Machine Learning and Data Mining Via Mathematical Programing Based Support Vector Machines

By

Glenn Fung

A dissertation submitted in partial fulfillment of the

requirements for the degree of

Doctor of Philosophy

(Computer Sciences)

at the

UNIVERSITY OF WISCONSIN – MADISON

2003

Abstract

Several issues that arise in machine learning and data mining are addressed using mathematical programming based support vector machines (SVMs). We address the following important problems. Instead of a standard SVM that classifies points by assigning them to one of two disjoint halfspaces, points are classified by assigning them to the closest of two parallel planes (in input or feature space) that are pushed apart as far as possible. This formulation leads to an extremely fast and simple algorithm for generating a linear or nonlinear classifier that merely requires the solution of a single system of nonsingular linear equations. Multiclass and incremental extensions of this proximal formulation are also presented.

Prior knowledge, in the form of multiple polyhedral sets each belonging to one of two categories, is introduced into a reformulation of a linear SVM classifier. The resulting formulation is solved efficiently by a linear program and results in enhanced testing set correctness.

A finite concave minimization algorithm is proposed for constructing classifiers that use a minimal number of data points both in generating and characterizing classifiers. The algorithm is theoretically justified by linear programming perturbation theory and a leave-one-out error bound, as well as by effective computational results on several real world datasets. Another very fast Newton based stand-alone algorithm to solve this problem is also presented.

The problem of incorporating unlabeled data into a support vector machine is formulated as a concave minimization problem on a polyhedral set for which a stationary point is quickly obtained by solving a few (5 to 7) linear programs.

We also propose an implicit Lagrangian formulation of a support vector machine classifier that results in a highly effective iterative scheme and that is solved here by a finite Newton method. The proposed method, which is extremely fast and terminates in 6 or 7 iterations, can handle classification problems in very high dimensional spaces, e.g. over 28,000, in a few seconds on a 400 MHz Pentium II machine. The method can also handle problems with large datasets and requires no specialized software other than a commonly available solver for a system of linear equations. Finite termination of the proposed method is established.

To sum up, we present several mathematical programming based algorithms that address various important SVM related issues such as: speed, scalability, data dependence and sparse representation, use of unlabeled data and knowledge incorporation.

Acknowledgements

I would like to express my gratitude to many people, beginning with my beloved wife Ceri, whose unconditional love and support have given me a solid base to confront every challenge and complete every difficult task on the road with confidence. Thanks to mamá Ling who is always there and who is largely responsible for who I am. Many thanks to all my families (Fischer, Fung, Jenkins, Goodrich) for all their support, and thanks to all the panitax for being so close despite the physical distance.

I am very grateful to an excellent professor, Javier Maguregui who introduced me to Madison, an incredible place that I am very attached to and that I love. He was was a very caring person who always wanted the best for his students. Prof. Maguregui recently passed away.

Thanks to Olvi Mangasarian, my advisor, for all the knowledge he shared with me. He showed me a new world of Mathematical Programming applications, including the fascinating area of support vector machines. I want also want to give special thanks to Robert Meyer – not only a great professor but an exceptional person – for all his support. I am also indebted to Michael Ferris for teaching me the practical and technical skills that have helped me to close the gap between theory and implementation ; and to Jude Shavlik for all the new ideas and suggestions and for introducing me to the area of artificial intelligence.

A number of colleagues in the department have also been of great help. Of them, I thank Dave Musicant, Yuh-Jye Lee, Jin-ho Lim, Meta Voelker and Michael Thompson. I also want to thank to all my good Madison friends that made me feel like at home: Mario and Katy Pi, Dan Lerner, Leito Guzman, Bernardo Uribe, Cold R. (Riofrio) and

Sonja Hanson.

Thanks to all the staff and friends at WORT for all that I have learned there; you have been an integral part of my education here in Madison.

Finally, thanks to Tula, an extraordinary being, for all the love, bites and fun that kept me always entertained and laughing the last two years.

This research was partially supported by National Science Foundation Grants CCR-9729842 and CCR-0138308, by Air Force Office of Scientific Research Grants F49620-97-1-0326 and F49620-00-1-0085, and by the Microsoft Corporation.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

In the last ten years, the dramatic rise in the use of the Internet, the improvement in technologies related to communications, and more recently, the advances related to the understanding of human biology and genetics have created increased dependence on information in our society. The huge amount of data generated daily contains important information that accumulates incrementally in databases and is not easy to extract. The field of data mining developed as a means of extracting information and knowledge from databases to discover patterns or concepts that are not evident or easy to obtain.

As stated in [108], machine learning provides the technical basis of data mining by extracting information from the raw data in the databases. The process usually consists of the following: transforming the data to a suitable format, cleaning it, and inferring or making conclusions regarding the data.

Two major sub-fields of machine learning are supervised learning and unsupervised learning. Within the category of supervised learning, one of the primary tools developed in recent years, support vector machines (SVMs)[10, 17, 19, 22, 57, 65, 92, 103], have proven to be powerful tools for data classification. There are several difficulties associated with the use of a standard SVM for classification. In this thesis we propose several math programming based algorithms that address several of these difficulties.

The first difficulty is related to the fact that standard SVMs require the solution of either a quadratic or a linear program, which require specialized codes such as [21].

Standard SVMs classify points by assigning them to one of two disjoint halfspaces. These halfspaces are either in the original input space of the problem for linear classifiers, or in a higher dimensional feature space for nonlinear classifiers [19, 65, 103]. As an alternative to standard SVMs, we propose here a *proximal* SVM (PSVM) which classifies points depending on proximity to one of two parallel planes that are pushed as far apart as possible. In Chapter 2, using our geometrically motivated proximal formulation, we give extensive computational implementation and results not contained in earlier theoretical and statistical related work on regularized networks [26, 27]. Our specific formulation leads to a strongly convex objective function, which is not always the case in other regularized network approaches [26, 27]. Strong convexity is an important factor in simplifying the proximal code provided here as well as resulting in very fast computational times. Furthermore, unlike other recent similar work [95–97], our formulation does not require any restrictive conditions on the kernel function such as Mercer's positive definiteness condition [103]. Obtaining a linear or nonlinear PSVM classifier requires nothing more sophisticated than solving a single system of linear equations. Efficient and fast linear equation solvers are freely available [1], or are part of standard commercial packages such as MATLAB [75], and can solve large systems very fast. By taking advantage of the structure of the $(n+1) \times (n+1)$ symmetric positive definite matrix constituting the system of linear equations, where n is the usually small dimensional input space of the proximal SVM classifier [33], we are able also to propose two extensions of PSVM:

- An incremental algorithm that can handle extremely large datasets, of the order of 10^9 in an incremental fashion. Old data can be easily retired while new data can just as easily be incorporated into the classifier.

- A multiclass PSVM that is based on the well known "one-from-the-rest" approach.

This approach is a natural choice in order to take advantage of its fast performance. However, there is a drawback associated with this one-from-the-rest approach. The resulting two-class problems are often very unbalanced, leading in some cases to poor performance. We propose balancing the k classes and a novel Newton refinement modification to PSVM in order to deal with this problem. Computational results indicate that these two modifications preserve the speed of PSVM while often leading to significant test set improvement over a plain PSVM one-from-the-rest application. The modified approach is considerably faster than other one-from-the-rest methods that use conventional SVM formulations, while still giving comparable test set correctness.

In Chapter 3 we present a novel approach to incorporating prior knowledge in the form of polyhedral *knowledge sets* in the input space of the given data. These knowledge sets, which can be as simple as cubes or more complex polyhedral sets, belong to one of two categories into which all the data is divided. Thus, a single knowledge set can be interpreted as a generalization of a training example, which typically consists of a single *point* in input space. In contrast, each of our knowledge sets consists of a *region* in the same space. By using a powerful tool from mathematical programming, theorems of the alternative [59, Chapter 2], we are able to embed such prior data into a linear program that can be efficiently solved by any of the publicly available linear programming solvers. Furthermore, we also propose an extension to a nonlinear classifier.

In Chapter 4, we address the problem of constructing classifiers that use a minimal number of data points both in generating and characterizing the separating surface. We propose three algorithms, the first two algorithms in Sections 4.1 and 4.2 are theoretically justified on the basis of linear programming perturbation theory and both formulated as

concave minimization problems, then solved using a successive linearization algorithm. In the linear case, a sparse representation results in a considerable reduction in the number of support vectors and/or in the number of input space feature dependencies. In the nonlinear case, this can result in substantial reduction in kernel data-dependence (over 94% in six of the seven public datasets tested on) and test set correctness equal to that obtained by using a conventional support vector machine classifier that depends on many more data points. This reduction in data dependence results in a much faster classifier that requires less storage. In Section 4.3 we propose a third algorithm: a fast Newton method, that suppresses input space features, for a linear programming formulation of support vector machine classifiers. The proposed stand-alone method can handle classification problems in very high dimensional spaces, such as 28,032 dimensions, and generates a classifier that depends on very few input features, such as 7 out of the original 28,032. The method can also handle problems with a large number of data points and requires no specialized linear programming packages, but merely a linear equation solver. For nonlinear kernel classifiers, the method utilizes a minimal number of kernel functions in the classifier that it generates.

The problem of incorporating unlabeled data into an SVM classifier is addressed in Chapter 5. This latter problem is formulated as a concave minimization problem on a polyhedral set for which a stationary point is quickly obtained by solving a few (5 to 7) linear programs. Such stationary points turn out to be very effective, as evidenced by our computational results.

Finally, in Chapter 6, an implicit Lagrangian [74] formulation of a support vector machine classifier that led to a highly effective iterative scheme [71] is solved by a finite Newton method. The proposed method, which is extremely fast and terminates in 6

or 7 iterations, can handle classification problems in very high dimensional spaces, e.g. over 28,000, in a few seconds on a 400 MHz Pentium II machine. The method can also handle problems with large datasets and requires no specialized software other than a commonly available solver for a system of linear equations. Finite termination of the proposed method is also established.

1.1 Notation and Mathematical Background

- All vectors will be column vectors unless transposed to a row vector by a prime superscript $'$. The scalar (inner) product of two vectors x and y in R^n will be denoted as $x'y$.

- For a vector x in the n-dimensional real space R^n, $|x|$ will denote a vector of absolute values of the components $x_i,\ i = 1, \ldots, n$ of x.

- The sign function $sign(x)$ is defined as $sign(x)_i = 1$ if $x_i > 0$ else $sign(x)_i = -1$ if $x_i \leq 0$,

- For a vector x in R^n, the plus function x_+ denotes the vector in R^n with components $\max\{0, x_i\}$.

- For a vector x in R^n, x_* denotes the vector in R^n with components $(x_*)_i$ equal 1 if $x_i > 0$ and 0 otherwise.

- For $x \in R^n$ and $1 \leq p < \infty$, the norm $\|x\|_p$ will denote the p-norm:

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

and

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

- 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 $A_{\cdot j}$ will denote the jth column of A.

- A vector of ones in a real space of arbitrary dimension will be denoted by e. A vector of zeros in a real space of arbitrary dimension will be denoted by 0. The identity matrix in a real space of arbitrary dimension will be denoted by I.

- For $e \in R^m$ and $y \in R^m$ the notation $e'y$ will signify the summation $\sum_{i=1}^{m} y^i$.

- We shall employ the MATLAB "dot" notation [75] to signify application of a function to all components of a matrix or a vector. For example if $A \in R^{m \times n}$, then $A_\bullet^2 \in R^{m \times n}$ will denote the matrix of elements of A squared.

- 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 of $\mathcal{A}$ or $\mathcal{B}$ when the sets are strictly linearly separable.

- 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.

- The symbol $\wedge$ will denote the logical "and".

- For $A \in R^{m \times n}$ and $B \in R^{n \times k}$, the kernel $K(A, B)$ 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. We shall only assume that $K(A, A')$ is symmetric, that is $(K(A, A'))' = K(A, A')$.

- The base of the natural logarithm will be denoted by ε.

- We will make use of the following Gaussian kernel [19, 65, 103] that is frequently used in the SVM literature:

$$(K(A,B))_{ij} = \varepsilon^{-\mu\|A_i{}' - B_{.j}\|^2},\ i = 1\ldots,m,\ j = 1\ldots,k, \tag{1.1}$$

  where $A \in R^{m\times n}$, $B \in R^{n\times k}$ and μ is a positive constant.

- For simplicity, the dimensionality of some vectors will not be explicitly given. The abbreviation "s.t." stands for "such that".

- If f is a real valued function defined on the n-dimensional real space R^n, the gradient of f at x is denoted by $\nabla f(x)$ which is a column vector in R^n and the $n \times n$ matrix of second partial derivatives of f at x is denoted by $\nabla^2 f(x)$.

- 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. However, one can define its **generalized Hessian** [28, 43, 66] which is the $n \times n$ symmetric positive semidefinite matrix:

$$\partial^2 f(x) = A'diag(Ax-b)_* A + P, \tag{1.2}$$

  where $diag(Ax-b)_*$ denotes an $m \times m$ diagonal matrix with diagonal elements $(A_ix - b_i)_*$, $i = 1,\ldots,m$. The generalized Hessian (1.2) has many of the properties of the regular Hessian [28, 43, 66] in relation to $f(x)$. For simplicity we use 0 when $A_ix - b_i = 0$ for $(A_ix - b_i)_*$ instead of a value in $(0, 1]$. This does not affect computational results.

- 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.

1.2 Computational Resources

Most of our computations were performed on the University of Wisconsin Data Mining Institute "locop1" machine, which utilizes a 400 Mhz Pentium II and allows a maximum of 2 Gigabytes of memory for each process. This computer runs on Windows NT server 4.0, with MATLAB 6 installed. Even though "locop1" is a multiprocessor machine, only one processor was used for all the experiments since MATLAB is a single threaded application and does not distribute any load across processors [75].

Chapter 2

Proximal Support Vector Machines for Binary, Multiclass and Incremental Classification

2.1 The linear Proximal Support Vector Machine

We consider the problem, depicted in Figure 2.1, of classifying m points in the n-dimensional real space R^n, represented by the $m \times n$ matrix A, according to membership of each point A_i in the class $A+$ or $A-$ as specified by a given $m \times m$ diagonal matrix D with plus ones or minus ones along its diagonal respectively. For this problem, the standard support vector machine with a linear kernel [19, 102] is given by the following quadratic program with parameter $\nu > 0$:

$$\begin{array}{rcl} \min\limits_{(w,\gamma,y)\in R^{n+1+m}} & \nu e'y + \frac{1}{2}w'w & \\ \text{s.t.} \quad D(Aw - e\gamma) + y & \geq & e \\ y & \geq & 0. \end{array} \tag{2.1}$$

As depicted in Figure 2.1, w is the normal to the bounding planes:

$$\begin{array}{rcl} x'w & = & \gamma + 1 \\ x'w & = & \gamma - 1, \end{array} \tag{2.2}$$

that bound most of the sets $A+$ and $A-$ respectively. The constant γ determines their location relative to the origin. When the two classes are strictly linearly separable, that is when the error variable $y = 0$ in (2.1) (which is not the case shown in Figure 2.1), the plane $x'w = \gamma + 1$ bounds all of the class $A+$ points, while the plane $x'w = \gamma - 1$ bounds all of the class $A-$ points as follows:

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

Consequently, the plane:

$$x'w = \gamma, \tag{2.4}$$

midway between the bounding planes (2.2), is a separating plane that separates $A+$ from $A-$ completely if $y = 0$, else only approximately as depicted in Figure 2.1. The quadratic term in (2.1), which is twice the reciprocal of the square of the 2-norm distance $\frac{2}{\|w\|}$ between the two bounding planes of (2.2) (see Figure 2.1), maximizes this distance, often called the "margin". Maximizing the margin enhances the generalization capability of a support vector machine [19, 102]. If the classes are linearly inseparable, which is the case shown in Figure 2.1, then the two planes bound the two classes with a "soft margin" (i.e. bound approximately with some error) determined by the nonnegative error variable y, that is:

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

The 1-norm of the error variable y is minimized parametrically with weight ν in (2.1) resulting in an approximate separating plane (2.4) as depicted in Figure 2.1. This plane

acts as a linear classifier as follows:

$$x'w - \gamma \begin{cases} > 0, & \text{then } x \in A+, \\ < 0, & \text{then } x \in A-, \\ = 0, & \text{then } x \in A+ \text{ or } x \in A-. \end{cases} \tag{2.6}$$

Our point of departure is similar to that of [69, 71], where the optimization problem (2.1) is replaced by the following problem:

$$\begin{array}{cc} \min\limits_{(w,\gamma,y)\in R^{n+1+m}} & \nu\frac{1}{2}\|y\|^2 + \frac{1}{2}(w'w + \gamma^2) \\ \text{s.t.} & D(Aw - e\gamma) + y \;\geq\; e \end{array} \tag{2.7}$$

Note that no explicit nonnegativity constraint is needed on y, because if any component y_i is negative then the objective function can be decreased by setting that $y_i = 0$ while still satisfying the corresponding inequality constraint. Note further that the 2-norm of the error vector y is minimized instead of the 1-norm, and the margin between the bounding planes is maximized with respect to both orientation w and relative location to the origin γ. Extensive computational experience [52, 53, 68, 69, 71] indicates that this formulation is just as good as the classical formulation (2.1) with some added advantages such as strong convexity of the objective function. Our key idea here is to make a simple but fundamental change in the formulation (2.7), namely replace the inequality constraint by an equality as follows:

$$\begin{array}{cc} \min\limits_{(w,\gamma,y)\in R^{n+1+m}} & \nu\frac{1}{2}\|y\|^2 + \frac{1}{2}(w'w + \gamma^2) \\ \text{s.t.} & D(Aw - e\gamma) + y \;=\; e \end{array} \tag{2.8}$$

This modification, even though very simple, changes the nature of optimization problem significantly. In fact it turns out that we can write an explicit exact solution to the

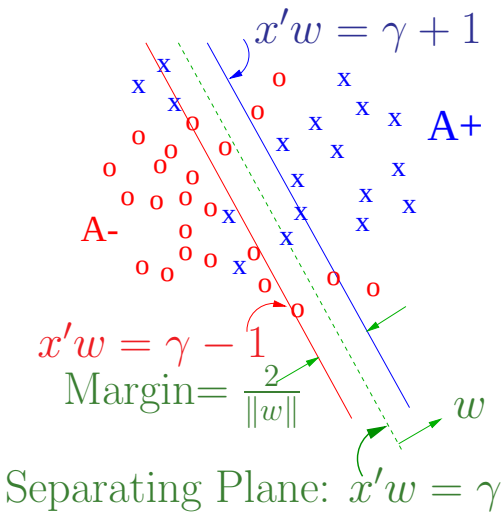

Figure 2.1: The Standard Support Vector Machine Classifier in the w-space of R^n: The approximately bounding planes of equation (2.2) with a soft (i.e. with some error) margin $\frac{2}{\|w\|}$, and the plane of equation (2.4) approximately separating $A+$ from $A-$.

problem in terms of the problem data as we show below. In contrast, it is impossible to do so in the conventional inequality based because of their combinatorial nature. Geometrically the formulation (2.8) is depicted in Figure 2.2, which can be interpreted as follows. The planes $x'w - \gamma = \pm 1$ are not bounding planes anymore, but can be thought of as "proximal" planes, around which the points of each class are clustered and which are pushed as far apart as possible by the term $(w'w + \gamma^2)$ in the objective function which is nothing other than the reciprocal of the 2-norm distance squared between the two planes in the (w, γ) space of R^{n+1}.

We note that our formulation (2.8) can be also interpreted as a regularized least squares solution [98] of the system of linear *equations* $D(Aw - e\gamma) = e$, that is finding an approximate solution (w, γ) with least 2-norm. Similarly the standard SVM formulation (2.1) can be interpreted, by using linear programming perturbation theory [67], as a least 2-norm approximate solution to the system of linear *inequalities* $D(Aw - e\gamma) \geq e$. Neither of these interpretations, however, is based on the idea of maximizing the margin,

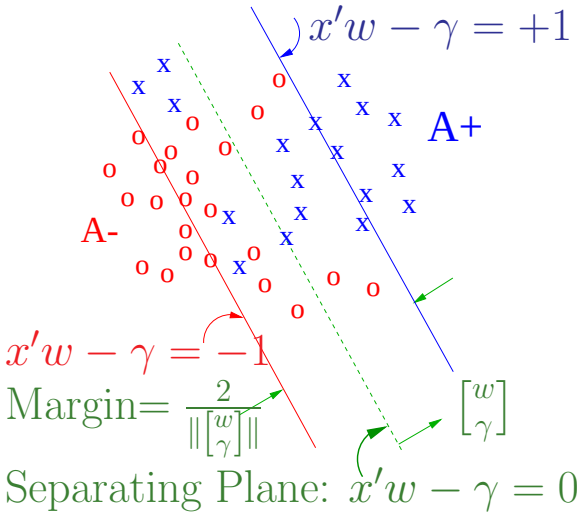

Figure 2.2: The Proximal Support Vector Machine Classifier in the (w, γ)-space of R^{n+1}: The planes $x'w - \gamma = \pm 1$ around which points of the sets $A+$ and $A-$ cluster and which are pushed apart by the optimization problem (2.8).

the distance between the parallel planes (2.2), which is a key feature of support vector machines [19, 65, 103].

The Karush-Kuhn-Tucker (KKT) necessary and sufficient optimality conditions [59, p. 112] for our equality constrained problem (2.8) are obtained by setting equal to zero the gradients with respect to (w, γ, y, u) of the Lagrangian:

$$L(w, \gamma, y, u) = \frac{\nu}{2}\|y\|^2 + \frac{1}{2}\|\begin{bmatrix} w \\ \gamma \end{bmatrix}\|^2 - u'(D(Aw - e\gamma) + y - e). \tag{2.9}$$

Here, $u \in R^m$ is the Lagrange multiplier associated with the equality constraint of (2.8). Setting the gradients of L equal to zero gives the following KKT optimality conditions:

$$\begin{aligned} w - A'Du &= 0 \\ \gamma + e'Du &= 0 \\ \nu y - u &= 0 \\ D(Aw - e\gamma) + y - e &= 0 \end{aligned} \tag{2.10}$$

The first three equations of (2.10) give the following expressions for the original problem variables (w, γ, y) in terms of the Lagrange multiplier u:

$$w = A'Du, \;\; \gamma = -e'Du, \;\; y = \frac{u}{\nu}. \tag{2.11}$$

Substituting these expressions in the last equality of (2.10) allows us to obtain an explicit expression for u in terms of the problem data A and D as follows:

$$u = (\frac{I}{\nu} + D(AA' + ee')D)^{-1}e = (\frac{I}{\nu} + HH')^{-1}e, \tag{2.12}$$

where H is defined as:

$$H = D[A \quad -e]. \tag{2.13}$$

Having u from (2.12), the explicit solution (w, γ, y) to our problem (2.8) is given by (2.11). Because the solution (2.12) for u entails the inversion of a possibly massive $m \times m$ matrix, we make immediate use of the Sherman-Morrison-Woodbury formula [40, p. 51] for matrix inversion, as was done in [29, 69, 71], which results in:

$$u = \nu(I - H(\frac{I}{\nu} + H'H)^{-1}H')e. \tag{2.14}$$

This expression, as well as another simple expression (2.28) for $\left[\begin{smallmatrix} w \\ \gamma \end{smallmatrix}\right]$ below, involve the inversion of a much smaller dimensional matrix of order $(n+1) \times (n+1)$ and completely solves the classification problem. This is a key difference relative to other approaches

that also substitute the inequalities by equalities [41]. For concreteness we explicitly state our very simple algorithm.

Algorithm 2.1.1 Linear Proximal SVM *Given m data points in R^n represented by the $m \times n$ matrix A and a diagonal matrix D of ± 1 labels denoting the class of each row of A, we generate the linear classifier (2.6) as follows:*

(i) Define H by (2.13) where e is an $m \times 1$ vector of ones and compute u by (2.14) for some positive ν. Typically ν is chosen by means of a tuning (validating) set.

(ii) Determine (w, γ) from (2.11).

(iii) Classify a new x by using (2.6).

For standard SVMs, support vectors consist of all data points which are the complement of the data points that can be dropped from the problem without changing the separating plane (2.4) [65, 103]. Thus, for the standard SVM formulation (2.1), support vectors correspond to data points for which the Lagrange multipliers are nonzero because, solving (2.1) with these data points only will give the same answer as solving it with the entire dataset. In our proximal formulation (2.8) however, the Lagrange multipliers u are merely a multiple of the error vector y: $u = \nu y$ as given by (2.11). Consequently, because all components of y are typically nonzero since none of the data points usually lie on the proximal planes $x'w = \pm 1$, the concept of support vectors needs to be modified as follows. Because $(w, \gamma) \in R^{n+1}$ are given as linear functions of y by (2.10), it follows by the basis theorem for linear equations [38, Theorem 2.11][73, Lemma 2.1], applied to the last equality of (2.10) for a fixed value of the error vector y, that at most $n + 1$ linearly independent data points are needed to determine the basic nonzero components

of $(w, \gamma) \in R^{n+1}$. Guided by this fact that only a small number of data points can characterize any specific (w, γ), we define the concept of ϵ-*support vectors* as those data points A_i for which error vector y_i is less than ϵ in absolute value. We typically pick ϵ small enough such that about 1% of the data are ϵ-support vectors. Re-solving our proximal SVM problem (2.8) with these data points and a ν adjusted (typically upwards) by a tuning set gives test set correctness that is essentially identical to that obtained by using the entire dataset.

We note that with explicit expressions (w, γ, y, u) in terms of problem data given by (2.11) and (2.14), we are able to get also an explicit expression for the leave-one-out-correctness *looc* [91], that is the fraction of correctly classified data points if each point in turn is left out of the PSVM formulation (2.8) and then is classified by the classifier (2.6). Omitting some algebra, we have the following leave-one-out-correctness:

$$looc = \frac{e'h_*}{m}, \tag{2.15}$$

where the "*" denotes the "step" function defined in the Introduction, and for $i = 1, \ldots, m$:

$$h_i = \frac{D_i H H^{i\prime} D^i u^i}{\nu} = D_i H H^{i\prime} D^i (I - H^i (\frac{I}{\nu} + H^{i\prime} H^i)^{-1} H^{i\prime}) e. \tag{2.16}$$

Here, H is defined by (2.13), H_i denotes row i of H, while H^i denotes H with row H_i removed from H, and u^i is defined by (2.14) with H replaced by H^i. Similarly, D_i denotes row i of D.

We extend now some of the above results to nonlinear proximal support vector machines.

