1-norm nonlinear SVM sharing $K(A_i, B')$ data for each entity versus a 1-norm nonlinear SVM using only the examples A_i of each entity. Points below the diagonal represent situations in which the error rate for sharing is better than the error rate for not sharing.

that PPSVM obtains classifiers with lower error than the average of the classifiers using only each entity's examples in six of the seven experiments. The parameter ν was selected from $\{10^i | i = -7, \ldots, 7\}$ for each dataset was selected using ten-fold cross validation on the training set when data was shared, and leave-one-out cross validation on the training set when data was not shared. We used leave-one-out cross validation when data was not shared because each entity only had about 25 examples in its training set.

Figure 4.8 shows similar results for experiments using Gaussian kernels, with details in Table 4.4. We used the same datasets as for the experiments described above. To save time, we used the tuning strategy from [46], as described in Section 4.1, with the exception that the initial range of $\log_{10} \mu$ was set independently for each entity using only that entity's examples. We chose to use leave-one-out cross validation to evaluate each (ν, μ) pair when not sharing data because only about 25 examples were available to each entity in that situation.

Dataset Examples × Input Features	No Sharing Error	Sharing Error
Cleveland Heart 297 × 13	0.2418	0.1567
Ionosphere 351 × 34	0.1747	0.0790
WDBC 569 × 30	0.0861	0.0316
Arrhythmia 452 × 279	0.3919	0.2873
Pima Indians 768 × 8	0.3203	0.2599
Bupa Liver 345 × 6	0.4752	0.3832
German Credit 1000 × 24	0.3653	0.2473

Table 4.4 Comparison of error rates for entities not sharing and sharing their datasets using a 1-norm nonlinear Gaussian SVM.

Chapter 5

Feature-Selecting k-Median Algorithm

The k-median clustering algorithm for unlabeled data can be considered as a constrained optimization problem. This problem has a concave objective function, and an algorithm for finding a local solution in a finite number of steps is given by Bradley et al. [9]. For problems with large numbers of features, such as those arising in text or gene expression data, there may be many redundant features. In this chapter, a simple but fundamental modification to the algorithm in [9] is used to select features while maintaining a clustering similar to that using the entire set of features [82].

5.1 Feature-selecting k-median (FSKM) theory and algorithm

The k-median clustering algorithm [9] consists of two basic steps. Given k initial or intermediate cluster centers, the *first* step consists of assigning each point to the closest cluster center using the 1-norm distance. The *second* step consists of generating k new cluster centers, each being the median of each cluster. It is the second step that we shall modify, in order to remove possibly irrelevant input space features from the problem, as follows. Since the median of a cluster is the point (or set of points) that minimizes the sum of the 1-norm distances to all the points in the cluster, we shall perturb this minimization problem by adding to its objective function a weighted term with weight ν consisting of the 1-norm distance to global median of zero for the entire dataset. As the weight ν gets sufficiently large, all the features will become zero and are eliminated from the problem. Conversely, if $\nu = 0$, then we have the ordinary k-median algorithm. We derive now the

optimality condition for minimizing the nondifferentiable objective function for the perturbed objective function for this step of the modified k-median algorithm.

Let the given dataset, consisting of m points in R^n, be represented by the matrix $A \in R^{m\times n}$. We shall assume without loss of generality that a median of the m rows of A is $0 \in R^n$. Assume further, that k clusters have been generated by the k-median algorithm and are represented by the k submatrices of A:

$$A^\ell \in R^{m(\ell)\times n}, A^\ell_i = A_{i\in J(\ell)},\ \ell = 1,\ldots,k, \tag{5.1}$$

where $J(\ell) \subset \{1,\ldots,m\},\ \ell = 1,\ldots,k$, is a partition of $\{1,\ldots,m\}$. The k perturbed optimization problems that need to be solved at this second step of the modified k-median algorithm consist of the following k unconstrained minimization problems. Find k cluster centers $c^\ell \in R^n,\ \ell = 1,\ldots,k$, with one or more components being zero, depending on the size of ν. Each $c^\ell \in R^n,\ \ell = 1,\ldots,k$ is a solution of:

$$\min_{c\in R^n} \sum_{i\in J(\ell)} \|A_i - c\|_1 + \nu\|c\|_1,\ \ell = 1,\ldots,k. \tag{5.2}$$

Since each of these problems is *separable* in the components $c_j,\ j = 1,\ldots,n$ of c, we can consider the following *1-dimensional* minimization problem for each component c_j, which we denote for simplicity by $c \in R^1$, and for $a_i := A_{ij},\ i \in J(\ell)$ for a fixed $j \in \{1,\ldots,n\}$ as follows:

$$\min_{c\in R^1} \sum_{i\in J(\ell)} |c - a_i| + \nu|c|,\ \ell = 1,\ldots,k. \tag{5.3}$$

Here $A_{J(\ell)}$ denotes the subset of the rows of A that are in cluster ℓ. Setting the subgradient (see Equations (1.1)-(1.3)) of the objective function of (5.3) equal to zero gives the following necessary and sufficient optimality condition for a fixed $j \in \{1,\ldots,n\}$ and for a fixed cluster

$\ell \in \{1, \ldots, k\}$:

$$\begin{aligned} & card\{i|c > a_{i\in J(\ell)}\} - card\{i|c < a_{i\in J(\ell)}\} \\ & \\ & +[-1,1] \cdot card\{i|c = a_{i\in J(\ell)}\} \\ & \\ & +\nu \cdot \left\{ \begin{array}{ll} -1 & \text{if } c < 0 \\ [-1,1] & \text{if } c = 0 \\ +1 & \text{if } c > 0 \end{array} \right\} = 0. \end{aligned} \tag{5.4}$$

Henceforth, $[-1, 1]$ denotes some point in the closed interval $\{x| - 1 \leq x \leq 1\}$. Thus, for a cluster center c to be zero, for a fixed $j \in \{1, \ldots, n\}$, $a_i := A_{ij}$, and for a fixed cluster $\ell \in \{1, \ldots, k\}$, we need to have:

$$\begin{aligned} & card\{i|0 > a_{i\in J(\ell)}\} - card\{i|0 < a_{i\in J(\ell)}\} \\ & +[-1,1] card\{i|0 = a_{i\in J(\ell)}\} + [-1,1] \cdot \nu = 0. \end{aligned} \tag{5.5}$$

Simplifying this expression by replacing the first $[-1, 1]$ by the zero subgradient and solving for ν, we have that:

$$\nu = \frac{card\{i|0 < a_{i\in J(\ell)}\} - card\{i|0 > a_{i\in J(\ell)}\}}{[-1,1]}, \tag{5.6}$$

which is satisfied if we set:

$$\nu \geq |card\{i|0 < a_{i\in J(\ell)}\} - card\{i|0 > a_{i\in J(\ell)}\}|. \tag{5.7}$$

Hence we can state the following result based on the above analysis.

Proposition 5.1.1 Cluster Center with Selected Features A solution c to the perturbed cluster center optimization problem (5.2) has zero components $c_j = 0$ for each $j \in \{1, \ldots, n\}$, such that:

$$\nu \geq |card\{i|0 < A_{i\in J(\ell),j}\} - card\{i|0 > A_{i\in J(\ell),j}\}|. \tag{5.8}$$

It follows that if we set ν large enough one or more input space features are killed. Hence we can gradually increase ν from zero and systematically kill at least one feature at a time. This property suggests the following algorithm.

Algorithm 5.1.2 FSKM: Feature-Selecting k-Median Algorithm

1. Shift the dataset $A \in R^{m \times n}$ such that $0 \in R^n$ is its median.

(i) Use the k-median the algorithm to cluster into k clusters.

(ii) For each input space component $j \in \{1, \ldots, n\}$ and for each cluster $A_{J(\ell)}$, $\ell \in \{1, \ldots, k\}$ compute:

$$\nu_j^\ell = |card\{i|0 < A_{i \in J(\ell),j}\} - card\{i|0 > A_{i \in J(\ell),j}\}|. \tag{5.9}$$

(iii) Delete feature(s) $\bar{j}$ by deleting column(s) $A_{\cdot \bar{j}}$ for which:

$$\nu_{\bar{j}} = \min_{1 \le j \le n} \max_{1 \le \ell \le k} \nu_j^\ell. \tag{5.10}$$

(iv) Stop if A has no columns remaining, else let $A = \bar{A} \in R^{m \times \bar{n}}$, $n = \bar{n}$, where $\bar{A}$ is the matrix with reduced columns.

(v) Go to (i).

We note that Step (iii) in the FSKM Algorithm above determines precisely which input space feature(s) will be deleted next, based on successively increasing values of the perturbation parameter ν. Thus, formula (5.10) of Step (iii) sets apart our algorithm from a lengthy greedy n-choose-1 approach that systematically deletes one feature at a time. Such a greedy approach chooses to delete the feature which minimizes the clustering error for the remaining features. This procedure is repeated n times until one feature is left. Hence, instead of n applications of the k-median algorithm needed by FSKM, a greedy approach would need $\frac{n(n+1)}{2}$ applications of the k-median algorithm.

We turn now to our computational results to show the effectiveness of the FSKM Algorithm.

5.2 Computational results

To illustrate the performance of our algorithm, we tested it on five publicly available datasets, four from the UCI Machine Learning Repository [96] and one available at [100].

We ran Algorithm 5.1.2 30 times on each dataset, and we report average results. If Algorithm 5.1.2 produced multiple candidate features for elimination in Step (iii), then only one randomly chosen feature from this set was eliminated. The k-median algorithm was initialized with centers chosen by the following procedure which is similar to that of [9]. For each feature, $4k$ bins of equal size were created. The data was sorted into these bins, and the k initial centers were chosen by taking the midpoint of the k most populous bins for each feature. Consider using this procedure for $k = 2$. One initial center will have each coordinate be the midpoint of the most populous bin for the corresponding feature, while the other initial center will have each coordinate be the midpoint of the second most populous bin for the corresponding feature. The decision to use $4k$ bins was made arbitrarily and not adjusted while developing the algorithm or performing the experiments.

Figure 5.1 gives results for the Wine dataset [96]. The two curves shown are the classification error and the clustering error. The *classification error* curve, marked by squares, is computed by labeling members of each cluster with the majority label of the cluster, where the labels are the actual class labels from the dataset. These class labels are used only in generating the classification error curve and *not* in obtaining the clusters. The error is the number of incorrectly classified examples divided by the number of examples in the dataset. The entire dataset is used both for the clustering and evaluation of the error. No data is left out. The *clustering error* curve, marked by circles, is computed by accepting the clusters produced by k-median on the full-featured dataset as the gold standard labeling, and then using the following procedure for computing the error without using any class labels, as would be the case for unlabeled data clustering. For each reduced dataset, members of each cluster are marked with the majority gold standard label of that cluster. The gold standard labels are used only in generating the clustering error curve and *not* in obtaining the clusters. Note that the clustering error on the full-featured dataset is always zero by definition. Error bars show one sample standard deviation above and below each point. Total time to generate the error curves which entails running the k-median algorithm 390 times and plotting the error

Figure 5.1 Error curves for the 3-class Wine dataset with 178 points in 13-dimensional space are plotted as a function of the number features selected by FSKM. The average range of ν computed by (5.10) was from 42 to 55. Note that the low variance between runs on this dataset makes the error bars essentially invisible.

curves, all within MATLAB [94], took 205.1 seconds on a 650MHz, 256MB RAM desktop machine running Red Hat Linux, Version 9.0.

The curves in Figure 5.1 show that the clustering error curve increases slightly as the input space dimensionality is reduced from 13 features to 4 features, and then increases very sharply as the data dimensionality is further reduced from 4 features down to 2 features. The classification error curve decreases slightly as the data dimensionality is reduced from 13 to 4 features, and then increases similarly to the clustering error curve as the number of features is reduced from 4 to 2. As the number of features is reduced from 2 to 1, both curves decrease. The number of features can be reduced to 4 from 13 while keeping the clustering error less than 4% and decreasing the classification error by 0.56 percentage points.

One key observation to make about Figure 5.1 and subsequent figures is the following. Since the real-world application of FSKM is to unlabeled data, we can only generate a *clustering* error curve similar to that of Figure 5.1. This curve will help us decide on the magnitude of error we wish to tolerate, which determines how many and which features to keep. The validity of such a procedure is based on the parallelism between the clustering error curve based on unlabeled data, and the classification error curve based on the labels of the datasets in the current experiments.

The results of our algorithm on the Votes dataset [96] are in Figure 5.2. The procedure for generating the curves is exactly the same as described above for the Wine dataset. Note that both the classification and clustering error increase slightly as the number of features is reduced from 16 to 12. Then the classification error increases briefly and then tends to decrease while the clustering error tends to increase slightly as the number of features is reduced from 12 to 3. Finally, both the classification and the clustering error increase more sharply as the number of features is reduced from 3 to 1. After reducing the number of features down to 3, the clustering error is less than 10%, and the classification error has only increased by 1.84 percentage points.

Results for the WDBC dataset [96] are in Figure 5.3. For this dataset, the classification error does not increase as much as the clustering error as the number of features is reduced

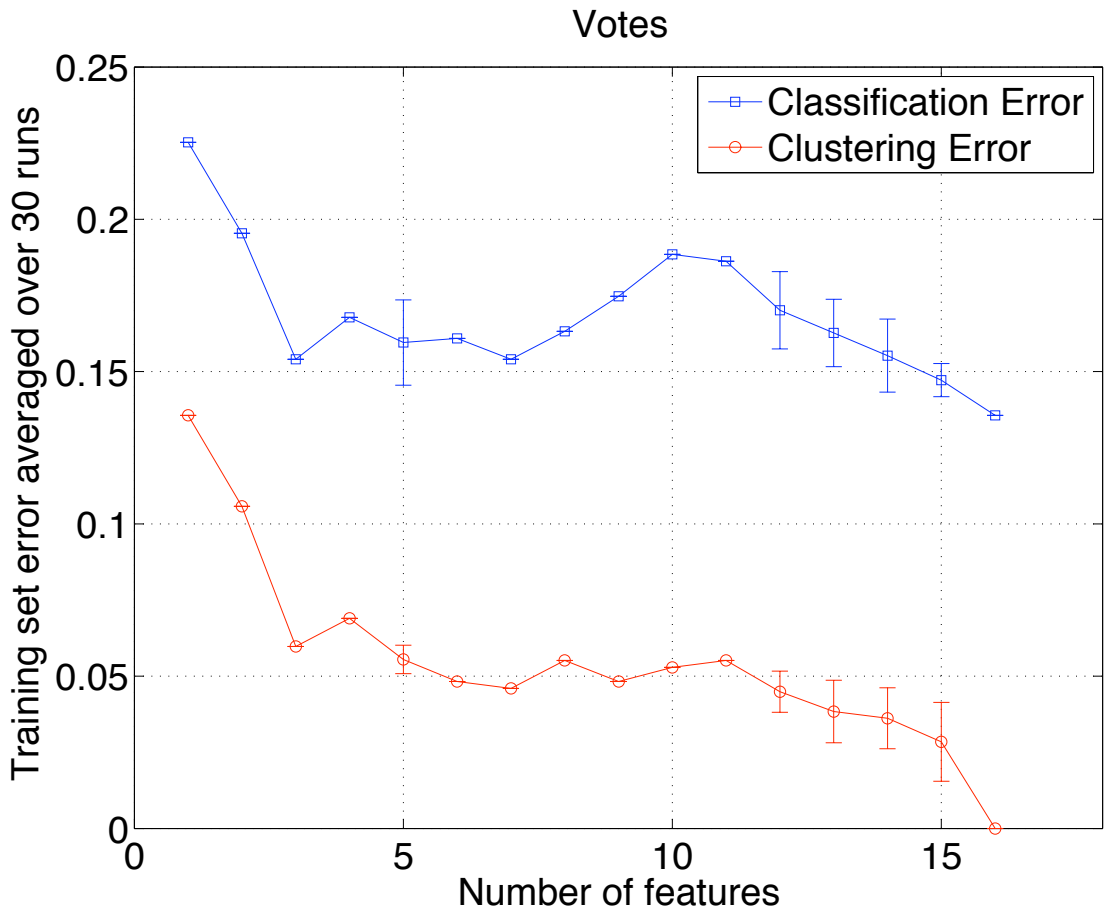


Figure 5.2 Error curves and variance bars for the 2-class Votes dataset with 435 points in 16-dimensional space are plotted as a function of the number features selected by FSKM. The average range of ν computed by (5.10) was from 0 to 192.

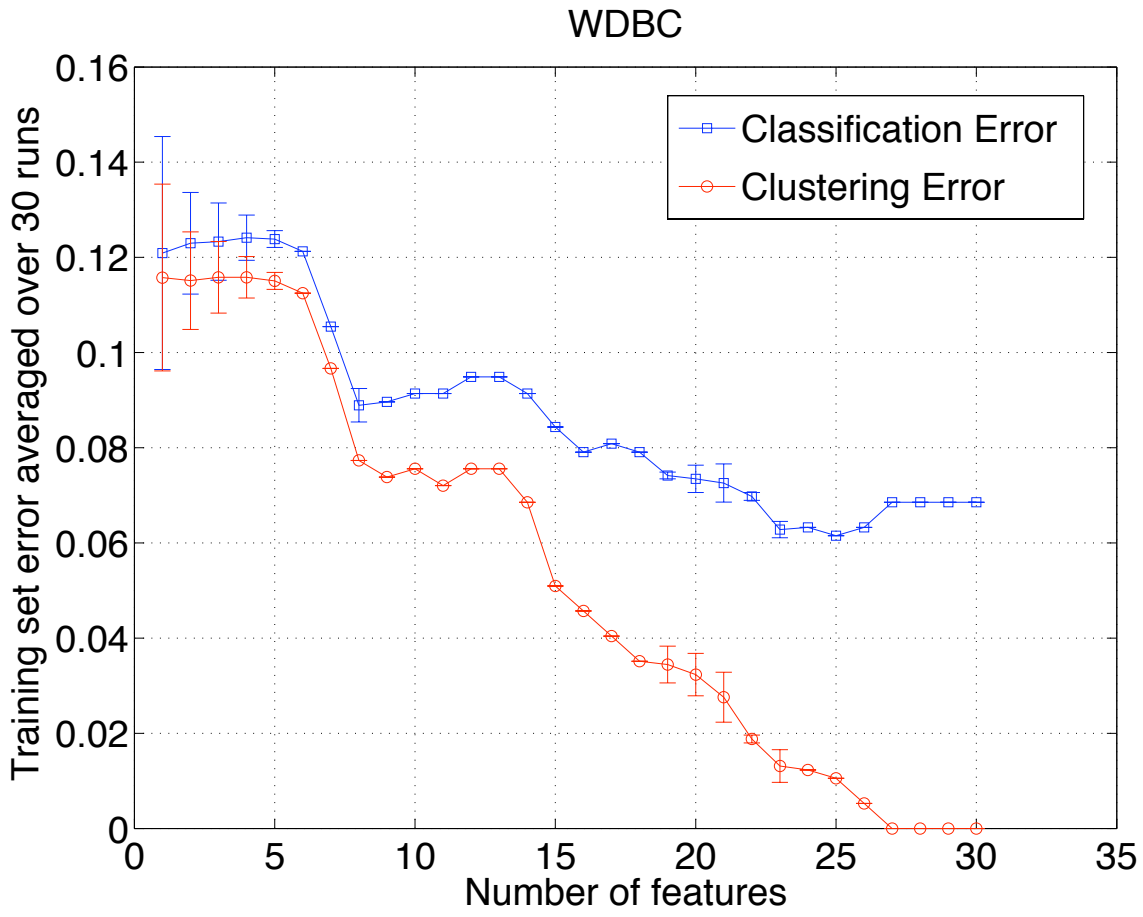


Figure 5.3 Error curves and variance bars for the 2-class WDBC dataset with 569 points in 30-dimensional space are plotted as a function of the number features selected by FSKM. The average range of ν computed by (5.10) was from 188 to 284.

from 30 to 14. At that point, the two curves mirror one another closely as the number of features is reduced further. Note that reducing the number of features from 30 to 27 causes no change in clustering or classification error. Reducing the number of features to 7 keeps the clustering error less than 10%, while increasing the classification error by 3.69 percentage points.

Figure 5.4 shows the results for the Star/Galaxy-Bright dataset [100]. For this dataset, the classification and clustering error curves behave differently. However, note that the clustering error curve tends to increase only slightly as the number of features decreases. This behavior is what we want. Overall, the classification error curve decreases noticeably until 6 features remain and then begins to increase, indicating that some of the features may be obstructing the classification task. The problem can be reduced to 4 features and still keep the clustering error under 10% while decreasing the classification error by 1.42 percentage points from the initial error using 14 features.

Results for the Cleveland Heart dataset [96] are in Figure 5.5. Note that although the increase in clustering error when reducing from 13 features to 9 features is very large, subsequent increases are not so severe. In addition, the classification error curve behaves similarly to the clustering error curve in the sense that both curves have the greatest increase going from 13 features to 9 features. Using FSKM to remove 5 features causes the clustering error to be less than 17%, and increases classification error by 7.74 percentage points.