2.2 Nonlinear Proximal Support Vector Machines

To obtain our nonlinear proximal classifier we modify our equality constrained optimization problem (2.8) as in [53, 65] by replacing the primal variable w by its dual equivalent $w = A'Du$ from (2.11) to obtain:

$$\begin{array}{rcl} \min\limits_{(u,\gamma,y)\in R^{m+1+m}} & \nu\frac{1}{2}\|y\|^2 + \frac{1}{2}(u'u+\gamma^2) & \\ \text{s.t.} & D(AA'Du - e\gamma) + y & = & e, \end{array} \tag{2.17}$$

where the objective function has also been modified to minimize weighted 2-norm sums of the problem variables (u, γ, y). If we now replace the linear kernel AA' by a nonlinear kernel $K(A, A')$ as defined in the Introduction, we obtain:

$$\begin{array}{rcl} \min\limits_{(u,\gamma,y)\in R^{m+1+m}} & \nu\frac{1}{2}\|y\|^2 + \frac{1}{2}(u'u+\gamma^2) & \\ \text{s.t.} & D(K(A, A')Du - e\gamma) + y & = & e. \end{array} \tag{2.18}$$

Using the shorthand notation:

$$K := K(A, A'), \tag{2.19}$$

the Lagrangian for (2.18) can be written similarly to (2.9) as:

$$L(u, \gamma, y, v) = \frac{\nu}{2}\|y\|^2 + \frac{1}{2}\|\begin{bmatrix} u \\ \gamma \end{bmatrix}\|^2 - v'(D(KDu - e\gamma) + y - e). \tag{2.20}$$

Here, $v \in R^m$ is the Lagrange multiplier associated with the equality constraint of (2.18). Setting the gradients of this Lagrangian with respect to (u, γ, y, v) equal to zero gives the following KKT optimality conditions:

$$\begin{array}{rcl} u - DK'Dv & = & 0 \\ \gamma + e'Dv & = & 0 \\ \nu y - v & = & 0 \\ D(KDu - e\gamma) + y & = & e. \end{array} \tag{2.21}$$

The first three equations of (2.21) give the following expressions for (u, γ, y) in terms of the Lagrange multiplier v:

$$u = DK'Dv, \;\; \gamma = -e'Dv, \;\; y = \frac{v}{\nu}. \tag{2.22}$$

Substituting these expressions in the last equality of (2.21) gives an explicit expression for v in terms of the problem data A and D as follows:

$$v = (\frac{I}{\nu} + D(KK' + ee')D)^{-1}e = (\frac{I}{\nu} + GG'))^{-1}e, \tag{2.23}$$

where G is defined as:

$$G = D[K \quad -e]. \tag{2.24}$$

Note the similarity between G above and H as defined in (2.13). This similarity allows us to obtain G from the expression (2.13) for H by replacing A by K in (2.13). This can be taken advantage of in the MATLAB code 2.2.2 of Algorithm 2.1.1 which is written for the linear classifier (2.6). *Thus, to generate a nonlinear classifier by Algorithm 2.2.1 merely replace A by K in the algorithm.*

Having the solution v from (2.23), the solution (u, γ, y) to our problem (2.18) is given by (2.22). Unlike the situation with linear kernels, the Sherman-Morrison-Woodbury formula is useless here because the kernel matrix $K = K(A, A')$ is a square $m \times m$ matrix, so the inversion must take place in a potentially high-dimensional R^m. However, the reduced kernel techniques of [52], reduce the $m \times m$ dimensionality of the kernel $K = K(A, A')$ to a much smaller $m \times \bar{m}$ dimensionality of a rectangular kernel $K = K(A, \bar{A}')$, where $\bar{m}$ is as small as 1% of m and $\bar{A}$ is an $\bar{m} \times n$ random submatrix of of A.

This reduced technique can be justified by the theory of random projections [24, 25] and the idea behind it is similar to using low-rank kernel representations [31].

Such reduced kernels not only make most large problems tractable, but they also often lead to improved generalization by avoiding data overfitting. The effectiveness of these reduced kernels is demonstrated by means of a numerical test problem in the next section of the paper.

The nonlinear separating surface corresponding to the kernel $K(A, A')$ [65, Equation (8.1)] and can be deduced from the linear separating surface (2.4) and $w = A'Du$ from (2.11) as follows:

$$x'w - \gamma = x'A'Du - \gamma = 0. \tag{2.25}$$

If we replace $x'A'$ by the corresponding kernel expression $K(x', A')$, and substitute from (2.22) for u and γ: $u = DK'Dv$ and $\gamma = -e'Dv$ we obtain the nonlinear separating surface:

$$\begin{aligned} K(x', A')Du - \gamma &= K(x', A')DDK(A, A')'Dv + e'Dv \\ &= (K(x', A')K(A, A')' + e')Dv = 0. \end{aligned} \tag{2.26}$$

The corresponding *nonlinear* classifier to this nonlinear separating surface is then:

$$(K(x', A')K(A, A')' + e')Dv \begin{cases} > 0, & \text{then } x \in A+, \\ < 0, & \text{then } x \in A-, \\ = 0, & \text{then } x \in A+ \text{ or } x \in A-. \end{cases} \tag{2.27}$$

We now give an explicit statement of our nonlinear classifier algorithm.

Algorithm 2.2.1 Nonlinear Proximal SVM *Given m data points in R^n represented by the $m \times n$ matrix A and a diagonal matrix D of ± 1 labels denoting the class of each row of A, we generate the nonlinear classifier (2.27) as follows:*

(i) Choose a kernel function $K(A, A')$, typically the Gaussian kernel (1.1).

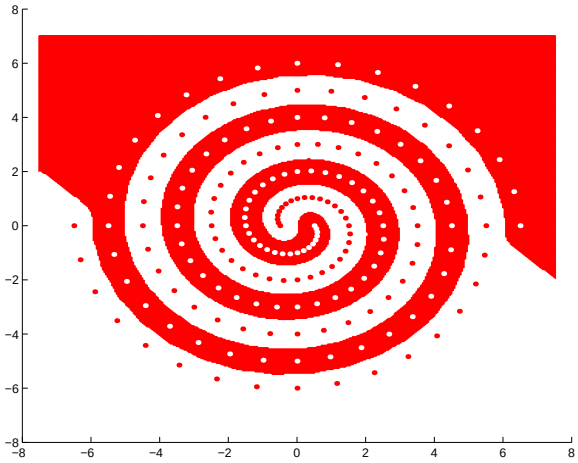

Figure 2.3: The spiral dataset consisting of 97 black points and 97 white points intertwined as two spirals in 2-dimensional space. PSVM with a Gaussian kernel generated a sharp nonlinear spiral-shaped separating surface.

(ii) Define G by (2.24) where $K = K(A, A')$ and e is an $m \times 1$ vector of ones. Compute v by (2.23) for some positive ν. (Typically ν is chosen by means of a tuning set.)

(iii) The nonlinear surface (2.26) with the computed v constitutes the nonlinear classifier (2.27) for classifying a new point x.

The nonlinear classifier (2.27), which is a direct generalization of the linear classifier (2.6), works quite effectively as indicated by the numerical examples presented in the next section.

2.2.1 Numerical Implementation and Comparisons

Our algorithms require the solution of a single square system of linear equations of the size of the number of input attributes n in the linear case, and of the size of the number of data points m in the nonlinear case. When using a rectangular kernel [53], the size of the problem can be reduced from m to k with $k << m$ for the nonlinear case. Because

of the simplicity of our algorithm, we give below the actual MATLAB implementation that was used in our experiments and which consists of 6 lines of native MATLAB code.

Code 2.2.2 PSVM MATLAB Code

```
function [w,gamma] = psvm(A,D,nu)
 % PSVM:linear and nonlinear classification
 % INPUT: A, D, nu. OUTPUT: w, gamma
 % [w, gamma] = psvm(A,D,nu);
  [m,n]=size(A);e=ones(m,1);H=D*[A -e];
  r=sum(H)'; %r=H'*e;
  r=(speye(n+1)/nu+H'*H)\r; %solve (I/nu+H'*H)r=H'*e
  u=nu*(1-(H*r)); s=D*u;
  w=(s'*A)';                %w=A'*D*u
  gamma=-sum(s);            %gamma=-e'*D*u
```

Note that the command line in the MATLAB code above: `r=(speye(n+1)/nu+H'*H)\r;` computes directly the factor $(\frac{I}{\nu}+H'H)^{-1}H'e$ of (2.14). This is much more economical and stable than computing the inverse $(\frac{I}{\nu}+H'H)^{-1}$ explicitly then multiplying it by $H'e$. Using MATLAB, the calculations $H'e$ and $A's$ involve the transpose of typically large matrices which can be time consuming. Instead, we calculate `r=sum(H)'` and `w=(s'*A)'` respectively, the transposes of these vectors.

We further note that the MATLAB code above not only works for a linear classifier, but also for a nonlinear classifier as well. In the nonlinear case, the matrix $K(A, A')$

is used as input instead of A, [65, Equations (1), (10)], and the pair $(\hat{u}, \gamma)$, where $\hat{u} = K(A, A')Du$, is returned instead of (w, γ). The nonlinear separating surface is then given by (2.26) as:

$$K(x, A')\hat{u} - \gamma = 0.$$

Rectangular kernels [52] can also be handled by this code. The input then is the rectangular matrix $K(A, \bar{A}')$, where $\bar{A} \in R^{m \times k}$, $k << m$ and the given output is the pair $(\hat{u}, \gamma)$ with $\hat{u} \in R^k$ and $\hat{u} = \bar{D}\bar{u}$, where $\bar{D}$ and $\bar{u}$ are the D and u associated with the reduced matrix $\bar{A}$.

A final note regarding a further simplification of PSVM. If we substitute the expression (2.14) for u in (2.11), we obtain after some algebra the following simple expression for w and γ in terms of the problem data:

$$\begin{bmatrix} w \\ \gamma \end{bmatrix} = (\frac{I}{\nu} + E'E)^{-1}E'De, \tag{2.28}$$

where $E = DH$ and hence $H = DE$, $(D = D^{-1})$. Thus:

$$E = DH = [A \quad -e], \text{ and } H = DE = D[A \quad -e]. \tag{2.29}$$

This direct explicit solution of our PSVM problem (2.8) can be written as the following single line of MATLAB code, which also does not perform the explicit matrix inversion $(\frac{I}{\nu} + E'E)^{-1}$, and is slightly faster than the above MATLAB code:

```
r=(I/nu+E'*E)\(diag(D)'*E)';w=r(1:n);gamma=r(n+1);
```
(2.30)

Here, according to MATLAB commands, $diag(D)$ is an $m \times 1$ vector generated from the diagonal of the matrix D. Computational testing results using this one-line MATLAB code (2.30) are slightly better than those obtained with Code 2.2.2 and are the ones

reported in the tables below. We comment further that the solution (2.28) can also be obtained directly from (2.8) by using the equality constraint to eliminate y from the problem and solving the resulting unconstrained minimization problem in the variables w and γ by setting to zero the gradients with respect to w and γ. We turn now to our computations.

The datasets used for our numerical tests were the following:

- Seven publicly available datasets from the UCI Machine Learning Repository [77]: WPBC, Ionosphere, Cleveland Heart, Pima Indians, BUPA Liver, Mushroom, Tic-Tac-Toe.
- The Census dataset is a version of the US Census Bureau "Adult" dataset, which is publicly available from the Silicon Graphics website [16].
- The Galaxy Dim dataset used in galaxy discrimination with neural networks from [79]
- Two large datasets (2 million points and 10 attributes) created using David Musicant's NDC Data Generator [78].
- The Spiral dataset proposed by Alexis Wieland of the MITRE Corporation and available from the CMU Artificial Intelligence Repository [107].

We outline our computational results now in five groups as follows.

1. **Table 2.2.1: Comparison of seven different methods on the Adult dataset** In this experiment we compared the performance of seven different methods for linear classification on different sized versions of the Adult dataset. Reported results on the SOR [68], SMO [84] and SVM^{light} [50] are from [68]. Results for LSVM [71]

results were computed here using "locop1", whereas SSVM [53] and RLP [4] are from [53]. The SMO experiments were run on a 266 MHz Pentium II processor under Windows NT 4 using Microsoft's Visual C++ 5.0 compiler. The SOR experiments were run on a 200 MHz Pentium Pro with 64 megabytes of RAM, also under Windows NT 4 and using Visual C++ 5.0. The SVM^{light} experiments were run on the same hardware as that for SOR, but under the Solaris 5.6 operating system. Bold type indicates the best result and a dash (-) indicates that the results were not available from [68]. Although the timing comparisons are approximate because of the different machines used, they do indicate that PSVM has a distinct edge in speed, e.g. solving the largest problem in 7.4 seconds, which is much faster than any other method. Times and ten-fold testing correctness are shown in Table 2.2.1. Times are for the ten-folds.

2. **Table 2.2.1: Comparative performances of LSVM [71] and PSVM on a large dataset**

   Two large datasets consisting of 2 million points and 10 attributes were created using the NDC Data Generator [78]. One of them is called NDC-easy because it is highly linearly separable (around 90%). The other one is called NDC-hard since it has linear separability of around 70%. As is shown in Table 2.2.1 the linear classifiers obtained using both methods performed almost identically. Despite the 2 million size of the datasets, PSVM solved the problems in about 20 seconds each compared to LSVM's times of over 650 seconds. In contrast, SVM^{light} [50] failed on this problem due to memory problems [71] on the same machine.

3. **Table 2.2.1: Comparison of PSVM, SSVM and LSVM and SVM^{light}, using a Linear Classifier**

In this experiment we compared four methods: PSVM,

SSVM, LSVM and SVM^{light} on seven publicly available datasets from UCI Machine Learning Repository [77] and [79]. As shown in Table 2.2.1, the correctness of the four methods were very similar but the execution time including ten-fold cross validation for PSVM was smaller by as much as one order of magnitude or more than the other three methods tested. Since LSVM, SSVM and PSVM are all based on similar formulations of the classification problem the same value of ν was used for all of them. For SVM ^{light} the trade-off between trading error and margin is represented by a parameter C. The value of C was chosen by tuning. A paired t-test [76] at 95% confidence level was performed to compare the performance of PSVM and the other algorithms tested. The p-values obtained show that there is no significant difference between PSVM and the other methods tested.

4. **Figure 2.3: PSVM on the Spiral Dataset**

   We used a Gaussian kernel in order to classify the spiral dataset. This dataset consisting of 194 black and white points intertwined in the shape of a spiral is a synthetic dataset [107]. However, it apparently is a difficult test case for data mining algorithms and is known to give neural networks severe problems [42]. In contrast, a sharp separation was obtained using PSVM as can be seen in Figure 2.3.

5. **Table 2.2.1: Nonlinear Classifier Comparison using PSVM, SSVM and LSVM**

For this experiment we chose four datasets from the UCI Machine Learning Repository for which it is known that a nonlinear classifier performs significantly better that a linear classifier. We used PSVM, SSVM and LSVM in order to find a Gaussian-kernel-based nonlinear classifier to classify the data. In all datasets tested, the three methods performed similarly as far as ten-fold cross validation is concerned. However, execution time of PSVM was much smaller than that of other two methods. Note that for the mushroom dataset that consists of $m = 8124$ points with $n = 22$ attributes each, the square 8124×8124 kernel matrix does not fit into memory. In order to address this problem, we used a rectangular kernel with $\bar{A} \in R^{215\times 8124}$ instead, as described in [52]. In general, our algorithm performed particularly well with a rectangular kernel since the system solved is of size $k \times k$, with $k << m$ and where k is the much smaller number of rows of $\bar{A}$. In contrast with a full square kernel matrix the system solved is of size $m \times m$. A paired t-test [76] at 95% confidence level was performed to compare the performance of PSVM and the other algorithms tested. The p-values obtained show that there is no significant difference between PSVM and the other methods tested as far as ten-fold testing correctness is concerned.

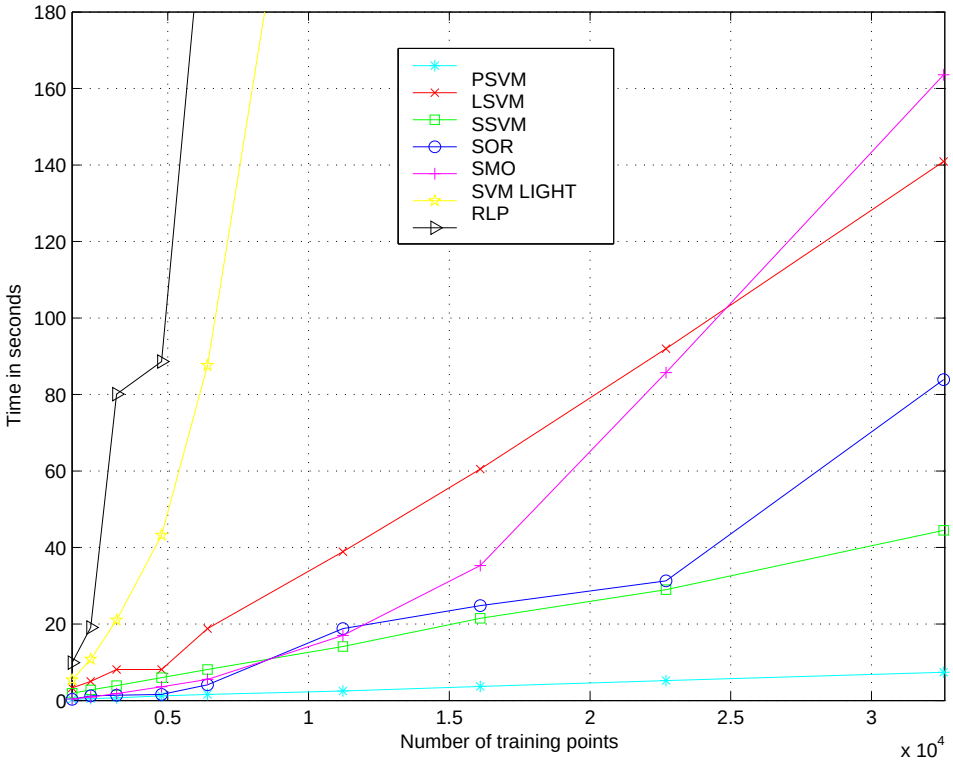

Figure 2.4: Running times on the Adult dataset obtained by seven different methods using a linear classifier. As in table 2.2.1 the slope of the curve corresponding to PSVM indicate that it is much faster than any other method.

Dataset size	Testing Correctness % Running Time *Sec.*						
(Training, Testing)	Method						
n = no. of attributes	PSVM	LSVM	SSVM	SOR	SMO	SVM^{light}	RLP
$(1605, 30957)$	84.00	**84.27**	**84.27**	84.06	-	84.25	78.68
$n = 123$	**0.3**	3.3	1.9	**0.3**	0.4	5.4	9.9
$(2265, 30297)$	84.13	**84.66**	84.57	84.24	-	84.43	77.19
$n = 123$	**0.5**	5.0	2.8	1.2	0.9	10.8	19.12
$(3185, 29377)$	84.25	84.55	**84.63**	84.23	-	84.40	77.83
$n = 123$	**0.7**	8.1	3.9	1.4	1.8	21.0	80.1
$(4781, 27781)$	84.35	**84.55**	**84.55**	84.28	-	84.47	79.15
$n = 123$	**1.2**	8.1	6.0	1.6	3.6	43.2	88.6
$(6414, 26148)$	84.49	**84.68**	84.60	84.30	-	84.43	71.85
$n = 123$	**1.6**	18.8	8.1	4.1	5.5	87.6	218.8
$(11221, 21341)$	84.48	**84.84**	84.79	84.37	-	84.68	60.00
$n = 123$	**2.5**	38.9	14.1	18.8	17.0	306.6	449.2
$(16101, 16461)$	84.78	**85.01**	84.96	84.62	-	84.83	72.52
$n = 123$	**3.7**	60.5	21.5	24.8	35.3	667.2	632.6
$(22697, 9865)$	85.16	**85.35**	**85.35**	85.06	-	85.17	77.43
$n = 123$	**5.2**	92.0	29.0	31.3	85.7	1425.6	991.9
$(32562, 16282)$	84.56	**85.05**	85.02	84.96	-	**85.05**	83.25
$n = 123$	**7.4**	140.9	44.5	83.9	163.6	2184.0	1561.1

Table 2.1: Testing set correctness and running times on the larger Adult dataset obtained by seven different methods using a linear classifier. Timing comparisons are approximate because of the different machines used, but they do indicate that PSVM has a distinct edge, e.g. solving the largest problem in 7.4 seconds, much faster than any other method. Best results are shown in bold.

Data Set $m \times n$	PSVM Train Test Time (Sec.)	SSVM Train Test Time (Sec.) p-value *	LSVM Train Test Time (Sec.) p-value*
Ionosphere 351×34	96.5% 95.2% **4.60**	**97.0 %** **95.8 %** 25.25 0.71	**97.0 %** **95.8%** 14.58 0.71
BUPA Liver 345×6	75.7% 73.6% **4.34**	**75.8%** **73.7%** 20.65 0.97	**75.8%** **73.7%** 30.75 0.97
Tic-Tac-Toe 958×9	98.0% **98.4%** **74.95**	98.0% **98.4%** 395.30 1	**98.2%** 94.7% 350.64 1
Mushroom ** 8124×22	88.0% 88.0% **35.50**	**89.0%** **88.8%** 307.66 0.09	87.6 87.8 503.74 0.79

Table 2.2: PSVM, SSVM and LSVM training and ten-fold testing correctness and running times using a nonlinear classifier. Execution times include ten-fold training. Same value of ν was used in all the methods. Best results are in bold.

* **Paired t-test p-values are calculated for each method relative to PSVM for ten-fold test correctness.**

** **A Rectangular kernel [53] of the size** 8124×215 **was used here instead of the square** 8124×8124 **kernel which does not fit into memory.**

Data Set $m \times n$	PSVM Train Test Time (Sec.)	SSVM Train Test Time (Sec.) p-value *	LSVM Train Test Time (Sec.) p-value*	SVM^{light} Train Test Time (Sec.) p-value*
WPBC (60 mo.) 110×32	**70.8%** **68.5%** **0.02**	**70.8%** **68.5%** 0.17 1	**70.8%** **68.5%** 0.53 1	62.7% 62.7% 3.85 0.30
Ionosphere 351×34	90.7% 87.3% **0.17**	94.3 % **88.7 %** 1.23 0.55	**94.4 %** **88.7 %** 1.40 0.55	91.4 % 88.0 % 2.19 0.71
Cleveland Heart 297×13	87.0% 85.9% **0.01**	87.3% 86.2% 0.7 0.91	87.3% 86.2% 0.78 0.91	**87.7%** **86.5 %** 1.44 0.80
Pima Indians 768×8	77.9% 77.5% **0.02**	**78.2%** **77.6%** 0.78 0.95	**78.2%** **77.6%** 2.18 0.95	77.0 % 76.4 % 37.00 0.59
BUPA Liver 345×6	70.1% 69.4% **0.02**	70.2% **70.0%** 0.78 0.84	70.2% **70.0%** 2.18 0.72	**70.6%** 69.5% 6.65 0.96
Galaxy Dim 4192×14	93.7% 93.5% **0.34**	**95.0%** **95.0%** 5.21 4×10^{-4}	**95.0%** **95.0%** 21.56 4×10^{-4}	94.2 % 94.1 % 28.33 0.14
Mushroom 8124×22	81.0% 81.0% **1.15**	**81.7%** **81.5%** 11.73 0.49	**81.7%** **81.5%** 61.62 0.49	81.5% **81.5%** 145.59 0.48

Table 2.3: PSVM, SSVM, LSVM and SVM^{light} training and ten-fold testing correctness and running times using a linear classifier. Execution times include ten-fold training. Same value of ν was used in all the methods. The value of the parameter C in SVM^{light} was chosen by tuning. Best results are in bold. * Paired t-test p-values are calculated for each method relative to PSVM for ten-fold test correctness.

Method	Dataset	Training Correctness %	Testing Correctness %	Time (CPU) sec.
LSVM	NDC-easy	90.86	91.23	658.5
PSVM	NDC-easy	90.80	91.13	20.8
LSVM	NDC-hard	69.80	69.44	655.6
PSVM	NDC-hard	69.84	69.52	20.6

Table 2.4: LSVM and PSVM performance on two, 2 million point NDC datasets with 10-attributes. A linear classifier with parameter $\nu = 0.1$ was used in both methods on the same locop1 machine. SVM^{light} failed to solve this problem due to memory problems.

2.3 Incremental Proximal Support Vector Machine

We describe now how a simple procedure can be applied to the proximal SVM, described in the previous section, that will allow it to generate a new linear classifier (2.6) by retiring old data while simultaneously adding new data. To do that let us augment the input vector $x \in R^n$ by -1 and define an augmented input vector $z \in R^{n+1}$ as follows:

$$z := \begin{bmatrix} x \\ -1 \end{bmatrix}. \tag{2.31}$$

Coupling this definition with the solution expression for $\begin{bmatrix} w \\ \gamma \end{bmatrix}$ given by (2.11) and (2.12), the linear classifier (2.6) can be written explicitly in terms of the input data as follows:

$$sign(z'(\frac{I}{\nu} + E'E)^{-1}E'De\ ) \begin{cases} = 1, \ \text{ then } x \in A+, \\ \\ = -1, \ \text{ then } x \in A-, \end{cases} \tag{2.32}$$

where, as before, $E = [A \quad -e]$. Two key properties of the classifier (2.32) are:

(i) The size of the data to be stored is of order of $(n+1)^2$ for $E'E$ and of order $(n+1)$ for $E'De$, even if the number of data points is of the order of millions.

(ii) Time to solve the system of linear equations determined by the positive definite matrix $\frac{I}{\nu} + E'E$ is of order $(n+1)^3$.

In order to reduced numerical errors on the calculation of $E'E$ scaling of the data matrix A may be necessary.

Since n is typically less than 100 and often around 10, the above classifier is extremely effective for massive datasets with millions of data points. Based on these simple facts we describe an incremental algorithm that is capable of retiring any desired portion of the data while at the same time adding new data to generate an appropriately altered

classifier. To keep the notation simple assume that the current classifier is based on an input dataset $E \in R^{m\times(n+1)}$ and a corresponding diagonal matrix $D \in R^{m\times m}$ of ± 1. Suppose that an "old" subset of this data represented by the submatrix $E^1 \in R^{m^1\times(n+1)}$ of E and a corresponding diagonal submatrix $D^1 \in R^{m^1\times m^1}$ of ± 1 of D needs to be retired and removed from the characterization of our classifier, leaving its complement in E to determine the new classifier together with new data. The "new" set of data points is represented by a new matrix $E^2 \in R^{m^2\times(n+1)}$ and a corresponding diagonal matrix $D^2 \in R^{m^2\times m^2}$ of ± 1 that needs to be added to the characterization of our new classifier. The classifier (2.32) can be updated to reflect the retirement of E^1 and the addition of E^2 as stated below in our incremental algorithm.

Algorithm 2.3.1 Linear Incremental SVM *Given m data points in R^n represented by the $m \times n$ matrix A and a diagonal matrix D of ± 1 labels denoting the class of each row of A, we generate an incremental linear classifier by retiring old data represented by the submatrix $E^1 \in R^{m^1\times(n+1)}$ of $E = [A \quad -e]$ and a corresponding diagonal submatrix $D^1 \in R^{m^1\times m^1}$ of D of ± 1 and adding new data represented by a new matrix $E^2 \in R^{m^2\times(n+1)}$ and a corresponding diagonal matrix $D^2 \in R^{m^2\times m^2}$ of ± 1 as follows:*

$$sign(z'(\tfrac{I}{\nu} + E'E - E^{1\prime}E^1 + E^{2\prime}E^2)^{-1}(E'De - E^{1\prime}D^1e + E^{2\prime}D^2e\ )) = \begin{cases} 1, & \textit{then } x \in A+, \\ -1, & \textit{then } x \in A-. \end{cases} \tag{2.33}$$

Note that for each block of data, say $E^i \in R^{m^i\times(n+1)}$, all we need to store is the relatively small $(n+1)\times(n+1)$ matrix $E^{i\prime}E^i$ and the $(n+1)\times 1$ vector $E^{i\prime}D^ie$. These quantities, which are of order than $(n+1)^2$ and $n+1$ respectively, allow us to retire and add data to classifiers as often as we wish. Furthermore, this incremental process allows us to handle arbitrarily large datasets, by successively adding blocks of data in the form of $E^{i\prime}E^i$ and

$E^{i'}D^ie$, as will be demonstrated by the numerical results of the next section. Note also that $E^{i'}E^i$ which is of order $(n+1)^2$ can be considered to be a compression of the much larger dataset E^i which is of order $m^i(n+1)$. Since it takes $2(n+1)^2m^i$ operations to compute $E^{i'}E^i$ and $2(n+1)m^i$ operations to compute $E^{i'}D^ie$, the amount of computation of the incremental algorithm only grows linearly in the number of data points $m = \sum m^i$ as shown in Figure 2.6.

2.3.1 Numerical Testing

Because we are principally interested in massive problems, we focus our numerical tests on two datasets synthetically generated and totally stored on disk. These sets are generated by Musicant's NDC (normally distributed clustered) dataset generator [78]. The data is generated as clusters of normally distributed points in R^n with an adjustable linear separability.