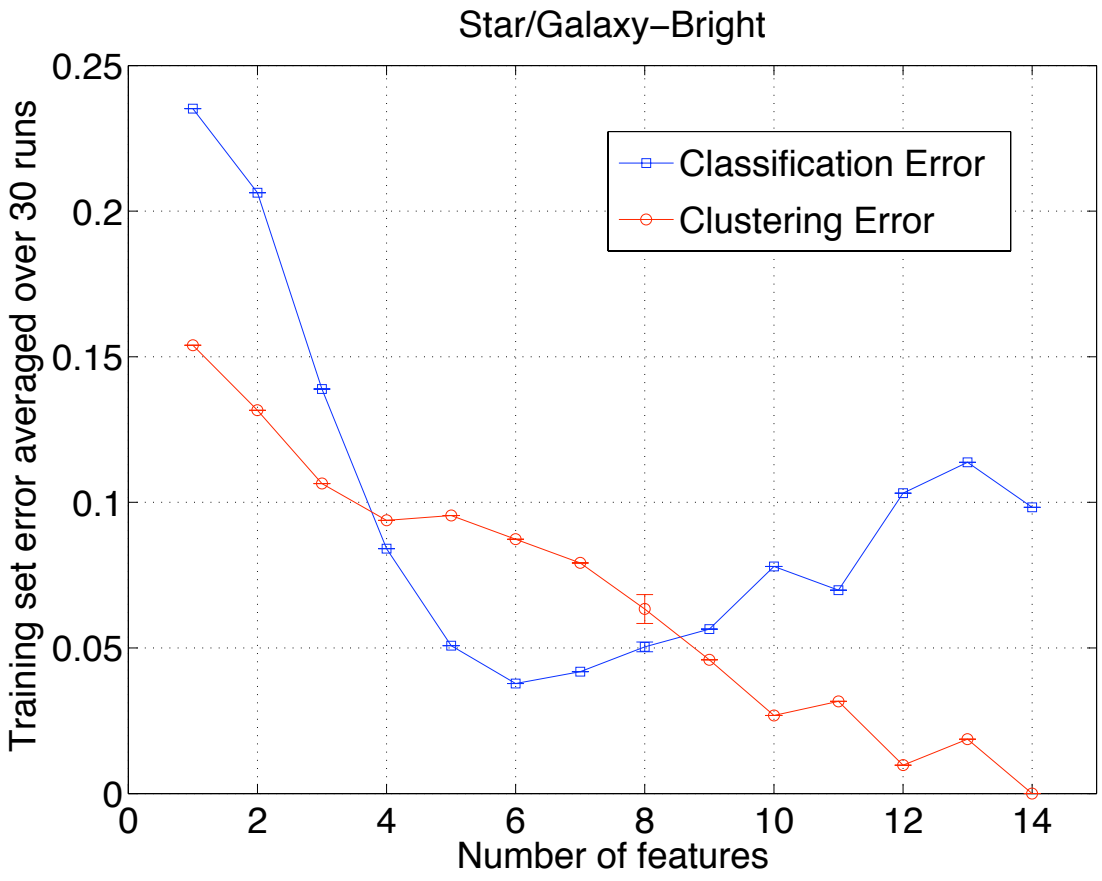


Figure 5.4 Error curves and variance bars for the 2-class Star/Galaxy-Bright dataset with 2462 points in 14-dimensional space are plotted as a function of the number features selected by FSKM. The average range of ν computed by (5.10) was from 658 to 1185.

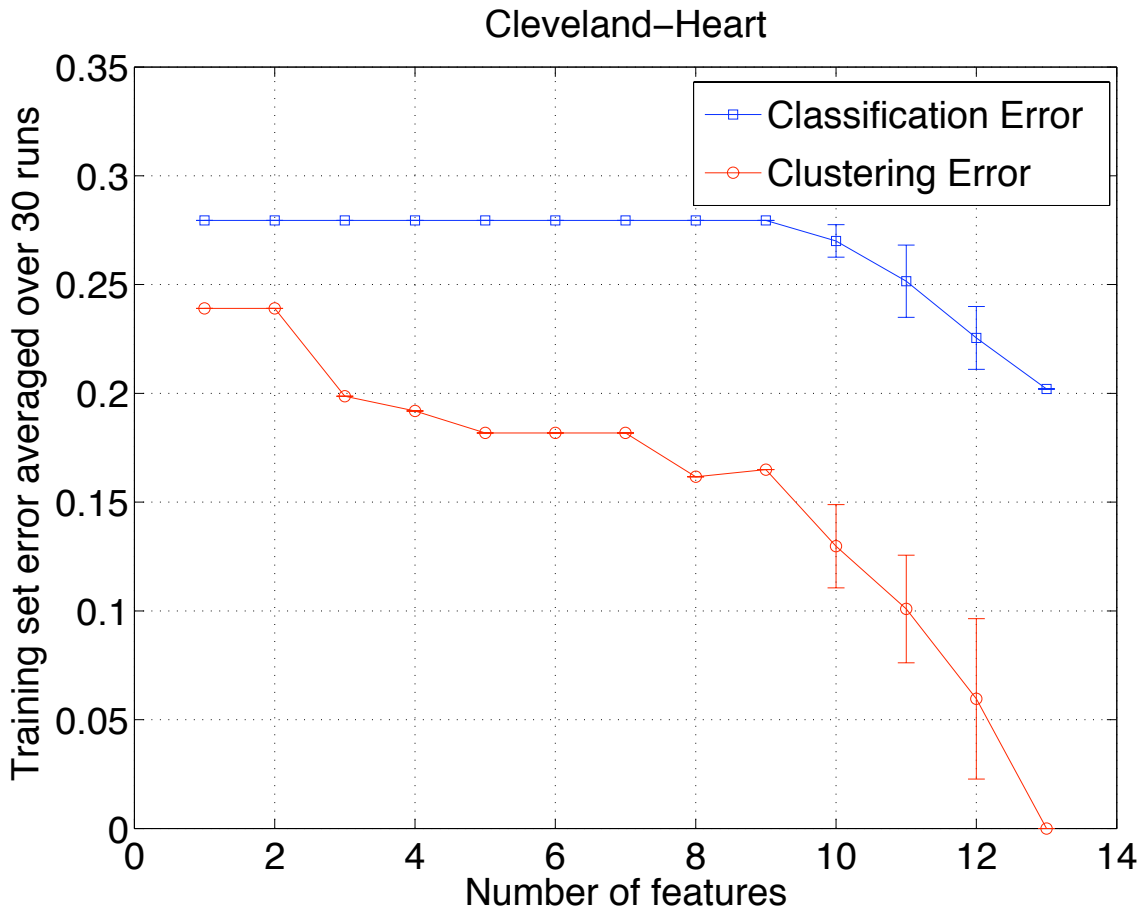


Figure 5.5 Error curves and variance bars for the 2-class Cleveland Heart dataset with 297 points in 13-dimensional space are plotted as a function of the number features selected by FSKM. The average range of ν computed by (5.10) was from 0 to 113.

Chapter 6

Feature Selection for Nonlinear Kernel Support Vector Machines

Feature selection is a fairly straightforward procedure for *linear* support vector machine (SVM) classifiers. For example, a 1-norm support vector machine linear classifier obtained by either linear programming or concave minimization will easily reduce features [8]. However, when similar techniques are applied to *nonlinear* SVM classifiers, the resulting reduction is *not* in the number of input space features but in the number of kernel functions needed to generate the nonlinear classifier [30]. This may be interpreted as a reduction in the dimensionality of the higher dimensional transformed space, but does not result in any reduction of input space features. It is precisely this reduction that we are after in this chapter, namely, a reduced number of input space features that we need to input into a nonlinear SVM classifier. We shall achieve this by replacing the usual nonlinear kernel $K(A, A')$, where A is the $m \times n$ data matrix, by $K(AE, EA')$ where E is an $n \times n$ diagonal matrix of ones and zeros. The proposed algorithm alternates between computing the continuous variables (u, γ) of the nonlinear kernel classifier $K(x'E, EA')u - \gamma = 0$, by using linear programming, and the integer diagonal matrix E of ones and zeros by successive minimization sweeps through its components. The algorithm generates a decreasing sequence of objective function values that converge to a local solution that minimizes the usual data fit and the number of kernel functions used while *also* minimizing the number of features used [85]. A possibly related result, justifying the use of reduced features, is that of random projection on a subspace of features for Gaussian mixtures which states that data

from a mixture of k Gaussians can be projected into $O(log\ k)$ dimensions while retaining approximate separability of the clusters [14].

There has been considerable recent interest in feature selection for SVMs. Weston et al. propose reducing features based on minimizing generalization bounds via a gradient approach [121]. In [26], Frölich and Zell introduce an incremental approach based on ranking features by their effect on the margin. An approach based on a Bayesian interpretation of SVMs is presented by Gold et al. [38], and an approach based on smoothing spline ANOVA kernels is proposed by Zhang [127]. In [41], Guyon et al. use a wrapper method designed for SVMs. Another possibility is to use a filter method such as Relief [105] in conjunction with an SVM. None of these approaches utilize the straightforward and easily implementable mixed-integer programming formulation proposed here.

6.1 Reduced Feature Support Vector Machine (RFSVM) Formulation and Algorithm

We consider a given set of m points in the n-dimensional input feature space R^n represented by the matrix $A \in R^{m\times n}$. Each point represented by A_i, $i = 1, \ldots, m$, belongs to class +1 or class -1 depending on whether D_{ii} is 1 or -1, where $D \in R^{m\times m}$ is a given diagonal matrix of plus or minus ones. We shall attempt to discriminate between the classes +1 and -1 by a nonlinear classifier induced by a *completely arbitrary* kernel $K(A, A')$ and parameters $u \in R^m$ and $\gamma \in R$, by using (1.14). Recall that minimizing $\|u\|_1$ leads to a minimal number of kernel functions used in the classifier (1.4) by zeroing components of the variable u [30]. However, our primary concern here is to use as few components of the input space vector x as possible in the nonlinear classifier (1.4). We proceed to do that now as follows.

We introduce a diagonal matrix $E \in R^{n\times n}$ with ones or zeros on its diagonal. The zeros correspond to suppressed input space features and the ones correspond to features utilized by the nonlinear classifier (1.4) which we modify as follows:

$$K(x'E, EA')u - \gamma = 0. \tag{6.1}$$

In turn, the linear program (1.14) becomes the following mixed-integer nonlinear program:

$$\begin{array}{rl} \min\limits_{u,\gamma,y,s,E} & \nu e'y + e's + \sigma e'Ee \\ \text{s.t.} & D(K(AE, EA')u - e\gamma) + y \;\geq\; e, \\ & -s \;\leq\; u \;\leq\; s, \\ & y \;\geq\; 0, \\ & E \;=\; diag(1 \text{ or } 0), \end{array} \tag{6.2}$$

where σ is a positive parameter that weights the feature suppression term $e'Ee = \sum_{i=1}^{n} E_{ii}$. Mixed-integer programs are basically NP-hard. However, we can easily obtain a local solution by fixing E and solving the resulting linear program (6.2) for (u, γ, y, s), then fixing (u, γ, y, s) and sweeping through the components of E altering them successively only if such alteration decreases the objective function. Repeating this process leads to the following algorithm which, in addition to suppressing input space features, suppresses components of the variable u because of the 1-norm term in the objective function and hence utilizes a minimal number of kernel function components $K(AE, (EA')_{\cdot j})$, $j = 1, \ldots, m$.

We state our algorithm now. More implementation details are given in Section 6.2.

Algorithm 6.1.1 Reduced Feature SVM (RFSVM) Algorithm

(1) Pick a random $E = diag(1 \text{ or } \ 0)$ with cardinality of E inversely proportional to σ. Pick a fixed integer k, typically very large, for the number of sweeps through E, and a stopping tolerance tol, typically $1e-6$.

(2) Solve the linear program (6.2) for a fixed E and denote its solution by (u, γ, y, s).

(3) For $\ell = 1, \ldots, kn$ and $j = 1 + (\ell - 1) \text{mod } n$:

  (a) Replace E_{jj} by 1 if it is 0 and by 0 if it is 1.

  (b) Compute:

$$f(E) = \nu e'(e - D(K(AE, EA')u - e\gamma))_+ + \sigma e'Ee,$$

before and after changing E_{jj}.

(c) Keep the new E_{jj} only if $f(E)$ decreases by more than tol. Else undo the change in E_{jj}. Go to (a) if $j < n$.

(d) Go to (4) if the total decrease in $f(E)$ is less than or equal to *tol* in the last n steps.

(4) Solve the linear program (6.2) for a fixed E and denote its solution by (u, γ, y, s). Stop if objective function decrease of (6.2) is less than *tol*.

(5) Go to (3).

Remark 6.1.2 We note that $f(E)$ in the RFSVM algorithm is equivalent to $\nu e'y + \sigma e'Ee$ when y takes on its optimal value generated by the first and the next-to-the-last sets of constraints of (6.2). Note that $f(E)$ still depends on E even for the case when $\sigma = 0$.

We establish now convergence of the RFSVM algorithm for $tol = 0$, however computationally we use $tol = 1e - 6$.

Proposition 6.1.3 RFSVM Convergence

For $tol = 0$, the nonnegative nonincreasing values of the sequence of objective function values $\{\nu e'y^r + e's^r + \sigma e'E^r e\}_{r=1}^{r=\infty}$, where the superscript r denotes iteration number of step (4) of Algorithm 6.1.1, converge to $(\nu e'\bar{y} + e's + \sigma e'\bar{E}e)$ where $(\bar{u}, \bar{\gamma}, \bar{y}, \bar{s}, \bar{E})$ is any accumulation point of the sequence of iterates $\{u^r, \gamma^r, y^r, s^r, E^r\}$ generated by Algorithm 6.1.1. The point $(\bar{u}, \bar{\gamma}, \bar{y}, \bar{s}, \bar{E})$ has the following local minimum property:

$$\begin{array}{rcl} (\nu e'\bar{y} + e's + \sigma e'\bar{E}e) = \min\limits_{u,\gamma,y,s} & \nu e'y + e's + \sigma e'\bar{E}e & \\ \text{s.t.} \quad D(K(A\bar{E}, \bar{E}A')u - e\gamma) + y & \geq & e \\ -s \leq u & \leq & s \\ y & \geq & 0, \end{array} \tag{6.3}$$

and for $p = 1, \ldots, n$:

$$f(\bar{E}) \leq f(E), \text{ for } E_{pp} = 1 - \bar{E}_{pp}, E_{jj} = \bar{E}_{jj}, j \neq p. \tag{6.4}$$

Proof That the sequence $\{\nu e'y^r + e's^r + \sigma e'E^r e\}_{r=1}^{r=\infty}$ converges follows from the fact that it is nonincreasing and bounded below by zero. That (6.3) is satisfied follows from the fact that each point of the sequence $\{u^r, \gamma^r, y^r, s^r, E^r\}$ satisfies (6.3) with $(\bar{u}, \bar{\gamma}, \bar{y}, \bar{s}, \bar{E})$ replaced by $\{u^r, \gamma^r, y^r, s^r, E^r\}$ on account of step (4) of Algorithm 6.1.1. That (6.4) is satisfied follows from the fact that each point of the sequence $\{E^r\}$ satisfies (6.4) with $(\bar{E})$ replaced by $\{E^r\}$ on account of step (3) of Algorithm 6.1.1. Hence every accumulation point $(\bar{u}, \bar{\gamma}, \bar{y}, \bar{s}, \bar{E})$ of $\{u^r, \gamma^r, y^r, s^r, E^r\}$ satisfies (6.3) and (6.4). □

It is important to note that by repeating steps (3) and (4) of Algorithm 6.1.1, a feature dropped in one sweep through the integer variables may be added back in another cycle, and conversely. Thus, our algorithm is not merely a naïve greedy approach because the choices of one iteration may be reversed in later iterations, and we have observed this phenomenon in our experiments. However, cycling is avoided by choosing $tol > 0$, which ensures that the sequence of objective values generated by Algorithm 6.1.1 is strictly decreasing. It is also important to note that when changing the integer variables in step (3), only the objective function needs to be recomputed, which is much faster than solving the linear program in step (4). In fact, as we shall discuss in Section 6.2, we have found that the cycle through the integer variables in step (3) tends to be repeated more often than the linear program of step (4). We turn now to computational testing of our approach.

6.2 Computational results

We illustrate the effectiveness of our Reduced Feature SVM (RFSVM) on two datasets from the UCI Machine Learning Repository [96] and on synthetic data generated using Michael Thompson's NDCC generator [115]. The UCI datasets are used to compare the feature selection and classification accuracy of RFSVM to the following two algorithms: recursive feature elimination (RFE), a wrapper method designed for SVMs [41], and Relief, a filter method [105]. A feature-reducing linear kernel 1-norm SVM (SVM1) [8], and a nonlinear kernel 1-norm SVM (NKSVM1) [72] with no feature selection are used as baselines.

The synthetic NDCC data is used to illustrate the effectiveness of RFSVM on problems with large numbers of features, including a problem with 1000 features, 900 of which are irrelevant.

6.2.1 UCI datasets

We use the UCI datasets to compare RFSVM to two other algorithms. RFE and Relief are used to illustrate how RFSVM maintains classification accuracy for different degrees of feature selection. SVM1 and NKSVM1 are used to establish baselines for feature selection and classification accuracy. For the sake of efficiency, we use the experimental methodology described below to compare the algorithms. We first briefly describe RFE and Relief.

6.2.1.1 RFE

Recursive Feature Elimination (RFE) is a wrapper method designed for SVMs [41]. First an SVM (u, γ) is learned using all features, then features are ranked based on how much the margin $u'K(A, A')u$ changes when each feature is removed separately. Features which have a small effect on the margin are considered less relevant. A given percentage of the least relevant features are removed, and the entire procedure is repeated with the remaining features. In our experiments, we remove one feature at a time until the reported number of features is reached. Note that our RFSVM procedure uses the objective value $f(E)$ to determine whether to include or remove each feature, and removes or keeps features if the objective function decreases by more than a threshold, without first ranking the features. Furthermore, once a feature is removed by RFE it is never again considered for inclusion in the final classifier, while any feature removed during a sweep through the integer variables E in our Algorithm 6.1.1 may be included by a later sweep.

6.2.1.2 Relief

Relief is a filter method for selecting features [105]. Features are ranked by computing weights as follows. For a randomly chosen training example, find the nearest example with the same class (the *nearest hit*), and the nearest example in the other class (the *nearest miss*).

Then update the weight of each feature by subtracting the absolute value of the difference in feature values between the example and the nearest hit, and adding the absolute value of the difference between the example and the nearest miss. This procedure is then repeated several times, with a different random example each time. Features with high weight are considered more relevant. Relief may be used with any binary classification algorithm, but in the following we use it exclusively with a 1-norm Gaussian kernel SVM.

6.2.1.3 Methodology

To save time, we tuned each algorithm by using $\frac{1}{11}$ of each dataset as a tuning set, and performed ten-fold cross validation on the remaining $\frac{10}{11}$. The tuning set was used to choose the parameters ν and μ on the first fold, and the chosen parameters were then used for the remaining nine folds. In order to avoid bias due to the choice of the tuning set, we repeated the above procedure five times using a different, randomly selected, tuning set each time. This procedure allows us to efficiently investigate the behavior of the feature selection algorithms RFSVM, RFE, and Relief on the datasets. Since the algorithms exhibit similar behavior on the datasets, we believe that our results support the conclusion that RFSVM is effective for learning nonlinear classifiers with reduced input space features.

For all the algorithms, we chose ν and the Gaussian kernel parameter μ from the set $\{2^i | i \in \{-7, \ldots, 7\}\}$. For each dataset, we evaluated the accuracy and number of features selected at $\sigma \in \{0, 1, 2, 4, 8, 16, 32, 64\}$. The diagonal of E was randomly initialized so that $\max\{\frac{n}{\sigma}, 1\}$ features were present in the first linear program, where n is the number of input space features for each dataset. As σ increases, the penalty on the number of features begins to dominate the objective. We only show values of σ for which we obtained reliable results. For RFE, we removed 1 feature per iteration. For Relief, we used 1000 iterations to determine the feature weights.

6.2.1.4 Results and discussion

Figure 6.1 gives curves showing the accuracy of RFSVM versus the number of input space features used on the Ionosphere and Sonar datasets. Each point on the curve is obtained by averaging five ten-fold cross validation experiments for a fixed σ. The square points denote the accuracy of NKSVM1, an ordinary nonlinear classifier which uses all the input space features. The points marked by triangles represent the accuracy and feature reduction of SVM1, a linear classifier which is known to reduce features [8]. Results for RFE are denoted by '+', and results for Relief are denoted by '♦' while results of our RFSVM algorithm are denoted by circles. Note that RFSVM is potentially able to obtain a higher accuracy than the linear classifier using approximately the same number of features on the Ionosphere and Sonar datasets. Note also that even for $\sigma = 0$, RFSVM was able to reduce features based only on decrease in the objective term $e'y$. RFSVM is comparable in both classification accuracy and feature selection to RFE and Relief.

To illustrate the efficiency of our approach, we report the CPU time taken on the Ionosphere dataset. On this dataset, the RFSVM algorithm required an average of 6 cycles through the integer variables on the diagonal of the matrix E, and the solution of 3 linear programs. The averages are taken over the classifiers learned for each fold once the parameters were selected. Using the MATLAB profiler, we found that the CPU time taken for one complete experiment on the Ionosphere dataset was 60.8 minutes. The experiment required 1960 runs of the RFSVM algorithm. Of this time, approximately 75% was used in evaluating the objective function, and 15% was used in solving linear programs. Our experience with the RFSVM algorithm is that the bottleneck is often the objective function evaluations rather than the linear programs, which suggests that significant speedups could be obtained by using more restrictive settings of the number of sweeps k and the tolerance tol for decreasing $f(E)$. These measurements were taken using MATLAB 7.2 [94] under CentOS Linux 4.3 running on an Intel 3.0 GHz Pentium 4. The linear programs were solved using CPLEX 9.0 [48] and the Gaussian kernels were computed using a compiled function written in C.

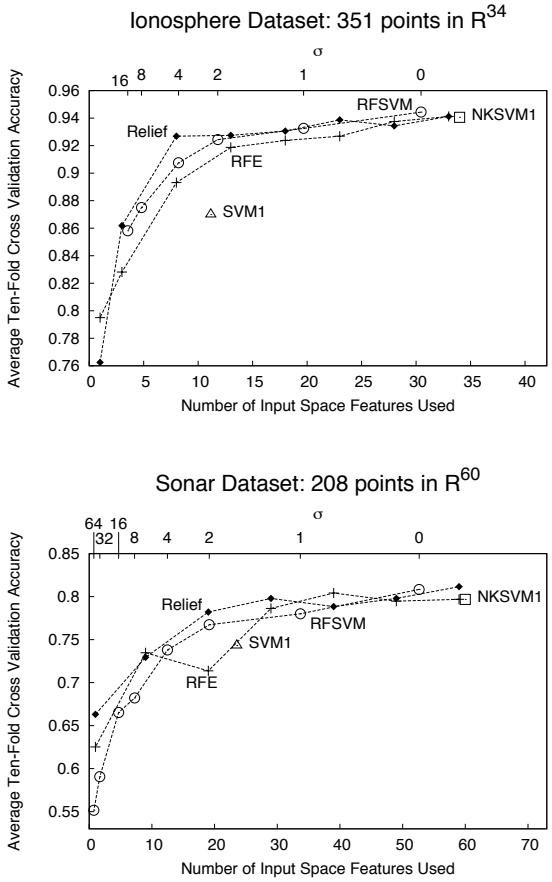


Figure 6.1 Ten-fold cross validation accuracy versus number of features used on the Ionosphere and Sonar datasets. Results for each algorithm are averages over five ten-fold cross validation experiments, each using a different $\frac{1}{11}$ of the data for tuning only, and the remaining $\frac{10}{11}$ for ten-fold cross validation. Circles mark the average number of features used and classification accuracy of RFSVM for each value of σ. '+', '♦', '□', and '△' represent the same values for RFE, Relief, NKSVM1, and SVM1, respectively.

6.2.2 NDCC data

The NDCC dataset generator creates datasets by placing normal distributions at the vertices of concentric 1-norm cubes [115]. The resulting datasets are not linearly separable, thus making them attractive testbeds for nonlinear classifiers. We create datasets to test feature selection by adding random normal features to an NDCC dataset and then normalizing all features to have mean 0 and standard deviation 1. The order of the features is shuffled. Each dataset has 200 training points, 200 tuning points, and 1000 testing points. Accuracy of RFSVM and NKSVM1 on the dataset is measured by choosing ν and μ from the set $\{2^i | i \in \{-7, \ldots, 7\}\}$ using the tuning set, and then evaluating the chosen classifier on the 1000 testing points. To save time, we arbitrarily set σ in RFSVM to 1 before performing any experiments.

Figure 6.2 shows a comparison of RFSVM and NKSVM1 on NDCC data with 20 true features as the number of irrelevant features increases. Note that the accuracy of NKSVM1 decreases more than the accuracy of RFSVM as more irrelevant features are added. When 480 irrelevant features are added, the accuracy of RFSVM is 74%, 45% higher than NKSVM1.

We also investigated the performance of RFSVM on NDCC data with 1000 features, 900 of which were irrelevant. To improve the running time of RFSVM on problems with such large numbers of features, we implemented optimizations which took advantage of the form of the Gaussian kernel. We also used the Condor distributed computing system [59], which allowed us to evaluate Algorithm 6.1.1 for several tuning parameters simultaneously. Over 10 datasets, the average classification accuracy of RFSVM was 70%, while the average classification accuracy of NKSVM1 was 53%. Thus, the feature selection provided by RFSVM leads to a 32% improvement over a classifier with no feature selection. We expect that even better accuracy could be obtained by tuning σ, and heuristics to choose σ are an important topic of future research.

When using Condor, we used the freely available CLP linear programming solver [24] to solve the linear programs and the MATLAB compiler version 4.5 to produce a stand-alone executable which ran Algorithm 6.1.1 for given values of ν and μ. On the same machine

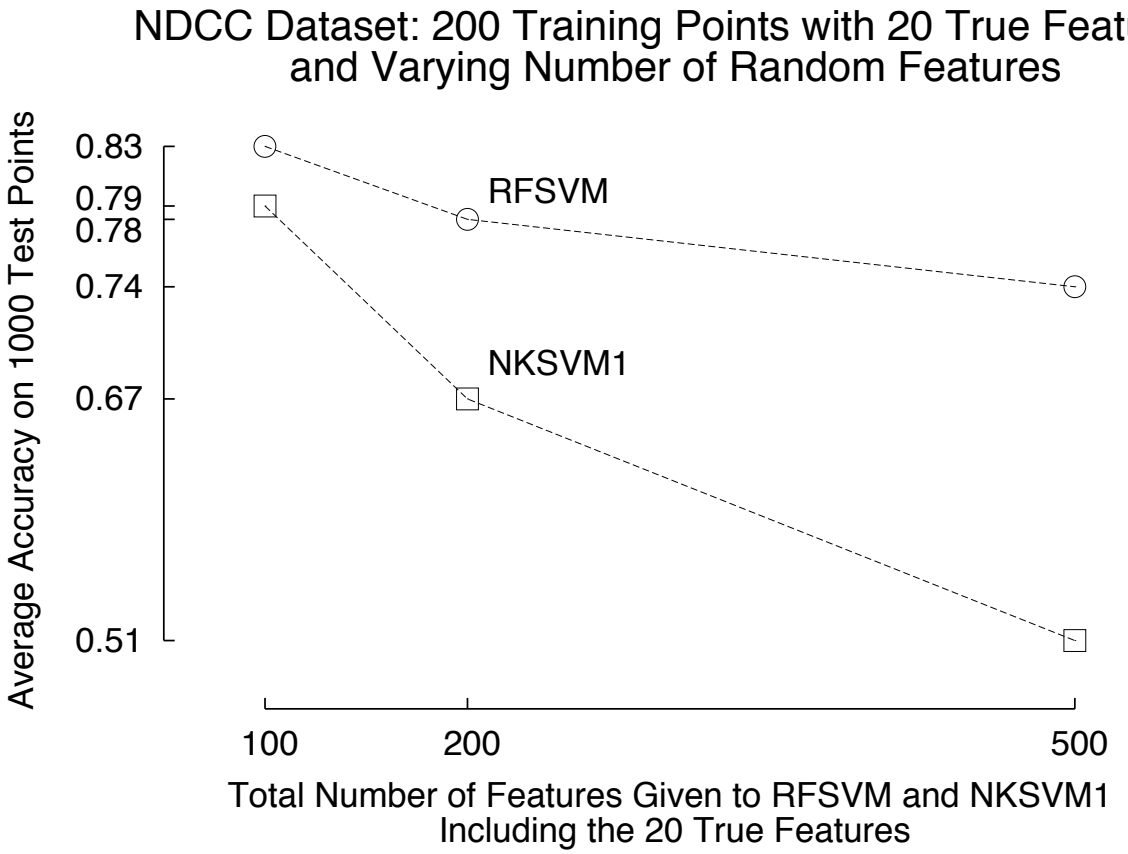


Figure 6.2 RFSVM1 and NKSVM1 on NDCC data with 20 true features and 80, 180, and 480 irrelevant random features. Each point is the average of the test set accuracy over two independently generated datasets.

described above, the average time to run this executable for the parameters chosen by the tuning procedure was 115 seconds. Further speedups may be possible for some kernels, including the Gaussian kernel, by using approximations such as [110].

Chapter 7

Generalized Eigenvalue Proximal Support Vector Machines

In standard SVM classification [72, 109] the classifier is given by a plane midway between two parallel bounding planes that bound two disjoint halfspaces. Each halfspace contains points mostly of one class, and the bounding planes are as far apart as possible. Proximal support vector classification [27, 114, 20] is another approach in which two parallel planes are generated such that each plane is closest to one of the two datasets and the two planes are as far apart as possible. The classifying plane is again midway between the two planes. This chapter describes a method which uses the generalized eigenvalue problem to drop the parallelism condition on the proximal planes, leading to an extremely simple problem formulation with a very effective and fast algorithm [83].

7.1 The multiplane linear kernel classifier

We consider the problem of classifying m points in the n-dimensional real space R^n, represented by the $m_1 \times n$ matrix A belonging to class 1 and the $m_2 \times n$ matrix B belonging to class 2, with $m_1 + m_2 = m$. For this problem, a standard support vector machine with a linear classifier [72, 109] is given by a plane midway between two *parallel* bounding planes that bound two disjoint halfspaces each containing points mostly of class 1 or 2. In another somewhat less standard approach, the proximal support vector classification [27, 114, 20], two *parallel* planes are generated such that each plane is closest to one of two datasets to be

classified and such that the two planes are as far apart as possible. The classifying plane is again midway between the parallel proximal planes, and is given by (1.17).

In this chapter we drop the parallelism condition on the proximal planes and require that each plane be as close as possible to one of the datasets and as far as possible from the other one. Thus we are seeking two planes in R^n:

$$x'w^1 - \gamma^1 = 0,\ x'w^2 - \gamma^2 = 0, \tag{7.1}$$

where the first plane is closest to the points of class 1 and furthest from the points in class 2, while the second plane is closest to the points in class 2 and furthest from the points in class 1. To obtain the first plane of (7.1) we minimize the sum of the squares of 2-norm distances between each of the points of class 1 to the plane divided by the squares of 2-norm distances between each of the points of class 2 to the plane. This leads to the following optimization problem:

$$\min_{(w,\gamma)\neq 0} \frac{\|Aw - e\gamma\|^2 / \|\left[\begin{smallmatrix} w \\ \gamma \end{smallmatrix}\right]\|^2}{\|Bw - e\gamma\|^2 / \|\left[\begin{smallmatrix} w \\ \gamma \end{smallmatrix}\right]\|^2}, \tag{7.2}$$

where $\|\cdot\|$ denotes the 2-norm and it is implicitly assumed that $(w,\gamma) \neq 0 \Longrightarrow Bw - e\gamma \neq 0$. This assumption will be made explicit below. We note that the numerator of the minimization problem (7.2) is the sum of squares of 2-norm distances in the (w,γ)-space of points in class 1 to the plane $x'w - \gamma = 0$, while the denominator of (7.2) is the sum of squares of 2-norm distances in the (w,γ)-space of points in class 2 to the same plane [71]. Simplifying (7.2) gives:

$$\min_{(w,\gamma)\neq 0} \frac{\|Aw - e\gamma\|^2}{\|Bw - e\gamma\|^2}. \tag{7.3}$$

We now introduce a Tikhonov regularization term [116] that is often used to regularize least squares and mathematical programming problems [76, 68, 20, 114] that reduces the norm of the problem variables (w,γ) that determine the proximal planes (7.1). Thus for a nonnegative parameter δ we regularize our problem (7.3) as follows:

$$\min_{(w,\gamma)\neq 0} \frac{\|Aw - e\gamma\|^2 + \delta\|\left[\begin{smallmatrix} w \\ \gamma \end{smallmatrix}\right]\|^2}{\|Bw - e\gamma\|^2}. \tag{7.4}$$

A possible geometric interpretation of the formulation (7.4) is that the first equation of (7.1) is obtained as a closest plane to the dataset represented by A with distances to points of A normalized by the sum of the distances to the points of B.

By making the definitions:

$$G := [A \quad -e]'[A \quad -e] + \delta I, H := [B \quad -e]'[B \quad -e], z := \begin{bmatrix} w \\ \gamma \end{bmatrix}, \tag{7.5}$$

the optimization problem (7.3) becomes:

$$\min_{z \neq 0} r(z) := \frac{z'Gz}{z'Hz}, \tag{7.6}$$

where G and H are symmetric matrices in $R^{(n+1)\times(n+1)}$. The objective function of (7.6) is known as the *Rayleigh quotient* [101, p. 357] and has some very useful properties which we now cite.

Proposition 7.1.1 [101, Theorem 15.9.2]**(Rayleigh Quotient Properties)** Let G and H be arbitrary symmetric matrices in $R^{(n+1)\times(n+1)}$. When H is positive definite the Rayleigh quotient of (7.6) enjoys the following properties:

(i) (**Boundedness**) The Rayleigh quotient ranges over the interval $[\lambda_1, \lambda_{n+1}]$ as z ranges over the unit sphere, where λ_1 and λ_{n+1} are the minimum and maximum eigenvalues of the generalized eigenvalue problem:

$$Gz = \lambda Hz, \quad z \neq 0. \tag{7.7}$$

(ii) (**Stationarity**)

$$\nabla r(z) = 2\frac{(Gz - r(z)Hz)}{z'Hz} = 0. \tag{7.8}$$

Thus $r(z)$ is stationary at, and only at, the eigenvectors of the generalized eigenvalue problem (7.7).

We note the following consequence of this proposition. Under the rather unrestrictive assumption that the columns of the matrix $[B \quad -e]$ are linearly independent, the global

minimum of problem (7.6) is achieved at an eigenvector of the generalized eigenvalue problem (7.7) corresponding to a smallest eigenvalue λ_1. If we denote this eigenvector by z^1 then $[w^{1\prime} \;\; \gamma^1]' = z^1$ determines the plane $w^{1\prime}x - \gamma^1 = 0$ of (7.1) which is closest to all the points of the dataset 1 and furthest away from the points of dataset 2.

By an entirely similar argument we define an analogous minimization problem to (7.4) for determining (w^2, γ^2) for the plane $x'w^2 - \gamma^2 = 0$ of (7.1) which is closest to the points of set 2 and furthest from set 1 as follows.

$$\min_{(w,\gamma)\neq 0} \frac{\|Bw - e\gamma\|^2 + \delta\|\left[\begin{smallmatrix} w \\ \gamma \end{smallmatrix}\right]\|^2}{\|Aw - e\gamma\|^2}. \tag{7.9}$$

By defining:

$$L := [B \;\; -e]'[B \;\; -e] + \delta I, M := [A \;\; -e]'[A \;\; -e], \tag{7.10}$$

and z as in (7.5), the optimization problem (7.9) becomes:

$$\min_{z\neq 0} s(z) := \frac{z'Lz}{z'Mz}, \tag{7.11}$$

where L and M are again symmetric matrices in $R^{(n+1)\times(n+1)}$. The minimum of (7.11) is achieved at an eigenvector corresponding to a smallest eigenvalue of the generalized eigenvalue problem:

$$Lz = \lambda Mz, \; z \neq 0. \tag{7.12}$$

We can now state the following proposition.

Proposition 7.1.2 (Proximal Multiplane Classification) Let $A \in R^{m_1\times n}$ represent the dataset of class 1, and $B \in R^{m_2\times n}$ represent the dataset of class 2. Define G, H, L, M and z as in (7.5) and (7.10). Assume that $[A \quad -e]$ and $[B \quad -e]$ have linearly independent columns. Then, the proximal planes (7.1) are obtained by the two MATLAB [94] commands: eig(G,H) and eig(L,M), each of which generates $n+1$ eigenvalues and eigenvectors of the generalized eigenvalue problems (7.7) and (7.12). The proximal planes (7.1) are obtained by:

$$\begin{bmatrix} w^1 \\ \gamma^1 \end{bmatrix} = z^1, \;\; \begin{bmatrix} w^2 \\ \gamma^2 \end{bmatrix} = z^2, \tag{7.13}$$

where z^1 is an eigenvector of the generalized eigenvalue problem (7.7) corresponding to a smallest eigenvalue, and z^2 is an eigenvector of the generalized eigenvalue problem (7.12) corresponding to a smallest eigenvalue.

We note that the linear independence condition is not restrictive for a great many classification problems for which $m_1 >> n$ and $m_2 >> n$. We also note that it is merely a sufficient but not a necessary condition for the above proposition to hold. Thus, in the XOR example given below, the linear independence condition is not satisfied. However we are able to obtain a perfect 2-plane classifier using Proposition 7.1.2 above.

Example 7.1.3 (Zero-Error XOR Classifier) Given the matrices:

$$A = \begin{bmatrix} 0 & 0 \\ 1 & 1 \end{bmatrix}, \quad B = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \tag{7.14}$$

we define G, H of (7.5) and L, M of (7.10) as follows.

$$G = M = \begin{bmatrix} 1 & 1 & -1 \\ 1 & 1 & -1 \\ -1 & -1 & 2 \end{bmatrix}, \quad H = L = \begin{bmatrix} 1 & 0 & -1 \\ 0 & 1 & -1 \\ -1 & -1 & 2 \end{bmatrix}, \tag{7.15}$$

Then, the generalized eigenvalue problems (7.7) and (7.12) have the following respective minimum eigenvalues and corresponding eigenvectors:

$$\begin{aligned} \lambda_1 = 0, \quad z^{1\prime} = [1 \quad -1 \quad 0], \\ \lambda_2 = 0, \quad z^{2\prime} = [1 \quad 1 \quad 1]. \end{aligned} \tag{7.16}$$

These give two lines (planes) in R^2, each of which containing the two data points from one set only:

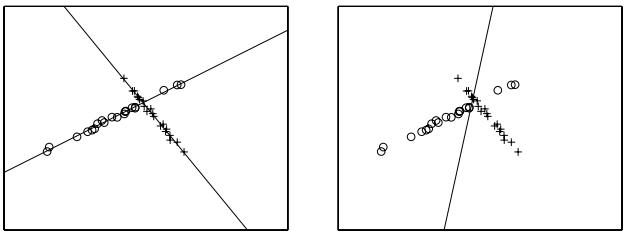
$$x_1 - x_2 = 0, \; x_1 + x_2 = 1. \tag{7.17}$$

We note that a standard 1-norm linear SVM generates a single line classifier for the XOR example that misclassifies one point [6]. Thus proximal separability does not imply linear

separability, nor is the converse true. However it is also possible for two sets to be both proximally and linearly separable.

We give another simple example to visually illustrate the effectiveness of our generalized eigenvalue proximal SVM (GEPSVM). We call this example "Cross Planes" because the data is obtained by perturbing points originally lying on two intersecting planes (lines).

Example 7.1.4 (Cross Planes Classifier) The data consists of points that are close to one of two intersecting "cross planes" in R^2. Figure 7.1 illustrates the dataset and the planes found by GEPSVM. We note that training set correctness for GEPSVM is 100% and for PSVM is 80%. We note that this example, which is a perturbed generalization of the XOR example, can serve as a difficult test case for typical linear classifiers just as the XOR example does. The reason PSVM did so poorly in comparison to GEPSVM on this example is because its proximal planes have to be parallel, meaning that PSVM generates a single linear classifier plane midway between the two proximal planes. By looking at the data points in Figure 7.1 it is obvious that the single plane of PSVM cannot do as well as the nonparallel planes of GEPSVM.



GEPSVM: 100% correct Linear PSVM: 80% correct

Figure 7.1 The "cross planes" learned by GEPSVM and the decision boundary learned by a 1-norm linear SVM together with their correctness on the training dataset.

We turn now to multisurface nonlinear classification.