The first dataset consists of 1 billion points in 10-dimensional input space. The purpose of this dataset is to demonstrate the capability of our incremental SVM algorithm to obtain linear classifier by 500 increments of 2 million points each. The 1-billion dataset was generated first, and stored in 500 subsets of size 2 million points each. A testing set of size 10 million, was also generated with the same distribution used to generate the 1 billion dataset and hence, resulting in essentially the same linear separability. It is important to note that every block of data is read from disk only once, and that once the new block of data is processed, the only data that has to be kept in memory is a matrix of size 11×11 and a vector of size 11×1 in our case here. Every time a new block is read from disk, new incoming data is processed by the algorithm, the 11×11 matrix and 11×1 vector are updated, and after this, all the processed data is discarded, leaving the

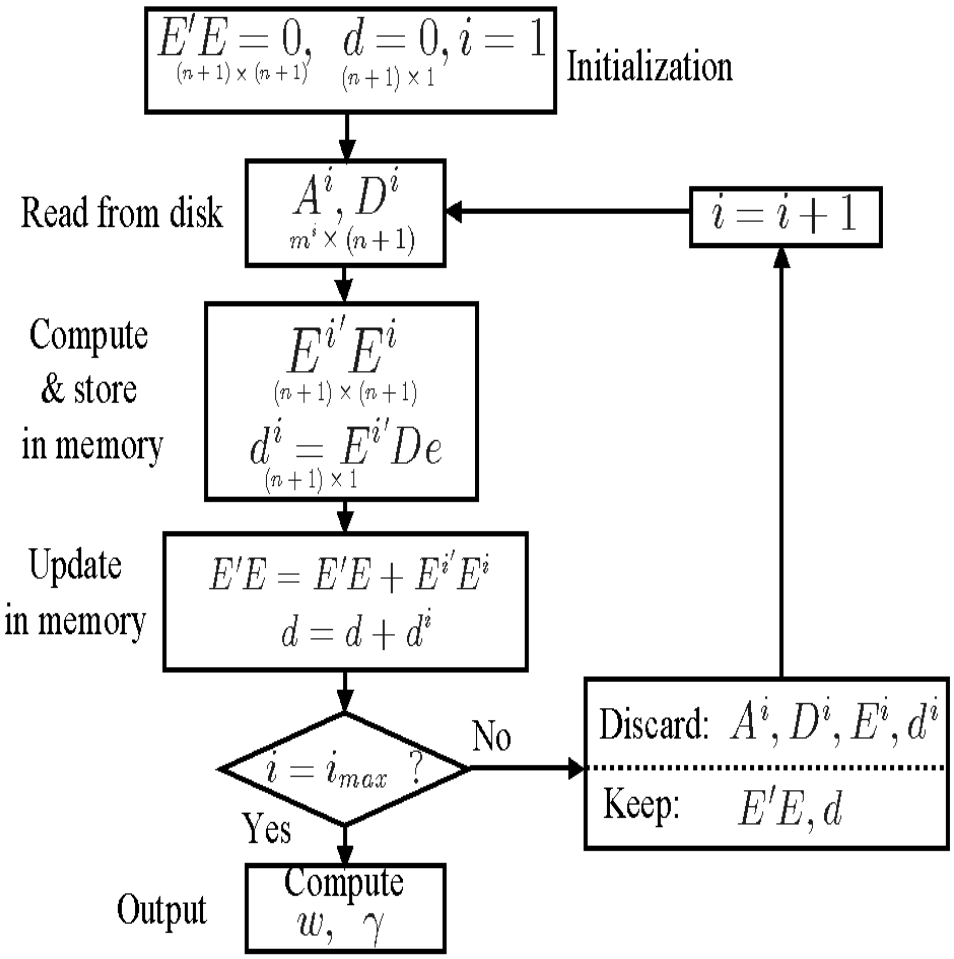

Figure 2.5: Flow chart for the Linear Incremental Algorithm 2.3.1.

memory ready for new incoming data. This process is graphically depicted in Figure 2.5.

The proposed incremental algorithm solved this 1-billion point problem in less than 2 hours and 26 minutes (8747 seconds). About 43 minutes, slightly less than 30% of the total time, was spent reading data from disk. Training set correctness was 90.78 % while testing set correctness was 90.79%. A graph exhibiting the linear dependence of computational time of the algorithm on the number of data points used to generate the solution $\begin{bmatrix} w \\ \gamma \end{bmatrix}$ of (2.12) and (2.11) is presented in Figure 2.6.

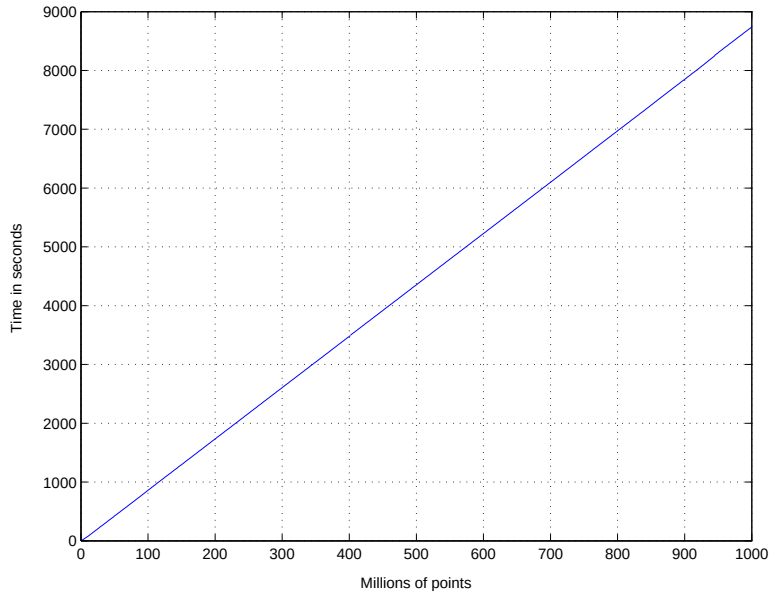

Figure 2.6: Computational time of the Linear Incremental Algorithm 2.3.1 as a function of the number of data points used to compute the solution $\begin{bmatrix} w \\ \gamma \end{bmatrix}$ of (2.28) for the 1-billion point dataset. The straight line depicts actual computational times obtained for 500 different problems each incremented by 2 million points.

The second dataset of 60 million points in 10-dimensional input space is intended to simulate two months of data. We use this dataset to demonstrate how our algorithm can incrementally retire 1 million old points and add 1 million new points in 30 successive steps which simulate daily retirement of the oldest data and addition of the newest data. In the previous experiment all the data was assumed to be on disk from the beginning and just one final separating hyperplane was required. In this case at the beginning we assume that we just have 30 days of data and every day the oldest block of data has to be retired (1-million points) and one new block of data (1-million points) corresponding to the new data just collected has to be added. Furthermore, we assume that at every time that an update of the data is made, a new separating hyperplane has to be calculated. The data was generated in such a way that the starting and final separating hyperplanes differ considerably. This means that the incoming data is constantly changing and the

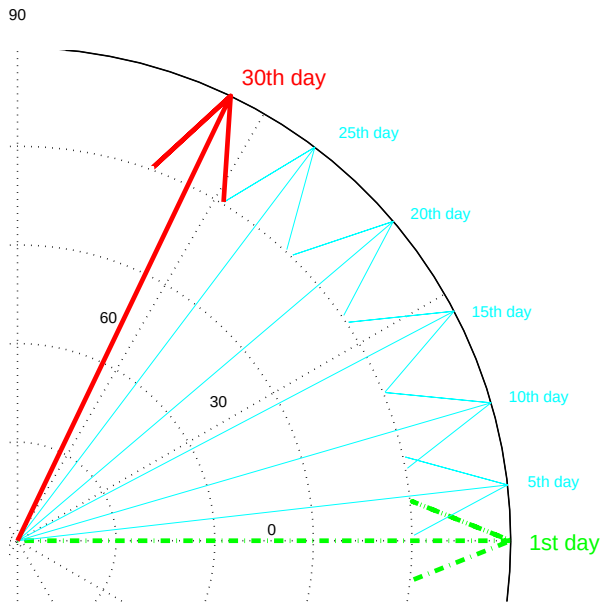

Figure 2.7: Normals to the separating hyperplanes $x'w^i = \gamma^i,\ i = 1, \ldots, 30$, corresponding to 5-day intervals. Separating hyperplanes change due to the incorporation of new data and the retirement of old data as described in Algorithm 2.3.1.

separating criteria to be learned changes dynamically with respect to time. A gradual change in the resulting hyperplane after incrementally retiring 1 million old points and adding 1 million new points can be observed. In order to show this gradual change, we measure the differences between planes by calculating the angle α between their normals. Thus, given two hyperplane normals $w^i, w^j \in R^n$ we compute:

$$\cos\ \alpha = \frac{|w^{i\prime} w^j|}{\|w^i\| \|w^j\|}$$

A plot showing the angles between the normals to the hyperplanes as the data changes is depicted in Figure 2.7.

2.4 Multicategory Proximal Support Vector Machine Classifiers

It is the purpose of this section to apply the simple 2-class PSVM classifier to k-category classification by using a one-from-rest (OFR) separation for each class [8]. However, due to the fact that the number of points belonging to one class is usually much smaller than the number of points in the union of the remaining classes, the resulting two-class problems are very unbalanced. It is important to note that this unbalanced property also appears often in real life datasets. PSVM fits each class with one of two distant parallel planes and errors in both classes are penalized similarly in the objective function. Because of the unbalanced classes, PSVM tends to fit better the class with more data points and it underestimates the overall error of the class with fewer data points. This often results in a poor PSVM performance. In order to override this difficulty, we propose a balanced modification of PSVM which weights each class equally no matter how many points are in each class. In addition, we propose a very fast Newton refinement algorithm, which is applicable to any SVM classification approach, and which leads to a better classifier. Experimental results show that incorporation of these two modifications into a plain PSVM one-from-the-rest approach improves significantly test set correctness while while maintaining its speed.

In contrast, other one-from-the-rest and SVM k-class classifiers [5, 8, 14] require the solution of either a large single or k smaller quadratic or linear programs that need specialized optimization codes such as CPLEX [21]. On the other hand, obtaining a linear or nonlinear PSVM classifier as we propose here, requires nothing more sophisticated than solving k systems of linear equations. Efficient and fast linear equation solvers are freely

available [1] or are part of standard commercial packages such as MATLAB [75], and can solve very large systems. We note that in [94, 97], multiclass least squares formulations are proposed that explicitly require Mercer's positive definiteness condition [19, 102] on the kernels used which is not needed here. In addition, the problem in [97] is formulated as single large constrained optimization problem in contrast to the k smaller uncoupled and unconstrained OFR approach used here. Various multiclass schemes are investigated in [39, 106]. We also note that, in concept, PSVM can be interpreted as ridge regression [46] which is essentially regularized least squares [98]. However, ridge regression in its general form lacks the geometric justification and interpretation of PSVM which consists of constructing two parallel planes, each proximal to one of two classes of data points, while simultaneously pushing these plane as far apart as possible. A ridge regression application similar to PSVM is given in [101], which however uses a variation of the EM-algorithm to solve the classification problem, whereas we use a straightforward solution of the normal equations of regularized least squares. Interesting numerical comparison of multiclass methods is given in [47].

2.4.1 PSVM Modification for Unbalanced Classes

In order to improve PSVM performance when one of classes has many more data points than the other one, which is usually the case in the two-class subproblems that the OFR approach generates, we propose the following simple balancing approach.

Let m_1 and m_2 be the number of points in classes 1 and -1 respectively. We first

define an $m \times m$ diagonal matrix N as follows:

$$N_{ii} = \begin{cases} \frac{1}{m_1}, & \text{if } d_{ii} = 1, \\ \frac{1}{m_2}, & \text{if } d_{ii} = -1. \end{cases} \tag{2.34}$$

We then formulate the following *balanced* PSVM problem:

$$\min_{(w,\gamma)\in R^{n+1}} \frac{\nu}{2}(D(Aw - e\gamma) - e)'N(D(Aw - e\gamma) - e) + \frac{1}{2}\left\|\begin{bmatrix} w \\ \gamma \end{bmatrix}\right\|^2 \tag{2.35}$$

This formulation is equivalent to re-weighting the examples on the training set, so that examples in a class with fewer members have more "influence" on the objective function that points in a majority class.

Setting the gradient with respect to w and γ equal to zero and noting that $D^2 = I$ and $DND = N$ we obtain the following necessary and sufficient optimality conditions for (2.35):

$$\begin{aligned} \nu A'N(Aw - e\gamma - De) + w &= 0 \\ \nu e'N(-Aw + e\gamma + De) + \gamma &= 0 \end{aligned} \tag{2.36}$$

We describe now a computational enhancement to PSVM which is also applicable to other SVM classifiers as well.

2.4.2 Newton Refinement

The simple computational refinement that we have implemented, and which is applicable to any type of SVM classifier, consists of taking a solution obtained by either a linear or nonlinear classifier, say for simplicity a solution $\begin{bmatrix} \bar{w} \\ \bar{\gamma} \end{bmatrix}$ to the PSVM problem (2.8), which generates a separating plane $x'\bar{w} - 1 \cdot \bar{\gamma} = 0$ as shown in Figure 2.2. The idea here is to move this plane parallel to itself in such a way to improve the separation of the two

sets $A+$ and $A-$. One way to measure such improvement is by counting the number of misclassified points as was done in [18]. A simpler way is to slightly alter the objective function of (2.8) so that the first term is zero if all the points are correctly classified by the separating plane. This is easily achieved by setting nonnegative components of $D(Aw - e\gamma) - e$, which correspond to correctly classified points, equal to zero, that is: $(-D(Aw - e\gamma) + e)_+ = 0$, where as defined in the Introduction, $(z)_+ = \max\{0, z\}$. Thus the minimization problem (2.8) becomes:

$$\min_{(w,\gamma)\in R^{n+1}} \frac{\nu}{2}\|((-D(Aw - e\gamma) + e)_+\|^2 + \frac{1}{2}\left\|\begin{bmatrix} w \\ \gamma \end{bmatrix}\right\|^2, \tag{2.37}$$

which is the optimization problem underlying the smooth support vector machine algorithm [53]. Since we are only interested in merely refining our solution while maximizing the margin $\begin{bmatrix} \bar{w} \\ \bar{\gamma} \end{bmatrix}$ of (2.8), we replace w by $\lambda\bar{w}$ in (2.37) and obtain our refinement problem:

$$\min_{(\lambda,\gamma)\in R^2} \; f(\lambda, \gamma) = \frac{\nu}{2}\|((-D(\lambda A\bar{w} - e\gamma) + e)_+\|^2 + \frac{1}{2}\left\|\begin{bmatrix} \lambda\bar{w} \\ \gamma \end{bmatrix}\right\|^2 \tag{2.38}$$

This is a simple strongly convex problem in the 2-dimensional space of (λ, γ), that can be very quickly solved by a fast Newton method, the quadratic convergence and effectiveness of which has been established in [53] for the full problem (2.37) in the $n+1$ dimensional space (w, γ). We briefly describe this approach now. We first need the expressions for the gradient and generalized Hessian matrix [28, 43] of $f(\lambda, \gamma)$ as follows. We first define:

$$d(\lambda, \gamma) = (-D(\lambda A\bar{w} - e\gamma) + e), \tag{2.39}$$

then the 2×1 gradient and the 2×2 generalized Hessian matrix of $f(\lambda, \gamma)$ are given by:

$$\nabla f(\lambda, \gamma) = \begin{bmatrix} -\nu\bar{w}'A'D(d(\lambda, \gamma))_+ + \| \bar{w} \|^2 \lambda \\ \nu e'D(d(\lambda, \gamma))_+ + \gamma \end{bmatrix}, \tag{2.40}$$

and,

$$\partial^2 f(\lambda,\gamma) = \begin{bmatrix} \nu\bar{w}'A'EA\bar{w} + \parallel \bar{w} \parallel^2 & -\nu\bar{w}'A'Ee \\ -\nu e'EA\bar{w} & \nu e'Ee + 1 \end{bmatrix}, \tag{2.41}$$

where E is the diagonal matrix:

$$E = Ddiag((d(\lambda,\gamma))_*)D = diag((d(\lambda,\gamma))_*), \tag{2.42}$$

and the $(\cdot)_*$ is the step function defined in the Introduction and which is taken here as a specific subgradient [85, 88] of the plus function $(\cdot)_+$ and is used to generate the generalized Hessian matrix in the same manner as in [51, 53].

A key difference between PSVM and SVM, is that with PSVM the conventional concept of support vectors (the data points corresponding to the positive multipliers) does not hold [33]. However, it is interesting to note that after this refinement is applied to the PSVM solution, the concept of support vectors applies to the new solution. If the pair (λ^*, γ^*) is the solution obtained by (2.38), then the corresponding dual multipliers associated with this problems are given by [53]:

$$u = (-D(\lambda^* A\bar{w} - e\gamma^*) + e)_+ \tag{2.43}$$

Then, the support vectors for the problem (2.38) are the data points of A corresponding to positive components of u.

The Newton refinement procedure can then be summarized as follows.

Algorithm 2.4.1 Newton Refinement *Given a solution* $\begin{bmatrix} \bar{w} \\ \bar{\gamma} \end{bmatrix}$ *to the PSVM 2-class problem (2.8) refine it as follows:*

(i) Start with $\lambda^0 = 1$ *and* $\gamma^0 = \bar{\gamma}$. *maxiter (maximum number of iterations).*

(ii) Iterate (iii) until either $j = maxiter$ *or:*

$$\left\| \begin{bmatrix} \lambda^j \\ \gamma^j \end{bmatrix} - \begin{bmatrix} \lambda^{j+1} \\ \gamma^{j+1} \end{bmatrix} \right\| \leq 10^{-3}, \tag{2.44}$$

in which case $\begin{bmatrix} w \\ \gamma \end{bmatrix} = \begin{bmatrix} \lambda^{j+1}\bar{w} \\ \gamma^{j+1} \end{bmatrix}$ *is the refined solution to (2.8).*

(iii) Calculate the new iterates:

$$\begin{bmatrix} \lambda^{j+1} \\ \gamma^{j+1} \end{bmatrix} = \begin{bmatrix} \lambda^j \\ \gamma^j \end{bmatrix} - \nabla^2 f(\lambda^j, \gamma^j)^{-1} \nabla f(\lambda^j, \gamma^j) \tag{2.45}$$

With obvious modifications this algorithm can be applied to refine a solution $\begin{bmatrix} \bar{u} \\ \bar{\gamma} \end{bmatrix}$ of the nonlinear PSVM (2.51) as well.

In order to illustrate the proposed modifications we generated a small unbalanced artificial two-dimensional two-class dataset. The dataset consist of 100 points, 85 of which are in class $A+$ and 15 points in class $A-$. When the problem is solved using plain PSVM (2.8), the influence of the 85 points in class $A+$ prevails over that of the much smaller set of data points in $A-$. As a result, 14 out of 15 points in class $A-$ are misclassified. The total training set correctness is 86%, with only 6.6% correctness for the smaller class $A-$ and 100% correctness for the larger class $A+$. The resulting separating plane is shown in Figure 2.8. When a balanced PSVM (2.35) is used we can see an improvement over the plain PSVM, in the sense that a separating plane is obtained that correctly classifies *all* the points in class $A-$. However due to the significant difference in the cardinality of the two classes and the distribution of their points, a subset of 16 points in class $A+$ is now misclassified. The total training set correctness is 84%, with 100% correctness for $A-$ points and 81.2% correctness for $A+$ points. The resulting separating plane is shown in Figure 2.9. If now in addition to balancing, the Newton refinement is also applied, we obtain a separating plane that misclassifies only two points. The total training set correctness is 98%. The resulting separating plane is shown in Figure 2.10.

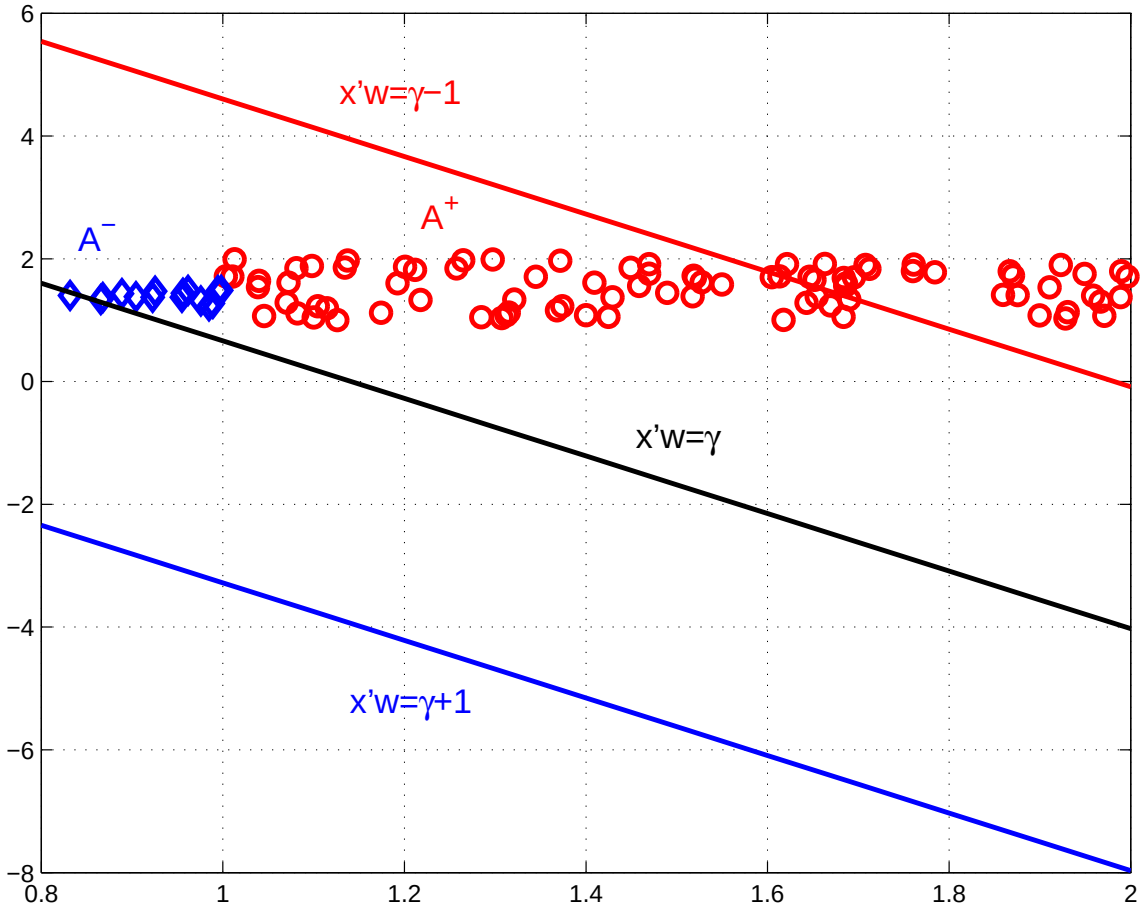

Figure 2.8: An unbalanced dataset consisting of 100 points, 85 of which in class $A+$ represented by hollow circles, and 15 points of which in class $A-$ represented by hollow diamonds. The separating plane is obtained by using a plain PSVM (2.8). The class $A-$ is practically ignored by the solution. The total training set correctness is 86% with 6.6% correctness for $A-$ and 100% correctness for $A+$.

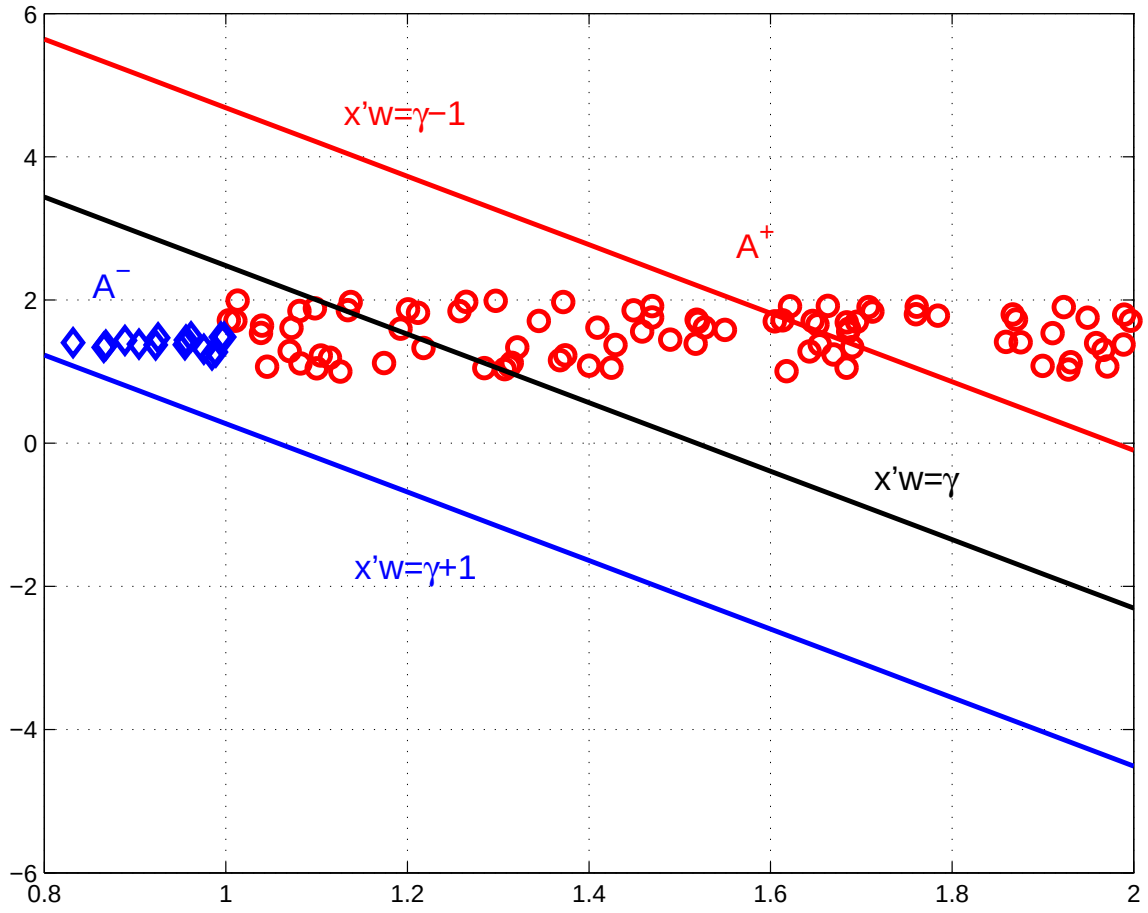

Figure 2.9: Linear classifier improvement by balancing is demonstrated on the same dataset of Figure 2.8. The separating plane is obtained by using a balanced PSVM (2.35). Even though the class $A-$ is correctly classified in its entirety, the overall performance is still rather unsatisfactory due to significant difference in the distribution of points in each of the classes. Total training set correctness is 84%.

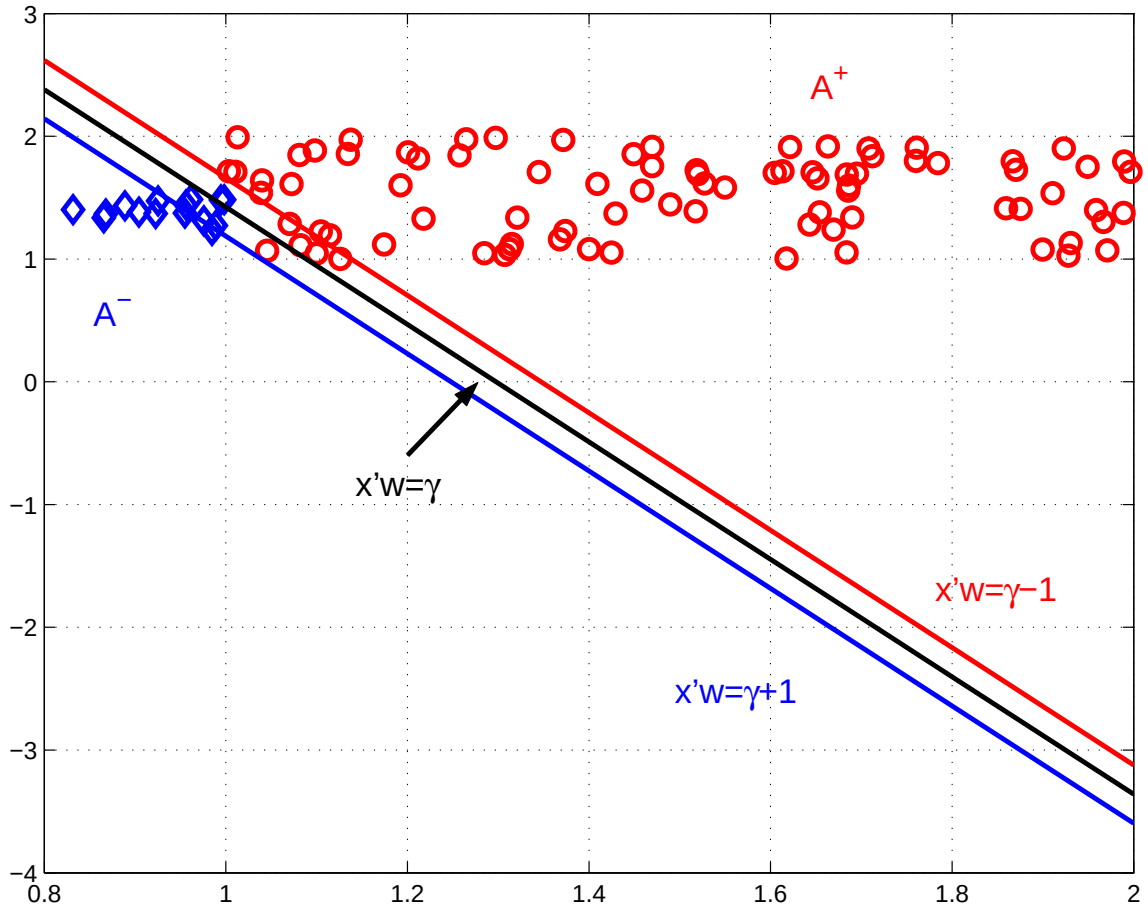

Figure 2.10: Very significant linear classifier improvement as a consequence of balancing and the use of the Newton refinement is demonstrated on the same dataset of Figures 2.8 and 2.9. The separating plane is obtained using both modifications to PSVM: balancing and Newton refinement . The total training set correctness is now 98% compared to 86% for plain PSVM and 84% for balanced PSVM.