7.2 The multisurface nonlinear kernel classifier

To extend our results to nonlinear multisurface classifiers we consider the following kernel-generated proximal surfaces instead of the planes (7.1):

$$K(x', C')u^1 - \gamma^1 = 0,\ K(x', C')u^2 - \gamma^2 = 0, \tag{7.18}$$

where

$$C := \begin{bmatrix} A \\ B \end{bmatrix}, \tag{7.19}$$

and K is an arbitrary kernel as defined in the Introduction. We note that the planes of (7.1) are a special case of (7.18) if we use a linear kernel $K(x', C) = x'C$ and define $w^1 = C'u^1$ and $w^2 = C'u^2$. By using the same arguments as those of Section 7.1, our minimization problem for generating a kernel-based nonlinear surface that is closest to one data set and furthest from the other leads to the minimization problem that generalizes (7.4) to the following:

$$\min_{(u,\gamma)\neq 0} \frac{\|K(A, C')u - e\gamma\|^2 + \delta\|\left[\begin{smallmatrix} u \\ \gamma \end{smallmatrix}\right]\|^2}{\|K(B, C')u - e\gamma\|^2}. \tag{7.20}$$

By making the definitions:

$$\begin{array}{rcl} G & := & [K(A, C') \quad -e]'[K(A, C') \quad -e] + \delta I, \\ H & := & [K(B, C') \quad -e]'[K(B, C') \quad -e], \end{array} \tag{7.21}$$

where G and H are now matrices in $R^{(m+1)\times(m+1)}$, the optimization problem (7.20) becomes:

$$\min_{z\neq 0} r(z) := \frac{z'Gz}{z'Hz},\ \text{where } z := \begin{bmatrix} u \\ \gamma \end{bmatrix}, \tag{7.22}$$

which is exactly the same as problem (7.6), but with different definitions for G and H.

By reversing the roles of $K(A, C')$ and $K(B, C')$ in (7.20) we obtain the following minimization problem for the proximal surface $K(x', C')u^2 - \gamma^2 = 0$ of (7.18):

$$\min_{(u,\gamma)\neq 0} \frac{\|K(B, C')u - e\gamma\|^2 + \delta\|\left[\begin{smallmatrix} u \\ \gamma \end{smallmatrix}\right]\|^2}{\|K(A, C')u - e\gamma\|^2}. \tag{7.23}$$

By defining:

$$\begin{array}{rcl} L & := & [K(B,C') \quad -e]'[K(B,C') \quad -e] + \delta I, \\ M & := & [K(A,C') \quad -e]'[K(A,C') \quad -e], \end{array} \tag{7.24}$$

where L and M are again matrices in $R^{(m+1)\times(m+1)}$, the optimization problem (7.23) becomes:

$$\min_{z\neq 0} s(z) := \frac{z'Lz}{z'Mz}, \text{ where } z := \begin{bmatrix} u \\ \gamma \end{bmatrix}, \tag{7.25}$$

which is exactly the same as problem (7.11), but with different definitions for L and M.

An analogous proposition to Proposition 7.1.2 can now be given for solving (7.22) and (7.25).

Proposition 7.2.1 (Proximal Nonlinear Multisurface Classification) Let $A \in R^{m_1\times n}$ represent the dataset of class 1, and $B \in R^{m_2\times n}$ represent the dataset of class 2. Define G, H, L, M and z as in (7.21)-(7.22) and (7.24). Assume that $[K(B,C') \quad -e]$ and $[K(A,C') \quad -e]$ have linearly independent columns. Then, the proximal surfaces (7.18) are obtained by the two MATLAB [94] commands: eig(G,H) and eig(L,M), each of which generates the $m+1$ eigenvalues and eigenvectors of the respective generalized eigenvalue problems:

$$Gz = \lambda Hz, \quad z \neq 0, \tag{7.26}$$

and

$$Lz = \lambda Mz, \quad z \neq 0, \tag{7.27}$$

The proximal surfaces (7.18) are obtained by:

$$\begin{bmatrix} u^1 \\ \gamma^1 \end{bmatrix} = z^1, \quad \begin{bmatrix} u^2 \\ \gamma^2 \end{bmatrix} = z^2, \tag{7.28}$$

where z^1 is an eigenvector of the generalized eigenvalue problem (7.26) corresponding to a smallest eigenvalue, and z^2 is an eigenvector of the generalized eigenvalue problem (7.27) corresponding to a smallest eigenvalue.

We note immediately that if either m_1 or m_2 are large, the techniques of the reduced support vector machine classification [55] can be easily applied to reduce the dimensionality $m+1 = m_1 + m_2 + 1$ of the generalized eigenvalue problem (7.26) to $\bar{m}+1$ by replacing the kernels $K(A, C')$, $K(B, C')$ by the reduced kernels $K(A, \bar{C}')$, $K(B, \bar{C}')$ respectively, where $\bar{C}$ is matrix formed by taking a small random sample of the rows of C.

We turn to our numerical tests and comparisons now.

7.3 Numerical testing and comparisons

To demonstrate the performance of our approach, we report results on publicly available datasets from the UCI Repository [96] and from [100], as well as two synthetic datasets. One synthetic dataset is David Musicant's NDC [98], and the other is a simple extension of our "Cross Planes" example above to R^7. Table 7.1 shows a linear kernel comparison of GEPSVM versus PSVM [27] and SVM-Light [49]. For a linear kernel, all three algorithms have a single parameter: δ for GEPSVM, ν for PSVM, and C for SVM-Light. This parameter was selected from the values $\{10^i | i = -7, -6, \ldots, 7\}$ by using 10% of each training fold as a tuning set. For GEPSVM only, this tuning set was not returned to the training fold to learn the final classifier once the parameter was selected. This choice was made by observing the performance of all three classifiers on datasets not shown here. GEPSVM tended to perform better without retraining on the entire training fold, while the other two algorithms benefited from the additional data. In addition to reporting the average accuracies across the ten folds, we performed paired t-tests [95] comparing PSVM to GEPSVM and SVM-Light to GEPSVM. The p-value for each test is the probability of the observed or a greater difference between two test set correctness values occurring, under the assumption of the null hypothesis that there is no difference between the test set correctness distributions. Thus, the smaller the p-value, the less likely that the observed difference resulted from identical test set correctness distributions. A typical threshold for p-values is 0.05. For example, the p-value of the test comparing GEPSVM and PSVM on the Galaxy Bright dataset was 0.031226, which is less than 0.05, leading us to conclude that GEPSVM and PSVM have different accuracies on

this dataset. We note that on the NDC and real-world datasets, the performance difference between GEPSVM and the other algorithms is statistically insignificant, with the exception of Galaxy Bright, where GEPSVM is significantly better than PSVM. This indicates that allowing the proximal planes to be nonparallel allows the classifier to better represent this dataset when needed.

Table 7.2 Compares GEPSVM, PSVM, and SVM-Light using a Gaussian kernel. The kernel parameter μ was chosen from the values $\{10^i | i = -4, -3, -2, -1\}$ for all three algorithms. The parameter ν for PSVM and C for SVM-Light was selected from the set $\{10^i | i = -4, -3, \ldots, 2\}$, while the parameter δ for GEPSVM was selected from the set $\{10^i | i = -2, -1, \ldots, 4\}$. Parameter selection was done by comparing the accuracy of each combination of parameters on a tuning set consisting of a random 10% of each training set. As in the linear kernel comparison above, this tuning set was not returned to the training fold to retrain the classifier before evaluating on the test fold for GEPSVM, but was for PSVM and SVM-Light. We note that GEPSVM has performance that is comparable to PSVM and SVM-Light on the real-world datasets, and the difference between GEPSVM and the other algorithms is not statistically significant on these datasets. As expected, nonlinear GEPSVM greatly outperformed nonlinear PSVM and SVM-Light on the Cross Planes dataset.

Table 7.3 contains a typical sample of the computation times of the three methods compared in Table 7.1. We report the average of times to learn the linear kernel classifier for each fold with the parameter selected by the tuning procedure described above on the Cylinder Bands dataset [96]. These times were obtained on a machine running Matlab 7 on Red Hat Linux 9.0 with a Pentium III 650 MHz processor and 256 megabytes of memory. Complexity of the generalized eigenvalue problem is of order n^3 [39, Section 7.7] which is similar to that of solving the system of linear equations resulting from PSVM, although the constant multiplying $O(n^3)$ for the generalized eigenvalue problem is larger. For an interior point method used for solving a 2-norm SVM quadratic program, the complexity is of order $n^{3.5}$ based on a linear complementarity problem formulation of the quadratic program [51]. These facts help explain the computation times of Table 7.3, where PSVM is over one order

Data Set $m \times n$	GEPSVM Correctness	PSVM Correctness p-value	SVM-Light Correctness p-value
Cross Planes 300×7	**98.0%**	55.3%* 5.24671e-07	45.7%* 1.4941e-08
NDC 300×7	86.7%	88.3% 0.244333	**89.0%** 0.241866
Cleveland Heart 297×13	81.8%	**85.2%** 0.112809	83.6% 0.485725
Cylinder Bands 540×35	71.3%	71.7% 0.930192	**76.1%** 0.229676
Pima Indians 768×8	73.6%	**75.9%** 0.274187	75.7% 0.380633
Galaxy Bright 2462×14	**98.6%**	97.3%* 0.031226	98.3% 0.506412
Mushroom 8124×22	81.1%	80.9% 0.722754	**81.5%** 0.356003

Table 7.1 Linear kernel GEPSVM, PSVM [27], and SVM-Light [49] ten-fold testing correctness and p-values. The p-values are from a t-test comparing each algorithm to GEPSVM. Best correctness results are in bold. An asterisk (*) denotes significant difference from GEPSVM based on p-values less than 0.05.

Data Set $m \times n$	GEPSVM Correctness	PSVM Correctness p-value	SVM-Light Correctness p-value
Cross Planes 300×7	**99.0%**	73.7%* 0.00025868	79.3%* 8.74044e-06
WPBC (60 mo.) 110×32	62.7%	**64.5%** 0.735302	63.6% 0.840228
BUPA Liver 345×6	63.8%	67.9% 0.190774	**69.9%** 0.119676
Votes 435×16	94.2%	94.7% 0.443332	**95.6%** 0.115748
Haberman's Survival 306×3	75.4%	**75.8%** 0.845761	71.7% 0.0571092

Table 7.2 Nonlinear kernel GEPSVM, PSVM [27], and SVM-Light [49] ten-fold testing correctness and p-values. The p-values were calculated using a t-test comparing each algorithm to GEPSVM. Best results are in bold. An asterisk (*) denotes significant difference from GEPSVM based on p-values less than 0.05.

GEPSVM Time (seconds)	PSVM Time (seconds)	SVM-Light Time (seconds)
0.96	**0.08**	75.4

Table 7.3 Average time to learn one linear kernel GEPSVM, PSVM [27], and SVM-Light [49] on the Cylinder Bands dataset [96].

of magnitude faster than GEPSVM, which is nearly two orders of magnitude faster than SVM-Light.

As final remarks we note that for our multiplane linear kernel classifiers of Section 7.1, very large datasets can be handled by GEPSVM provided the input space dimension n is moderate in size, say of the order of few hundreds. This is so because the generalized eigenvalue problem (7.7) is in the space R^{n+1}. Thus, even for two randomly generated matrices G and H of the order of 1000×1000, MATLAB was able to solve the generalized eigenvalue problem (7.7) in less than 75 seconds on a Pentium 4 1.7Ghz machine. For our multisurface nonlinear kernel classifiers of Section 7.2, the reduced kernel techniques of [55] can be used to handle such datasets as discussed at the end of Section 7.2.

Chapter 8

Multiple-Instance Classification

The multiple-instance classification problem was introduced in [16, 2, 62]. In this chapter, we consider the problem to consist of classifying *positive and negative bags of points in the n-dimensional real space R^n on the following basis.* Each bag contains a number of points and a classifier correctly classifies all the bags if for each positive bag *at least one* point in the bag is classified as positive, and for each negative bag *all* the points in the bag are classified as negative. Various formulations of the multiple-instance classification problem have been proposed, including integer programming [1], expectation maximization [128], kernel formulations [36], and lazy learning [128]. Ray and Craven [104] provide an empirical comparison of several multiple-instance classification algorithms and their non-multiple-instance counterparts.

This chapter introduces a novel mathematical programming formulation of the multiple-instance classification problem that leads to an efficient successive linearization algorithm. This algorithm, first implemented in [84], typically converges in a few steps to a local solution of the problem, which for a linear classifier utilizes as little as one percent of problem features. Our formulation uses a linear or nonlinear kernel support vector machine (SVM) classifier [109, 12] and is based on the following simple ideas. For a linear classifier, a positive bag is classified correctly *if and only if* some convex combination of points in the bag lies on the positive side of a separating plane. For a nonlinear kernel classifier, a similar statement applies to the higher dimensional space induced by the kernel. This leads to a constrained optimization problem where the objective function and constraints are linear except for a set

of bilinear constraints corresponding to the positive bags. A local solution to this formulation is obtained by solving a sequence of linear programs that terminate in a few iterations.

Andrews et al. [1] have previously investigated extending support vector machines to the multiple-instance classification problem using mixed-integer programming. They use integer variables either to select the class of points in positive bags or to identify one point in each positive bag as a "witness" point that must be placed on the positive side of the decision boundary. Each of these representations leads to a natural heuristic for approximately solving the resulting mixed-integer program. In contrast, we introduce continuous variables to represent the convex combination of each positive bag which must be placed on the positive side of the separating plane. This representation leads to an optimization problem with both linear and bilinear constraints, and we give an algorithm which converges to a local solution of this problem. Andrews et al. extend the quadratic programming SVM, while we begin with the linear programming SVM [72]. The use of the linear programming SVM allows us to solve a sequence of linear programs as opposed to quadratic programs. We include results in Section 8.3 which demonstrate that linear programs are much more computationally efficient than quadratic programs. Further, our results show that the use of the 1-norm in the linear programming SVM as opposed to the square of the 2-norm in the quadratic programming SVM leads to feature reduction in our linear kernel multiple-instance classifiers.

8.1 Problem formulation

8.1.1 Linear kernel classifier

Let the positive bags be represented by the k matrices $B^i \in R^{m^i \times n}$, $i = 1, \ldots, k$, where row ℓ, $B^i_\ell \in R^n$, of the matrix B^i represents point ℓ of bag i with $\ell = 1, \ldots, m^i$. Similarly, we shall represent the negative bags by the $m - k$ matrices $C^i \in R^{m^i \times n}$, $i = k+1, \ldots, m$, where row ℓ, $C^i_\ell \in R^n$, of the matrix C^i represents point ℓ of bag i with $\ell = 1, \ldots, m^i$. We shall first use a linear classifier given by the separating plane:

$$x'w = \gamma, \tag{8.1}$$

where $w \in R^n$ is the normal to a plane that attempts to separate the positive and negative bags, while γ determines the location of the plane relative to the origin in R^n. The separation will be achieved by attempting to place all the points in the negative bags in the halfspace $\{x'w \leq \gamma - 1\}$ and at the same time placing *some* convex combination of points of each positive bag in the halfspace $\{x'w \geq \gamma + 1\}$ while maximizing the margin (distance) $\frac{2}{\|w\|_1}$ between the bounding planes $x'w = \gamma \pm 1$ using the ∞-norm to measure the margin, as described in [71]. We use the vectors $v^i \geq 0,\ e'v^i = 1,\ i = 1, \ldots, k$, to denote the convex combination of the i-th positive bag which we will attempt to place in the positive halfspace. This leads to the following mathematical program with some positive parameter ν that weights data fitting versus generalization:

$$\begin{array}{rrcl} \min\limits_{w,\gamma,y,v^1,\ldots,v^k} & \nu e'y + \|w\|_1 & & \\ \text{s.t.} & {v^i}'B^i w - \gamma + y_i & \geq & 1,\ i = 1, \ldots, k, \\ & -C^i_\ell w + \gamma + y^i_\ell & \geq & 1,\ i = k+1, \ldots, m,\ \ell = 1, \ldots, m^i, \\ & e'v^i = 1,\ v^i & \geq & 0,\ i = 1, \ldots, k, \\ & y & \geq & 0. \end{array} \tag{8.2}$$

Here, the vector y with components $y_i,\ i = 1, \ldots, k$ and $y^i_\ell,\ i = k+1, \ldots, m,\ \ell = 1, \ldots, m^i$ represents nonnegative slack variables that are driven towards zero by the objective function term $\nu e'y$. The objective function term $\|w\|_1$ represents twice the reciprocal of the soft margin between the bounding planes $x'w = \gamma \pm 1$. Other than the first set of k constraints which are bilinear, the optimization problem has a linear objective function and constraints. We note that the term $\|w\|_1$ is easily converted to a linear term $e's$ with the added constraint $s \geq w \geq -s$. We shall propose and establish convergence to a local solution of our formulation (8.2) of the multiple-instance problem via a successive linearization algorithm in the next section after we have formulated the nonlinear kernel problem.

8.1.2 Nonlinear kernel classifier

We now describe how to generate a nonlinear classifier via a nonlinear kernel formulation. We replace the separating plane (8.1) by the nonlinear separating surface:

$$K(x', H')u = \gamma, \tag{8.3}$$

where $u \in R^m$ is a dual variable and the $m \times n$ matrix H is defined as:

$$H' = [B^{1'} \dots B^{k'}\ C^{k+1'} \dots C^{m'}] \tag{8.4}$$

and $K(x', H')$ is an arbitrary kernel map from $R^n \times R^{n \times m}$ into R^m. We note that the linear classifier (8.1) is recovered from (8.3) if we use the linear kernel $K(x', H') = x'H'$ and define $w = H'u$. With this nonlinear kernel formulation, the mathematical program for generating the nonlinear classifier becomes the following, upon kernelizing (8.2):

$$\begin{array}{rrcl} \min\limits_{u,\gamma,y,v^1,\dots,v^k} & \nu e'y + \|u\|_1 & & \\ \text{s.t.} & v^{i'}K(B^i, H')u - \gamma + y_i & \geq & 1,\ i = 1, \dots, k, \\ & -K(C^i_\ell, H')u + \gamma + y^i_\ell & \geq & 1,\ i = k+1, \dots, m,\ \ell = 1, \dots, m^i, \\ & e'v^i = 1,\ v^i & \geq & 1,\ i = 1, \dots, k, \\ & y & \geq & 0. \end{array} \tag{8.5}$$

We note again that the only nonlinearity is in the first k bilinear constraints. Also, note that the use of the 1-norm, $\|u\|_1$, leads to model simplification by suppressing components of u. In contrast, the weighted 1-norm of u, $\|K(H, H')^{\frac{1}{2}}u\|_1$, does not ensure such simplification, particularly if the kernel K is not positive definite. A necessary and sufficient condition for strict nonlinear kernel separation is that $y < e$, which is true if and only if $v^{i'}K(B^i, H')u - \gamma > 0$ for $i = 1, \dots, k$ and $K(C^i_\ell, H')u - \gamma < 0$ for $i = k+1, \dots, m$, $\ell = 1, \dots, m^i$.

8.2 Multiple-instance classification algorithm

Since only the first constraints in our multiple-instance formulation are nonlinear, and in fact are bilinear, an obvious method of solution suggests itself as follows. Alternately hold

one set of variables that constitute the bilinear terms constant while varying the other set. This leads to the successive solution of linear programs that underly our algorithm which we specify now.

Algorithm 8.2.1 MICA: Multiple-Instance Classification Algorithm with a Nonlinear Kernel

(0) For fixed $\nu > 0$, initialize $v^{i0} = \frac{e}{m^i}$, $i = 1, \ldots, k$. Set counter $r = 0$.
Note: This initial choice of $v^{10}, \ldots, v^{k0}$ results in using the mean for each positive bag in (i) below.

(i) For fixed $v^{1r}, \ldots, v^{kr}$, where v^{ir} is iterate r for v^i, solve the following linear program for (u^r, γ^r, y^r):

$$\min_{u,\gamma,y} \quad \nu e'y + \|u\|_1, \tag{8.6a}$$

$$\text{s.t.} \quad v^{ir'}K(B^i, H')u - \gamma + y_i \geq 1,\ i = 1, \ldots, k, \tag{8.6b}$$

$$-K(C^i_\ell, H')u + \gamma + y^i_\ell \geq 1,\ i = k+1, \ldots, m,\ \ell = 1, \ldots, m^i, \tag{8.6c}$$

$$y \geq 0. \tag{8.6d}$$

(ii) For u^r fixed at the value obtained in (i), solve the following linear program for $(\gamma, y, v^{1(r+1)}, \ldots, v^{k(r+1)})$:

$$\min_{\gamma,y,v^1,\ldots,v^k} \quad e'y, \tag{8.7a}$$

$$\text{s.t.} \quad v^{i'}K(B^i, H')u^r - \gamma + y_i \geq 1,\ i = 1, \ldots, k, \tag{8.7b}$$

$$-K(C^i_\ell, H')u^r + \gamma + y^i_\ell \geq 1,\ i = k+1, \ldots, m, \ell = 1, \ldots, m^i, \tag{8.7c}$$

$$e'v^i = 1,\ v^i \geq 0,\ i = 1, \ldots, k, \tag{8.7d}$$

$$y \geq 0. \tag{8.7e}$$

(iii) Stop if $\|(v^{1(r+1)} - v^{1r}, \ldots, v^{k(r+1)} - v^{kr})\|_2$ is less than some desired tolerance. Else replace $(v^{1r}, \ldots, v^{kr})$ by $(v^{1(r+1)}, \ldots, v^{k(r+1)})$, r by $r+1$ and go to (i).

Since the objective function of our original multiple-instance formulation (8.5) is bounded below by zero and is nonincreasing in the iterations (i) and (ii) of the MICA Algorithm 8.2.1, it must converge. We can state the following convergence result.