To extend this formulation to k classes, all we need is to redefine the following for separating class r from the rest:

$$\begin{aligned} A &= \begin{bmatrix} A^1 \\ \vdots \\ A^k \end{bmatrix} \\ A+ &= A^r, \\ A- &= \begin{bmatrix} A^1 \\ \vdots \\ A^{r-1} \\ A^{r+1} \\ \vdots \\ A^k \end{bmatrix} \\ r &\in \{1, \ldots, k\}, \end{aligned} \tag{2.46}$$

where, $A^r \in R^{m^r \times n}$ represents the m^r points in class r. We then define for $m = m^1 + \ldots + m^k$ the $m \times m$ diagonal matrix D of ± 1 as follows:

$$\begin{aligned} D_{ii} &= 1 \text{ for } A_i \in A^r \\ D_{ii} &= \text{-1 for } A_i \notin A^r \\ r &\in \{1, \ldots, k\} \end{aligned} \tag{2.47}$$

We note that since the multicategory classification problem, $A-$ has many more rows than $A+$, a normalization is usually carried out by dividing the error vector y_i by m^r for $A_i \in A^r$ and by $(m - m^r)$ for $A_i \notin A^r$. Here, m^r is the number of points in class r which is represented by the matrix $A^r \in R^{m^r \times n}$.

Once the k minimization problems (2.8) are solved (with A and D defined as in (2.46) and (2.47)) by solving the linear system of equations (2.36), k unique separating planes are generated:

$$x'w^r - \gamma^r = 0, \quad r = 1, \ldots, k. \tag{2.48}$$

A given new point $x \in R^n$ is assigned to class s, depending on which of the k halfspaces generated by the k planes (2.48) it lies deepest in, that is:

$$x'w^s - \gamma^s = \max_{r=1,\ldots,k} x'w^r - \gamma^r. \tag{2.49}$$

For concreteness we explicitly state our multicategory PSVM algorithm.

Algorithm 2.4.2 Linear Multicategory Proximal SVM *Given m data points in R^n, each belonging to one of k classes and represented by k matrices A^r of order $m^r \times n$, $r = 1, \ldots, k$, with $m^1 + \ldots + m^k = m$, we generate the linear classifier (2.49) as follows:*

(i) Solve k independent nonsingular systems of $(n+1)$ linear equations (2.36) in $(n+1)$ unknowns, with A and D defined as in (2.46) and (2.47), for some positive value of ν. (Typically ν is chosen by means of a tuning set.)

(ii) Apply the Newton Refinement 2.4.1 to each solution $(\bar{w}^r, \bar{\gamma}^r)$, $(r = 1 \ldots k)$ obtained on step (i) to get the refined solutions (w^r, γ^r) .

(iii) The point x belongs to class s as determined by the criterion (2.49).

We extend now the above results to nonlinear proximal support vector machines that result in nonlinear proximal surfaces instead of planes in the input space.

2.4.3 Nonlinear Multicategory Proximal Support Vector Machines

To obtain our nonlinear proximal classifier we modify our proximal minimization problem (2.8) as in [33, 65] by first replacing the primal variable w by its dual equivalent , $w = A'Du$, and modifying the last term of the objective function to be the norm of the new dual variable u and γ. We obtain then the following problem:

$$\min_{(u,\gamma)\in R^{m+1}} \frac{\nu}{2}\|D(AA'Du - e\gamma) - e\|^2 + \frac{1}{2}\left\|\begin{bmatrix} u \\ \gamma \end{bmatrix}\right\|^2. \tag{2.50}$$

If we now replace the linear kernel AA' by a nonlinear kernel $K(A, A')$, as defined in the Introduction, we obtain:

$$\min_{(u,\gamma)\in R^{m+1}} \frac{\nu}{2}\|D(K(A, A')Du - e\gamma) - e\|^2 + \frac{1}{2}\left\|\begin{bmatrix} u \\ \gamma \end{bmatrix}\right\|^2. \tag{2.51}$$

As in the linear kernel case, we extend the above two category case to k categories by redefining A and D as in (2.46) and (2.47) to obtain k minimization problems. Setting the gradient with respect to u and γ to zero and noting again that $D^2 = I$ gives the following necessary and sufficient optimality conditions for (2.51):

$$\begin{aligned} \nu D(K(A, A')'K(A, A')Du - e\gamma - De) + u &= 0 \\ \nu e'(-K(A, A')Du + e\gamma + De) + \gamma &= 0. \end{aligned} \tag{2.52}$$

Once the k minimization problems (2.51) are solved (with A and D defined as in (2.46) and (2.47)) by solving the k independent linear systems of equations (2.52), k unique proximal surfaces are generated:

$$K(x', A')Du^r - \gamma^r = 0, \quad r = 1, \ldots, k. \tag{2.53}$$

A given new point $x \in R^n$ is assigned to class s depending on which of the k nonlinear halfspaces generated by the k surfaces (2.53) it lies deepest in, that is:

$$K(x', A')Du^s - \gamma^s = \max_{r=1,\ldots,k} K(x', A')Du^r - \gamma^r. \tag{2.54}$$

For concreteness we explicitly state our multicategory nonlinear PSVM algorithm.

Algorithm 2.4.3 Nonlinear Multicategory Proximal SVM *Given m data points in R^n, each belonging to one of k classes and represented by k matrices A^r of order $m^r \times n$, $r = 1, \ldots, k$, with $m^1 + \ldots + m^k = m$, we generate the nonlinear classifier (2.54) as follows:*

(i) Solve k independent nonsingular systems of $(m+1)$ linear equations (2.52) in $(m+1)$ unknowns, with A and D defined as in (2.46) and (2.47), for some positive value of ν. (Typically ν is chosen by means of a tuning set.)

(ii) Apply the Newton refinement algorithm 2.4.1 to each solution $(\bar{u}^r, \bar{\gamma}^r)$, $(r = 1 \ldots k)$ obtained on step (i) to get the refined solutions (u^r, γ^r) .

(ii) The point x belongs to class s as determined by the criterion (2.54).

When each of the k subproblems become large enough so as not to fit in memory, then the $m \times m$ kernel $K(A, A')$ is replaced by the considerably smaller $m \times \bar{m}$ rectangular kernel $K(A, \bar{A}')$, where $\bar{A}$ consists of as little as 15% of randomly chosen rows of A. This leads to the extremely fast and effective Reduced Support Vector Machine (RSVM) algorithm as described in [52] and presented next.

Algorithm 2.4.4 RSVM Algorithm

(i) Choose a random subset matrix $\bar{A} \in R^{\bar{m} \times n}$ *of the original data matrix* $A \in R^{m \times n}$. *Typically* $\bar{m}$ *is 1% to 15% of* m, *and* $\bar{A}$ *consists of the union of random samples of each class that maintain the original relative sizes of the* k *classes.*

(ii) Solve the following modified version of the PSVM (2.51) where A' **only** *is replaced by* $\bar{A}'$ *with corresponding* $\bar{D} \subset D$:

$$\min_{(\bar{u},\gamma) \in R^{m+1}} \frac{\nu}{2} \| D(K(A, \bar{A}')\bar{D}\bar{u} - e\gamma) - e \|^2 + \frac{1}{2} \left\| \begin{bmatrix} \bar{u} \\ \gamma \end{bmatrix} \right\|^2 \tag{2.55}$$

which is equivalent to solving (2.51) with A' **only** *replaced by* $\bar{A}'$.

The separating k *surface is given by (2.53) with* A' *replaced by* $\bar{A}'$ *as follows:*

$$K(x', \bar{A}')\bar{D}\bar{u}^r = \gamma^r, \tag{2.56}$$

where $(\bar{u}, \gamma) \in R^{\bar{m}+1}$ *is the unique solution of (2.55), and* $x \in R^n$ *is a free input space variable of a new point.*

We turn now to our numerical results.

2.4.4 Numerical Implementation and Comparisons

Our algorithms require the solution of k square systems of linear equations, where k is the number of classes to be classified. Each one of the linear systems of equations involved is of the size of the number of input attributes n plus one in the linear case, and of the size of the number of data points m plus one in the nonlinear case. When using a rectangular kernel [53], the size of the problem can be reduced from m to $\bar{m}$ with $\bar{m} < m$ for the nonlinear case.

The real life datasets used for our numerical tests are the following:

- Four publicly available datasets from the UCI Machine Learning Repository [77]: Wine, Glass, Iris, Vowel, with 3, 6, 3 and 11 categories respectively.

- Two publicly available datasets from the Statlog Project Databases, also available from UCI [77]: Vehicle and Segment, with 4 and 7 categories respectively.

Properties of each dataset such as number of points, number of features and number of classes are given in Table 2.4.4.

Numerical Experiments Using Multicategory Linear Classifiers

We compared the performances of the methods described below.

- **Linear OFRQP**: One-From-Rest Quadratic Programming classifier using a standard support vector machine formulation for each subproblem and solved using a MATLAB-CPLEX interface [21]. CPLEX is a state of the art software widely used to solve linear and quadratic programs.

- **Linear MPSVM** : Multicategory Proximal SVM One-From-Rest classifier using a Linear Proximal support vector machine (PSVM) for each subproblem. Usually, each one-from-rest problem is an unbalanced two-class classification problem. This means that the number of points $m-$ in $A-$ is much larger than the number of points $m+$ in $A+$. In order to address this problem, we apply balancing, which is, a weight factor added to each error term in the objective function of (2.8) that is inversely proportional to the number of points in each class. We call this MPSVM modification Balanced MPSVM (**B-MPSVM**) and is given in (2.35). In order to further improve the performance of B-PSVM, for each two-class classification subproblem we use the Newton Refinement 2.4.1. Although the refinement step is

very simple and fast, in almost all the tested cases this refinement combined with the balancing procedure improved test set correctness of the MPSVM by as much as 16.8% (Table 2.4.4, Iris). We called this MPSVM modification Balanced and Refined MPSVM (**BR-MPSVM**).

The value of the parameter ν in each of these methods was chosen by using a tuning set extracted from the training set. In order to find an optimal value for ν the following tuning procedure was employed on each fold:

- A random tuning set of the the size of 10% of the training data was chosen and separated from the training set.
- Several SVMs were trained on the remaining 90% of the training data using values for ν equal to 2^i, where $i = 0, 1, \ldots, 25$.
- The value of ν that gave the best SVM correctness on the tuning set was chosen.
- A final SVM was trained using the chosen value of ν and all the training data. The resulting SVM was tested on the testing data.

The linear BR-MPSVM running time was often one order of magnitude less than the standard OFRQP time. Furthermore, there was no a significant statistical difference between both methods as far test set correctness was concerned. This is shown by the p-values obtained using a 95% confidence interval paired t-test for the tenfold test set correctness. Experiments indicated that both modifications, balancing and refinement achieved significant accuracy improvements over the plain MPSVM, while maintaining relatively fast performances. Testing set correctness, training set correctness, CPU times and p-values are given in Table 2.4.4.

Data Set $m \times n$ # of Classes	OFRQP Train Test Time (Sec.)	MPSVM Train Test Time (Sec.) p-value†	B-MPSVM Train Test Time (Sec.) p-value†	BR-MPSVM Train Test Time (Sec.) p-value†
Wine 178×13 3	**100.0%** 96.1 % 1.39	**100.0 %** 98.9 % **0.02** 0.20	99.9% 98.9% **0.02** 0.80	**100.0%** **99.4%** 0.11 0.10
Glass 214×9 6	**72.9 %** **67.2 %** 1.80	66.5 % 60.6 % **0.02** 0.19	68.29 % 61.6 % 0.03 0.28	68.9% 63.0 % 0.14 0.35
Iris 150×4 3	**98.7 %** **98.0 %** 0.73	85.6% 83.3% **0.02** $1.2e-6$	86.9 % 86.7 % **0.02** $2.0e-4$	97.6% 97.3% 0.11 0.66
Vowel 528×10 11	**68.7 %** 57.2 % 5.56	54.6% 45.5 % **0.05** $9.9e-3$	56.1% 47.0% **0.05** $1.8e(-2)$	64.5% **57.6%** 0.14 0.93
Vehicle 846×18 4	**83.3 %** **79.0 %** 2.88	79.1% 76.2 % **0.11** $8.8e-2$	81.0% 77.4% **0.11** 0.33	81.1 % 77.5 % 0.34 0.30
Segment 2310×19 7	**93.0 %** **91.9 %** 18.57	85.5% 84.8 % **0.22** $7.5e-7$	90.3% 90.1% 0.31 $2.2e(-2)$	91.3% 90.8% 0.67 0.14

Table 2.5: OFRQP, MPSVM,B-MPSVM,BR-MPSVM linear classifier training correctness, tenfold testing correctness and running times. Execution times include tenfold training. Best results are in bold.

† **p-values calculated with respect to OFRQP for tenfold testing correctness, using a paired t-test with 95% confidence interval.**

Data Set $m \times n$ # of Classes	Nonlinear OFRQP Train Test Time (Sec.)	Nonlinear BR-MPSVM Train Test Time (Sec.) p-value†
Wine 178×13 3	99.2 % 97.7 % 5.39	**100.0 %** **100.0 %** **0.45** $2.5e-2$
Glass 214×9 6	**88.5%** **70.0 %** 9.05	78.09% 69.1% **0.59** 0.84
Iris 150×4 3	98.1 % 98.0 % 3.01	**99.5%** **98.7%** **0.31** 0.62
Vowel 528×10 11	**100.0%** 94.3 % 221.34	**100.0%** **98.5%** **6.62** 0.67
Vehicle * 846×18 4	**89.5%** 80.5 % 148.01	88.6% **82.2%** **1.17** 0.78
Segment ** 2310×19 7	**99.9 %** 96.1% 5562.31	98.3% **97.0%** 11.65 0.16

Table 2.6: Nonlinear OFRQP and Nonlinear BR-MPSVM training correctness, tenfold testing correctness and running times. Execution times include tenfold training.

* **For this nonlinear MPSVM classifier, RSVM [52] with an 85% kernel reduction was used here in order to obtain a smaller rectangular kernel problem that will fit in memory** (846×127 **instead of** 846×846**).**

** **For this nonlinear MPSVM classifier, RSVM [52] with an 85% kernel reduction was used here in order to obtain a smaller rectangular kernel problem that will fit in memory** (2310×350 **instead of** 2310×2310**).**

† **p-values calculated with respect to OFRQP for tenfold testing correctness, using a paired t-test with 95% confidence interval.**

Numerical Experiments Using Multicategory Nonlinear Classifiers

For the nonlinear case, we compared again nonlinear OFRQP and nonlinear PSVM and its modifications. In all experiments, a Gaussian kernel was used. In order to find an optimal value for ν and the Gaussian kernel parameter μ, a tuning procedure similar to that employed for the linear case was employed. Values for ν where taken equal to 2^i, where $i = 5, 6, \ldots, 35$. Values for μ where taken equal to 2^i, where $i = -7, -6, \ldots, 1$. Since the difference between the plain MPSVM and the modified MPSVM was not significant, Table 2.4.4 shows comparisons between the following methods only:

- **Nonlinear OFRQP**: One-From-Rest Quadratic Programming classifier using a standard nonlinear support vector machine for each subproblem which is solved by a MATLAB-CPLEX.
- **Nonlinear BR-MPSVM** : Balanced Refined Multicategory PSVM One-From-Rest classifier using a nonlinear PSVM including both modifications, balancing and Newton refinement.

On the larger datasets (Vehicle, Segment) a rectangular kernel [52] was used on both methods in order to reduce even more the computational time while maintaining the correctness achieved by using the full kernel.

The nonlinear BP-MPSVM classifier was obtained in shorter time than the nonlinear OFRQP classifier in all the datasets tested. Furthermore, the BR-MPSVM algorithm was statistically better or equal to the nonlinear OFRQP on test set correctness. CPU times and p-values are given in Table 2.4.4.

In order to show graphically for the nonlinear case that BP-MPSVM can produce significant improvement over MPSVM, we created an artificial 2-dimensional example

where this improvement can be visually observed. The example consists of 500 data points in 2 dimensions belonging to one of three classes. Class 1 consists of 400 points, class 2 consists of 50 points and class 3 consists of 50 points. Figure 2.11 depicts a non-linear classification obtained using MSPVM without any modifications using a Gaussian kernel. Since the classes are unbalanced, we observe that the majority of the x class is misclassified by the algorithm leading to 91.8% training set correctness. On the other hand, Figure 2.12 depicts a nonlinear classification obtained by BP-MPSVM that utilizes balancing and Newton refinement which gives a significantly improved 98.8% training set correctness.

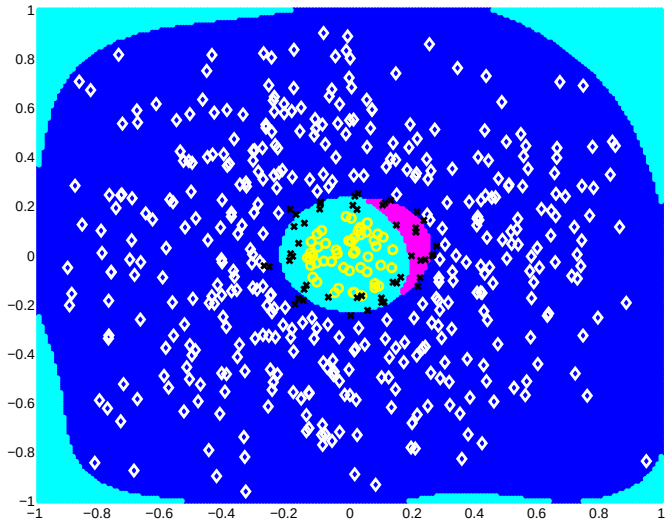

Figure 2.11: Example consisting of 500 data points in 2 dimensions belonging to one of three classes. Nonlinear Gaussian kernel classifiers using MSPVM without balancing or Newton refinement generated a torus containing mostly white diamonds, a crescent containing black x's, and an ellipse containing mostly yellow circles. Since the classes are unbalanced, most of the x class is misclassified by the algorithm and resulting in a 91.8% overall training set correctness.

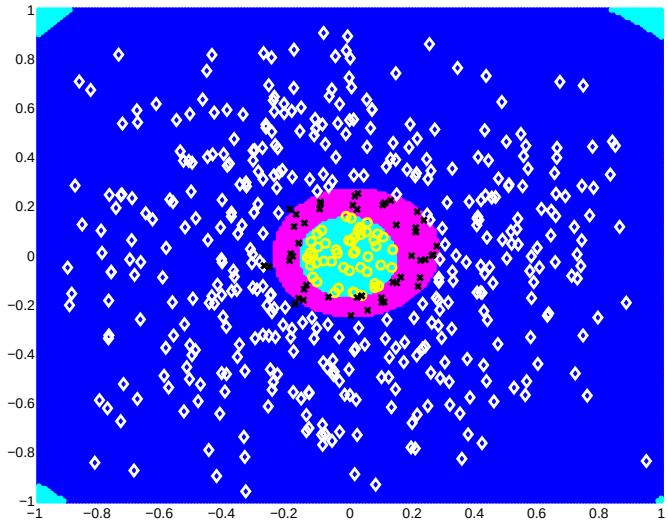

Figure 2.12: The same example as that of Figure 2.11 classified by a nonlinear BR-MPSVM which uses MSPVM plus balancing and Newton refinement. This resulted in a torus containing mostly white diamonds, another torus containing black x's and an ellipse containing mostly yellow circles. Overall training set correctness is 98.8%.

Chapter 3

Knowledge Based Support Vector Machines

3.1 The Linear Support Vector Machine and Prior Knowledge

We consider the problem, depicted in Figure 3.1, of classifying m points in the n-dimensional input space R^n, represented by the $m \times n$ matrix A, according to membership of each point A_i in the class $A+$ or $A-$ as specified by a given $m \times m$ diagonal matrix D with plus ones or minus ones along its diagonal. For this problem, the linear programming support vector machine [9, 65] with a linear kernel, which is a variant of the standard support vector machine (2.1) [19, 103], is given by the following linear program with parameter $\nu > 0$:

$$\begin{array}{rll} \min\limits_{(w,\gamma,y)\in R^{n+1+m}} & \nu e'y + \|w\|_1 & \\ \text{s.t.} & D(Aw - e\gamma) + y \geq e & \\ & y \geq 0, & \end{array} \tag{3.1}$$

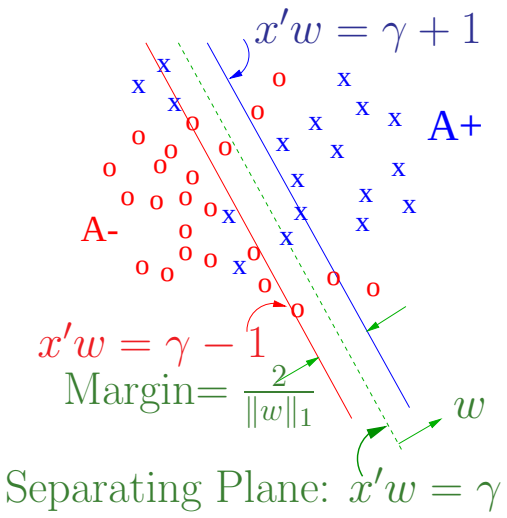

Figure 3.1: The linear programming support vector machine classifier in the w-space of R^n: The approximately bounding planes of equation (2.2) with a soft (i.e. with some error) margin $\frac{2}{\|w\|_1}$, and the plane of equation (2.4) approximately separating points in $A+$ from points in $A-$.

where $\|\cdot\|_1$ denotes the 1-norm as defined in the Introduction. That this problem is indeed a linear program, can be easily seen from the equivalent formulation:

$$\begin{array}{rll} \min\limits_{(w,\gamma,y,t)\in R^{n+1+m}} & \nu e'y + e't & \\ \text{s.t.} & D(Aw - e\gamma) + y \geq e & \\ & t \geq w \geq -t & \\ & y \geq 0. & \end{array} \tag{3.2}$$

For economy of notation we shall use the first formulation (3.1) with the understanding that computational implementation is via (3.2).

The 1-norm term $\|w\|_1$ in (3.1), which is half the reciprocal of the distance $\frac{2}{\|w\|_1}$ measured using the ∞-norm distance [63] between the two bounding planes of (2.2) (see Figure 3.1), maximizes the "margin". As was pointed out in chapter 2 maximizing the margin enhances the generalization capability of a support vector machine [19, 103].

The 1-norm of the error variable y is minimized parametrically with weight ν in (3.1),

resulting in an approximate separating plane (2.4) as depicted in Figure 3.1. This plane acts as a linear classifier as follows:

$$sign(x'w - \gamma\ ) \begin{cases} = 1, \ \text{ then } x \in A+, \\ \\ = -1, \ \text{ then } x \in A-, \end{cases} \tag{3.3}$$

where $sign(\cdot)$ is the sign function defined in the Introduction.

Suppose now that we have prior information of the following type. All points x lying in the polyhedral set determined by the linear inequalities:

$$Bx \leq b, \tag{3.4}$$

belong to class $A+$. Such inequalities generalize simple box constraints such as $a \leq x \leq d$. Looking at Figure 3.1 or at the inequalities (2.3) we conclude that the following implication must hold:

$$Bx \leq b \implies x'w \geq \gamma + 1. \tag{3.5}$$

That is, the *knowledge set* $\{x \mid Bx \leq b\}$ lies on the $A+$ side of the bounding plane $x'w = \gamma + 1$. This implication is equivalent to the following statement for a given (w, γ):

$$Bx \leq b, \ \ x'w < \gamma + 1, \ \text{ has no solution } x. \tag{3.6}$$

This statement in turn is *implied* by the following statement:

$$B'u + w = 0,\ b'u + \gamma + 1 \leq 0,\ u \geq 0, \text{ has a solution } (u, w). \tag{3.7}$$

To see this simple backward implication: (3.6)$\Longleftarrow$(3.7), we suppose the contrary that there exists an x satisfying (3.6) and obtain the contradiction $b'u > b'u$ as follows:

$$b'u \geq u'Bx = -w'x > -\gamma - 1 \geq b'u, \tag{3.8}$$

where the first inequality follows by premultiplying $Bx \leq b$ by $u \geq 0$. In fact, under the natural assumption that the prior knowledge set $\{x \mid Bx \leq b\}$ is nonempty, the forward implication: (3.6)$\Longrightarrow$(3.7) is also true, as a direct consequence of the nonhomogeneous Farkas theorem of the alternative [59, Theorem 2.4.8]. We state this equivalence as the following key proposition to our knowledge-based approach.

Proposition 3.1.1 Knowledge Set Classification. *Let the set* $\{x \mid Bx \leq b\}$ *be nonempty. Then for a given* (w, γ)*, the implication (3.5) is equivalent to the statement (3.7). In other words, the set* $\{x \mid Bx \leq b\}$ *lies in the halfspace* $\{x \mid w'x \geq \gamma + 1\}$ *if and only if there exists* u *such that* $B'u + w = 0,\ b'u + \gamma + 1 \leq 0$ *and* $u \geq 0$.

Proof We establish the equivalence of (3.5) and (3.7) by showing the equivalence (3.6) and (3.7). By the nonhomogeneous Farkas theorem [59, Theorem 2.4.8] we have that (3.6) is equivalent to either:

$$B'u + w = 0,\ b'u + \gamma + 1 \leq 0,\ u \geq 0, \text{ having solution } (u, w), \tag{3.9}$$

or

$$B'u = 0,\ b'u < 0,\ u \geq 0, \text{ having solution } u. \tag{3.10}$$

However, the second alternative (3.10) contradicts the nonemptiness of the knowledge-set $\{x \mid Bx \leq b\}$, because for $x \in \{x \mid Bx \leq b\}$ and u solving (3.10) we obtain the contradiction:

$$0 \geq u'(Bx - b) = x'B'u - b'u = -b'u > 0. \tag{3.11}$$

Hence (3.10) is ruled out and we have that (3.6) is equivalent to (3.9) which is (3.7). □

This proposition will play a key role in incorporating knowledge sets, such as $\{x \mid Bx \leq b\}$, into one of two categories in a support vector classifier formulation as demonstrated in the next section.

3.1.1 Knowledge-Based SVM Classification

We describe now how to incorporate prior knowledge in the form of polyhedral sets into our linear programming SVM classifier formulation (3.1).