Proposition 8.2.2 Convergence to a Local Minimum Value. The nonnegative nonincreasing values of the sequence of objective function values $\{\nu e'y^r + \|u^r\|_1\}_{r=1}^{r=\infty}$ converges to $(\nu e'\bar{y} + \|\bar{u}\|_1)$ where $(\bar{u}, \bar{\gamma}, \bar{y}, \bar{v}^1, \ldots, \bar{v}^k)$ is any accumulation point of the sequence of iterates $\{(u^r, \gamma^r, y^r, v^{1r}, \ldots, v^{kr})\}$ generated by the MICA Algorithm 8.2.1. The point $(\bar{u}, \bar{\gamma}, \bar{y}, \bar{v}^1, \ldots, \bar{v}^k)$ has the following local minimum property:

$$\nu e'\bar{y} + \|\bar{u}\|_1 = \min_{u,\gamma,y} \quad \nu e'y + \|u\|_1, \tag{8.8a}$$

$$\text{s.t.} \quad (\bar{v}^i)'K(B^i, H')u - \gamma + y_i \geq 1, \; i = 1, \ldots, k, \tag{8.8b}$$

$$-K(C^i_\ell, H')u + \gamma + y^i_\ell \geq 1, \quad i = k+1, \ldots, m, \quad \ell = 1, \ldots, m^i, \tag{8.8c}$$

$$y \geq 0. \tag{8.8d}$$

Proof That the sequence $\{\nu e'y^r + \|u^r\|_1\}$ converges follows from the fact that it is nonincreasing and bounded below by zero. That (8.8) is satisfied follows from the fact each point of the sequence $\{(u^r, \gamma^r, y^r, v^{1r}, \ldots, v^{kr})\}$ satisfies (8.8) with $(\bar{u}, \bar{y}, \bar{v}^1, \ldots, \bar{v}^k)$ replaced by $(u^r, y^r, v^{1r}, \ldots, v^{kr})$ on account of step (i) of the MICA Algorithm 8.2.1. Hence, any accumulation point $(\bar{u}, \bar{\gamma}, \bar{y}, \bar{v}^1, \ldots, \bar{v}^k)$ of $\{(u^r, \gamma^r, y^r, v^{1r}, \ldots, v^{kr})\}$ satisfies (8.8). □

In practice the MICA Algorithm 8.2.1 terminates very quickly, typically before ten iterations.

Before we turn to our numerical results, it is important to point out some significant differences between our MICA Algorithm 8.2.1 and the mi-SVM and MI-SVM mixed integer programming formulations introduced by Andrews et al. in [1]. A key difference is that MICA employs the 1-norm, rather than the 2-norm used by mi-SVM and MI-SVM. The 1-norm SVM formulation is known to lead to sparse solutions [8, 129], which corresponds to using few input features when a linear classifier is used. Our experimental results will

demonstrate this behavior. The 1-norm also allows MICA to be solved by a succession of linear programs, rather than more complex quadratic programs. We include results which show that state of the art optimization software solves linear programs much faster than it solves quadratic programs. Linear programs are also simpler than quadratic programs in the sense that every linear program is a quadratic program, but not conversely. Also, MICA uses an arbitrary convex combination of points in the positive bags to represent each such bag. This representation is done by means of a continuous nonnegative variable v^i for each positive bag. This is fundamentally different from both mi-SVM which uses integer variables to assign labels to each point in each positive bag, and from MI-SVM which chooses a single "witness" point to represent each positive bag. Furthermore, while MI-SVM chooses the witness point to be the point furthest from the decision boundary, MICA does not necessarily choose a convex combination furthest from the decision boundary. Note that the objective function of (8.7) is the hinge loss (i. e. the plus function: maximum $\{0, \cdot\}$, where the dot stands for the terms appearing to the left of y_i and y_ℓ^i minus 1 in Equation (8.7)). Each y_i, $i = 1, \ldots, k$, will attain a minimum of zero at the solution for *any* v^i which satisfies ${v^i}' K(B^i, H')u^r \geq \gamma + 1$, that is at least one element of bag i is on the correct side of the bounding surface. Conversely, v^i will act similarly to the selection of a "witness" point in MI-SVM only in the case that every element in the bag B^i is on the wrong side of the bounding surface at the solution of (8.7), that is $K(B^i, H')u^r < e\gamma + e$. Furthermore, note that γ is also updated in (8.7), but not in the corresponding step of MI-SVM. Finally, MICA and mi-SVM use one slack variable per negative instance, while MI-SVM uses one slack variable per negative bag. Our formulation leads to a simple algorithm that always converges to a local solution.

8.3 Numerical testing

To demonstrate the capabilities of our formulation, we report results on twelve datasets, two from the UCI machine learning repository [96], and ten from [1]. Detailed information about these datasets is summarized in Table 8.1. We use the datasets from [1] to evaluate our linear classification algorithm. Three of these datasets are from an image annotation task

Table 8.1 Description of the datasets used in the experiments. Elephant, Fox, Tiger, and the TST datasets are used in [1], while Musk-1 and Musk-2 are available from [96]. + Bags denotes the number of positive bags in each dataset, while + Instances denotes the total number of instances in all the positive bags. Similarly, - Bags and - Instances denote corresponding quantities for the negative bags.

Data Set	+ Bags	+ Instances	- Bags	- Instances	Features
Elephant	100	762	100	629	143
Fox	100	647	100	673	143
Tiger	100	544	100	676	143
TST1	200	1580	200	1644	6668
TST2	200	1715	200	1629	6842
TST3	200	1626	200	1620	6568
TST4	200	1754	200	1637	6626
TST7	200	1746	200	1621	7037
TST9	200	1684	200	1616	6982
TST10	200	1818	200	1635	7073
Musk-1	47	207	45	269	166
Musk-2	39	1017	63	5581	166

in which the goal is to determine whether or not a given animal is present in an image. The other seven datasets are from the OHSUMED data and the task is to learn binary concepts associated with the Medical Subject Headings of MEDLINE documents. The two datasets from the UCI repository [96] are the Musk datasets, which are commonly used in multiple-instance classification. We report results on these datasets for our nonlinear classification algorithm.

8.3.1 Linear kernel classification results

We compare our linear classification algorithm to the linear versions of mi-SVM and MI-SVM [1]. Both mi-SVM and MI-SVM use mixed-integer programming to learn a linear classifier, and as such are natural candidates for comparison with MICA. Since Andrews et al. also report results on Zhang and Goldman's expectation-maximization approach EM-DD [128] on these datasets [1], we include those results here as well. Finally, Ray and Craven [104] demonstrate that in some domains, algorithms that make no use of multiple-instance information may outperform their multiple-instance counterparts. Thus, we include a linear programming formulation of an SVM (SVM1) [8, 72] in our comparisons as the natural non-multiple-instance counterpart to MICA.

Table 8.2 reports results comparing MICA to mi-SVM, MI-SVM, EM-DD, and SVM1. Accuracy results for mi-SVM, MI-SVM and EM-DD were taken from [1]. Accuracy for each algorithm was measured by averaging ten ten-fold cross validation runs. The regularization parameters for MICA and SVM1 were selected from the set $\{2^i | i = -7, \ldots, 7\}$ by ten-fold cross validation on each training fold for the image annotation datasets, and by using a random ten percent of each training fold as a tuning set on the OHSUMED datasets. The final classifier for each fold was trained using all the data in the training fold. MICA was stopped if the difference between the v variables was less than 10^{-4} or if $r > 80$. SVM1 was trained by assuming all instances in each positive bag had a positive label, but for tuning and testing the classification rule was the same as for MICA. Standard deviations for MICA and SVM1 were calculated by treating each of the 100 cross validation runs as

Table 8.2 Linear kernel MICA, mi-SVM [1], MI-SVM [1], EM-DD [128], and SVM1 testing accuracy and number of features used averaged over ten ten-fold cross validation experiments. For MICA and SVM1, the standard deviation (SD) of each accuracy result is given in parenthesis. The datasets are those used by Andrews et al. in [1]. The number of features used is available on all datasets for MICA and SVM1, and on the Elephant, Fox, and Tiger datasets for mi-SVM and MI-SVM. Best accuracy on each dataset is in bold. Note the substantial reduction in features by MICA and SVM1.

Data Set	MICA	mi-SVM	MI-SVM	EM-DD	SVM1
	% Correct (SD)	% Correct	% Correct	% Correct	% Correct (SD)
#Features	# Features	# Features	# Features		# Features
Elephant	80.5% (8.5)	**82.2%**	81.4%	78.3%	78.5% (9.6)
143	59.0	143.0	143.0		18.2
Fox	**58.7%** (11.3)	58.2%	57.8%	56.1%	56.7% (9.4)
143	78.0	143.0	143.0		73.6
Tiger	82.6% (7.9)	78.4%	**84.0%**	72.1%	77.3% (8.5)
143	50.8	143.0	143.0		46.0
TST1	**94.5%** (3.3)	93.6%	93.9%	85.8%	94.2% (4.1)
6668	50.5				41.1
TST2	**85.0%** (6.2)	78.2%	84.5%	84.0%	77.5% (6.4)
6842	97.2				68.4
TST3	86.0% (5.8)	87.0%	82.2%	69.0%	**87.3%** (5.0)
6568	123.7				39.4
TST4	**87.7%** (5.4)	82.8%	82.4%	80.5%	81.0% (7.4)
6626	59.9				69.0
TST7	78.9% (6.9)	**81.3%**	78.0%	75.4%	79.2% (6.4)
7037	145.3				37.3
TST9	61.4% (8.8)	**67.5%**	60.2%	65.5%	65.8% (7.8)
6982	302.1				119.6
TST10	82.3% (5.8)	79.6%	79.5%	78.5%	**83.6%** (6.5)
7073	132.2				37.6

an independent sample, as suggested by Bouckaert [7]. We provide the standard deviations to aid visualization of our results, and will use the nonparametric test described below to precisely compare the classification accuracy of all five algorithms on the ten datasets. We note that MICA had the best accuracy on four datasets, while mi-SVM had the best accuracy on three datasets. SVM1 had the best accuracy on two datasets, and MI-SVM had the best accuracy on one dataset. EM-DD did not have the best accuracy on any dataset.

In order to evaluate the difference between the algorithms more precisely, we used the Friedman test [25] on the results reported in Table 8.2. The Friedman test is a nonparametric test that compares the average ranks of the algorithms, where the algorithm with the highest accuracy on a dataset is given a rank of 1 on that dataset, and the algorithm with the worst accuracy is given a rank of 5. For example, on the Elephant dataset, mi-SVM has rank 1, MICA has rank 3, and EM-DD has rank 5. The average rank for MICA was 2.1, for mi-SVM 2.3, for SVM1 2.9, for MI-SVM 3.1, and for EM-DD 4.6. The Friedman test indicated that these results were significantly different at the five percent level, so we went on to perform a Bonferroni-Dunn post-hoc test [19] to compare MICA with the other algorithms. We found that the only algorithm with statistically significant difference from MICA was EM-DD. Demšar [15] gives an introduction to these tests, and demonstrates their applicability to the comparison of machine learning algorithms. These results indicate that MICA, which is the only algorithm that incorporates *both* multiple-instance information and substantial feature reduction, has accuracy comparable to the other linear classifiers and better than EM-DD on these datasets.

Table 8.2 also reports the number of features used by MICA and SVM1 on all datasets, and by the mixed-integer programming formulations on the image annotation datasets (Elephant, Fox, and Tiger). The feature selection results for mi-SVM and MI-SVM were computed by running a single ten-fold cross validation experiment on each dataset. The regularization parameter was chosen from the set $\{2^i | i = -7, \ldots, 7\}$ by a tuning set consisting of a random ten percent of each training fold, and the final classifier was trained on all the data in the training fold for each fold. Our implementations achieved accuracies within a

Table 8.3 Average running time in seconds of linear kernel MICA, mi-SVM [1], and MI-SVM [1]. Times were computed for each combination of the ten datasets in Table 8.2 and the fifteen regularization parameter values in $\{2^i | i = -7, \ldots, 7\}$. Best result is in bold.

Algorithm	MICA	mi-SVM	MI-SVM
Average Elapsed Time in Seconds	**44.9**	339.7	90.4

few percentage points of the reported accuracies in [1]. We do not believe that changing the tuning procedure or running more experiments would change the number of features used, since our results are in line with previous comparisons between 1-norm and 2-norm penalties for SVMs [8, 129]. We note that although there were 230 features reported in the three image annotation datasets, 87 are zero in every instance for each dataset. Thus, neither MI-SVM nor mi-SVM reduce features on these datasets. While we did not test mi-SVM and MI-SVM on the OHSUMED data, the above results and cited work indicate that they will not reduce features as drastically as MICA or SVM1. We note that in some cases MICA used less than one percent of the features, and never used more than five percent on the OHSUMED data or more than 55% of the 143 non-zero features on the image annotation data.

8.3.2 Computational efficiency

To demonstrate the efficiency of our proposed formulation, we describe its behavior on the Elephant dataset. The average time to learn a classifier once the parameter ν was chosen was 25.2 seconds, and the average number of MICA iterations required was 5.8. Note that each iteration involves solving two linear programs. These results were obtained on a Pentium III 650 MHz desktop machine with 256MB RAM running Tao Linux, Version 1. The algorithm was implemented in MATLAB [94] and the linear programs were solved using the dual simplex method of the CPLEX linear programming solver [48].

To further investigate the benefits of linear programming, we compared two SVM formulations on the TST4 dataset. One formulation was the linear programming formulation SVM1 used in Table 8.2, and the other was a quadratic programming formulation SVM2

which was identical to SVM1 except for using the square of the 2-norm instead of the 1-norm as the penalty on the weight vector w. Timing and feature selection information for these algorithms were obtained on a machine which had a Pentium IV 3 GHz processor with 1GB RAM running CentOS Linux 4.4. Both algorithms were implemented in MATLAB and the linear and quadratic programs were solved using CPLEX 9.0. Over a single ten-fold cross validation experiment, the average time to solve the linear program for SVM1 after the parameters were selected by the tuning procedure was 2.2 seconds, while the average time to solve the corresponding quadratic program for SVM2 was 28.6 seconds, a factor of 13 slower than the time to solve the linear program for SVM1. Furthermore, SVM2 used on average 6241 of the 6626 features in the TST4 dataset, while SVM1 used an average of 69 features as indicated in Table 8.2. These results support the claim that the use of the 1-norm in the term $\|u\|_1$ in MICA results in faster running times and sparser solutions than would the use of the square of the 2-norm $\|u\|^2$.

To compare the running time of MICA with mi-SVM and MI-SVM, we used our implementations of mi-SVM and MI-SVM in MATLAB mentioned above. We tested each algorithm on the ten datasets listed in Table 8.2 using the entire dataset as the training set, and for every value of the regularization parameter in the set $\{2^i | i = -7, \ldots, 7\}$. Thus, each of the three algorithms was run 150 times. Table 8.3 gives the average running time for each algorithm, which was obtained on the same computer mentioned above. We note that comparing the logarithm of the running times using a randomized block design with the dataset and value of the regularization parameter as blocks we found a significant difference between the running times of MICA and mi-SVM and MICA and MI-SVM, both with p-values much less than 10^{-4}. We used the logarithm transform in order to stabilize the variance. This result indicates that the differences between MICA and mi-SVM and MICA and MI-SVM are not due to random effects of the environment. Although we report elapsed time, we found the same effect when considering CPU time or the CPU time spent in CPLEX by each of the three algorithms. Furthermore, we measured the number of iterations each

Table 8.4 Nonlinear kernel MICA, mi-SVM [1], MI-SVM [1], EM-DD [128], DD [93] MI-NN [103], IAPR [16], and MIK [36] ten-fold testing accuracy on the Musk-1 and Musk-2 datasets. Best accuracy is in bold.

Data Set	MICA	mi-SVM	MI-SVM	EM-DD	DD	MI-NN	IAPR	MIK
Musk-1	84.4%	87.4%	77.9%	84.8%	88.0%	88.9%	**92.4%**	91.6%
Musk-2	**90.5%**	83.6%	84.3%	84.9%	84.0%	82.5%	89.2%	88.0%

algorithm took. On average, MICA took 7.5 iterations, mi-SVM took 9.8 iterations, and MI-SVM took 5.1 iterations. MICA spent an average of 28% of its CPU time in CPLEX, while mi-SVM spent an average of 81% and MI-SVM spent an average of 47%. MICA had the lowest total time in 88% of the 150 combinations of datasets and regularization parameter values, and the lowest CPLEX CPU time in 90% of the combinations. From these results, we conclude that MICA's advantage in running time over mi-SVM and MI-SVM is primarily due to the use of linear programming over quadratic programming.

8.3.3 Nonlinear kernel classification

Although our numerical testing is focused on linear classification, Table 8.4 gives ten-fold cross validation accuracy results for MICA using a Gaussian kernel and previously published results of several other algorithms on the Musk-1 and Musk-2 datasets which are available from the UCI repository [96]. The results for EM-DD are taken from [1]. The MIK entry reports the best leave-ten-out result among the multi-instance kernel methods of Gartner et al. [36]. The IAPR entry reports the results obtained by Dietterich et al. using the Iterated Discrimination Axis-Parallel Rectangle algorithm [16]. The parameters ν and μ of MICA were both chosen from the set $\{2^i | i = -7, \ldots, 7\}$ using a subset of the training set as a tuning set for each fold. MICA was stopped if the difference between the v variables was less than 10^{-4} or if $r > 80$. The full kernel matrix of the Musk-2 dataset did not fit into memory as required by CPLEX, so to speed computation and reduce the risk of overfitting, a reduced kernel was used on both datasets. As established in [55], reduced kernels consisting of only 10% of the dataset resulted in effective nonlinear support vector machine classifiers.

Hence, we used a randomly selected reduced kernel that consisted of 10% of the rows of H. Although we do not perform statistical tests on the nonlinear datasets, the results in Table 8.4 indicate that the nonlinear kernel formulation of MICA is able to attain accuracy comparable to that of previously published multiple-instance algorithms on these datasets.

Chapter 9

Exactness Conditions for a Convex Differentiable Exterior Penalty for Linear Programming

In [75], a classical exterior penalty formulation for the dual of a linear program was shown to yield an exact solution to the primal linear program provided that the penalty parameter was sufficiently large, but finite. However, no precise value for the penalty parameter was given that would guarantee an exact primal solution. In this chapter, we give sufficient conditions on the penalty parameter so that the corresponding computed primal variable is an exact solution. In Section 9.1 we derive our sufficient conditions for the penalty parameter to be large enough to generate an exact primal solution. Section 9.2 briefly details a generalized Newton algorithm as well as a new and fast iterative algorithm that solves a sequence of linear equations. Both methods, first implemented in [88], will be used to obtain our numerical results.

9.1 Sufficient conditions for dual exterior penalty function exactness

We shall consider the solvable linear program (LP):

$$\min_{y \in R^{\ell}} d'y \;\; s.t. \;\; By \geq b, \tag{9.1}$$

where $d \in R^{\ell}$, $B \in R^{m \times \ell}$, $b \in R^{m}$, and its dual:

$$\max_{u \in R^{m}} b'u \;\; s.t. \;\; B'u = d, \; u \geq 0. \tag{9.2}$$

The classical asymptotic exterior penalty problem for the dual linear program is:

$$\min_{u \in R^m} \epsilon(-b'u) + \frac{1}{2}(\|B'u - d\|^2 + \|(-u)_+\|^2). \tag{9.3}$$

Dividing Equation (9.3) by ϵ^2 and letting:

$$\frac{u}{\epsilon} \rightarrow u, \quad \alpha = \frac{1}{\epsilon}, \tag{9.4}$$

Equation (9.3) becomes:

$$\min_{u \in R^m} f(u) = -b'u + \frac{1}{2}(\|B'u - \alpha d\|^2 + \|(-u)_+\|^2). \tag{9.5}$$

Applying Proposition 1 of [75] to the dual exterior penalty problem (9.5) we get:

Proposition 9.1.1 Exact Primal Solution Computation Let the primal LP (9.1) be solvable. Then the dual exterior penalty problem (9.5) is solvable for all $\alpha > 0$. For any $\alpha \geq \bar{\alpha}$ for some finite $\bar{\alpha} > 0$, any solution u of (9.5) generates an exact solution to primal LP (9.1) as follows:

$$y = B'u - \alpha d. \tag{9.6}$$

In addition, this y minimizes:

$$\|y\|^2 + \|By - b\|^2, \tag{9.7}$$

over the solution set of the primal LP (9.1).

What we are after here are sufficient conditions to ensure that $\alpha \geq \bar{\alpha}$, which we proceed to obtain now by first establishing the following lemma based on the convexity of the penalty function (9.5).

Lemma 9.1.2 Optimality Condition for Solving the Exterior Penalty (9.5) A necessary and sufficient condition for u to be a solution of the exterior penalty function (9.5) is that:

$$\nabla f(u) = -b + B(B'u - \alpha d) + Pu = 0, \tag{9.8}$$

where $P \in R^{m \times m}$ is a diagonal matrix of ones and zeros defined as follows:

$$P = diag\Big(sign\big((-u)_+\big)\Big). \tag{9.9}$$

Proof Setting the gradient of the convex differentiable exterior penalty function (9.5) equal to zero gives:

$$-b + B(B'u - \alpha d) - (-u)_+ = 0, \tag{9.10}$$

which, upon making use of the definition (9.9) of P, gives (9.8). □

Note that the matrix P in the condition (9.8) depends on u and hence cannot be used directly to solve for u. However, P will be used to derive the conditions needed to ensure that $\alpha \geq \bar{\alpha}$, as we shall proceed to do now. We note first that by Proposition 9.1.1, Equation (9.8) is solvable for any $\alpha > 0$ whenever the linear program (9.1) is solvable. Hence, for such a case an explicit solution to (9.8) is given as follows:

$$u = (BB' + P)\backslash(\alpha Bd + b), \tag{9.11}$$

where the MATLAB [94] backslash notation $u = H\backslash h$ denotes a solution to a solvable system of linear equations $Hu = h$. If we now substitute for u in (9.6) we obtain the following expression for the solution y of the primal linear program (9.1) when $\alpha \geq \bar{\alpha}$:

$$y = B'\Big((BB' + P)\backslash b\Big) + \alpha\Big(B'\big((BB' + P)\backslash(Bd)\big) - d\Big). \tag{9.12}$$

We note immediately that y depends on α through the explicit term α in the above Equation (9.12), as well as through the dependence of P on u in its definition (9.9), and the dependence of u on α through (9.11). Hence, in order for y to be independent of α, as would be the case when $\alpha \geq \bar{\alpha}$, we have the following result which follows directly from the expression (9.12) for y.

Proposition 9.1.3 Sufficient Conditions for Independence on α The optimal solution y given by (9.12) for a solvable primal linear program (9.1) is independent of α if:

$$\textit{The diagonal matrix } P \text{ is independent of } \alpha \tag{9.13}$$

and

$$B'\big((BB' + P)\backslash(Bd)\big) - d = 0. \tag{9.14}$$

We note immediately that condition (9.14) is easy to verify once the exterior penalty problem (9.3) is solved. However, although condition (9.13) is difficult to verify without varying α, it turns out that in all our computational results presented in this Section 9.4, condition (9.14) is satisfied whenever an exact solution to the linear program (9.1) is obtained. Hence, for all intents and purposes, (9.13) can be ignored. Similarly, even though we have not established the necessity of conditions (9.13) and (9.14), the necessity of (9.14) appears to hold in our computational results.

We now describe our computational algorithms.

9.2 Computational algorithms

In this section we present two algorithms for solving the linear program (9.1) by solving the dual exterior penalty problem (9.5). Both algorithms utilize the sufficient condition (9.14) for independence on the penalty parameter α as a stopping criterion.

9.2.1 Generalized Newton algorithm

Our first algorithm will make use of the generalized Newton algorithm, utilized in [75] for 1-norm support vector machine classification problems, to solve our unconstrained minimization problem (9.5) with an appropriate value of the penalty parameter α by utilizing the new sufficient condition (9.14) for exactness. For that purpose we define $f(u)$ as the objective function of (9.5), that is:

$$f(u) = -b'u + \frac{1}{2}(\|B'u - \alpha d\|^2 + \|(-u)_+\|^2). \tag{9.15}$$

The gradient and generalized Hessian as defined in the Introduction are given as follows.

$$\nabla f(u) = -b + B(B'u - \alpha d) - (-u)_+. \tag{9.16}$$

$$\partial^2 f(u) = BB' + diag\Big(sign\big((-u)_+\big)\Big). \tag{9.17}$$

We now incorporate our new sufficient condition (9.14) into the generalized Newton algorithm [75, Algorithm 3] for solving the unconstrained minimization problem (9.5). In particular the sufficient condition (9.14) will be used as a stopping criterion in Step (II) of the following Algorithm 9.2.1, as well as in Algorithm 9.2.3 further on.

Algorithm 9.2.1 Generalized Newton Algorithm for (9.5) Let $f(u)$, $\nabla f(u)$ and $\partial^2 f(u)$ be defined by (9.15),(9.16) and (9.17). Set the parameter values α, δ, tol and imax (typically $\alpha = 100$, $\delta = 1e{-}6$, tol=$1e{-}3$ and imax=5000). Start with a random $u^0 \in R^m$. For $i = 0, 1, \ldots$:

(I) $u^{i+1} = u^i - \lambda_i(\partial^2 f(u^i) + \delta I)^{-1}\nabla f(u^i) = u^i + \lambda_i t^i$,
where the Armijo stepsize $\lambda_i = \max\{1, \frac{1}{2}, \frac{1}{4}, \ldots\}$ is such that:

$$f(u^i) - f(u^i + \lambda_i t^i) \geq -\frac{\lambda_i}{4}\nabla f(u^i)' t^i, \tag{9.18}$$

and t^i is the modified Newton direction:

$$t^i = -(\partial^2 f(u^i) + \delta I)^{-1}\nabla f(u^i). \tag{9.19}$$

In other words, start with $\lambda_i = 1$ and keep multiplying λ_i by $\frac{1}{2}$ until (9.18) is satisfied.

(II) Stop if $\|\nabla f(u^i)\| \leq tol$ & $norm\Big(B'\big((BB' + P^i)\backslash(Bd)\big) - d\Big) \leq tol$ where $P^i = diag\Big(sign\big((-u^i)_+\big)\Big)$.

(III) If $i = imax$ then $\alpha \to 10\alpha$, $imax \to 2 \cdot imax$

(IV) $i \to i + 1$ and go to (I)

The iterates u^i of the above algorithm either terminate or converge as follows [75, Proposition 4].

Proposition 9.2.2 Generalized Newton Algorithm 9.2.1 Convergence Let $tol = 0$ and assume that (9.14) implies that $\alpha \geq \bar{\alpha}$. Then either Algorithm 9.2.1 terminates at an i such that u^i solves the exterior penalty problem (9.5) and consequently $y^i = B'u^i - \alpha d$ solves the primal linear program (9.1), or any accumulation point $\bar{u}$ of the sequence $\{u^i\}$

generated by Algorithm 9.2.1 solves the exterior penalty problem (9.5) and the corresponding $\bar{y} = B'\bar{u} - \alpha d$ solves the primal linear program (9.1).

Proof If Algorithm 9.2.3 terminates at some i, then, by the stopping criterion (II), $\nabla f(u^i) = 0$, and $B'\big((BB' + P^i)\backslash(Bd)\big) - d = 0$. Hence, $P^{i+1} = P^i$, condition (9.14) is satisfied for P^i, and u^i solves (9.8). Consequently, u^i solves the dual penalty minimization problem (9.5), and since, by assumption, condition (9.14) implies that $\alpha \geq \bar{\alpha}$, it follows that $y^i = B'u^i - \alpha d$ solves the primal linear program (9.1). □

For the case when $\{i\}$ does not terminate, since the corresponding sequence of diagonal matrices of ones and zeros $\{P^i\}$ has a finite number of possible configurations, at least one such configuration must occur infinitely often and, by [75, Proposition 4], for any accumulation point $\bar{u}$ of $\{u^i\}$ there exists a subsequence $\{u^{i_j}, P^{i_j}\}$ with constant P^{i_j} that converges to $(\bar{u}, \bar{P})$ such that $\nabla f(\bar{u}) = 0$ and $B'\big((BB' + \bar{P})\backslash(Bd)\big) - d = 0$. Consequently, $\bar{u}$ solves the minimization problem (9.15) and since, by assumption, (9.14) implies that $\alpha \geq \bar{\alpha}$, it follows that $\bar{y} = B'\bar{u} - \alpha d$ solves the primal linear program (9.1). □

We now give a considerably simpler algorithm based on the successive direct solution of a linear equation obtained from the sufficient condition (9.8).

9.2.2 Direct linear equation (DLE) algorithm

Our direct linear equation algorithm consists of successively solving the necessary and sufficient optimality condition (9.8) for updated values of the diagonal matrix P. The sufficient condition (9.14) for independence on α is satisfied in our computational examples.

Algorithm 9.2.3 DLE: Direct Linear Equation Algorithm for (9.8) Set the parameter values α, δ, tol, tol1 and imax (typically $\alpha = 100$, $\delta = 1e{-}8$, tol=$1e{-}16$, tol1=$1e{-}3$ and imax=500). Start with a random $u^0 \in R^m$. For $i = 0, 1, \ldots$:

(I) Set P^i to:

$$P^i = diag\Big(sign\big((-u^i)_+\big)\Big). \tag{9.20}$$

(II) Solve for u^{i+1} as follows:

$$u^{i+1} = (BB' + P^i)\backslash(b + \alpha Bd). \tag{9.21}$$

(III) $u^{i+1} \rightarrow \big(u^i + \lambda_i(u^{i+1} - u^i)\big)$, where the Armijo stepsize $\lambda_i = \max\{1, \frac{1}{2}, \frac{1}{4}, \ldots\}$ is such that:

$$f(u^i) - f\big(u^i + \lambda_i(u^{i+1} - u^i)\big) \geq -\frac{\lambda_i}{4}\nabla f(u^i)'(u^{i+1} - u^i). \tag{9.22}$$

(IV) Stop if $\|u^{i+1} - u^i\| \leq tol$ & $norm\Big(B'\big((BB' + P^i)\backslash(Bd)\big) - d\Big) \leq tol1$.

(V) If $i = imax$ then $\alpha \rightarrow 10\alpha$, $imax \rightarrow 2 \cdot imax$.

(VI) $i \rightarrow i + 1$ and go to (I).

We state now the following convergence result for the DLE Algorithm 9.2.3. The result is based on the fact that the direction $(u^{i+1} - u^i)$ is a descent direction for the dual penalty function (9.15), and satisfies the convergence conditions required by [70, Theorem 2.1] .

Proposition 9.2.4 DLE Algorithm 9.2.3 Convergence Let $tol = tol1 = 0$ and assume that (9.14) implies $\alpha \geq \bar{\alpha}$. Then either Algorithm 9.2.3 terminates at an i such that u^i solves (9.5) and consequently $y^i = B'u^i - \alpha d$ solves the primal linear program (9.1), or any accumulation point $\bar{u}$ of the sequence $\{u^i\}$ generated by Algorithm 9.2.3 solves the exterior penalty problem (9.5) and the corresponding $\bar{y} = B'\bar{u} - \alpha d$ solves the primal linear program (9.1), provided the matrices of the sequence $\{(BB' + P^i)\}$ are nonsingular.

Proof If Algorithm 9.2.3 terminates at some i, then $u^{i+1} = u^i$ and $B'\big((BB'+P^i)\backslash(Bd)\big)-d = 0$. Hence $P^{i+1} = P^i$, $u^i = (BB' + P^i)\backslash(b + \alpha Bd)$, and (9.14) holds for $u = u^i$ and $P = P^i$. Thus the pair (u^i, P^i) solves (9.8). Consequently, u^i solves the dual penalty minimization problem (9.5), and since, by assumption, condition (9.14) implies that $\alpha \geq \bar{\alpha}$, it follows that $y^i = B'u^i - \alpha d$ solves the primal linear program (9.1). □

For the case when $\{i\}$ does not terminate, for each accumulation point $\bar{u}$ of $\{u^i\}$, there exists a subsequence $\{u^{i_j}, P^{i_j}\}$ of $\{u^i, P^i\}$ for which $\{P^{i_j}\}$ is constant and equal to $\bar{P}$, and $\{u^{i_j}, P^{i_j}\}$ converges to $(\bar{u}, \bar{P})$. This is because $\{P^i\}$ is a sequence of diagonal matrices of

ones and zeros with a finite number of values. We will now show that $\nabla f(\bar{u}) = 0$ and consequently $\bar{u}$ minimizes the dual exterior penalty function (9.15).

From (9.8) we have that:

$$\nabla f(u^i) = (BB' + P^i)u^i - (\alpha Bd + b). \tag{9.23}$$

By the assumption that $(BB' + P^i)$ is nonsingular, it follows from (9.21), before the Armijo stepsize is taken, that:

$$u^{i+1} = (BB' + P^i)^{-1}(\alpha Bd + b). \tag{9.24}$$

Define now the direction t^i as:

$$t^i := u^{i+1} - u^i = (BB' + P^i)^{-1}(\alpha Bd + b) - u^i. \tag{9.25}$$

Consequently, after some algebra we have that:

$$-\nabla f(u^i)'t^i = u^{i'}(BB' + P^i)u^i - 2u^{i'}(\alpha Bd + b) + (\alpha Bd + b)'(BB' + P^i)^{-1}(\alpha Bd + b). \tag{9.26}$$

We will now show that the right hand side of (9.26) is a forcing function of $\nabla f(u^i)$, that is, it is nonnegative and if it approaches zero then $\|\nabla f(u^i)\|$ approaches zero. Define the quantities a and M as:

$$a = (\alpha Bd + b),\ M = (BB' + P^i). \tag{9.27}$$

Then, Equation (9.26) reduces to:

$$-\nabla f(u^i)'t^i = u^{i'}Mu^i - 2u^{i'}a + a'M^{-1}a. \tag{9.28}$$

Define now

$$h = M^{-\frac{1}{2}}a,\ g = M^{\frac{1}{2}}u^i. \tag{9.29}$$

Hence

$$\begin{aligned} -\nabla f(u^i)'t^i &= g'g - 2g'M^{-\frac{1}{2}}M^{\frac{1}{2}}h + h'h = \|g - h\|^2 \\ &= \|M^{\frac{1}{2}}u^i - M^{-\frac{1}{2}}a\|^2 \\ &= \|M^{-\frac{1}{2}}(Mu^i - a)\|^2 \\ &= \|M^{-\frac{1}{2}}\big((BB' + P^i)u^i - (\alpha Bd + b)\big)\|^2 \\ &= \|(BB' + P^i)^{-\frac{1}{2}}\nabla f(u^i)\|^2. \end{aligned} \tag{9.30}$$

The last term of (9.30) is a forcing function of $\nabla f(u^i)$. Hence, by [70, Theorem 2.1] it follows that for each accumulation point $\bar{u}$, $\nabla f(\bar{u}) = 0$, and for a subsequence $\{u^{i_j}, P^{i_j}\}$ with constant P^{i_j} that converges to $(\bar{u}, \bar{P})$, the pair $(\bar{u}, \bar{P})$ solves (9.8) and $B'\big((BB' + \bar{P})\backslash(Bd)\big) - d = 0$. Consequently, $\bar{u}$ solves the minimization problem (9.5) and since (9.14) implies that $\alpha \geq \bar{\alpha}$, it follows that $\bar{y} = B'\bar{u} - \alpha d$ solves the primal linear program (9.1). □

Before presenting our computational results we state results corresponding to those of Section 9.1 for linear programs with explicit nonnegative variable constraints. We do this because it will allow us to find an exact solution for the linear program (9.2), that is the dual of (9.1), which will allow us to efficiently obtain solutions of (9.1) when $m >> \ell$.

9.3 Linear programs with nonnegative variables

In solving the linear program (9.1) for the case when $m >> \ell$, the algorithms of Section 9.2 are quite inefficient because these algorithms involve the inversion of very large m-by-m matrices that contain BB' in (9.19) or (9.21). To avoid this difficulty we shall instead first solve the dual linear program (9.2) exactly by solving an exterior penalty problem for the primal linear program (9.1). As shown below, this involves the inversion of much smaller ℓ-by-ℓ matrices. Once an exact dual solution u is obtained an exact primal solution is obtained by solving the following unconstrained minimization problem:

$$\min_{y \in R^\ell} \frac{1}{2}\|B^1 y - b^1\|^2 + \frac{1}{2}\|(-B^2 y + b^2)_+\|^2 + \frac{1}{2}(d'y - b'u)^2, \tag{9.31}$$

where the superscripts 1 and 2 denote components of the primal constraints $By \geq b$ corresponding to optimal dual variable components $u^1 > 0$ and $u^2 = 0$ respectively. It is easy to check that solving (9.31) involves the inversion of much smaller ℓ-by-ℓ matrices, as will be the case for the computations below.

In order to develop the above approach, we begin with the solvable dual linear program (9.2) with nonnegative variables (hence the title of this section) and consider the classical exterior penalty function for the corresponding primal problem (9.1):

$$\min_{y \in R^{\ell}} \epsilon d'y + \frac{1}{2}\|(-By + b)_+\|^2. \tag{9.32}$$

Dividing by ϵ^2 and letting:

$$\frac{y}{\epsilon} \rightarrow y \; and \; \frac{1}{\epsilon} \rightarrow \alpha, \tag{9.33}$$

the penalty problem (9.32) becomes:

$$\min_{y \in R^{\ell}} d'y + \frac{1}{2}\|(-By + \alpha b)_+\|^2. \tag{9.34}$$

Applying Proposition 1 of [75] to the primal exterior penalty problem (9.34), we get:

Proposition 9.3.1 Exact Dual Solution Computation Let the dual LP (9.2) be solvable. Then the primal exterior penalty problem (9.34) is solvable for all $\alpha > 0$. For any $\alpha \geq \bar{\alpha}$ for some finite $\bar{\alpha} > 0$, any solution y of (9.34) generates an exact solution to dual LP (9.2) as follows:

$$u = (-By + \alpha b)_+. \tag{9.35}$$

In addition, this u minimizes:

$$\|u\|^2, \tag{9.36}$$

over the solution set of the dual LP (9.2).

Without giving proofs of results similar to those of Section 9.1, we now state results corresponding to Lemma 9.1.2 and Proposition 9.1.3.

Lemma 9.3.2 Optimality Condition for Solving the Primal Exterior Penalty (9.34) A necessary and sufficient condition for u to be a solution of the exterior penalty function (9.34) is that:

$$d - B'Q(-By + \alpha b) = 0, \tag{9.37}$$

where $Q \in R^{\ell \times \ell}$ is a diagonal matrix of ones and zeros defined as follows:

$$Q = diag\Big(sign\big((-By + \alpha b)_+\big)\Big). \tag{9.38}$$

We note that Equation (9.37) is solvable for any $\alpha > 0$ whenever the linear program (9.1) is solvable. Hence, for such a case an explicit solution to (9.37) is given as follows:

$$y = (B'QB)\backslash(\alpha B'Qb - d). \tag{9.39}$$

If we now substitute for y in (9.35) we obtain the following expression for an exact solution u of the dual linear program (9.2) when $\alpha \geq \bar{\alpha}$:

$$u = \Big(B\Big((B'QB)\backslash d\Big) - \alpha\Big(B\big((B'QB)\backslash(B'Qb)\big) - b\Big)\Big)_+ \tag{9.40}$$

We note that u depends on α through the explicit term α in the above Equation (9.40), as well as through the dependence of Q on y in the definition (9.38), and the dependence of y on α in (9.39). Hence, in order for u to be independent of α, as would be the case when $\alpha \geq \bar{\alpha}$, we have the following result which follows directly from the expression (9.40) for u being a constant function of α: for a constant Q, the subgradient [106] of the expression for u (9.40) with respect to α vanishes as stated in (9.42) below.

Proposition 9.3.3 Sufficient Conditions for Independence on α The optimal solution u given by (9.40) for a solvable dual linear program (9.2) is independent of α if:

$$\textit{The diagonal matrix } Q \text{ is independent of } \alpha \tag{9.41}$$

and

$$diag\Big(sign(u)\Big)\Big(B\big((B'QB)\backslash(B'Qb)\big) - b\Big) = 0, \tag{9.42}$$

where u is defined in (9.40).

We turn now to our computational results based on Algorithms 9.2.1 and 9.2.3.

9.4 Computational results

To illustrate the effectiveness of our approach, we report results on randomly generated linear programs. In these linear programs we selected the number of variables, constraints, nonzero coefficients, and generated optimal values for the primal and dual variables. We

compare our Newton LP approach (Algorithm 9.2.1) and direct linear equation (DLE) approach (Algorithm 9.2.3) with CPLEX 9.0 [48], a state-of-the-art linear programming package.

All methods were run in a random order on each linear program. The reported results were obtained on an Intel Pentium 4 processor with 1 gigabyte of RAM running CentOS Linux 4.5 and MATLAB 7.3. For the DLE method, we set $\delta = 1e{-}8$, $tol = 1e{-}16$, $tol1 = 1e{-}3$, $imax = 500$, and $\alpha = 100$. For the Newton LP method, we set $\delta = 1e{-}6$, $tol = 1e{-}3$, $imax = 5000$, and $\alpha = 100$. For problems with more constraints than variables, we used $imax = 500$ for the Newton LP method. If the sufficient condition was not less than tol after the Newton method terminated, we tried again with 10α. This case occurred only a few times over all the experiments we report. We used only stepless methods. Although these parameters effect the solution time and accuracy, we did not experiment much with different settings and chose these particular values because they gave acceptable results on our experiments. Default parameters were used for CPLEX.