We assume that we are given the following *knowledge sets*:

$$\begin{array}{l} k \text{ sets belonging to } A+ : \{x \mid B^i x \leq b^i\},\ i = 1, \ldots, k \\ \ell \text{ sets belonging to } A- : \{x \mid C^i x \leq c^i\},\ i = 1, \ldots, \ell \end{array} \tag{3.12}$$

By Proposition 3.1.1 this knowledge is equivalent to the following requirements with respect to the bounding planes (2.2):

$$\begin{array}{l} \text{There exist } u^i,\ i = 1, \ldots, k,\ v^j,\ j = 1, \ldots, \ell, \text{ such that:} \\ B^{i\prime} u^i + w = 0,\ b^{i\prime} u^i + \gamma + 1 \leq 0,\ u^i \geq 0,\ i = 1, \ldots, k \\ C^{j\prime} v^j - w = 0,\ c^{j\prime} v^j - \gamma + 1 \leq 0,\ v^j \geq 0,\ j = 1, \ldots, \ell \end{array} \tag{3.13}$$

All we need to do now in order to incorporate the knowledge sets (3.12) into the linear programming formulation (3.1) of an SVM classifier, is to add the conditions (3.13) as constraints to (3.1) as follows:

$$\begin{array}{rrcl} \min\limits_{w,\gamma,y,u^i,v^j} & \nu e'y + \|w\|_1 & & \\ \text{s.t.} & D(Aw - e\gamma) + y & \geq & e \\ & y & \geq & 0 \\ & B^{i\prime} u^i + w & = & 0 \\ & b^{i\prime} u^i + \gamma + 1 & \leq & 0 \\ & u^i & \geq & 0,\ i = 1, \ldots, k \\ & C^{j\prime} v^j - w & = & 0 \\ & c^{j\prime} v^j - \gamma + 1 & \leq & 0 \\ & v^j & \geq & 0,\ j = 1, \ldots, \ell \end{array} \tag{3.14}$$

This linear programming formulation will ensure that each of the knowledge sets $\{x \mid B^i x \leq b^i\},\ i = 1, \ldots, k$ and $\{x \mid C^i x \leq c^i\},\ i = 1, \ldots, \ell$ lie on the appropriate side of the bounding planes (2.2). However, there is no guarantee that such bounding planes exist that will precisely separate these two classes of knowledge sets, just as there is no *a priori* guarantee that the original points belonging to the sets $A+$ and $A-$ are linearly separable. We therefore add error variables $r^i, \rho^i, i = 1, \ldots, k, s^j, \sigma^j, j = 1, \ldots, \ell$, just like the error variable y of the SVM formulation (3.1), and attempt to drive these error variables to zero by modifying our last formulation above as follows:

$$\begin{array}{rrcl} \min\limits_{w,\gamma,y,u^i,r^i,\rho^i,v^j,s^j,\sigma^j} & \nu e'y + & & \\ & \mu(\sum\limits_{i=1}^{k}(r^i + \rho^i) & + & \sum\limits_{j=1}^{\ell}(s^j + \sigma^j)) + \|w\|_1 \\ \text{s.t.} & D(Aw - e\gamma) + y & \geq & e \\ & y & \geq & 0 \\ & -r^i \leq B^{i\prime}u^i + w & \leq & r^i \\ & b^{i\prime}u^i + \gamma + 1 & \leq & \rho^i \\ & u^i, r^i, \rho^i & \geq & 0,\ i = 1, \ldots, k \\ & -s^j \leq C^{j\prime}v^j - w & \leq & s^j \\ & c^{j\prime}v^j - \gamma + 1 & \leq & \sigma^j \\ & v^j, s^j, \sigma^j & \geq & 0,\ j = 1, \ldots, \ell \end{array} \tag{3.15}$$

This is our final knowledge-based linear programming formulation which incorporates the knowledge sets (3.12) into the linear classifier with weight μ, while the (empirical) error term $e'y$ is given weight ν. Note that for simplicity we chose the same weighting factor μ for all the slack variables associated with the knowledge sets. There is also possible to define a different weighting factor mu_i or mu_j for each prior knowledge set. As usual, the value of these two parameters, ν, μ, are chosen by means of a tuning set extracted from

the training set. If we set $\mu = 0$ then the linear program (3.15) degenerates to (3.1), the linear program associated with an ordinary linear SVM. However, if set $\nu = 0$, then the linear program (3.15) generates a linear SVM that is strictly based on knowledge sets, but not on any specific training data. This might be a useful paradigm for situations where training datasets are not easily available, but expert knowledge, such as doctors' experience in diagnosing certain diseases, is readily available. This will be demonstrated in the breast cancer dataset of subsection 3.1.2.

Note that the 1-norm term $\|w\|_1$ can be replaced by one half the 2-norm squared, $\frac{1}{2}\|w\|_2^2$, which is the usual margin maximization term for ordinary support vector machine classifiers [19, 103]. However, this changes the linear program (3.15) to a quadratic program which typically takes much longer to solve.

For completeness we state our knowledge-based algorithm as follows.

Algorithm 3.1.2 Knowledge-Based Linear SVM (KSVM). *Given m data points in R^n represented by the $m \times n$ matrix A and a diagonal matrix D of ± 1 labels denoting the class of each row of A, and given the knowledge sets (3.12), we generate the linear classifier (3.3) as follows:*

(i) Compute $\begin{bmatrix} w \\ \gamma \end{bmatrix}$ by solving the linear program (3.15) for some positive values of ν and μ. Typically ν and μ are chosen by means of a tuning (validating) set.

(ii) Classify a new x by using (3.3) and the solution $\begin{bmatrix} w \\ \gamma \end{bmatrix}$ from Step (i) above.

For standard SVMs, support vectors consist of all data points which are the complement of the data points that can be dropped from the problem without changing the separating plane (2.4) [65, 103]. Thus for our knowledge-based linear programming formulation (3.15), support vectors correspond to data points (rows of the matrix A) for which the

Lagrange multipliers are nonzero, because solving (3.15) with these data points only will give the same answer as solving (3.15) with the entire matrix A.

The concept of support vectors has to be modified as follows for our knowledge sets. Since each knowledge set in (3.12) is represented by a matrix B^i or C^j, each row of these matrices can be thought of as characterizing a plane boundary of the knowledge set. In our formulation (3.15) above, such rows are wiped out if the corresponding component of the variables u^i or v^j are zero at an optimal solution. We call the complement of these components of the the knowledge sets (3.12) *support constraints*. Deleting constraints (rows of B^i or C^j), for which the corresponding components of u^i or v^j are zero, will not alter the solution of the knowledge-based linear program (3.15). This in fact is corroborated by numerical tests that were carried out.

We demonstrate the geometry of incorporating knowledge sets into a classification problem by considering a synthetic example in R^2 with $m = 200$ points, 100 of which are in $A+$ and the other 100 in $A-$. Figure 3.2 depicts ordinary linear separation using the linear SVM formulation (3.1). We now incorporate three knowledge sets into the the problem: $\{x \mid B^1x \leq b^1\}$ belonging to $A+$ and $\{x \mid C^1x \leq c^1\}$ and $\{x \mid C^2x \leq c^2\}$ belonging to $A-$, and solve our linear program (3.15). We depict the new linear separation in Figure 3.3 and note the substantial change generated in the linear separation by the incorporation of these three knowledge sets.

3.1.2 Numerical Testing for the Linear Case

Our numerical testing was performed on two publicly available datasets [77]. A description of the datasets and the experiments performed is presented next.

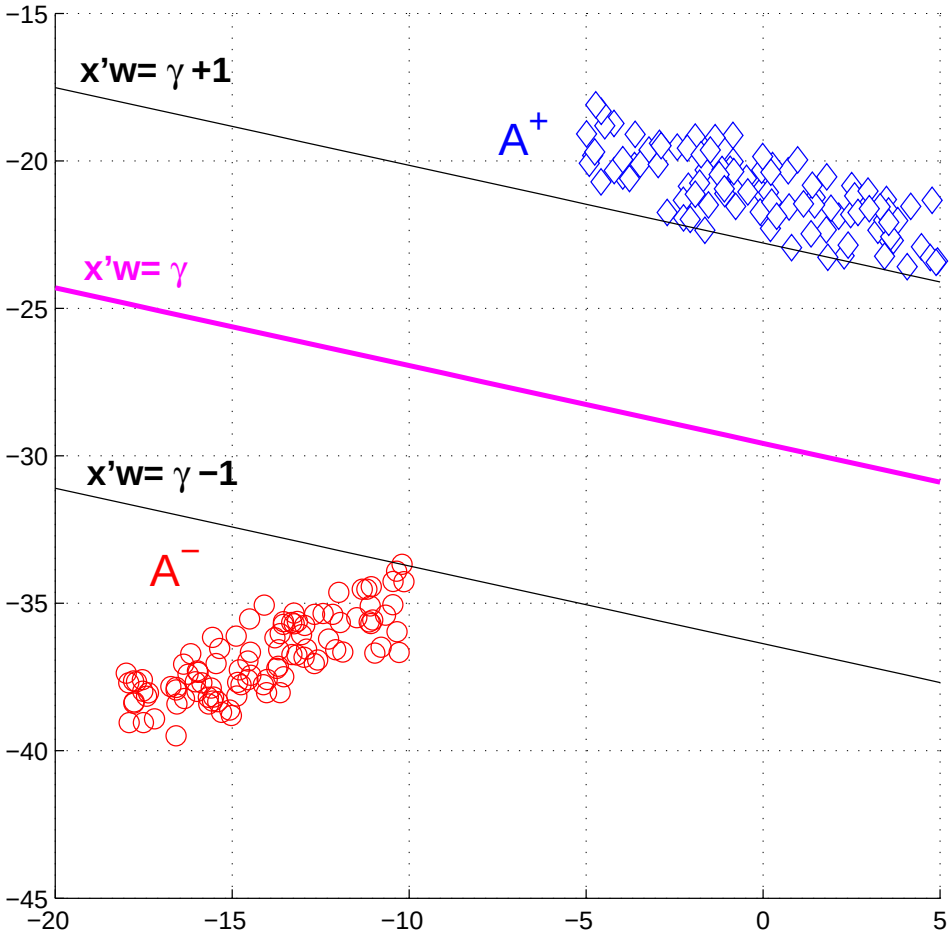

Figure 3.2: A linear SVM separation for 200 points in R^2 using the linear programming formulation (3.1).

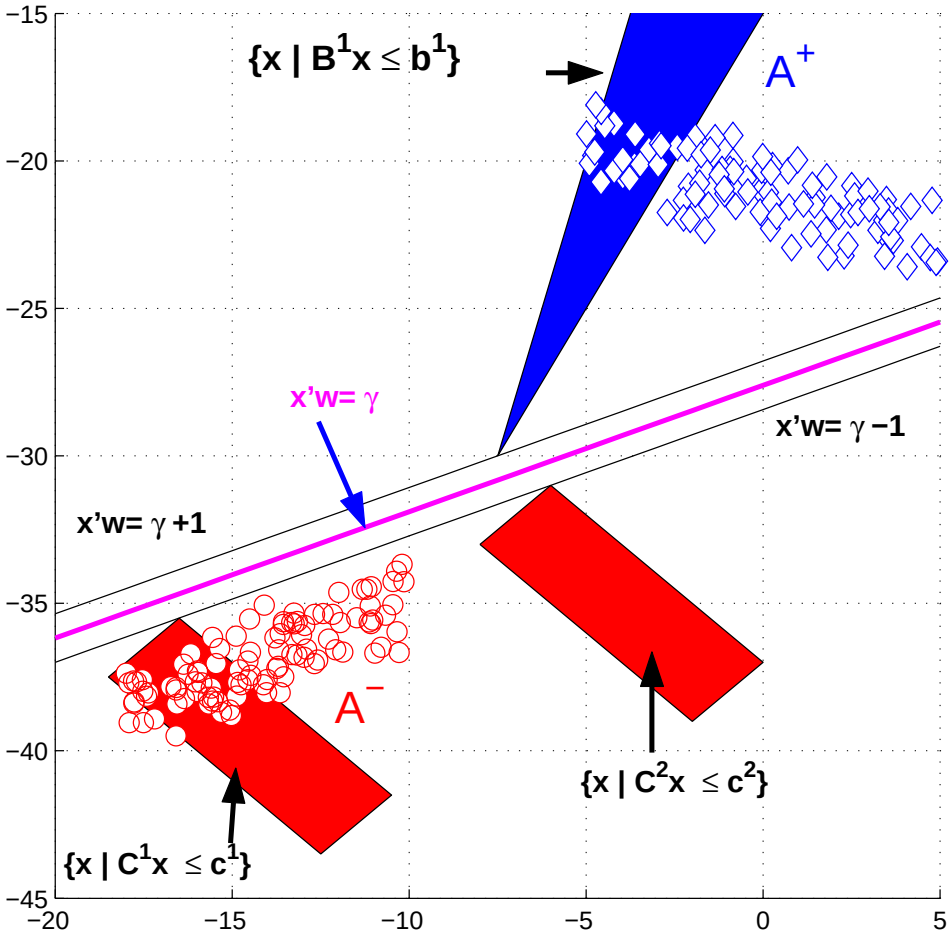

Figure 3.3: A linear SVM separation for the same 200 points in R^2 as those in Figure 3.2 but using the linear programming formulation (3.15) which incorporates three knowledge sets: $\{x \mid B^1x \leq b^1\}$ into the halfspace of $A+$, and $\{x \mid C^1x \leq c^1\}$, $\{x \mid C^2x \leq c^2\}$ into the halfspace of $A-$, as depicted in the figure above. Note the substantial difference between the linear classifiers $x'w = \gamma$ of Figures 3.2 and 3.3.

Promoter Recognition Dataset

The first dataset [100] is from the domain of DNA sequence analysis and is called the promoter recognition dataset. A promoter is a short DNA sequence that precedes a gene sequence. Hence, it is important distinguish between *promoter* and *nonpromoter* sequences, since this will make it possible to identify the starting locations of genes in long, uncharacterized sequences of DNA.

Description of the data

For classification purposes of this experiment it is sufficient to consider the DNA as a linear sequence of characters belonging to a four elements set $\{A, G, C, T\}$. These four elements are commonly referred to nucleotides.

Each promoter consists of a sequence of 57 consecutive DNA nucleotides. The locations are numbered with respect to a fixed meaningful, reference point. Starting at position -50 (p_{-50}) and ending at position +7 (p_7). Negative numbers indicate sites preceding the reference point while positive numbers indicate sites following the reference point. Since each one of the 57 feature takes nominal values from the set $\{A, G, C, T\}$, a real valued representation is needed to apply our KSVM formulation. Each nominal value is mapped into a four dimensional binary vector depending on the nucleotide that is being represented. This simple and widely used "1 of N" mapping scheme for converting nominal attributes into real-valued attributes is illustrated in Figure 3.4.

Once this simple conversion is applied to the dataset, the feature space is transformed from a 57-dimensional space with nominal values into a $57 \times 4 = 228$ real-valued dimensional space. So, each data point is now a point in R^{228}. See Figure 3.5. The training

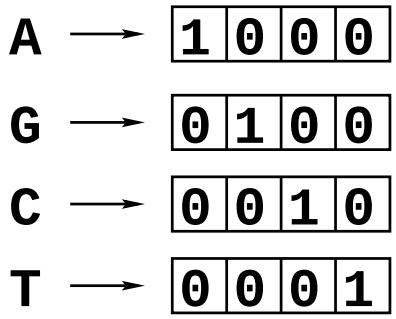

Figure 3.4: Real-valued representation of the nucleotide set $\{A, G, C, T\}$.

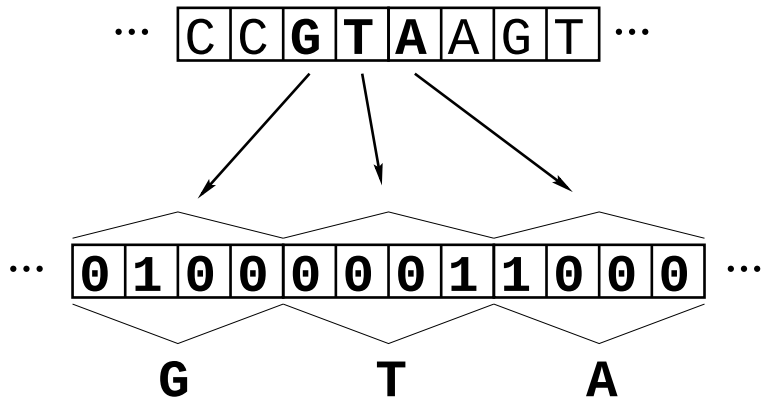

Figure 3.5: Real-valued representation of a promoter, from a 57 nominal-valued vector to a $57 \times 4 = 228$ real-valued vector.

example consist of 53 sample promoters and 53 nonpromoter sequences for a total of 106 data points.

The Prior Knowledge

The prior knowledge for the promoters dataset is also publicly available [77] using a prolog notation. However in order to make these rules accessible to a wider audience we will use standard logic notation to describe them below. There are two facts that will imply that the sequence is very likely to be a promoter: If the sequence has a region where a certain protein (RNA polymerase) make $CONTACT$ and the helical DNA sequence has a valid conformation ($CONF$) so the two pieces of the contact region spatially align. This rule can be written as:

$$CONTACT \wedge CONF \Longrightarrow PROMOTER \tag{3.16}$$

There are two regions named $M35$ and $M10$ in the original representation of the sequences in which the RNA polymerase makes $CONTACT$, this means:

$$M35 \wedge M10 \Longrightarrow CONTACT \tag{3.17}$$

The following 4 rules R_1, R_2, R_3, R_4 imply possible contact in the region $M35$:

$$R_1: \quad \begin{array}{l} (p_{-37} = C) \wedge (p_{-36} = T) \wedge (p_{-35} = T) \wedge \\ (p_{-34} = G) \wedge (p_{-33} = A) \wedge (p_{-32} = C) \end{array} \quad \Longrightarrow \quad M35$$

$$R_2: \begin{array}{l} (p_{-36} = T) \wedge (p_{-35} = T) \wedge (p_{-34} = G) \wedge \\ (p_{-32} = C) \wedge (p_{-31} = A) \end{array} \implies M35$$

$$R_3: \begin{array}{l} (p_{-36} = T) \wedge (p_{-35} = T) \wedge (p_{-34} = G) \wedge \\ (p_{-33} = A) \wedge (p_{-32} = C) \wedge (p_{-31} = A) \end{array} \implies M35$$

$$R_4: \begin{array}{l} (p_{-36} = T) \wedge (p_{-35} = T) \wedge (p_{-34} = G) \wedge \\ (p_{-33} = A) \wedge (p_{-32} = C) \end{array} \implies M35$$

The following 4 rules R_5, R_6, R_7, R_8 imply possible contact in the region $M10$:

$$R_5: \begin{array}{l} (p_{-14} = T) \wedge (p_{-13} = A) \wedge (p_{-12} = T) \wedge \\ (p_{-11} = A) \wedge (p_{-10} = A) \wedge (p_{-09} = T) \end{array} \implies M10$$

$$R_6: \begin{array}{l} (p_{-13} = T) \wedge (p_{-12} = A) \wedge (p_{-10} = A) \wedge \\ (p_{-08} = T) \end{array} \implies M10$$

$$R_7: \begin{array}{l} (p_{-13} = T) \wedge (p_{-12} = A) \wedge (p_{-11} = T) \wedge \\ (p_{-10} = A) \wedge (p_{-09} = A) \wedge (p_{-08} = T) \end{array} \implies M10$$

$$R_8: \ (p_{-12} = T) \wedge (p_{-11} = A) \wedge (p_{-07} = T) \implies M10$$

The following 4 rules $R_9, R_{10}, R_{11}, R_{12}$ imply characteristics that produce acceptable conformations:

$$
\begin{aligned}
R_9: \quad & (p_{-47} = C) \wedge (p_{-46} = A) \wedge (p_{-45} = A) \wedge \\
& (p_{-43} = T) \wedge (p_{-42} = T) \wedge (p_{-40} = A) \wedge \\
& (p_{-39} = C) \wedge (p_{-22} = G) \wedge (p_{-18} = T) \wedge \\
& (p_{-16} = C) \wedge (p_{-08} = G) \wedge (p_{-07} = C) \wedge \\
& (p_{-06} = G) \wedge (p_{-05} = C) \wedge (p_{-04} = C) \wedge \\
& (p_{-02} = C) \wedge (p_{-01} = C) \qquad \Longrightarrow \quad CONF
\end{aligned}
$$

$$
R_{10}: \quad (p_{-45} = A) \wedge (p_{-44} = A) \wedge (p_{-41} = A) \quad \Longrightarrow \quad CONF
$$

$$
\begin{aligned}
R_{11}: \quad & (p_{-49} = A) \wedge (p_{-44} = T) \wedge (p_{-27} = T) \wedge \\
& (p_{-22} = T) \wedge (p_{-18} = T) \wedge (p_{-16} = A) \wedge \\
& (p_{-15} = C) \wedge (p_{-01} = C) \qquad \Longrightarrow \quad CONF
\end{aligned}
$$

$$
\begin{aligned}
R_{12}: \quad & (p_{-45} = A) \wedge (p_{-41} = A) \wedge (p_{-28} = T) \wedge \\
& (p_{-27} = T) \wedge (p_{-23} = T) \wedge (p_{-21} = A) \wedge \\
& (p_{-20} = A) \wedge (p_{-17} = T) \wedge (p_{-15} = T) \wedge \\
& (p_{-04} = T) \qquad \Longrightarrow \quad CONF
\end{aligned}
$$

It is important to note that this prior knowledge matches none of the given examples of the training set. Hence, this set of rules is useless as a classifier by itself. However, they do capture a significant amount of information about promoters and it has been shown that incorporating them into a classifier results in a better and more accurate classifier using knowledge-based neural networks [100]. We will demonstrate a similar result using our simpler linear support vector machine linear classifier.

Conversion of the rules to sets of the form $Bx \leq b$

To convert the above rules $R_1, \ldots, R_{12}$ to a set of matrix inequalities we group them as follows:

$$\begin{array}{ccccc} M35: & & M10: & & CONF: \\ \begin{bmatrix} R_1 \\ or \\ R_2 \\ or \\ R_3 \\ or \\ R_4 \end{bmatrix} & \wedge & \begin{bmatrix} R_5 \\ or \\ R_6 \\ or \\ R_7 \\ or \\ R_8 \end{bmatrix} & \wedge & \begin{bmatrix} R_9 \\ or \\ R_{10} \\ or \\ R_{11} \\ or \\ R_{12} \end{bmatrix} \end{array} \quad \Longrightarrow \quad PROMOTER$$

$$4 \quad \times \quad 4 \quad \times \quad 4 \quad = \quad 64 \text{ Rules.}$$

This results in 64 rules which imply the occurrence of a promoter. Each of these 64 rules can be represented as a constraint set $\{x|B^i x \leq b^i\}$ for $i = 1, \ldots, 64$, which define part of the promoter region in R^{228}.

Numerical Comparisons

The performance in terms of generalization ability of our proposed algorithm, is compared to six other learning algorithms using the same promoter dataset. These algorithms are:

1. **KBANN: Knowledge Based Artificial Neural Networks**

   A hybrid learning system that maps problem specific prior knowledge, represented in propositional logic into neural networks and then, refines this reformulated knowledge using back propagation [100].

2. **BP: Standard Back Propagation**

   Neural networks with a simple connected layer of hidden units [90].

3. **O'Neill's Method**

   An empirical method suggested by a biologist based in a collection of "filters" to be used for promoter recognition [80].

4. **NN: Nearest Neighbor Algorithm**

   PEBLS Nearest algorithm [20] with $k = 3$.

5. **ID3: Decision Tree**

   Quinlan's decision tree builder [86].

6. **$\mathbf{SVM_1}$: 1-norm SVM**

   Standard 1-norm support vector machine [9].

Following the methodology used in prior work [100], we tested our algorithm on this dataset using a "leave-one-out" cross validation methodology in which the entire training set of 106 elements is repeatedly divided into a training set of size 105 and a test set of size 1, in all 106 possible ways. The values of ν and μ associated with both KSVM and SVM_1 where obtained by a tuning procedure. The tuning procedure consists of removing a subset of the training data, solving (3.15) for various various values of ν and μ and choosing values of these parameters based on best accuracy on the removed "tuning" set.

The number of times that a test element is misclassified for each method is counted as an error and reported in Table 3.1.2.

Method	Number of Errors (out of 106)
KBANN*	4
KSVM	5
BP*	8
SVM_1	9
O'Neill*	12
NN*	13
ID3*	19

Table 3.1: Comparison of KSVM leave-one-out total error with various classification algorithms.

* Results reported in [100].

Note that our proposed KSVM is second best among all tested methods with 5 misclassification while KBANN is best with 4 misclassification even though our classifier is a simple linear classifier, $sign(x'w-\gamma)$, while a neural network classifier is a more complex nonlinear classifier. Furthermore, we note that KSVM is relatively simpler to implement than KBANN and requires merely a commonly available linear programming solver. In addition, KSVM which is a linear support vector machine classifier, improves by 44.4% the error of an ordinary linear 1-norm SVM classifier that does not use knowledge sets.

Wisconsin Breast Cancer Prognosis Dataset

Description of the Data

The second dataset used in our numerical tests was the Wisconsin breast cancer prognosis dataset. Each data point consists of 30 nuclear features plus the diameter of the

excised tumor and the number of metastasized lymph nodes. There are 110 instances corresponding to 41 patients whose cancer had recurred (RECUR) in less than 60 months and 69 patients whose cancer had not recurred (NONRECUR) in less than 60 months. We tested our algorithm on this dataset using a ten-fold cross validation methodology in which the entire training set of 110 elements is divided ten times into a training set of size 99 and a test set of size 11. The testing correctness reported is an average taken over the testing correctness obtained in each fold. The parameters ν and μ associated with both KSVM and SVM_1 where obtained by a tuning procedure similar to that of Section 3.1.2.

The Prior Knowledge

The prior knowledge we used in this experiment are rules used by doctors [55]. These rules are related to two of the features of the dataset: tumor size (feature 31), that is the diameter of the excised tumor in centimeters and lymph node status which refers to the number of metastasized axillary lymph nodes (feature 32). The rules are:

$RECUR$ Rule:

$$(LYMPH \geq 5) \wedge (TUMOR \geq 4) \implies RECUR$$

$NONRECUR$ rule:

$$(LYMPH = 0) \wedge (TUMOR \leq 1.9) \implies NONRECUR$$

Experimental Results

It is important to note that the rules described above can be applied directly to classify only 32 of the given 110 given points of the training dataset and correctly classify 22 of these 32 points. The remaining 78 points are not classifiable by the above rules. Hence, if the rules are applied as a classifier by themselves the classification accuracy would be 20%. As such, these rules are not very useful by themselves and doctors use them in conjunction with other rules [55]. However, using our approach the rules were converted to linear inequalities and used in our KSVM algorithm without any use of the data, i.e. $\nu = 0$ in the linear program (3.15). The resulting linear classifier achieved 66.4% accuracy. Similar results were obtained using both the data and the rules in our formulation. The correctness achieved by standard SVM using all the data is 66.2% [9]. This result is remarkable because our knowledge-based formulation can be applied to problems where training data may not be available whereas expert knowledge may be readily available in the form of knowledge sets. This fact makes this method considerably different from previous hybrid methods like KBANN where training examples are needed in order to refine prior knowledge.

3.2 Knowledge-Based Nonlinear Kernel Classifiers

In this section we extend the previous work shown in the previous section of incorporating multiple polyhedral sets as prior knowledge for a linear classifier to nonlinear kernel-based classifiers. This extension is not an obvious one, since it depends critically on the theory of *linear* inequalities and cannot be incorporated directly into a nonlinear kernel classifier. However, if the "kernel trick" is employed *after* one uses a theorem of the alternative for linear inequalities [59, Chapter 2], then incorporation of polyhedral knowledge sets into a nonlinear kernel classifier can be achieved. We note that conventional datasets are not essential to our formulation and can be surrogated by samples taken from the knowledge sets. We tested our formulation on standard type test problems. The first test problem is the exclusive-or (XOR) problem consisting of four points in 2-dimensional input space plus four polyhedral knowledge sets, all of which get classified perfectly by a Gaussian kernel knowledge-based classifier. The second test problem is the checkerboard problem consisting of 16 two-colored squares. Typically this problem is classified based on 1000 points. Here, by using only 16 points plus prior knowledge, our knowledge-based nonlinear kernel classifier, generates a sharp classifier that is as good as that obtained by using 1000 points.

3.2.1 Prior Knowledge in a Nonlinear Kernel Classifier

We turn now to the incorporation of prior knowledge in the form of a polyhedral set into a nonlinear kernel classifier. But first, we show that a routine incorporation of such knowledge leads to a nonlinear system of nonconvex inequalities that are not very useful.

Suppose that the polyhedral $\{x \mid Bx \leq b\}$ where $B \in R^{\ell \times n}$ and $b \in R^{\ell}$, must lie in the halfspace $\{x \mid x'w \geq \gamma + 1\}$ for some given $w \in R^n$ and $\gamma \in R$. We thus have the

implication:

$$Bx \leq b \implies x'w \geq \gamma + 1. \tag{3.18}$$

By letting w take on its dual representation $w = A'Du$ [53, 65], the implication (3.18) becomes:

$$Bx \leq b \implies x'A'Du \geq \gamma + 1. \tag{3.19}$$

If we now "kernelize" this implication by letting $x'A' \longrightarrow K(x', A')$, where K is some nonlinear kernel as defined in the Introduction, we then have the implication, for a given A, D, u and γ, that:

$$Bx \leq b \implies K(x', A')Du \geq \gamma + 1. \tag{3.20}$$

This is equivalent to the following nonlinear, and generally nonconvex, system of inequalities *not* having a solution x for a given A, D, u and γ:

$$Bx \leq b,\ K(x', A')Du < \gamma + 1. \tag{3.21}$$

Unfortunately, the nonlinearity and nonconvexity of the system (3.21) precludes the use of any theorem of the alternative for either linear or convex inequalities [59]. We thus have to backtrack to the implication (3.19) and rewrite it equivalently as the following system of homogeneous linear inequalities not having a solution $(x, \zeta) \in R^{n+1}$ for a given fixed u and γ:

$$\begin{array}{rrl} Bx & -b\zeta \leq & 0, \\ u'DAx & -(\gamma+1)\zeta < & 0, \\ & -\zeta < & 0. \end{array} \tag{3.22}$$

Here, the positive variable ζ is introduced in order to make the inequalities (3.22) homogeneous in (x, ζ), thus enabling us to use a desired theorem of the alternative [59] for such linear inequalities. It follows by Motzkin's Theorem of the Alternative [59], that (3.22) is equivalent to the following system of linear inequalities having a solution in $(v, \eta, \tau) \in R^{\ell+1+1}$ for a given fixed u and γ:

$$\begin{array}{rrrcl} B'v & +(A'Du)\eta & & = & 0, \\ -b'v & -(\gamma+1)\eta & -\tau = & & 0, \\ v & & \geq & & 0, \\ & & 0 \neq & (\eta, \tau) \geq & 0. \end{array} \tag{3.23}$$

If $\eta = 0$, then $\tau > 0$. Dividing by τ and redefining v as $\frac{v}{\tau}$, it follows from (3.23) that there exists a v such that: $B'v = 0,\ -b'v > 0,\ v \geq 0$, which contradicts the natural assumption that the knowledge set $\{x \mid Bx \leq b\}$ is nonempty. Otherwise, we have the contradiction:

$$0 = v'Bx \leq b'v < 0. \tag{3.24}$$

Hence $\eta > 0$ and $\tau \geq 0$. Dividing the inequalities of (3.23) by η and redefining v as $\frac{v}{\eta}$, we have from (3.23) that the following system of linear equalities has a solution v for a given u and γ:

$$\begin{array}{rrcl} B'v & +A'Du & = & 0, \\ b'v & +\gamma+1 & \leq & 0, \\ v & & \geq & 0. \end{array} \tag{3.25}$$

Under the rather natural assumption that A has linearly independent columns, this in turn is equivalent to following system of linear equalities having a solution v for a given

u and γ:

$$\begin{array}{rcl} AB'v \quad +AA'Du & = & 0, \\ b'v \quad +\gamma+1 & \leq & 0, \\ v & \geq & 0. \end{array} \tag{3.26}$$

Note that the linear independence is needed only for (3.26) to imply (3.25). Replacing the the linear kernels AB' and AA' by the general nonlinear kernels $K(A, B')$ and $K(A, A')$, we obtain that the following system of linear equalities has a solution v for a given u and γ:

$$\begin{array}{rcl} K(A,B')v \quad +K(A,A')Du & = & 0, \\ b'v \quad +\gamma+1 & \leq & 0, \\ v & \geq & 0. \end{array} \tag{3.27}$$

This is the set of constraints that we shall impose on our nonlinear classification formulation as a surrogate for the implication (3.20). Since the derivation of the conditions were not directly obtained from (3.20), it is useful to state precisely what the conditions (3.27) are equivalent to. By using a similar reasoning that employs theorems of the alternative as we did above, we can derive the following equivalence result which we state without giving its explicit proof. The proof is very similar to the arguments used above.

Proposition 3.2.1 Knowledge Set Classification *Let*

$$\{y \mid K(B, A')y \leq b\} \neq \emptyset. \tag{3.28}$$

Then the system (3.27) having a solution v, for a given u and γ, is equivalent to the implication:

$$K(B, A')y \leq b \implies u'DK(A, A')y \geq \gamma + 1. \tag{3.29}$$

We note that the implication is not precisely the implication (3.20) that we started with, but can be thought of as a kernelized version of it. To see this we state a corollary to the above proposition which shows what the implication means for a linear kernel AA'.

Corollary 3.2.2 Linear Knowledge Set Classification

Let

$$\{y \mid Bx \leq b,\ x = A'y\} \neq \emptyset. \tag{3.30}$$

For a linear kernel $K(A, A') = AA'$, the system (3.27) having a solution v, for a given u and γ, is equivalent to the implication:

$$Bx \leq b,\ x = A'y \implies w'x \geq \gamma + 1,\ w = A'Du,\ x = A'y. \tag{3.31}$$

We immediately note that the implication (3.31) is equivalent to the desired implication (3.18) for linear knowledge sets, under the rather unrestrictive assumption that A has linearly independent columns. That the columns of A are linearly independent is equivalent to assuming that in the input space, features are not linearly dependent on each other. If they were, then linearly dependent features could be easily removed from the problem.

We turn now to a linear programming formulation of a nonlinear kernel classifier that incorporates prior knowledge in the form of multiple polyhedral sets.

3.2.2 Knowledge-Based Linear Programming Formulation of Nonlinear Kernel Classifiers

A standard [9, 65] linear programming formulation of a nonlinear kernel classifier is given by:

$$\begin{array}{rll} \min\limits_{u,\gamma,r,y} & \nu e'y + e'r & \\ \text{s.t.} & D(K(A,A')Du - e\gamma) + y \;\geq\; e, & \\ & -r \leq u \;\leq\; r, & \\ & y \;\geq\; 0. & \end{array} \tag{3.32}$$

The (u, γ) taken from a solution (u, γ, r, y) of (3.32) generates the nonlinear separating surface $K(x', A')Du = \gamma$. Suppose now that we are given the following *knowledge sets*:

$$\begin{array}{l} p \text{ sets belonging to } A+ : \{x \mid B^i x \leq b^i\},\ i = 1, \ldots, p, \\ q \text{ sets belonging to } A- : \{x \mid C^i x \leq c^i\},\ i = 1, \ldots, q. \end{array} \tag{3.33}$$

It follows from the implication (3.20) for $B = B^i$ and $b = b^i$ for $i = 1, \ldots, p$ and its consequence, the existence of a solution to (3.27), and a similar implication for the sets $\{x \mid C^i x \leq c^i\},\ i = 1, \ldots, q$, that the following holds:

$$\begin{array}{l} \text{There exist } s^i,\ i = 1, \ldots, p,\ t^j,\ j = 1, \ldots, q, \text{ such that:} \\ K(A, B^{i\prime})s^i + K(A, A')Du = 0,\ b^{i\prime}s^i + \gamma + 1 \leq 0,\ s^i \geq 0,\ i = 1, \ldots, p, \\ K(A, C^{j\prime})t^j - K(A, A')Du = 0,\ c^{j\prime}t^j - \gamma + 1 \leq 0,\ t^j \geq 0,\ j = 1, \ldots, q. \end{array} \tag{3.34}$$

We now incorporate the knowledge sets (3.33) into the nonlinear kernel classifier linear program (3.32) by adding conditions (3.34) as constraints to (3.32) as follows:

$$\begin{array}{rrcl} \min\limits_{u,\gamma,r,(y,s^i,t^j)\geq 0} & \nu e'y + e'r & & \\ \text{s.t.} & D(K(A,A')Du - e\gamma) + y & \geq & e, \\ & -r \leq u & \leq & r, \\ & K(A,B^{i'})s^i + K(A,A')Du & = & 0, \\ & b^{i'}s^i + \gamma + 1 & \leq & 0,\ i = 1,\ldots,p, \\ & K(A,C^{j'})t^j - K(A,A')Du & = & 0, \\ & c^{j'}t^j - \gamma + 1 & \leq & 0,\ j = 1,\ldots,q. \end{array} \tag{3.35}$$

This linear programming formulation incorporates the knowledge sets (3.33) into the appropriate halfspaces in the higher dimensional feature space. However since there is no guarantee that we are able to place each knowledge set in the appropriate halfspace, we need to introduce error variables $z_1^i, \zeta_1^i, i = 1,\ldots,p, z_2^j, \zeta_2^j, j = 1,\ldots,q$, just like the error variable y of the SVM formulation (3.32), and attempt to drive these error variables to zero by modifying our last formulation above as follows:

$$\begin{array}{rrcl} \min\limits_{u,\gamma,r,z_1^i,z_2^j,(y,s^i,t^j,\zeta_1^i,\zeta_2^j)\geq 0} & \nu e'y + e'r + \mu(\sum\limits_{i=1}^{p}(e'z_1^i + \zeta_1^i) & & +\sum\limits_{j=1}^{q}(e'z_2^j + \zeta^j)) \\ \text{s.t.} & D(K(A,A')Du - e\gamma) + y & \geq & e, \\ & -r \leq u & \leq & r, \\ & -z_1^i \leq K(A,B^{i'})s^i + K(A,A')Du & \leq & z_1^i, \\ & b^{i'}s^i + \gamma + 1 & \leq & \zeta_1^i,\ i = 1,\ldots,p, \\ & -z_2^j \leq K(A,C^{j'})t^j - K(A,A')Du & \leq & z_2^j, \\ & c^{j'}t^j - \gamma + 1 & \leq & \zeta_2^j,\ j = 1,\ldots,q. \end{array} \tag{3.36}$$

This is our final knowledge-based linear programming formulation which incorporates the knowledge sets (3.33) into the linear classifier with weight μ, while the (empirical)

error term $e'y$ is given weight ν. As usual, the value of these two parameters, ν, μ, are chosen by means of a tuning set extracted from the training set.

Remark 3.2.3 Data-Based and Knowledge-Based Classifiers *If we set $\mu = 0$, then the linear program (3.36) degenerates to (3.32), the linear program associated with an ordinary data-based nonlinear kernel SVM. We can also make the linear program (3.36), which generates a nonlinear classifier, to be* **only knowledge-based** *and not dependent on any specific training data* **if** *we replace the matrix A appearing everywhere in (3.36) by a random sample of points taken from the knowledge sets (3.33) together with the associated diagonal matrix D. This might be a useful paradigm for situations where training datasets are not easily available, but expert knowledge, such as doctors' experience in diagnosing certain diseases, is readily available. In fact, using this idea of making A and D random samples drawn from the knowledge sets (3.33), the linear programming formulation (3.36)* **as is** *can be made* **totally** *dependent on prior knowledge only.*

We turn now to our numerical experiments.

3.2.3 Numerical Experience

The focus of this section is rather theoretical. However, in order to illustrate the power of the proposed formulation, we tested our algorithm on two synthetic examples for which most or all the data is constituted of knowledge sets. Experiments involving real world knowledge sets will be utilized in future work.

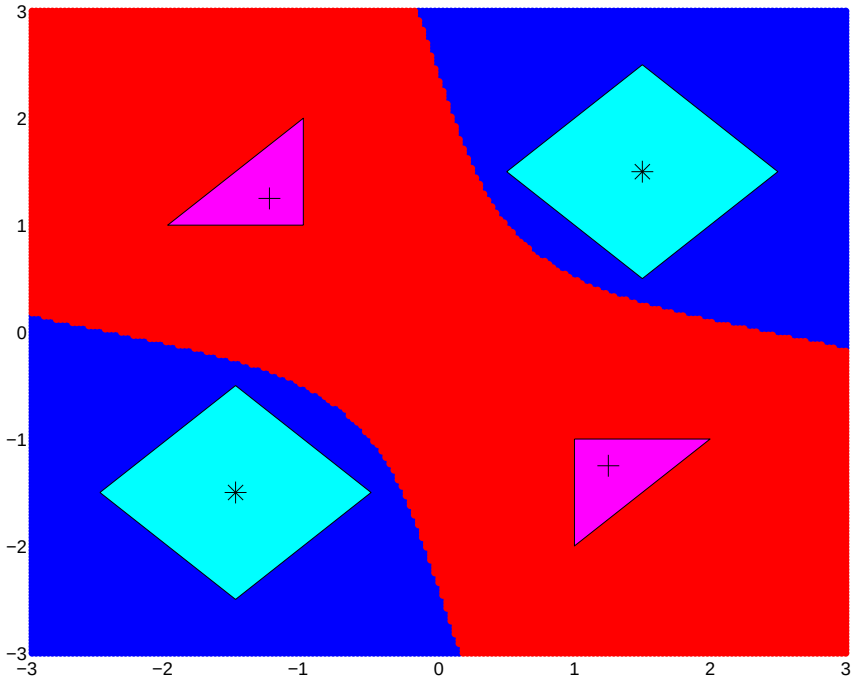

Figure 3.6: Totally or partially knowledge-based XOR classification problem. The nonlinear classifier obtained by using a Gaussian kernel in our linear programming formulation (3.36), completely separates the two pairs of prior knowledge sets as well the two pairs of points. The points can be treated either as samples taken from the knowledge sets or given independently of them.

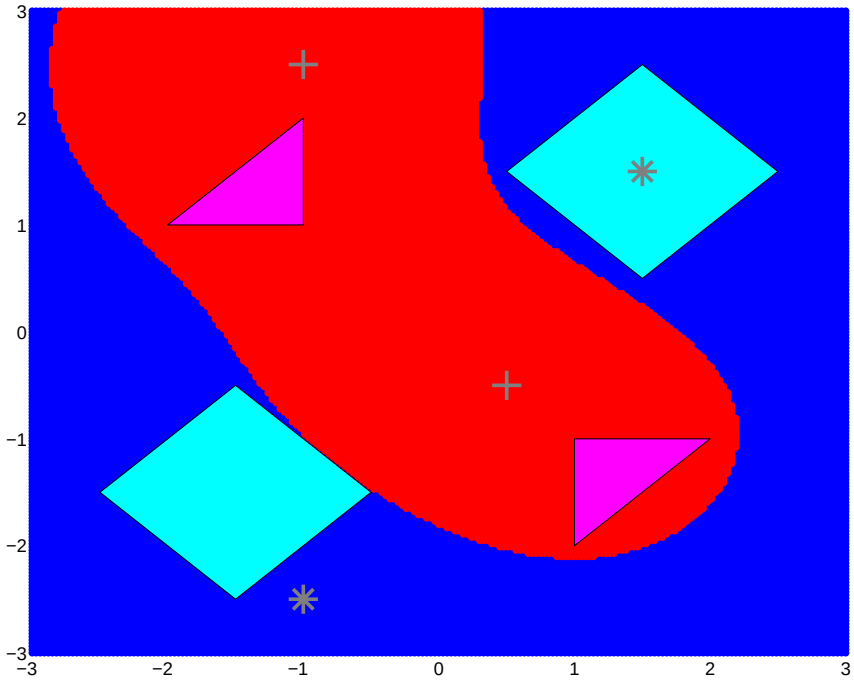

Figure 3.7: Another XOR classification problem where only one of the points is contained in a knowledge set. The nonlinear classifier obtained by using a Gaussian kernel in our linear programming formulation (3.36), completely separates the two pairs of prior knowledge sets as well the two pairs of points. Note the the resulting separating surface is strongly influenced by the knowledge sets in that the separating surface is constrained by a knowledge set, and in fact it is tangent to it.

Exclusive-Or (XOR) Knowledge Sets

This example generalizes the well known XOR example which consists of the four vertices of a rectangle in 2-dimensions, with the pair of vertices on the end of one diagonal belonging to one class (crosses) while the other pair belongs to another class (stars). Figure 3.2.2 depicts two such pairs of vertices symmetrically placed around the origin. It also depicts two pairs of knowledge sets with each pair belonging to one class (two triangles and two parallelograms respectively).

The given points in this XOR example can be considered in two different ways. We note that in line with Remark 3.2.3, this classifier can be considered either partially or totally dependent on prior knowledge, depending on whether the four data points are given independently of the knowledge sets or as points contained in them. Our knowledge-based linear programming classifier (3.36) with a Gaussian kernel yielded the depicted nonlinear separating surface that classified all given points and sets correctly.

Another realization of the XOR example is depicted in Figure 3.2.2. Here the data points are not positioned symmetrically with respect to the origin and only one of them is contained in a knowledge set. The resulting nonlinear separating surface for this XOR example is constrained by one of the knowledge sets, in fact it is tangent to one of the diamond-shaped knowledge sets.

Checkerboard

Our second example is the classical checkerboard dataset [44, 45, 52, 53] which consists of 1000 points taken from a 16-square checkerboard. The following experiment on this

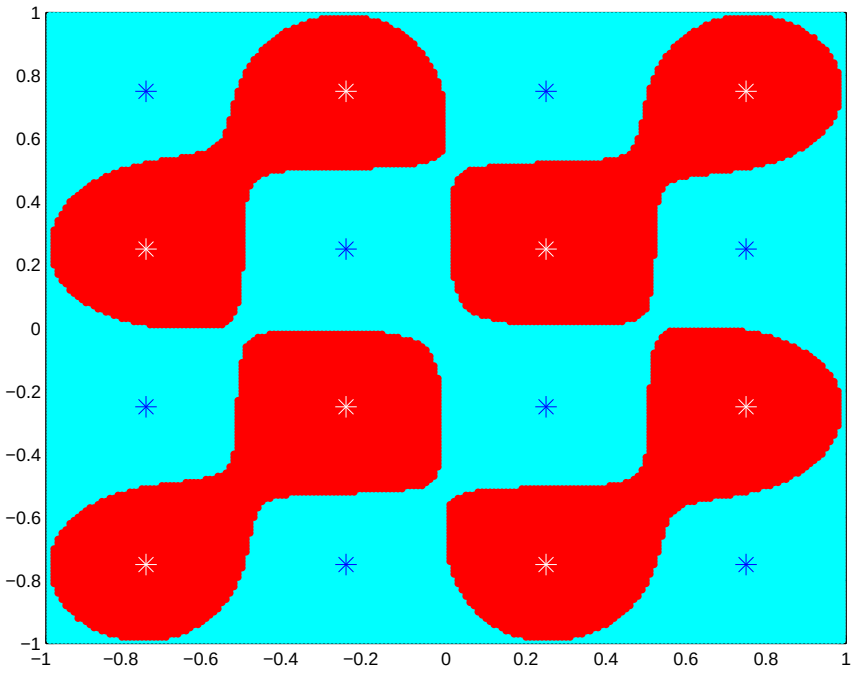

Figure 3.8: A poor nonlinear classifier based on 16 points taken from each of the 16 squares of a checkerboard. The nonlinear classifier was obtained by using a Gaussian kernel in a conventional linear programming formulation (3.32).

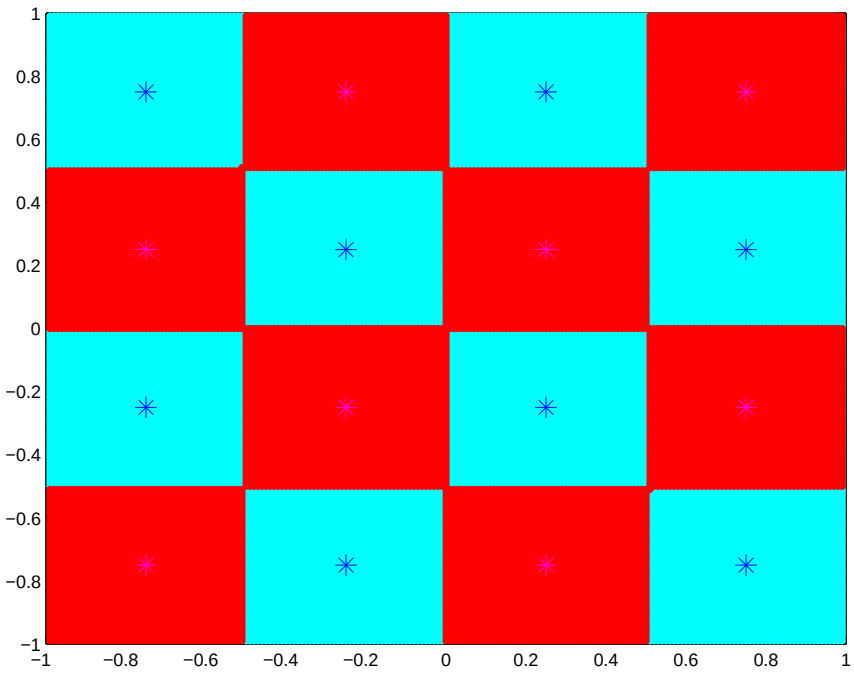

Figure 3.9: Totally or partially knowledge-based checkerboard classification problem. The nonlinear classifier was obtained by using a Gaussian kernel in our linear programming formulation (3.36) together with 16 points given either as a separate dataset or chosen from each of the 16 prior knowledge sets.

dataset shows the strong influence of knowledge sets on the separating surface.

We first took a subset of 16 points only, each one is the "center" of one of the 16 squares. Since we are using a nonlinear Gaussian kernel to model the separating surface, this particular choice of the training set is very appropriate for the checkerboard dataset. However, due to the nature of the Gaussian function it is hard for it to learn the "sharpness" of the checkerboard by using only a 16-point Gaussian kernel basis. We thus obtain a fairly poor Gaussian-based representation of the checkerboard depicted in Figure 3.2.3 with correctness of only 89.66% on a testing set of uniformly randomly generated 39601 points labeled according to the true checkerboard pattern.

On the other hand, if we use these same 16 points in the linear program (3.36) either as a distinct dataset in conjunction with prior knowledge in the form of only two of the 16 squares, one in each class, we obtain the sharply defined checkerboard depicted in Figure 3.2.3, with a correctness of 98.50% on the 39601-point testing set.

Note that increasing both the number and the size of the prior knowledge squares does not appear to improve correctness in a significant way. This is probably due to the fact that the checkerboard shape is "periodic" and the square pattern repeats itself. This means that the SVM "learns" the pattern with only two of the squares regarding of the square sizes once they are big enough. A centered square of 25% the original area is sufficient for this task.

It is interesting to note that our linear programming formulation (3.36) is capable of transforming complex prior knowledge, made up here of the union of 8 squares for each class, into a single nonlinear classifier equation given by $K(x', A')Du = \gamma$.

Chapter 4

Sparse Classifiers: Data and Feature Selection

4.1 Data Selection for Support Vector Machine Classifiers

When applying support vector machines, the separating surface that is obtained, depends only on a subset of the original data. This subset of data, which is all that is needed to generate the separating surface, constitutes the set of support vectors. In this section we give a method for selecting as small a set of support vectors as possible which completely determines a separating plane classifier. We term such a set of support vectors *minimal*, and the corresponding classifier, a *minimal support vector machine*. Such a classifier turns out to have improved testing set accuracy over one chosen by a standard support vector machine. Mathematically, support vectors are data points corresponding to constraints with positive multipliers in a constrained optimization formulation of a support vector machine. Computationally, the problem of determining a minimal set of support vectors does not appear to have been addressed before as proposed here. This is an important problem in applications such as fraud detection where the dataset may contain millions of data points. To make support vector machines viable for such applications, it is important

to identify a minimal set of support vectors, often an order of magnitude smaller than the original dataset, which determines the separating surface and allows the removal of redundant data. This dependence on a small subset of a given dataset, which often leads to an improved classifier, can be utilized in an incremental approach such as chunking [10, 70] where a small fraction of the data is maintained before merging and processing it with new incoming data. For the sake of simplicity and getting basic ideas across we will confine ourselves here to linear separating surfaces. The nonlinear case will be considered in detail in a subsequent section.

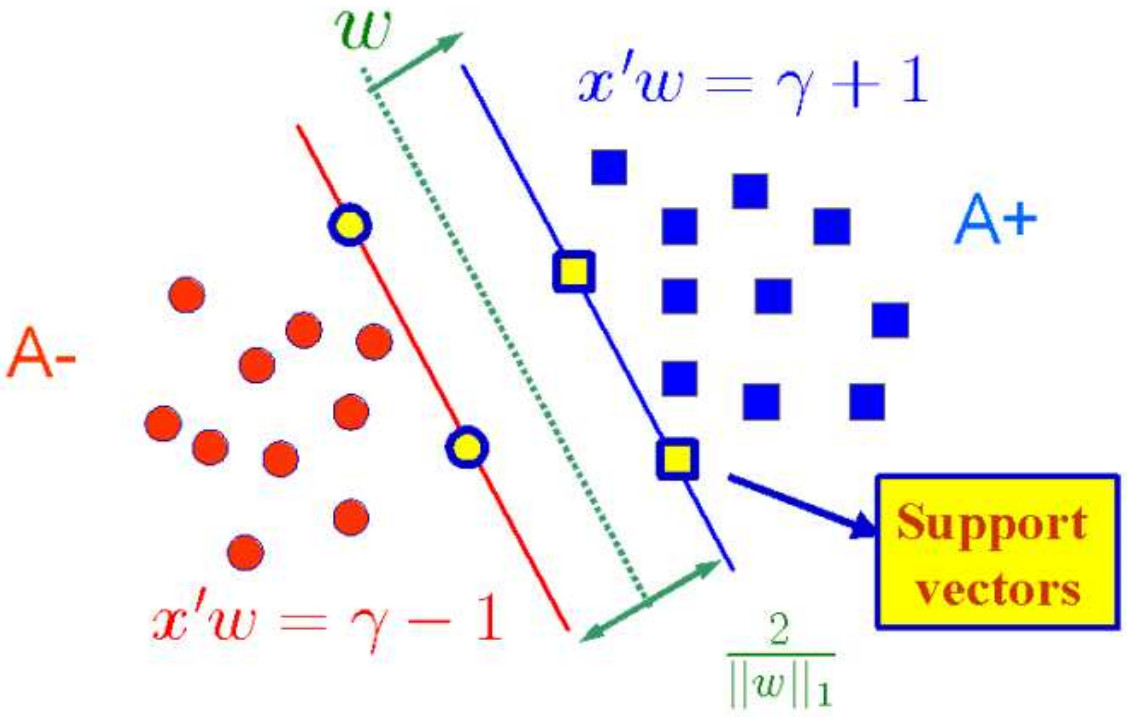

Figure 4.1: The linear programming support vector machine and the support vectors

4.1.1 MSVM: A Minimal Linear Support Vector Machine

Support vectors, which constitute the complement of the strictly separated points by the bounding planes (2.2), completely determine the separating surface. Minimizing the number of such exceptional points can lead to a minimum length description model [76, p. 66],[7] that depends on much fewer data points. Computational results indicate that such lean models generalize as well or better than models that depend on many more data points. We give in the next section of the paper an algorithm that minimizes the number of support vectors that determine the separating plane as well as the number of input space features.

In order to make use of a faster linear programming based approach, instead of the standard quadratic programming formulation (2.1), we will consider the linear programming formulation (3.1).

This SVM$\|\cdot\|_1$ reformulation in effect maximizes the margin, the distance between the two bounding planes of Figure 2.1, using a different norm, the ∞-norm, and results with a margin in terms of the 1-norm, $\frac{2}{\|w\|_1}$, instead of $\frac{2}{\|w\|_2}$ [63].