The sufficient conditions (9.14) and (9.42) play an important role in our algorithms in choosing the size of the penalty parameter α and as a stopping criterion. In fact, in all our tests the penalty parameter α was chosen sufficiently large so that it satisfied the appropriate sufficient condition (9.14) or (9.42). In every experiment we report, the relative difference between the returned and the true objective value, maximum constraint violation, and sufficient condition was less then $1e{-}3$, and in most experiments these values were of the order of $1e{-}7$ or smaller.

9.4.1 Square linear programs

Figure 9.1 shows results for our proposed Newton LP Algorithm 9.2.1 and DLE Algorithm 9.2.3 formulations, as well as CPLEX 9.0. Each point on the graph represents the average elapsed seconds over 10 square linear programs randomly generated with density 0.1. The standard deviations are too small to show on the plot, and our proposed approaches are both clearly faster than CPLEX for problems with more than 2000 variables and constraints. In

general, CPLEX returned more accurate solutions than our approaches. We used the barrier method for CPLEX, which was faster than both the primal and dual simplex algorithms for these problems. Note that both of our approaches are more than twice as fast as CPLEX for problems with 5000 variables and 5000 constraints.

9.4.2 Rectangular linear programs

Table 9.1 shows results for our proposed Newton LP Algorithm 9.2.1 and DLE Algorithm 9.2.3 methods, as well as CPLEX 9.0. We applied Algorithms 9.2.1 and 9.2.3 directly for problems with more variables than constraints, and used the technique described in Section 9.3 for problems with more constraints than variables. We used the primal simplex method of CPLEX for problems with more variables than constraints, and the dual simplex method of CPLEX for problems with more constraints than variables. The results in Table 9.1 show that our approach can solve problems with up to a million variables or constraints, although not as quickly as the commercially available CPLEX solvers in all cases.

We note that our proposed DLE Algorithm 9.2.3 method, which is implemented in 54 lines of MATLAB code, gives times which are faster than CPLEX for problems with more variables than constraints. In general, our proposed Newton method is slower on these problems, but is still able to give solutions in about half a minute. For problems with more constraints than variables, CPLEX is noticeably faster. Nevertheless, our proposed approaches are able to give accurate solutions to problems with as many as one million constraints and one hundred variables within a few minutes.

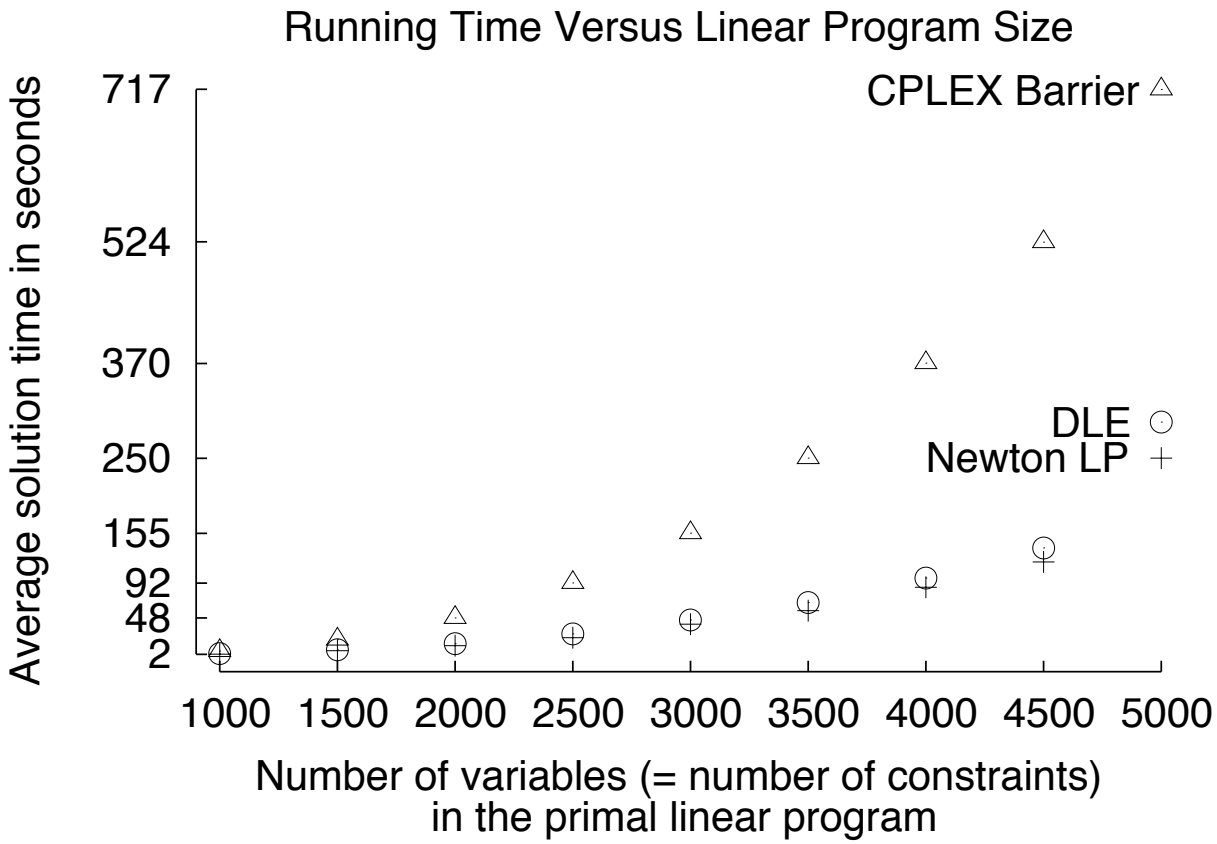


Figure 9.1 Average running times of our proposed approaches and the CPLEX 9.0 barrier method. Our Newton LP Algorithm 9.2.1 method is represented by '+', our DLE Algorithm 9.2.3 method is represented by '◯', and CPLEX is represented by '△'. Each point is the average of 10 randomly generated square linear programs.

Table 9.1 Average running times of our proposed approaches and the CPLEX 9.0 simplex method. Ten linear programs were randomly generated for each number of variables and constraints, and the average solution time in seconds is given with the standard deviation in parentheses for each algorithm. Primal methods were used for problems with more variables than constraints, and dual methods were used for problems with more constraints than variables.

Constraints	Variables	CPLEX	Newton LP	DLE
100	1,000	0.0246 (0.0006)	0.0438 (0.0003)	0.0251 (0.0001)
100	10,000	0.0522 (0.0001)	0.2024 (0.0004)	0.0682 (0.0001)
100	100,000	0.837 (0.003)	3.219 (0.012)	0.905 (0.004)
100	1,000,000	17.9 (0.6)	29.1 (0.2)	9.3 (0.2)
1,000	100	0.0969 (0.0025)	0.1391 (0.0027)	0.1032 (0.0004)
10,000	100	0.267 (0.001)	0.970 (0.012)	0.469 (0.003)
100,000	100	2.96 (0.05)	13.02 (0.12)	5.50 (0.04)
1,000,000	100	44.8 (0.3)	173.5 (2.2)	70.9 (0.4)

Chapter 10

Conclusion and Outlook

We have proposed optimization-based approaches to six important machine learning problems. These approaches result in new techniques that can be used to solve these classes of problems effectively and efficiently. We have also given conditions which allow us to give a precise termination condition for two fast algorithms for linear programming. In this chapter, we give some concluding remarks for each of the new techniques and some avenues of future research.

10.1 Knowledge-based kernel approximation

We presented a knowledge-based formulation of a nonlinear kernel SVM approximation. The approximation is obtained using a linear programming formulation with any nonlinear symmetric kernel and with no positive semidefiniteness (Mercer) condition assumed. The issues associated with sampling the knowledge sets in order to generate function values (that is, a matrix A and a corresponding vector y to be used in conventional approximation formulations) in situations where there are no conventional data points constitute an interesting topic for future research. Additional future work includes refinement of prior knowledge, such as continuing the work in [66], and applications to medical problems, computer vision, microarray gene classification, and efficacy of drug treatment, all of which have prior knowledge available.

10.2 Nonlinear knowledge in kernel machines

We have proposed a unified, computationally effective framework for handling general nonlinear prior knowledge in kernel approximation and kernel classification problems. We have reduced such prior knowledge to easily implemented linear inequality constraints in a linear programming formulation, and as linear equality constraints in a proximal formulation. We have demonstrated the effectiveness of our approach on two synthetic problems, on two important real-world problems arising in breast cancer prognosis, and on problems in which prior knowledge was generated from ordinary datasets. Possible future extensions are to consider different ways of solving the semi-infinite programs [37, 42], refinements to the prior knowledge as in [66], and even more general prior knowledge, such as that where the right hand side of the implications (3.2) and (3.6) are replaced by very general nonlinear inequalities involving the kernel function (1.4). Another important avenue of future work is to construct an interface which allows users to easily specify arbitrary regions to be used as prior knowledge.

10.3 Privacy-preserving classification via random kernels

We have proposed privacy-preserving SVM classifiers for vertically and for horizontally partitioned data based on random matrices. Each entity possesses either a different set of input features or a different set of individual rows used collectively to generate the SVM classifier. The proposed approaches use all the privately held data in a form that does not reveal what that data is. Computational comparisons indicate that the accuracy of our proposed approaches is comparable to full and reduced data classifiers. Furthermore, a marked improvement of accuracy is obtained by the privacy-preserving SVM compared to classifiers generated by each entity using its own data alone. Hence, by making use of a random kernel, the proposed approaches succeed in generating an accurate classifier based on privately held data without revealing any of that data.

10.4 Feature-selecting *k*-median algorithm

The feature-selecting k-median algorithm (FSKM) is a fast and efficient method for selecting features of unlabeled datasets that gives clusters that are similar to clusters obtained in the full dimensional space of the original data. In addition, features selected by FSKM may be useful for labeled feature selection. For example, the six features selected by FSKM for the Star/Galaxy-Bright dataset gave a classification error of 3.78% compared with 9.83% error with the full 14 features. Using the features chosen by FSKM could eliminate the costly search for the best 6 out of 14 features. Exhaustively searching for those 6 features would require $\binom{14}{6} = 3003$ k-median runs as opposed to our 9 k-median runs.

It is hoped that future research into the theory used here to justify the feature selection procedure of FSKM will have further application to other algorithms of machine learning and data mining.

10.5 Feature selection for nonlinear kernel support vector machines

We have presented a new approach to feature selection for nonlinear SVM classifiers for a completely arbitrary kernel. Our approach is formulated as an easily implementable mixed-integer program and solved efficiently by alternating between a linear program to compute the continuous parameter values of the classifier and successive sweeps through the objective function to update the integer variables representing the presence or absence of each feature. This procedure converges to a local minimum that minimizes both the usual SVM objective and the number of input space features used. Our results on two publicly available datasets and synthetic NDCC data show that our approach efficiently learns accurate nonlinear classifiers with reduced numbers of features. Extension of RFSVM to regression problems, further evaluation of RFSVM on datasets with very large numbers of features, use of different strategies to update the integer variables, and procedures for automatically choosing a value of σ for a desired percentage of feature reduction are important avenues of future research.

10.6 Generalized eigenvalue proximal support vector machines

We have proposed a novel approach to classification problems that relaxes the universal requirement that bounding or proximal planes generated by SVMs be parallel in the input space for linear kernel classifiers or in the higher dimensional feature space for nonlinear kernel classifiers. Each of our proposed nonparallel proximal planes are easily obtained using a single MATLAB command that solves the classical generalized eigenvalue problem. Classification accuracy results are comparable to those of classical support vector classification algorithms, and in some cases they are better. Also, in our experience the generalized eigenvalue problem can be solved more quickly than the optimization algorithm needed for SVM-Light [49]. The simple program formulation, computational efficiency, and accuracy of the generalized eigenvalue proximal support vector machine (GEPSVM) on real-world data indicate that it is an effective algorithm for classification. More recently, Guarracino et al. have studied a technique similar to GEPSVM but using a different regularization term so that only the solution of one generalized eigenvalue problem is required [40]. Analysis of the statistical properties of GEPSVM and extensions to multicategory classification are promising areas of future research.

10.7 Multiple-Instance Classification

We have introduced a mathematical programming formulation of the multiple-instance problem that has a linear objective with linear and bilinear constraints. Our mathematical program can be solved with a succession of fast linear programs that, in our experience, converges in a few iterations to a local solution. Results on previously published datasets indicate that our approach is very effective at finding linear classifiers. It can also be easily extended to finding nonlinear classifiers with potential high accuracy through the use of kernels. Improvements in the mathematical programming formulation and evaluation on more datasets are promising avenues of future research.

10.8 Exactness conditions for a convex differentiable exterior penalty for linear programming

We have presented sufficient conditions for a classical dual exterior penalty function of a linear program to provide an exact solution to a primal linear program. These conditions allow us to give a precise termination condition to a Newton algorithm for linear programming introduced in [75], and also to a new direct method based on solving the necessary and sufficient optimality condition (9.8). Experimental results indicate that both approaches are able to efficiently obtain accurate solutions on randomly generated linear programs with different numbers of constraints and variables. For some linear programs, our approaches implemented in MATLAB were as much as twice as fast as the commercial linear programming package CPLEX 9.0.

It is possible that our approaches could be extended to handle problems which are too big to fit in memory, so long as either the number of constraints or the number of variables is not too big, and the matrix multiplications are done externally. Other opportunities for future work include applying both approaches to real-world linear programs, exploring the use of direct methods for other optimization problems, and further improving the performance of our approaches.

10.9 Outlook

Mathematical programming is a very useful framework that can be used to solve important problems in machine learning and data mining. We have investigated techniques that deal with prior knowledge, privacy preservation, feature selection, multisurface classifiers, and multiple instance learning. We have also shown empirical results that demonstrate their utility. We have also described two methods for solving linear programs, which have proven to arise in the solution of several of the above problems. In addition to the approaches described above, interesting areas of future research include investigations into the statistical properties of the given algorithms, and the use of mathematical programming

techniques to generate novel formulations and solution techniques for these and other problems in machine learning and data mining.

LIST OF REFERENCES

[1] S. Andrews, I. Tsochantaridis, and T. Hofmann. Support vector machines for multiple-instance learning. In Suzanna Becker, Sebastian Thrun, and Klaus Obermayer, editors, *Advances in Neural Information Processing Systems 15*, pages 561–568. MIT Press, Cambridge, MA, October 2003.

[2] P. Auer. On learning from multi-instance examples: Empirical evaluation of a theoretical approach. In *Proceedings 14th International Conference on Machine Learning*, pages 21–29. Morgan Kaufmann, 1997.

[3] G. Baudat and F. Anouar. Kernel-based methods and function approximation. In *International Joint Conference on Neural Networks*, pages 1244–1249, Washington, D.C., 2001.

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

[5] 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.

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

[7] R. R. Bouckaert. Choosing between two learning algorithms based on calibrated tests. In *Proceedings 20th International Conference on Machine Learning*, pages 51–58, 2003.

[8] P. S. Bradley and O. L. Mangasarian. Feature selection via concave minimization and support vector machines. In J. Shavlik, editor, *Proceedings 15th International Conference on Machine Learning*, pages 82–90, San Francisco, California, 1998. Morgan Kaufmann. ftp://ftp.cs.wisc.edu/math-prog/tech-reports/98-03.ps.

[9] P. S. Bradley, O. L. Mangasarian, and W. N. Street. Clustering via concave minimization. In M. C. Mozer, M. I. Jordan, and T. Petsche, editors, *Advances in Neural Information Processing Systems -9-*, pages 368–374, Cambridge, MA, 1997. MIT Press. ftp://ftp.cs.wisc.edu/math-prog/tech-reports/96-03.ps.

[10] K. Chen and L. Liu. Privacy preserving data classification with rotation perturbation. In *Proceedings of the Fifth International Conference of Data Mining (ICDM'05)*, pages 589–592. IEEE, 2005.

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

[12] C. Cortes and V. Vapnik. Support vector networks. *Machine Learning*, 20:273–279, 1995.

[13] M. Craven and J. Shavlik. Learning symbolic rules using artificial neural networks. In *Proceedings 10th International Conference on Machine Learning*, pages 73–80, Amherst, MA, 1993. Morgan Kaufmann.

[14] S. Dasgupta. Learning mixtures of Gaussians. In *IEEE Symposium on Foundations of Computer Science (FOCS) 1999*, pages 634–644, 1999.

[15] J. Demšar. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research*, 7:1–30, 2006.

[16] T. G. Dietterich, R. H. Lathrop, and T. Lozano-Perez. Solving the multiple-instance problem with axis-parallel rectangles. *Artificial Intelligence*, 89:31–71, 1998.

[17] H. Drucker, C. J. C. Burges, L. Kaufman, A. Smola, and V. Vapnik. Support vector regression machines. In M. C. Mozer, M. I. Jordan, and T. Petsche, editors, *Advances in Neural Information Processing Systems -9-*, pages 155–161, Cambridge, MA, 1997. MIT Press.

[18] W. Du, Y. Han, and S. Chen. Privacy-preserving multivariate statistical analysis: Linear regression and classification. In *Proceedings of the Fourth SIAM International Conference on Data Mining*, pages 222–233, 2004. http://citeseer.ist.psu.edu/du04privacypreserving.html.

[19] O. J. Dunn. Multiple comparisons among means. *Journal of the American Statistical Association*, 56:52–64, 1961.

[20] T. Evgeniou, M. Pontil, and T. Poggio. Regularization networks and support vector machines. *Advances in Computational Mathematics*, 13:1–50, 2000.

[21] T. Evgeniou, M. Pontil, and T. Poggio. Regularization networks and support vector machines. In A. Smola, P. Bartlett, B. Schölkopf, and D. Schuurmans, editors, *Advances in Large Margin Classifiers*, pages 171–203, Cambridge, MA, 2000. MIT Press.

[22] F. Facchinei. Minimization of SC^1 functions and the Maratos effect. *Operations Research Letters*, 17:131–137, 1995.

[23] X. Feng and Z. Zhang. The rank of a random matrix. *Applied Mathematics and Computation*, 185:689–694, 2007.

[24] J. Forrest, D. de la Nuez, and R. Lougee-Heimer. *CLP User Guide*, 2004. http://www.coin-or.org/Clp/userguide/index.html.

[25] M. Friedman. The use of ranks to avoid the assumption of normality implicit in the analysis of variance. *Journal of the American Statistical Association*, 32:675–701, 1937.

[26] H. Fröhlich and A. Zell. Feature subset selection for support vector machines by incremental regularized risk minimization. In *International Joint Conference on Neural Networks (IJCNN)*, volume 3, pages 2041–2046, 2004.

[27] G. Fung and O. L. Mangasarian. Proximal support vector machine classifiers. In F. Provost and R. Srikant, editors, *Proceedings KDD-2001: Knowledge Discovery and Data Mining, August 26-29, 2001, San Francisco, CA*, pages 77–86, New York, 2001. Association for Computing Machinery. ftp://ftp.cs.wisc.edu/pub/dmi/tech-reports/01-02.ps.

[28] G. Fung and O. L. Mangasarian. Semi-supervised support vector machines for unlabeled data classification. *Optimization Methods and Software*, 15:29–44, 2001. ftp://ftp.cs.wisc.edu/pub/dmi/tech-reports/99-05.ps.

[29] G. Fung and O. L. Mangasarian. Incremental support vector machine classification. In H. Mannila R. Grossman and R. Motwani, editors, *Proceedings Second SIAM International Conference on Data Mining, Arlington, Virginia, April 11-13,2002*, pages 247–260, Philadelphia, 2002. SIAM. ftp://ftp.cs.wisc.edu/pub/dmi/tech-reports/01-08.ps.

[30] G. Fung and O. L. Mangasarian. A feature selection Newton method for support vector machine classification. *Computational Optimization and Applications*, 28(2):185–202, July 2004. ftp://ftp.cs.wisc.edu/pub/dmi/tech-reports/02-03.ps.

[31] G. Fung and O. L. Mangasarian. Multicategory proximal support vector machine classifiers. *Machine Learning*, 59:77–97, 2005. University of Wisconsin Data Mining Institute Technical Report 01-06, July 2001, ftp://ftp.cs.wisc.edu/pub/dmi/tech-reports/01-06.ps.

[32] G. Fung, O. L. Mangasarian, and J. Shavlik. Knowledge-based nonlinear kernel classifiers. Technical Report 03-02, Data Mining Institute, Computer Sciences Department, University of Wisconsin, Madison, Wisconsin, March 2003. ftp://ftp.cs.wisc.edu/pub/dmi/tech-reports/03-02.ps. *Conference on Learning Theory (COLT 03) and Workshop on Kernel Machines*, Washington D.C., August 24-27, 2003. Proceedings edited by M. Warmuth and B. Schölkopf, Springer Verlag, Berlin, 2003, 102-113.

[33] G. Fung, O. L. Mangasarian, and J. Shavlik. Knowledge-based nonlinear kernel classifiers. In M. Warmuth and B. Schölkopf, editors, *Conference on Learning Theory (COLT 03) and Workshop on Kernel Machines*, pages 102–113, Berlin, 2003. Springer–Verlag. ftp://ftp.cs.wisc.edu/pub/dmi/tech-reports/03-02.ps.