We will modify this linear program so as to generate an SVM with as few support vectors as possible by adding an error term $e'y_*$ to the objective function, where $_*$ denotes the step function as defined in Chapter 1, this is x_* denotes the vector in R^n with components $(x_*)_i$ equal 1 if $x_i > 0$ and 0 otherwise. The term $e'y_*$ suppresses misclassified points and results in our minimal support vector machine MSVM:

$$\begin{aligned} \min_{(w,\gamma,y,v)\in R^{n+1+m+n}} \quad & \nu e'y + e'v + \mu e'y_* \\ \text{s.t.} \quad & D(Aw - e\gamma) + y \geq e \\ & v \geq w \geq -v \\ & y \geq 0. \end{aligned} \tag{4.1}$$

Note that when the error vector y is zero all the points have been strictly separated by the plane $x'w = \gamma$. Thus, the separation error term in the objective function of (4.1) results in:

$$e'y_* = \sum_{i=1}^{m} y_{i*} = \bar{m}, \tag{4.2}$$

where $\bar{m}$ is the number of positive components of y_i, or equivalently the number of misclassified points by the bounding planes $x'w = \gamma \pm 1$. This number is directly related to the number of support vectors as shown below following equation (4.5). The positive parameter μ, chosen by a tuning set, multiplies the term $e'y_*$ which eliminates positive components of the error variable y. The justification for eliminating components of the error vector y, other than the intuitive idea of having a separating surface with as few misclassified points as possible, is as follows. If we define nonnegative multipliers $u \in R^m$ associated with the first set of constraints of the linear program (3.2), and multipliers $(r, s) \in R^{n+n}$ for the second set of constraints of (3.2), then the dual linear program [23] associated with the linear SVM formulation (3.2) is the following:

$$\begin{array}{rrcl} \max\limits_{(u,r,s)\in R^{m+n+n}} & e'u & & \\ \text{s.t.} & A'Du - r + s & = & 0 \\ & -e'Du & = & 0 \\ & u & \leq & \nu e \\ & r + s & = & e \\ & u,\ r,\ s & \geq & 0. \end{array} \tag{4.3}$$

Equality of the primal objective function of (3.2) and the dual objective function of (4.3) imply the (equality) complementarity conditions of the following Karush-Kuhn-Tucker

optimality conditions [59, p. 94] for (3.2):

$$\begin{array}{lcl} u'(D(Aw - e\gamma) + y - e) & = & 0 \\ u & \geq & 0 \\ D(Aw - e\gamma) + y - e & \geq & 0 \\ y'(\nu e - u) & = & 0 \\ y & \geq & 0 \\ \nu e - u & \geq & 0. \end{array} \tag{4.4}$$

These optimality conditions lead to the following implications for $i = 1, \ldots, m$:

$$\begin{array}{l} y_i > 0 \\ \qquad \Longrightarrow \qquad\qquad u_i = \nu > 0 \\ \qquad \Longrightarrow \quad D_{ii}(A_i w - \gamma) - 1 = -y_i < 0. \end{array} \tag{4.5}$$

Thus, a positive y_i implies a positive multiplier $u_i = \nu > 0$ and a corresponding support vector A_i that violates (2.3). Consequently eliminating positive components of y tends to minimize the number of multipliers at the upper bound ν as well as data points A_i that violate (2.3), that is, points that lie on the wrong sides of the bounding planes (2.2). Minimizing $e'y_*$ works remarkably well computationally in eliminating positive components of the multiplier u and consequently the number of misclassified points.

Even though the discontinuity of the step function term $e'y_*$ in (4.1) can be handled directly by an algorithm such as that of [62, Algorithm 1 SLA], we prefer to approximate it here by a smooth concave exponential on the nonnegative real line [61] as was done in the feature selection approach of [9] because of the proven track record of the smoothing approach in machine learning applications. For $y \geq 0$, the approximation of the step vector y_* of (4.1) by the concave exponential, $y_{i*} \approx 1 - \varepsilon^{-\alpha y_i}$, $i = 1, \ldots, m$, that is:

$$y_* \approx e - \varepsilon^{-\alpha y}, \; \alpha > 0, \tag{4.6}$$

where ε is the base of natural logarithms, leads to the following smooth reformulation of problem (4.1), the smooth MSVM:

$$\begin{aligned} \min_{(w,\gamma,y,v)\in R^{n+1+m+n}} \quad & \nu e'y + e'v + \mu e'(e - \varepsilon^{-\alpha y}) \\ \text{s.t.} \quad & D(Aw - e\gamma) + y \geq e \\ & v \geq w \geq -v \\ & y \geq 0. \end{aligned} \tag{4.7}$$

Note that:

$$e'(e - \varepsilon^{-\alpha y}) = m - \sum_{i=1}^{m} \varepsilon^{-\alpha y_i}. \tag{4.8}$$

It can be shown [11, Theorem 2.1] that, for a finite value of the parameter α (appearing in the concave exponential), the smooth problem (4.7) generates an *exact* solution of the nonsmooth problem (4.1). We note that this problem is the minimization of a concave objective function over a polyhedral set. Even though it is difficult to find a global solution to this problem, a fast successive linear approximation (SLA) algorithm [13, Algorithm 2.1] terminates finitely (usually in 4 to 7 steps) at a stationary point which satisfies the minimum principle necessary optimality condition for problem (4.7) [13, Theorem 2.2] and leads to a locally minimal number of support vectors, that is, a minimal number of data points A_i with positive multipliers u_i that completely determine the separating surface.

Algorithm 4.1.1 Successive Linearization Algorithm (SLA) for (4.7). *Choose* $\nu, \mu, \alpha > 0$. *Start with some* $(w^0, \gamma^0, y^0, v^0)$. *Having* $(w^i, \gamma^i, y^i, v^i)$ *determine the next iterate by solving the linear program:*

$$\begin{aligned} \min_{(w,\gamma,y,v)\in R^{n+1+m+n}} \quad & \nu e'y + e'v + \mu\alpha(\varepsilon^{-\alpha y^i})'(y - y^i) \\ s.t. \quad & D(Aw - e\gamma) + y \geq e \\ & v \geq w \geq -v \\ & y \geq 0. \end{aligned} \tag{4.9}$$

Stop when:

$$\nu e'(y - y^i) + e'(v - v^i) + \mu\alpha(\varepsilon^{-\alpha y^i})'(y - y^i) = 0. \tag{4.10}$$

Comment: The parameter α was set to 5. The parameters ν and μ were chosen with the help of a tuning set surrogate for a testing set to simultaneously minimize the number of support vectors, number of input space features and tuning set error.

We turn our attention now to numerical implementation and testing.

4.1.2 Numerical Implementation and Comparisons

Before applying Algorithm 4.1.1, which typically consists of solving 4 to 7 linear programs, the dimensionality of $w \in R^n$ was reduced by solving the 1-norm SVM (3.1), as a single linear program, with weight $\nu \in [0.01, 0.1]$ and discarding components of w less than 10^{-8} in magnitude. The reason for this dimensionality reduction, described more fully in [9], is the presence of the term $\|w\|_1$ in (3.1), which suppresses components of w.

The remaining components of w with the corresponding values of γ, y and v were used as the initial starting point $(w^0, \gamma^0, y^0, v^0)$ in Algorithm 4.1.1. After the termination of Algorithm 4.1.1, only support vectors were kept, that is A_i for which the multiplier $u_i > 10^{-8}$. This small set of support vectors generated the same stationary point for

problem (4.7) as that generated by the entire dataset. Such stationary points, which satisfy necessary optimality conditions, are typically good candidates to being a global solution to optimization problems of the type considered here.

The smooth minimal support vector machine (MSVM) (4.7) which generates a linear separating surface (2.3) by using a minimum number of data points was compared with the 1-norm support vector machine SVM $\|\cdot\|_1$ (3.1) as well as the the 1-norm support vector machine with feature selection FSV [9] which is problem (3.1) with the added feature-suppression term $\mu e'|w|_*$ in the objective function and smoothed to:

$$\mu e'(e - \varepsilon^{-\alpha|w|}) = \mu \sum_{i=1}^{n} (1 - \varepsilon^{-\alpha|w_i|}). \tag{4.11}$$

This smoothing, similar to that of (4.7)-(4.8) except that it is applied here to $|w|$ instead of y, leads to a selection of $\bar{n}(< n)$ input space features. The three classifiers MSVM (4.7), SVM$\|\cdot\|_1$ (3.1) and FSV [9, Eqn. (8)] were tested on seven datasets, the first five of which, WPBC, Ionosphere, Cleveland Heart, Pima Indians, and BUPA Liver are from the Irvine Machine Learning Repository [77], while the Galaxy Dim dataset is from [79], and the Census dataset is a version of the US Census Bureau "Adult" dataset, which is publicly available from Silicon Graphics' website [16]. For WPBC(60), 110 breast cancer patients were classified into those who had a recurrence of the disease within 60 months and those who had not. For the Census dataset, ten features were used to predict whether the income of a person was greater or equal to the mean income or below it. Our computational results are summarized in Table 4.1.2. for the three classifiers. We make the following observations based on numerical results:

1. For all test problems MSVM had the least number of support vectors. This translates into the least number of data points selected for determining the separating

surface and may be interpreted as a minimum description length model [76, p. 66],[7].

2. For the Ionosphere problem, the reduction in the number of support vectors of MSVM over SVM $\|\cdot\|_1$ is 81% with a corresponding increase of tenfold test set correctness of 5.6% with associated p value of 0.0003. (The p value measures the probability that two test results are the same, with sameness typically rejected if $p \leq 0.05$ [76]). For the seven datasets, the average reduction in the number of support vectors of MSVM over SVM $\|\cdot\|_1$ is 65.8%.

3. Tenfold testing set correctness of MSVM was as good or better on all seven datasets.

4. Computing times were higher for MSVM than those for SVM$\|\cdot\|_1$ and FSV. For example, one fold testing for the Galaxy Dim problem took 43.7 seconds on a 400 MHz Pentium II using MATLAB [75], while the corresponding times were 11.2 seconds for SVM$\|\cdot\|_1$ and 16.0 seconds for FSV. One justification of the additional time taken by MSVM is that it trades support vector storage space needed to generate a separating surface, with a one-time additional computational time expense.

Data Set $m \times n$	MSVM (Eqn. (4.7)) Train Test #Features #SV	SVM $\|\cdot\|_1$ (Eqn. (3.1)) Train Test #Features #SV	FSV [9, Eqn. (8)] Train Test #Features #SV
WPBC (60 mo.) 110×32	76.4% 68.3% 5.0 **29.6**	69.5% 62.1% 4.3 **69.4**	69.5% 62.1% 2.6 **69.1**
Ionosphere 351×34	91.4% 88.9% 7.0 **34.2**	84.6% 84.2% 7.2 **179.9**	91.2% 86.5% 8.1 **80.8**
Cleveland Heart 297×13	89.5% 86.9% 9.2 **38.5**	86.8% 85.8% 8.8 **109.8**	87.0% 85.2% 8.9 **92.4**
Pima Indians 768×8	76.6% 79.6% 7.5 **150.1**	77.1% 76.5% 6.8 **374.8**	76.9% 75.9% 5.0 **396.3**
BUPA Liver 345×6	72.7% 70.0% 6.0 **91.9**	71.3% 69.9% 6.0 **236.8**	70.0% 67.3% 4.5 **236.7**
Galaxy Dim 4192×14	95.0% 94.7% 5.0 **193.0**	94.4% 94.4% 5.0 **774.0**	94.9% 94.7% 4.9 **541.0**
Census $20,000 \times 10$	94.0% 94.1% 9.3 **1065.0**	93.9% 94.0% 9.8 **2745.5**	94.0% 93.8% 7.0 **2783.2**

Table 4.1: Tenfold training and testing correctness, number of features (#Features) and number of support vectors (#SV) used in seven public datasets by an MSVM classifier, and by a 1-norm SVM classifier without feature selection SVM $\|\cdot\|_1$ and with feature selection (FSV).

4.2 Minimal Kernel Classifiers

One of the main difficulties that confront large data classification by a nonlinear kernel is the possible dependence of the nonlinear separating surface on the entire dataset. This creates unwieldy storage and computational problems that may preclude the direct use of nonlinear kernels for large datasets or in applications where a very fast classifier is required such as in on-line credit card fraud detection.

For example, let $A \in R^{1000\times 10}$ represent a thousand-point dataset with 10 features characterizing each point. Then the above two difficulties translate into a nonlinear kernel matrix of size 1000×1000 with a million entries that leads to a dense nontrivial mathematical program. Even after solving this problem, the resulting nonlinear separating surface can potentially depend on most of the 1000 points that need to be stored and used in each classification of a new point.

In this section we propose a minimal kernel classifier method completely that addresses this difficulty by generating a nonlinear kernel-based classifier that reduces the classifier's data dependence by as much as 98.8%, compared to a conventional support vector machine classifier. In other words, the classifier depends on a very small number of kernel functions.

Such "minimum description length" models [87],[76, p. 66],[7] that depend on much fewer data points, often generalize as well or better than models that depend on many more data points, and are useful for incremental and chunking algorithms [10, 72] for massive datasets. In order to address the problem of solving huge mathematical programs, we use RSVM [52] to generate an initial reduced rectangular kernel that reduces dramatically the size of the problems to be solved while preserving and often, improving training and testing set correctness.

In this section we will consider the linear programming formulation for nonlinear kernels as described in chapter 3, equation 3.32.

The dual [59, p. 130] of this linear program is:

$$\begin{array}{rrrrrrr} \max\limits_{(t,r,s)\in R^{m+n+n}} & & e't & & & & \\ \text{s.t.} & DK(A,A')'Dt & -r & +s & = & 0 \\ & -e'Dt & & & = & 0 \\ & t & & & \leq & \nu e \\ & & r & +s & = & e \\ & t, & r, & s & \geq & 0 \end{array} \tag{4.12}$$

Note that, for any standard simplex algorithm [23], solution of either of the dual linear programs (3.32) or (4.12), automatically generates a solution of the other program at the termination of the algorithm.

We next derive our leave-one-out-correctness and leave-one-out-error bounds in terms of a solution to the above linear programs. Note that solving a *single* linear program (3.32) yields these bounds.

4.2.1 Leave-One-Out-Correctness (*looc*) and Leave-One-Out-Error (*looe*) Bounds

In this section we derive a lower bound on the leave-one-out-correctness (*looc*) of a solution to a support vector machine with a nonlinear kernel as well as an upper bound on the leave-one-out-error (*looe*), where $looc + looe = 1$. Our bounds, similar to those of [49, 81, 104, 105], are however easily and directly computable from a solution of the linear-programming-based formulation of the support vector machine formulation (3.32) above and are used as a justification for our concave minimization algorithm for a minimal

kernel classifier.

Proposition 4.2.1 Leave-One-Out-Correctness (*looc*) & Leave-One-Out-Error (*looe*) Bounds *Let (u, γ, y, v) be a solution of the linear program (3.32) and let (t, r, s) be a corresponding solution of its dual (4.12). The leave-one-out-correctness looc is bounded below as follows:*

$$looc \geq \frac{card((t \wedge u)_0)}{m} \tag{4.13}$$

and the leave-one-out-error looe is bounded above as follows:

$$looe \leq \frac{card((t \vee u)_+)}{m}, \tag{4.14}$$

where $card((t \wedge u)_0)$ denotes the cardinality of the set $\{i \mid t_i = 0 \text{ and } u_i = 0\}$, while $card((t \vee u)_+)$ denotes the cardinality of the set $\{i \mid t_i > 0 \text{ or } u_i \neq 0\}$.

Proof By the Karush-Kuhn-Tucker optimality conditions for (3.32) [59, p. 94] we have that:

$$\begin{array}{rcl} t'(D(K(A,A')Du - e\gamma) + y - e) & = & 0 \\ t & \geq & 0 \\ D(K(A,A')Du - e\gamma) + y - e & \geq & 0 \\ y'(\nu e - t) & = & 0 \\ y & \geq & 0 \\ \nu e - t & \geq & 0. \end{array} \tag{4.15}$$

These optimality conditions lead to the following implications for $i = 1, \ldots, m$:

$$\begin{array}{lrcl} y_i > 0 & & & \\ \Longrightarrow & t_i & = & \nu > 0 \\ \Longrightarrow & D_{ii}(K(A_i, A')Du - \gamma) - 1 & = & -y_i < 0. \end{array} \tag{4.16}$$

Thus, a positive y_i implies a positive multiplier $t_i = \nu > 0$ and a corresponding support vector A_i . Hence the number of support vectors equals or exceeds $\mathrm{card}(y_+)$, the number of positive components of y.

To establish (4.13) we observe that all data points A_i for which *both* the corresponding $t_i = 0$ (*i.e.* A_i is not a support vector) *and* $u_i = 0$ (*i.e.* $K(A, A_i')D_{ii}u_i = K(A, A_i')D_{ii} \cdot 0 = 0$), can be thrown out of the linear program (3.32) without changing the solution. For all such A_i we have by (4.15) that $y_i = 0$ and hence these A_i are correctly classified points, and if they were left out of the linear program (3.32) they would be correctly classified by the linear programming solution. Hence the leave-one-out correctness can be bounded below (because there could be other correctly classified A_i that could be thrown out also without changing the solution) by the cardinality of A_i for which both $t_i = 0$ and $u_i = 0$, that is:

$$looc \;\geq\; \frac{\mathrm{card}((t \wedge u)_0)}{m}, \tag{4.17}$$

which is the bound (4.13). Since $looc + looe = 1$ and

$$\mathrm{card}((t \wedge u)_0) + \mathrm{card}((t \vee u)_+) = m,$$

it follows from (4.17) that

$$1 - looe \;\geq\; \frac{\mathrm{card}((t \wedge u)_0)}{m} = \frac{m - \mathrm{card}((t \vee u)_+)}{m} = 1 - \frac{\mathrm{card}((t \vee u)_+)}{m},$$

from which follows the bound (4.14).□ Motivated by the above bound (4.14) on the *looe* and by linear programming perturbation theory [67], we present a minimal kernel algorithm that can obtain striking test set correctness results with a minimal use of data points as well as kernel components.

4.2.2 The Minimal Kernel Problem Formulation & Algorithm

By using an argument based on a finite perturbation of the objective function of a linear program, we look for solutions of the linear program (3.32), which in general has multiple solutions, that in addition suppress simultaneously as many components of the error of the primal variable u as well as the dual variable t. Empirical evidence [9] indicates that linear-programming-based classification problems are amenable to feature suppression. Since we do not want to solve both the primal and dual problems explicitly, which would be prohibitively costly for very large problems, we propose suppressing components of the error vector y as a reasonable surrogate for suppressing multiplier components t. The justification for this is that from the implication (4.16) we have that:

$$\{i \mid y_i > 0\} \subseteq \{i \mid t_i > 0\}.$$

Hence:

$$\frac{\text{card}((t \vee u)_+)}{m} \geq \frac{\text{card}((y \vee u)_+)}{m} = \frac{\text{card}((y \vee v)_+)}{m}, \tag{4.18}$$

where $\text{card}((y \vee u)_+)$ denotes the cardinality of the set $\{i \mid y_i > 0 \text{ or } u_i \neq 0\}$, $\text{card}((y \vee v)_+)$ denotes the cardinality of the set $\{i \mid y_i > 0 \text{ or } v_i > 0\}$, and the equality above follows from the fact that $v = |u|$ at a solution of the linear program (3.32). We propose to minimize the last term in (4.18) above instead of the actual upper bound on the *looe* given by the first term in (4.18). *This works remarkably well in reducing both data points needed and kernel size.* Consequently, we perturb the objective function of (3.32) by a step function $(\cdot)_*$ of y and of $v = |u|$, thus suppressing as many components of these variables as possible. This leads to the following minimization problem with a concave objective function on its feasible region:

$$\begin{aligned} \min_{(u,\gamma,y,v)\in R^{m+1+m+m}} \nu e'y + e'v + \mu(\nu e'y_* + e'v_*) &= \nu e'y_\# + e'v_\# \\ \text{s.t. } D(K(A,A')Du - e\gamma) + y &\geq e \\ v \geq u &\geq -v \\ y &\geq 0. \end{aligned} \tag{4.19}$$

Here, the "pound" function $(\cdot)_\#$ is defined as the following loss function in terms of the step function $(\cdot)_*$:

$$x_\# = |x| + \mu|x|_*, \text{ for some } \mu > 0. \tag{4.20}$$

Figure 4.2 depicts the shape of this loss function $x_\#$, which not only penalizes the amount of deviation from zero, but also *any* deviation from zero no matter how small by an initial penalty of μ which progresses linearly with the amount of deviation thereafter. Using linear programming perturbation theory we can state the following result that justifies our minimal kernel classifier algorithm.

Proposition 4.2.2 Minimal Kernel as Perturbed LP *For a given $\nu > 0$ there exist $\bar{\mu} > 0$, such that for all $\mu \in (0, \bar{\mu}]$, each solution of (4.19) is a solution of the linear programming kernel problem (3.32) with a minimal number of nonzero components of y and u among all its solutions.*

Proof Without loss of generality we assume that the feasible region of the linear program has no lines going to infinity in both directions and forgo the change of variables $w = \tilde{w} - e\zeta,\ \gamma = \tilde{\gamma} - \zeta,\ (\tilde{w}, \tilde{\gamma}, \zeta) \geq 0$. If we make this transformation then there are no lines in the feasible region that go to infinity in both directions, because all the variables would be nonnegative then.

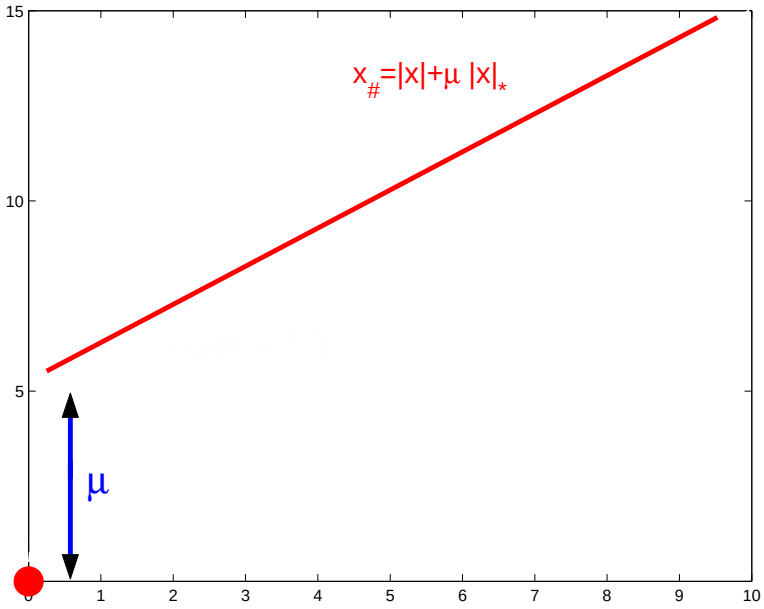

Figure 4.2: The loss function $x_{\#}$

Since the objective function of the problem (4.19) is concave on the feasible region and bounded below by zero it must have a vertex solution [88, Corollaries 32.3.3 and 32.3.4] for all nonnegative values of ν and μ.

Fix ν at some positive value and, to simplify notation and get the basic ideas across with as little detail as possible, let the concave program (4.19) be represented as follows:

$$\min_{z \in S} f(z) + \mu g(z), \tag{4.21}$$

where $z = (u, \gamma, y, v)$, S is the polyhedral feasible region of (4.19), f is linear and g is concave on S, that is:

$$f(z) := \nu e'y + e'v, \quad g(z) := \nu e'y_* + e'v_*.$$

Since the S has a finite number of vertices, for some decreasing sequence of positive numbers $\{\mu_0, \mu_1, \ldots, \} \downarrow 0$, some fixed vertex $\bar{z}$ of S will repeatedly solve the concave minimization problem (4.19). Hence for any $\mu \in (0, \mu_0]$, $\bar{z}$ is also a solution of (4.19),

because:

$$\mu = (1-\lambda)\mu_i + \lambda\mu_{i+1}, \text{ for some } i \text{ and some } \lambda \in [0,1],$$

and

$$f(\bar{z}) + \mu g(\bar{z}) = (1-\lambda)(f(\bar{z}) + \mu_i g(\bar{z})) + \lambda(f(\bar{z}) + \mu_{i+1} g(\bar{z})).$$

Hence $\bar{z}$ solves (4.21) for $\mu \in (0, \mu_0]$. We will now show that $\bar{z}$ also solves:

$$\min_{z \in S} f(z), \tag{4.22}$$

which is equivalent to the linear program (3.32). Define

$$\bar{f} := \min_{z \in S} f(z) \ \geq \ 0.$$

Suppose now that $f(\bar{z}) > \bar{f}$ and exhibit a contradiction. This will establish that $\bar{z}$ solves (4.22). Let

$$\epsilon := \frac{f(\bar{z}) - \bar{f}}{4} > 0 \text{ and } z \in S \text{ with } f(z) < \bar{f} + \epsilon, \tag{4.23}$$

and choose μ such that

$$\frac{1}{\mu} > \max\{\frac{g(z) - g(\bar{z})}{f(\bar{z}) - \bar{f} - 2\epsilon} \ , \ \frac{1}{\mu_0}\}. \tag{4.24}$$

We then have the following contradiction:

$$\mu g(z) + \bar{f} + \epsilon \ > \ \mu g(z) + f(z) \ \geq \ \mu g(\bar{z}) + f(\bar{z}) \ > \ \mu g(z) + \bar{f} + 2\epsilon,$$

where the first inequality follows from the last inequality of (4.23), the second inequality from the fact that $\bar{z}$ is a solution of (4.21), and the last inequality from (4.24). Hence $\bar{z}$ solves (4.22) which is equivalent to the linear program (3.32).

It remains to show that $\bar{z}$ also minimizes $g(z)$ as well over the set of minimizers of f on S. That is, we have picked among all solutions of the linear program (3.32) that one with a minimal number of nonzero components of y and u.

Let z be any solution of (4.22), that is $z \in S$ and $f(z) = f(\bar{z})$. Then,

$$\mu_0 g(z) = \mu_0 g(z) + f(z) - f(z) \ \geq \ f(z) + \mu_0 g(z) - f(\bar{z}) \ \geq f(\bar{z}) + \mu_0 g(\bar{z}) - f(\bar{z}) = \mu_0 g(\bar{z}).$$

Hence,

$$\bar{z} \in \arg\min_{z \in S} \{g(z) \mid f(z) \ \leq \ f(\bar{z}) = \min_{z \in S} f(z)\}.\square$$

Because of the discontinuity of the term $e'y_{\#}$, we approximate it by a concave exponential on the nonnegative real line. For $x \geq 0$, we approximate $x_{\#}$ of (4.20) by the concave exponential depicted in Figure 4.3. Thus for a vector $y \geq 0$:

$$y_{\#} \approx y + \mu(e - \varepsilon^{-\alpha y}), \ \ \alpha > 0, \tag{4.25}$$

where ε is the base of natural logarithms. This leads to the following smooth reformulation of problem (4.19):

$$\begin{aligned} \min_{(u,\gamma,y,v) \in R^{m+1+m+m}} \ & \nu e'(y + \mu(e - \varepsilon^{-\alpha y})) + e'(v + \mu(e - \varepsilon^{-\alpha v})) \\ \text{s.t. } & D(K(A, A')Du - e\gamma) + y \ \geq \ e \\ & v \geq u \ \geq \ -v \\ & y \ \geq \ 0. \end{aligned} \tag{4.26}$$

It can be shown [11, Theorem 2.1] that an exact solution to the original discontinuous problem (4.19) can be obtained by solving the above smooth problem (4.25) for any sufficiently large value of α, which is typically set to 5 in our numerical computations.

We now prescribe the highly effective and finite successive linearization algorithm (SLA) [11–13, 61, 64] for solving the above problem.