[34] G. Fung, O. L. Mangasarian, and J. Shavlik. Knowledge-based support vector machine classifiers. In Suzanna Becker, Sebastian Thrun, and Klaus Obermayer, editors, *Advances in Neural Information Processing Systems 15*, pages 521–528. MIT Press, Cambridge, MA, October 2003. ftp://ftp.cs.wisc.edu/pub/dmi/tech-reports/01-09.ps.

[35] G. Fung, O. L. Mangasarian, and A. Smola. Minimal kernel classifiers. *Journal of Machine Learning Research*, pages 303–321, 2002. University of Wisconsin Data Mining Institute Technical Report 00-08, November 200, ftp://ftp.cs.wisc.edu/pub/dmi/tech-reports/00-08.ps.

[36] Thomas Gartner, Peter A. Flach, Adam Kowalczyk, and Alex J. Smola. Multi-instance kernels. In Claude Sammut and Achim Hoffmann, editors, *Proceedings 19th International Conference on Machine Learning*, pages 179–186. Morgan Kaufmann, July 2002.

[37] M. A. Goberna and M. A. López. *Linear Semi-Infinite Optimization*. John Wiley, New York, 1998.

[38] C. Gold, A. Holub, and P. Sollich. Bayesian approach to feature selection and parameter tuning for support vector machine classifiers. *Neural Networks*, 18(5-6):693–701, 2005.

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

[40] M.R. Guarracino, C. Cifarelli, O. Seref, and P. Pardalos. A classification method based on generalized eigenvalue problems. *Optimization Methods and Software*, 22(1):73–81, 2007.

[41] I. Guyon, J. Weston, S. Barnhill, and V. Vapnik. Gene selection for cancer classification using support vector machines. *Machine Learning*, 46(1-3):389–422, 2002.

[42] R. Hettich and K. O. Kortanek. Semi-infinite programming: Theory, methods, and applications. *SIAM Review*, 35(3):380–429, 1993.

[43] J.-B. Hiriart-Urruty, J. J. Strodiot, and V. H. Nguyen. Generalized Hessian matrix and second-order optimality conditions for problems with C^{L1} data. *Applied Mathematics and Optimization*, 11:43–56, 1984.

[44] T. K. Ho and E. M. Kleinberg. Checkerboard dataset, 1996. http://www.cs.wisc.edu/math-prog/mpml.html.

[45] R. A. Horn and C. R. Johnson. *Matrix Analysis*. Cambridge University Press, Cambridge, England, 1985.

[46] C.-H. Huang, Y.-J. Lee, D.K.J. Lin, and S.-Y. Huang. Model selection for support vector machines via uniform design. In *Machine Learning and Robust Data Mining of Computational Statistics and Data Analysis*, Amsterdam, 2007. Elsevier Publishing Company. http://dmlab1.csie.ntust.edu.tw/downloads/papers/UD4SVM013006.pdf.

[47] S.Y. Huang and Y.-J. Lee. Theoretical study on reduced support vector machines. Technical report, National Taiwan University of Science and Technology, Taipei, Taiwan, 2004. yuh-jye@mail.ntust.edu.tw.

[48] ILOG, Incline Village, Nevada. *ILOG CPLEX 9.0 User's Manual*, 2003. http://www.ilog.com/products/cplex/.

[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, Cambridge, MA, 1999. MIT Press.

[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] M. Kojima, S. Mizuno, T. Noma, and A. Yoshise. *A Unified Approach to Interior Point Algorithms for Linear Complementarity Problems*. Springer-Verlag, Berlin, 1991.

[52] G. Kuhlmann, P. Stone, R. Mooney, and J. Shavlik. Guiding a reinforcement learner with natural language advice: Initial results in robocup soccer. In *Proceedings of the AAAI Workshop on Supervisory Control of Learning and Adaptive Systems*, San Jose, CA, 2004.

[53] Q. V. Le, A. J. Smola, and T. Gärtner. Simpler knowledge-based support vector machines. In *Proceedings 23rd International Conference on Machine Learning, Pittsburgh, PA, 2006*, 2006. http://www.icml2006.org/icml2006/technical/accepted.html.

[54] Y.-J. Lee and S.Y. Huang. Reduced support vector machines: A statistical theory. *IEEE Transactions on Neural Networks*, 18:1–13, 2007.

[55] Y.-J. Lee and O. L. Mangasarian. RSVM: Reduced support vector machines. In *Proceedings First SIAM International Conference on Data Mining, Chicago, April 5-7, 2001, CD-ROM*, 2001. ftp://ftp.cs.wisc.edu/pub/dmi/tech-reports/00-07.pdf.

[56] Y.-J. Lee, O. L. Mangasarian, and W. H. Wolberg. Breast cancer survival and chemotherapy: a support vector machine analysis. In *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, volume 55, pages 1–10. American Mathematical Society, 2000. ftp://ftp.cs.wisc.edu/pub/dmi/tech-reports/99-10.ps.

[57] Y.-J. Lee, O. L. Mangasarian, and W. H. Wolberg. Survival-time classification of breast cancer patients. Technical Report 01-03, Data Mining Institute, Computer Sciences Department, University of Wisconsin, Madison, Wisconsin, March 2001. ftp://ftp.cs.wisc.edu/pub/dmi/tech-reports/01-03.ps. *Computational Optimization and Applications* 25, 2003, 151-166.

[58] Y.-J. Lee, O. L. Mangasarian, and W. H. Wolberg. Survival-time classification of breast cancer patients. *Computational Optimization and Applications*, 25:151–166, 2003. ftp://ftp.cs.wisc.edu/pub/dmi/tech-reports/01-03.ps.

[59] M. Litzkow and M. Livny. Experience with the condor distributed batch system. In *Proceedings IEEE Workshop on Experimental Distributed Systems*, pages 97–101, Hunstville, AL, October 1990. IEEE Compter Society Press.

[60] K. Liu, H. Kargupta, and J. Ryan. Random projection-based multiplicative data perturbation for privacy preserving distributed data mining. *IEEE Transactions on Knowledge and Data Engineering*, 18:92–106, 2006.

[61] L. Liu, J. Wang, Z. Lin, and J. Zhang. Wavelet-based data distortion for privacy-preserving collaborative analysis. Technical Report 482-07, Department of Computer Science, University of Kentucky, Lexington, KY 40506, 2007. http://www.cs.uky.edu/ jzhang/pub/MINING/lianliu1.pdf.

[62] P. M. Long and L. Tan. PAC learning axis aligned rectangles with respect to product distributions from multiple instance examples. *Machine Learning*, 30(1):7–22, 1998.

[63] R. Maclin and J. Shavlik. Creating advice-taking reinforcement learners. *Machine Learning*, 22, 1996.

[64] R. Maclin, J. Shavlik, L. Torrey, T. Walker, and E. Wild. Giving advice about preferred actions to reinforcement learners via knowledge-based kernel regression. In *Proceedings 20th National Conference on Artificial Intelligence*, pages 819–824, 2005.

[65] R. Maclin, J. Shavlik, T. Walker, and L. Torrey. A simple and effective method for incorporating advice into kernel methods. In *Proceedings 21st National Conference on Artificial In telligence*, 2006.

[66] R. Maclin, E. Wild, J. Shavlik, L. Torrey, and T. Walker. Refining rules incorporated into knowledge-based support vector learners via successive linear programming. In *Proceedings of the Twenty-Second Conference on Artificial Intelligence*, Vancouver, British Columbia, July 2007.

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

[68] O. L. Mangasarian. Least norm solution of non–monotone complementarity problems. In *Functional Analysis, Optimization and Mathematical Economics*, pages 217–221. Oxford University Press, New York, 1990.

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

[70] O. L. Mangasarian. Parallel gradient distribution in unconstrained optimization. *SIAM Journal on Control and Optimization*, 33(6):1916–1925, 1995. ftp://ftp.cs.wisc.edu/tech-reports/reports/1993/tr1145.ps.

[71] O. L. Mangasarian. Arbitrary-norm separating plane. *Operations Research Letters*, 24:15–23, 1999. ftp://ftp.cs.wisc.edu/math-prog/tech-reports/97-07r.ps.

[72] 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.

[73] O. L. Mangasarian. Data mining via support vector machines, July 23-27, 2001. http://ftp.cs.wisc.edu/math-prog/talks/ifip3tt.ppt.

[74] O. L. Mangasarian. A finite Newton method for classification problems. *Optimization Methods and Software*, 17:913–929, 2002. ftp://ftp.cs.wisc.edu/pub/dmi/tech-reports/01-11.ps.

[75] O. L. Mangasarian. Exact 1-Norm support vector machines via unconstrained convex differentiable minimization. Technical Report 05-03, Data Mining Institute, Computer Sciences Department, University of Wisconsin, Madison, Wisconsin, August 2005. ftp://ftp.cs.wisc.edu/pub/dmi/tech-reports/05-03.ps. Journal of Machine Learning Research 7, 2006, 1517-1530.

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

[77] O. L. Mangasarian and D. R. Musicant. Lagrangian support vector machines. *Journal of Machine Learning Research*, 1:161–177, 2001. ftp://ftp.cs.wisc.edu/pub/dmi/tech-reports/00-06.ps.

[78] O. L. Mangasarian and D. R. Musicant. Large scale kernel regression via linear programming. *Machine Learning*, 46:255–269, 2002. ftp://ftp.cs.wisc.edu/pub/dmi/tech-reports/99-02.ps.

[79] O. L. Mangasarian, J. W. Shavlik, and E. W. Wild. Knowledge-based kernel approximation. *Journal of Machine Learning Research*, 5:1127–1141, 2004. ftp://ftp.cs.wisc.edu/pub/dmi/tech-reports/03-05.ps.

[80] O. L. Mangasarian, W. N. Street, and W. H. Wolberg. Breast cancer diagnosis and prognosis via linear programming. *Operations Research*, 43(4):570–577, July-August 1995.

[81] O. L. Mangasarian and M. E. Thompson. Massive data classification via unconstrained support vector machines. *Journal of Optimization Theory and Applications*, 131:315–325, 2006. ftp://ftp.cs.wisc.edu/pub/dmi/tech-reports/06-01.pdf.

[82] O. L. Mangasarian and E. W. Wild. Feature selection in k-median clustering. Technical Report 04-01, Data Mining Institute, Computer Sciences Department, University of Wisconsin, Madison, Wisconsin, January 2004. SIAM International Conference on Data Mining, Workshop on Clustering High Dimensional Data and Its Applications, April 24, 2004, La Buena Vista, FL, Proceedings Pages 23-28. http://www.siam.org/meetings/sdm04. ftp://ftp.cs.wisc.edu/pub/dmi/tech-reports/04-01.pdf.

[83] O. L. Mangasarian and E. W. Wild. Multisurface proximal support vector classification via generalized eigenvalues. Technical Report 04-03, Data Mining Institute, Computer Sciences Department, University of Wisconsin, Madison, Wisconsin, June 2004. ftp://ftp.cs.wisc.edu/pub/dmi/tech-reports/04-03.pdf, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(1), 2006, 69-74.

[84] O. L. Mangasarian and E. W. Wild. Multiple instance classification via successive linear programming. Technical Report 05-02, Data Mining Institute, Computer Sciences Department, University of Wisconsin, Madison, Wisconsin, May 2005. ftp://ftp.cs.wisc.edu/pub/dmi/tech-reports/05-02.pdf. Journal of Optimization Theory and Applications 137(1), 2008.

[85] O. L. Mangasarian and E. W. Wild. Feature selection for nonlinear kernel support vector machines. Technical Report 06-03, Data Mining Institute, Computer Sciences Department, University of Wisconsin, Madison, Wisconsin, July 2006. ftp://ftp.cs.wisc.edu/pub/dmi/tech-reports/06-03.pdf. IEEE Seventh International Conference on Data Mining (ICDM'07) October 28, 2007, Omaha, NE, Workshop Proceedings 231-236.

[86] O. L. Mangasarian and E. W. Wild. Nonlinear knowledge-based classification. Technical Report 06-04, Data Mining Institute, Computer Sciences Department, University of Wisconsin, Madison, Wisconsin, August 2006. ftp://ftp.cs.wisc.edu/pub/dmi/tech-reports/06-04.pdf, *IEEE Transactions on Knowledge and Data Engineering*, submitted.

[87] O. L. Mangasarian and E. W. Wild. Nonlinear knowledge in kernel machines. Technical Report 06-06, Data Mining Institute, Computer Sciences Department, University of Wisconsin, Madison, Wisconsin, November 2006. ftp://ftp.cs.wisc.edu/pub/dmi/tech-reports/06-06.pdf, *CRM Proceedings & Lecture Notes*, 45, 2008.

[88] O. L. Mangasarian and E. W. Wild. Exactness conditions for a convex differentiable exterior penalty for linear programming. Technical Report 07-01, Data Mining Institute, Computer Sciences Department, University of Wisconsin, Madison, Wisconsin, July 2007. ftp://ftp.cs.wisc.edu/pub/dmi/tech-reports/07-01.pdf.

[89] O. L. Mangasarian and E. W. Wild. Nonlinear knowledge in kernel approximation. *IEEE Transactions on Neural Networks*, 18:300–306, 2007. ftp://ftp.cs.wisc.edu/pub/dmi/tech-reports/05-05.pdf.

[90] O. L. Mangasarian and E. W. Wild. Privacy-preserving classification of horizontally partitioned data via random kernels. Technical Report 07-03, Data Mining Institute, Computer Sciences Department, University of Wisconsin, Madison, Wisconsin, November 2007.

[91] O. L. Mangasarian, E. W. Wild, and G. M. Fung. Proximal knowledge-based classification. Technical Report 06-05, Data Mining Institute, Computer Sciences Department, University of Wisconsin, Madison, Wisconsin, November 2006. ftp://ftp.cs.wisc.edu/pub/dmi/tech-reports/06-05.pdf.

[92] O. L. Mangasarian, E. W. Wild, and G. M. Fung. Privacy-preserving classification of vertically partitioned data via random kernels. Technical Report 07-02, Data Mining Institute, Computer Sciences Department, University of Wisconsin, Madison, Wisconsin, September 2007.

[93] O. Maron and A. L. Ratan. Multiple-instance learning for natural scene classification. In *15th International Conference on Machine Learning*, San Francisco, CA, 1998. Morgan Kaufmann.

[94] MATLAB. *User's Guide*. The MathWorks, Inc., Natick, MA 01760, 1994-2006. http://www.mathworks.com.

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

[96] P. M. Murphy and D. W. Aha. UCI machine learning repository, 1992. www.ics.uci.edu/∼mlearn/MLRepository.html.

[97] K. G. Murty and S. N. Kabadi. Some NP-complete problems in quadratic and nonlinear programming. *Mathematical Programming*, 39:117–129, 1987.

[98] D. R. Musicant. NDC: Normally distributed clustered datasets, 1998. http://www.cs.wisc.edu/dmi/svm/ndc/.

[99] J. A. Nelder and R. Mead. A simplex method for function minimization. *The Computer Journal*, 7:308–313, 1965.

[100] S. Odewahn, E. Stockwell, R. Pennington, R. Humphreys, and W. Zumach. Automated star/galaxy discrimination with neural networks. *Astronomical Journal*, 103(1):318–331, 1992.

[101] B. N. Parlett. *The Symmetric Eigenvalue Problem*. SIAM, Philadelphia, PA, 1998.

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

[103] J. Ramon and L. De Raedt. Multi-instance neural networks. In *Proceedings ICML-2000, Workshop on Attribute-Value and Relational Learning*, 2000.

[104] S. Ray and M. Craven. Supervised versus multiple instance learning: An empirical comparison. In *Proceedings 22nd International Conference on Machine Learning*, volume 119, pages 697–704, Bonn, Germany, 2005.

[105] M. Robnik-Šikonja and I. Kononenko. Theoretical and empirical analysis of ReliefF and RReliefF. *Machine Learning*, 53(1-2):23–69, 2003.

[106] R. T. Rockafellar. *Convex Analysis*. Princeton University Press, Princeton, New Jersey, 1970.

[107] H. Lipmaa S. Laur and T. Mielikäinen. Cryptographically private support vector machines. In D. Gunopulos L. Ungar, M. Craven and T. Eliassi-Rad, editors, *Twelfth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD 2006, Philadelphia, August 20–23, 2006. ACM*, pages 618–624, 2006. http://eprints.pascal-network.org/archive/00002133/01/cpsvm.pdf.

[108] B. Schölkopf, P. Simard, A. Smola, and V. Vapnik. Prior knowledge in support vector kernels. In M. Jordan, M. Kearns, and S. Solla, editors, *Advances in Neural Information Processing Systems 10*, pages 640 – 646, Cambridge, MA, 1998. MIT Press.

[109] B. Schölkopf and A. Smola. *Learning with Kernels*. MIT Press, Cambridge, MA, 2002.

[110] Y. Shen, A. Y. Ng, and M. Seeger. Fast gaussian process regression using kd-trees. In *NIPS 18*, 2006. http://ai.stanford.edu/∼ang/papers/nips05-fastgaussianprocess.pdf.

[111] A. Smola and B. Schölkopf. On a kernel-based method for pattern recognition, regression, approximation and operator inversion. *Algorithmica*, 22:211–231, 1998.

[112] E. Suthampan and S. Maneewongvatana. Privacy preserving decision tree in multi party environment. In *LNCS: Lecture Notes in Computer Scince*, volume 3689/2005, pages 727–732, Berlin/Heidelberg, 2005. Springer.

[113] R. Sutton and A. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, 1998.

[114] J. A. K. Suykens, T. Van Gestel, J. De Brabanter, B. De Moor, and J. Vandewalle. *Least Squares Support Vector Machines*. World Scientific Publishing Co., Singapore, 2002.

[115] M. E. Thompson. NDCC: Normally distributed clustered datasets on cubes, 2006. www.cs.wisc.edu/dmi/svm/ndcc/.

[116] A. N. Tikhonov and V. Y. Arsenin. *Solutions of Ill–Posed Problems*. John Wiley & Sons, New York, 1977.

[117] J. Vaidya and C. Clifton. Privacy preserving association rule mining in vertically partitioned data. In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, July 23-26, 2002, Edmonton, Alberta, Canada*, pages 639–644. Association for Computing Machinery, 2002.

[118] V. N. Vapnik. *The Nature of Statistical Learning Theory*. Springer, New York, second edition, 2000.

[119] V. N. Vapnik, S. E. Golowich, and A. Smola. Support vector method for function approximation, regression estimation and signal processing. In *Neural Information Processing Systems Volume 9*, pages 281–287, Cambridge, MA, 1997. MIT Press.

[120] G. Wahba. Support vector machines, reproducing kernel Hilbert spaces and the randomized GACV. In B. Schölkopf, C. J. C. Burges, and A. J. Smola, editors, *Advances in Kernel Methods - Support Vector Learning*, pages 69–88, Cambridge, MA, 1999. MIT Press. ftp://ftp.stat.wisc.edu/pub/wahba/index.html.

[121] J. Weston, S. Mukherjee, O. Chapelle, M. Pontil, T. Poggio, and V. Vapnik. Feature selection for SVMs. In *NIPS*, pages 668–674, 2000.

[122] A. Wieland. Twin spiral dataset. http://www-cgi.cs.cmu.edu/afs/cs.cmu.edu/project/ai-repository/ai/areas/neural/bench/cmu/0.html.

[123] W. H. Wolberg, W. N. Street, D. N. Heisey, and O. L. Mangasarian. Computerized breast cancer diagnosis and prognosis from fine-needle aspirates. *Archives of Surgery*, 130:511–516, 1995.

[124] M.-J. Xiao, L.-S. Huang, H. Shen, and Y.-L. Luo. Privacy preserving id3 algorithm over horizontally partitioned data. In *Sixth International Conference on Parallel and Distributed Computing Applications and Technologies (PDCAT'05)*, pages 239–243. IEEE Computer Society, 2005.

[125] H. Yu, X. Jiang, and J. Vaidya. Privacy-preserving SVM using nonlinear kernels on horizontally partitioned data. In *SAC '06: Proceedings of the 2006 ACM symposium on Applied computing*, pages 603–610, New York, NY, USA, 2006. ACM Press.

[126] H. Yu, J. Vaidya, and X. Jiang. Privacy-preserving svm classification on vertically partitioned data. In *Proceedings of PAKDD '06*, volume 3918 of *Lecture Notes in Computer Science*, pages 647 – 656. Springer-Verlag, January 2006.

[127] H. H. Zhang. Variable selection for support vector machines via smoothing spline ANOVA. *Statistica Sinica*, 16(2):659–674, 2006.

[128] Q. Zhang and S. A. Goldman. EM-DD: an improved miltiple-instance learning technique. In *Neural Information Processing Systems 2001*, pages 1073–1080, Cambridge, MA, 2002. MIT Press.

[129] J. Zhu, S. Rosset, T. Hastie, and R. Tibshirani. 1-Norm support vector machines. In Sebastian Thrun, Lawrence K. Saul, and Bernhard Schölkopf, editors, *Advances in Neural Information Processing Systems 16–NIPS2003*, pages 49–56. MIT Press, 2004.