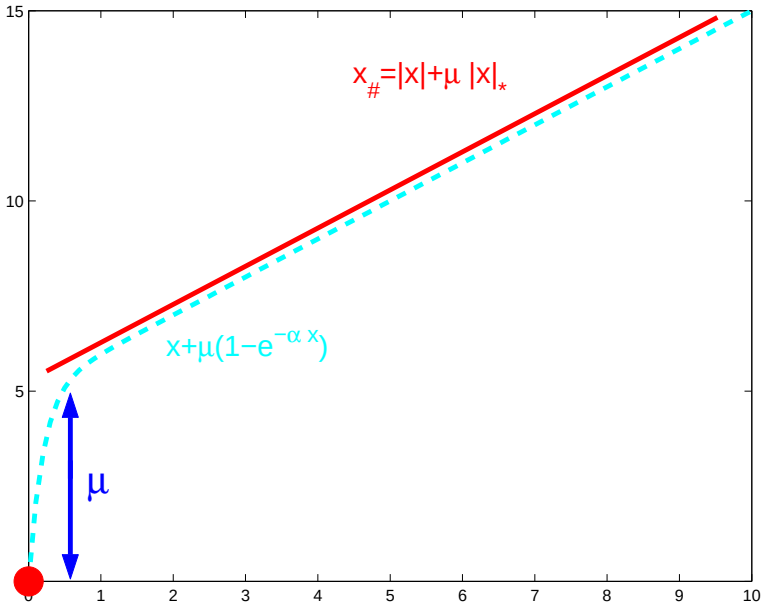

Figure 4.3: Loss function $x_{\#}$ approximation by $x + \mu(1 - \varepsilon^{\alpha x})$ on $x \geq 0$.

Algorithm 4.2.3 Minimal Kernel Algorithm *Start with an arbitrary* $(u^0, \gamma^0, y^0, v^0)$. *Having* $(u^i, \gamma^i, y^i, v^i)$ *determine the next iterate* $(u^{i+1}, \gamma^{i+1}, y^{i+1}, v^{i+1})$ *by solving the following linear program:*

$$\begin{aligned} \min_{(u,\gamma,y,v)\in R^{m+1+m+m}} \quad & \nu(e + \mu\alpha\varepsilon^{-\alpha y^i})'(y - y^i) + (e + \mu\alpha\varepsilon^{-\alpha v^i})'(v - v^i) \\ s.t.\ D(K(A, A')Du - e\gamma) + y &\geq e \\ v \geq u &\geq -v \\ y &\geq 0. \end{aligned} \tag{4.27}$$

Stop when:

$$\nu(e + \mu\alpha\varepsilon^{-\alpha y^i})'(y^{i+1} - y^i) + (e + \nu\alpha\varepsilon^{-\alpha v^i})'(v^{i+1} - v^i) = 0. \tag{4.28}$$

It can be shown [61] that this algorithm, which is essentially the repeated solution of the linear program (3.32), terminates in a finite number of steps (typically 5 to 8) at a point that satisfies the necessary optimality condition that the current iterate is a fixed point of the linearized problem.

Remark 4.2.4 *When $K(A, A')$ is large, for example where $m > 500$, it is convenient to use RSVM [52] in order to utilize a smaller reduced kernel $K(A, \bar{A}')$ where $\bar{A}' \in R^{\bar{m}\times n}$ and $\bar{m}$ is 5%-20% of m. In all cases, however, the initial $(u^0, \gamma^0, y^0, v^0)$ were obtained using the 1-norm formulation (3.32).*

Remark 4.2.5 *The Minimal Kernel Algorithm 4.2.3 terminates at a primal solution (u, v, y, γ) and a corresponding solution $(\tilde{t}, \tilde{r}, \tilde{s})$ to the dual of the last linearized problem (4.27). The reduced $m_1 \times m_2$ rectangular kernel $K(\bar{A}_{m_1}, \bar{A}_{m_2})$ for this last linearized problem has dimensions corresponding to:*

$$m_1 = card(\tilde{t}_+), \quad m_2 = card(u_+), \tag{4.29}$$

where:

$$\bar{A}_{m_1} := \{A_i \mid \tilde{t}_i > 0\} \text{ and } \bar{A}_{m_2} := \{A_i \mid |u|_i > 0\}. \tag{4.30}$$

Typically, m_1 and m_2 are considerably smaller than m. The number m_1 is the number of the first m constraints of the linear program (4.27) with positive dual multipliers at the solution of the last linear program solved by Algorithm 4.2.3. The number m_2 determines the number of data points that the classifier:

$$K(x', \bar{A}'_{m_2})Du_{m_2} - \gamma = 0, \tag{4.31}$$

depends on. We refer to m_2 as the number of kernel support vectors*. For a standard nonlinear quadratic programming support vector machine (2.1), $m_1 = m_2$. If the final linearization of (4.27) is re-solved with the smaller kernel $K(\bar{A}_{m_1}, \bar{A}_{m_2})$, the solution is the same as that obtained by Algorithm 4.2.3. However, if the standard 1-norm SVM formulation (3.32) is solved directly with the reduced kernel $K(\bar{A}_{m_1}, \bar{A}'_{m_2})$, then a result close to but not identical to that of Algorithm 4.2.3 is obtained because the objective function of (3.32) does not contain the linearization of the pound function.*

4.2.3 Computational Results

We tested Algorithm 4.2.3 on eight publicly available datasets in order to demonstrate that the algorithm gives equally correct test set results by using a drastically reduced kernel size compared to a kernel that uses the full dataset. A Gaussian kernel [19, 103] was used throughout the numerical tests:

$$K(A_i, A_j') = \varepsilon^{-\sigma\|A_i - A_j\|_2^2},$$

where σ is a positive parameter determined by using a tuning set for each dataset.

4.2.4 Results for the Checkerboard

The first dataset used was the Checkerboard [44, 45] consisting of 486 black points and 514 white points taken from a 16-square checkerboard. These 1000 points constituted the training dataset while the test set consisted of 39,601 points taken from a uniform 199×199 grid of points. This example was picked in order to demonstrate visually how a small subset of the original data can achieve an accurate separation.

4.2.5 Results on the USPS Dataset

Figure 4.4 depicts the separation obtained using a reduced kernel $K(A_{m_1}, A_{m_2})$ with $m_1 = 30$ and $m_2 = 27$. The 30 points constituting A_{m_1} and the 27 points constituting A_{m_2} are depicted in Figure 4.4 as circles and stars respectively. The total of these points is 5.7% of the original data and the rather accurate separating surface (4.31) depends merely on 27 points, that is 2.7% of the original data.

Although our algorithm is primarily intended for two-class problems, we have also applied it to the ten-category USPS (US Postal Service) dataset of hand-written numbers

[8, 103]. This well-known dataset consists of 7291 training patterns and 2007 testing patterns, collected from real-life zip codes. Each pattern consists of one of the ten numbers 0 to 9 and is represented by a digital image consisting of 16×16 pixels, which results in a 256-dimensional input space. In our tests here we used a one-from-the-rest approach, which led to the construction of 10 classifiers, each separating one class from the rest. The final classification was done by selecting the class corresponding to the classifier with the largest output value. The number of kernel support vectors reported in Table 1 is the total over the ten classifiers. We compare our algorithm with other kernel reducing methods such as the Sparse Greedy Matrix Approximation (**SGMA**) [93] and the Relevance Vector Machine (**RVM**) [99] as well as with the standard Support Vector Machine (**SVM**) [103]. In all the experiments in this section a Gaussian kernel was used. Table 1 compares the number of kernel support vectors for all these methods as well as the test set error. We note that our error rate is somewhat higher than that of the other methods. However, our number of kernel support vectors is the second smallest, in line with objectives of the paper. The average time to compute each of the ten classifiers for the USPS dataset was 24.6 minutes on our Pentium II 400MHz machine.

Method	**No. of Kernel Support Vectors**	**Test Error %**
MKC	376	6.7%
SVM [103]	1727	4.1%
SGMA [93]	590	*
RVM [99]	316	5.1%

Table 4.2: Comparison of total number of kernel support vectors (in ten classifiers) and test set error for 4 methods: Minimal Kernel Classifier (**MKC**), standard Support Vector Machine (**SVM**), Sparse Greedy Matrix Approximation (**SGMA**) and Relevance Vector Machine (**RVM**).
* No test error is reported in [93].

4.2.6 Results on Six Public Datasets

The next set of problems were from the University of California Irvine Repository [77] and varied in size between 297 to 8124 points, with input space dimensionality between 6 to 34. The purpose of the experiments was to show that the proposed Minimal Kernel Algorithm 4.2.3 can achieve three objectives:

(i) It can generate a nonlinear separating surface with less than 10% of the original data. This is a key property for incremental algorithms [35] where obsolete old data is retired before merging it with new incoming data.

(ii) Accuracy of ten-fold cross validation is as good or better than that of a nonlinear classifier that depends on a much bigger subset of the original training set.

(iii) Since the reduced kernel classifier depends on a very small subset of the original data, classification of a new point is done very quickly. This makes this method very attractive for applications where there are time constraints on the testing procedure or where there are storage constraints on the classifier.

The above and other results are given in Table 2 averaged over ten-fold runs for each dataset.

Data Set $m \times n$	Reduced rectangular kernel $m_1 \times m_2$	MKC Ten-fold test set correctness % (Ten-fold time sec.)	SVM Ten-fold test set correctness % (SV)*	Kernel Support vector reduction ** %	Testing time reduction †% (SVM-MKC time sec.)
Ionosphere 351×34	30.2×15.7	94.9% (172.3)	92.9% (288.2)	94.6%	94.9% (3.05-0.16)
Cleveland Heart 297×13	64.6×7.6	85.8% (147.2)	85.5 % (241.0)	96.9 %	96.3 % (0.84-0.03)
Pima Indians 768×8	$263.1 \times 7.8^{***}$	77.7 % (303.3)	76.6 % (637.3)	98.8%	98.8 % (3.95-0.05)
BUPA Liver 345×6	144.4×10.5	75.0 % (285.9)	72.7 % (310.5)	96.6%	97.5 % (0.59-0.02)
Tic-Tac-Toe 958×9	$31.3 \times 14.3^{***}$	98.4 % (150.4)	98.3 % (861.4)	98.3%	98.2 % (6.97-0.13)
Mushroom 8124×22	$933.8 \times 47.9^{***}$	89.3 % (2763.5)	oom (NA)	NA	NA

Table 4.3: Results for six UC Irvine datasets showing the percentage of reduction achieved over ten-fold runs. The last column gives testing time reduction resulting from using our minimal kernel classifier (MKC) instead of a regular full kernel classifier. The numbers m_1 and m_2 are averages over ten folds and refer to the dimensionality of the reduced kernel $K(\bar{A}_{m_1}, \bar{A}_{m_1})$ as explained in Remark 4.2.5. All linear programs were solved using CPLEX 6.5 [21]. NA denotes "Not Available", while oom denotes "out of memory".

* **Number of support vectors obtained using a standard quadratic programming nonlinear support vector machine (2.1).**

** **Comparison between the number of MKC kernel support vectors m_2 defined by (4.29) and the number of support vectors (SV) using the standard quadratic programming nonlinear support vector machine (2.1).**

*** **RSVM [52] was used here in order to obtain a smaller *initial* kernel for each of the Pima Indians dataset (768×150 instead of 768×768), the Tic-Tac-Toe dataset (958×96 instead of 958×958) and the Mushroom dataset (8124×400 instead of 8124×8124).**

† If T_{svm} **is the average single fold time over ten folds testing for the standard SVM classifier that depends on SV data points, and T_r is the average single fold time over ten folds testing using an MKC classifier that depends only on m_2 data points, then this percentage is given by:** $100 \times (1 - \frac{T_{svm}}{T_r})$

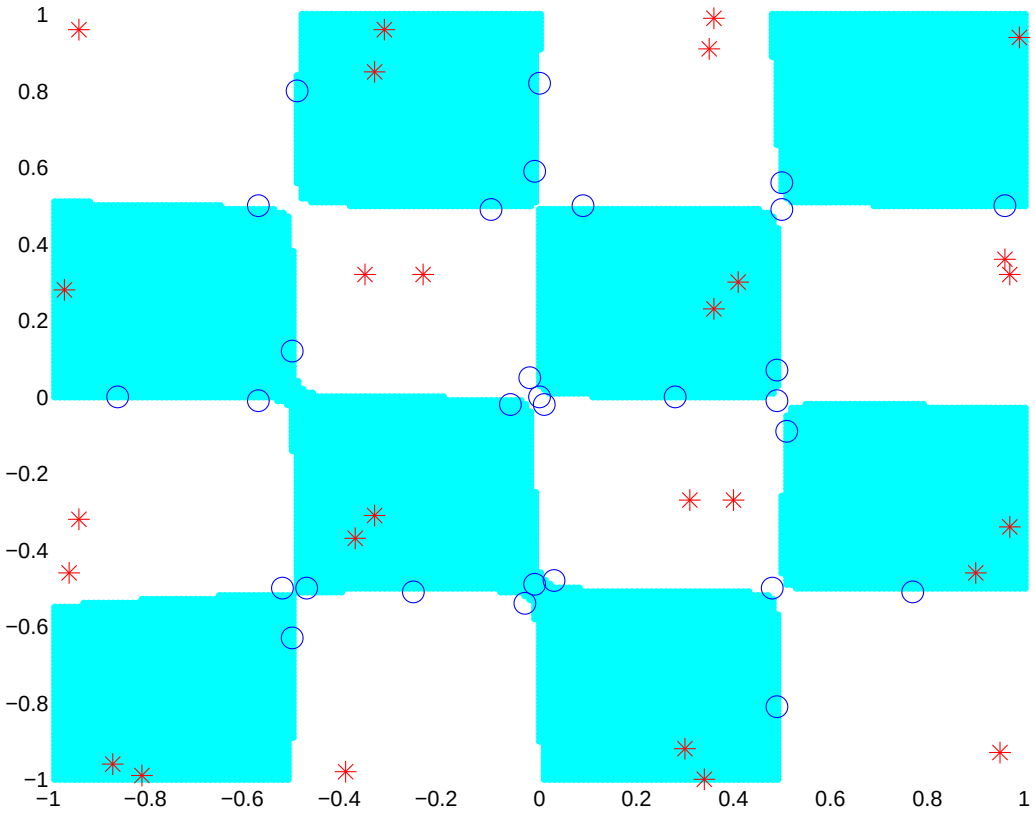

Figure 4.4: The checkerboard classifier depends on only 2.7% of the original data depicted as 27 stars, and is trained on only 3.0% of the original data depicted as 30 circles. The reduced kernel here is 30×27 compared to an original kernel for the full dataset of size $1,000 \times 1,000$. Running times were 109.9 seconds for the full kernel and 0.1 seconds for the reduced kernel.

4.3 A Feature Selection Newton Method for Support Vector Machine Classification

By minimizing an exterior penalty function of the dual of a linear programming formulation of a support vector machine (SVM) [9, 65], for a finite value of the penalty parameter, an exact least 2-norm solution to the SVM classifier is obtained. Our approach is based on a 1-norm SVM formulation that is known [9] to generate very sparse solutions. When a linear classifier is used, solution sparsity implies that the separating hyperplane depends on very few input features. This fact, makes this algorithm a very effective tool for feature selection for classification problems. On the other hand, when a nonlinear classifier is used, a sparse solution implies that few kernel functions determine the classifier. This makes the nonlinear classifier easier to store and faster to evaluate. The proposed Newton method requires only a linear equation solver and can be given in a few lines of a MATLAB [75] code. We note that a fast Newton method (NSVM) was also proposed recently in [34] that is based on a quadratic programming formulation of support vector machines. NSVM however, does not generate sparse solutions and hence does not suppress features at all. This contrasts sharply with the strong feature suppression property of the new algorithm proposed here.

4.3.1 Least 2-norm Solution of the Linear Programming SVM

In order to simplify the exterior penalty problem, here we will consider an slightly different formulation of the 1-norm formulation linear programming problem described in 3.1:

$$\begin{array}{rll} \min\limits_{(p,q,\gamma,y)} & \nu e'y + e'(p+q) & \\ \text{s.t.} & D(A(p-q) - e\gamma) + y & \geq \quad e \\ & p,q,y & \geq \quad 0, \end{array} \tag{4.32}$$

where the following substitution for w has been made:

$$w = p - q, \;\; p \geq 0, \; q \geq 0, \tag{4.33}$$

This is a different and a simpler linear program from previous linear programming SVM formulations [9, 65]. The dual of the linear program (4.32) is the following:

$$\begin{array}{rcccl} & \max\limits_{u \in R^m} & e'u & & \\ \text{s.t.} \quad -e & \leq & A'Du & \leq & e, \\ & & -e'Du & = & 0, \\ & & u & \leq & \nu e, \\ & & u & \geq & 0. \end{array} \tag{4.34}$$

The asymptotic exterior penalty problem [6, 30] for this linear program is the following nonnegatively constrained minimization problem:

$$\begin{array}{ll} \min\limits_{u \geq 0} & -\epsilon e'u + \frac{1}{2}\|(A'Du - e)_+\|^2 + \\ \frac{1}{2}\|(-A'Du - e)_+\|^2 & + \frac{1}{2}\|e'Du\|^2 + \frac{1}{2}\|(u - \nu e)_+\|^2, \end{array} \tag{4.35}$$

where ϵ is a positive penalty parameter that needs to approach zero for standard penalty application for solving the dual linear program (4.34). *However*, in our approach we shall establish the fact that ϵ will remain finite and we still can obtain an exact solution to

our linear programming SVM (4.32). To do that we first write the Karush-Kuhn-Tucker [59] necessary and sufficient optimality conditions for the penalty problem (4.35).

$$\begin{array}{rcl} 0 \leq u & \perp & (-\epsilon e + DA(A'Du - e)_+ \\ & & -DA(-A'Du - e)_+ \\ & & +Dee'Du + (u - \nu e)_+ \geq 0, \end{array} \tag{4.36}$$

where, as defined in the Introduction, $\perp$ denotes orthogonality. We will now show that these are also the necessary and sufficient conditions for finding an exact least 2-norm solution to the linear programming SVM (4.32) *without* ϵ approaching zero. To do that we first formulate the least 2-norm problem for (4.32) as follows:

$$\begin{array}{rl} \min\limits_{(p,q,\gamma,y)} & \nu e'y + e'(p+q) + \frac{\epsilon}{2}(\|p\|^2 + \|q\|^2 + \gamma^2 + \|y\|^2) \\ \text{s.t.} & D(A(p-q) - e\gamma) + y \geq e \\ & p, q, y \geq 0, \end{array} \tag{4.37}$$

with Karush-Kuhn-Tucker necessary and sufficient optimality conditions:

$$\begin{array}{rcl} 0 \leq \epsilon p & \perp & e + \epsilon p - A'Du \geq 0 \\ 0 \leq \epsilon q & \perp & e + \epsilon q + A'Du \geq 0 \\ & & \epsilon\gamma + e'Du = 0 \\ 0 \leq \epsilon y & \perp & \nu e + \epsilon y - u \geq 0 \\ 0 \leq u & \perp & DA(p-q) - De\gamma + y - e \geq 0. \end{array} \tag{4.38}$$

It follows, by [58, 67], that for any positive ϵ such that $\epsilon \in (0, \bar{\epsilon}]$ for some $\bar{\epsilon} > 0$, any (p, q, γ, y) satisfying the KKT conditions (4.38) for some $u \in R^m$, is the **exact** least 2-norm solution to the linear programming SVM (4.32). But, if we set in the KKT

conditions (4.36) for the penalty problem (4.35):

$$\begin{array}{rcl} p & = & \frac{1}{\epsilon}(A'Du - e)_+, \\ q & = & \frac{1}{\epsilon}(-A'Du - e)_+, \\ \gamma & = & -\frac{1}{\epsilon}e'Du, \\ y & = & \frac{1}{\epsilon}(u - \nu e)_+, \end{array} \tag{4.39}$$

and make use of the simple equivalence:

$$a = b_+ \iff 0 \leq a \perp (a - b) \geq 0, \tag{4.40}$$

then (4.39) together with the KKT conditions (4.36) for the exterior penalty problem (4.35), become precisely the KKT necessary and sufficient conditions (4.38) for the least 2-norm linear programming SVM (4.37). We have thus proven the following result.

Proposition 4.1 (Equivalence of Least 2-norm LP SVM to Dual Exterior Penalty) *A solution u to the dual exterior penalty (DEP) problem (4.35) for $\epsilon \in (0, \bar{\epsilon}]$ for some $\bar{\epsilon} > 0$, provides an* **exact** *least 2-norm solution to primal linear programming SVM (4.32) as follows:*

$$\begin{array}{rcl} w = p - q = & = & \frac{1}{\epsilon}((A'Du - e)_+ - (-A'Du - e)_+), \\ \gamma & = & -\frac{1}{\epsilon}e'Du, \\ y & = & \frac{1}{\epsilon}(u - \nu e)_+. \end{array} \tag{4.41}$$

We turn now to nonlinear kernel classifiers and use the notation of [65]. For $A \in R^{m \times n}$ and $B \in R^{n \times \ell}$, the **kernel** $K(A, B)$ maps $R^{m \times n} \times R^{n \times \ell}$ into $R^{m \times \ell}$. A typical kernel is the Gaussian kernel $\varepsilon^{-\mu \|A'_i - B_{\cdot j}\|^2}$, $i, j = 1, \ldots, m$, $\ell = m$, where ε is the base of natural logarithms, while a linear kernel is $K(A, B) = AB$. For a column vector x in R^n, $K(x', A')$ is a row vector in R^m, and the linear separating surface (2.4) is replaced by the nonlinear

surface:

$$K(x', A')Dv = \gamma, \tag{4.42}$$

where v is the solution of the dual problem (4.34). For a linear kernel $K(A, A') = AA'$, we have that $w = A'Dv$ [65] and the primal linear programming SVM (3.1) becomes upon using $w = p - q = A'Dv$ and the 1-norm of v in the objective instead that of w:

$$\begin{array}{rll} \min\limits_{(v,\gamma,y)} & \nu e'y + \|v\|_1 & \\ \text{s.t.} & D(AA'Dv - e\gamma) + y \;\geq\; e & \\ & y \;\geq\; 0, & \end{array} \tag{4.43}$$

Setting:

$$v = r - s, \;\; r \geq 0, \; s \geq 0, \tag{4.44}$$

the linear program (4.43) becomes:

$$\begin{array}{rll} \min\limits_{(r,s,\gamma,y)} & \nu e'y + e'(r+s) & \\ \text{s.t.} & D(AA'D(r-s) - e\gamma) + y \;\geq\; e & \\ & r, s, y \;\geq\; 0, & \end{array} \tag{4.45}$$

which is the linear kernel SVM in terms of the dual variable $v = r - s$. If we replace the linear kernel AA' in (4.45) by the nonlinear kernel $K(A, A')$ we obtain the nonlinear kernel linear program:

$$\begin{array}{rll} \min\limits_{(r,s,\gamma,y)} & \nu e'y + e'(r+s) & \\ \text{s.t.} & D(K(A, A')D(r-s) - e\gamma) + y \;\geq\; e & \\ & r, s, y \;\geq\; 0. & \end{array} \tag{4.46}$$

We immediately note that the linear program (4.46) is identical to the linear classifier SVM (4.32) if we let:

$$A \;\longrightarrow\; K(A, A')D, \tag{4.47}$$

in (4.32) and $n \longrightarrow m$. Hence the results outlined in Proposition 4.1 are applicable to a nonlinear kernel if we make the replacement (4.47) in (4.35) and (4.41) and let $p \longrightarrow r$, $q \longrightarrow s$, $w \longrightarrow v$ in (4.41) and use the nonlinear kernel classifier (4.42). As in the linear case, the 1-norm formulation (4.47) leads to a very sparse v. Every zero component v_i of v implies non-dependence of the nonlinear kernel classifier on the kernel function $K(x', A_i')$. This is because:

$$\begin{array}{rcl} K(x', A')Dv & = & \sum_{i=1}^{m} D_{ii} v_i K(x', A_i') \\ & = & \sum_{\{i|v_i \neq 0\}} D_{ii} v_i K(x', A_i'). \end{array} \tag{4.48}$$

We turn now to our algorithmic implementation of Proposition 4.1.

4.3.2 Newton Method for Linear Programming SVM (NLPSVM)

We shall solve the exterior the dual exterior penalty (4.35) for a finite value of the penalty parameter ϵ and by incorporating the nonnegativity constraint $u \geq 0$ into the objective function of (4.35) as a penalty term as follows:

$$\begin{array}{rl} \min\limits_{u} f(u) = & -\epsilon e'u + \frac{1}{2}\|(A'Du - e)_+\|^2 \\ & +\frac{1}{2}\|(-A'Du - e)_+\|^2 + \frac{1}{2}\|e'Du\|^2 \\ & +\frac{1}{2}\|(u - \nu e)_+\|^2 + \frac{\alpha}{2}\|(-u)_+\|^2. \end{array} \tag{4.49}$$

The gradient of this function is given by:

$$\begin{array}{rcl} \nabla f(u) & = & -\epsilon e + DA(A'Du - e)_+ - DA(-A'Du - e)_+ \\ & & +Dee'Du + (u - \nu e)_+ - \alpha(-u)_+, \end{array} \tag{4.50}$$

and its generalized Hessian as defined by (1.2) in the Introduction:

$$\begin{aligned}\partial^2 f(u) &= DA(diag((A'Du-e)_* + (-A'Du-e)_*)A'D \\ &\quad +Dee'D + diag((u-\nu e)_* + \alpha(-u)_*) \\ &= DA(diag(|A'Du|-e)_*)A'D \\ &\quad +Dee'D + diag((u-\nu e)_* + \alpha(-u)_*),\end{aligned} \tag{4.51}$$

where the last equality follows from the equality:

$$(a-1)_* + (-a-1)_* = (|a|-1)_*. \tag{4.52}$$

We are ready now to state our Newton algorithm.

Algorithm 4.1 Newton Algorithm for (4.35) *Let $f(u)$, $\nabla f(u)$ and $\partial^2 f(u)$ be defined by (4.49)-(4.51). Set the parameter values ν, ϵ, δ, tolerance* tol*, α and* imax *(typically: $\epsilon = 10^{-1}$, $tol = 10^{-3}$, $\alpha = 10^3$, $imax = 50$, while ν and δ are set by a tuning procedure described in Section 4.3.3). Start with any $u^0 \in R^m$. For $i = 0, 1, \ldots$:*

(I) $u^{i+1} = u^i - \lambda_i(\partial^2 f(u^i) + \delta I)^{-1}\nabla f(u^i) = u^i + \lambda_i d^i$,

where the Armijo stepsize $\lambda_i = \max\{1, \frac{1}{2}, \frac{1}{4}, \ldots\}$ is such that:

$$f(u^i) - f(u^i + \lambda_i d^i) \geq -\frac{\lambda_i}{4}\nabla f(u^i)' d^i, \tag{4.53}$$

and d^i is the modified Newton direction:

$$d^i = -(\partial^2 f(u^i) + \delta I)^{-1}\nabla f(u^i). \tag{4.54}$$

(II) Stop if $\|u^i - u^{i+1}\| \leq tol$ or $i = imax$. Else, set $i = i+1$, $\alpha = 2\alpha$ and go to (I).

(III) Define the least 2-norm solution of the linear programming SVM (4.32) by (4.41) with $u = u^i$.

We state a convergence result for this algorithm now.

Theorem 4.1 *Let tol* $= 0$*, imax* $= \infty$ *and let* $\epsilon > 0$ *be sufficiently small. Each accumulation point* $\bar{u}$ *of the sequence* $\{u^i\}$ *generated by Algorithm 4.1 solves the exterior penalty problem (4.35). The corresponding* $(\bar{w}, \bar{\gamma}, \bar{y})$ *obtained by setting* u *to* $\bar{u}$ *in (4.41) is the exact least 2-norm solution to the primal linear program SVM (4.32).*

Proof That each accumulation point $\bar{u}$ of the sequence $\{u^i\}$ solves the minimization problem (4.35) follows from exterior penalty results [6, 30] and standard unconstrained descent methods such as [60, Theorem 2.1, Examples 2.1(i), 2.2(iv)] and the facts that the direction choice d^i of (4.45) satisfies, for some $c > 0$:

$$\begin{aligned} -\nabla f(u^i)' d^i &= \nabla f(u^i)'(\delta I + \partial^2 f(u^i))^{-1} \nabla f(u^i) \\ &\geq c \|\nabla f(u^i)\|^2, \end{aligned} \tag{4.55}$$

and that we are using an Armijo stepsize (4.53). The last statement of the theorem follows from Proposition 4.1.□

Remark 4.3.1 Choice of ϵ *Determining the size of* $\bar{\epsilon}$*, such that the solution* u *of the quadratic program (4.37) for* $\epsilon \in (0, \bar{\epsilon}]$*, is the least 2-norm solution of the problem (4.32), is not an easy problem theoretically. However, computationally this does not seem to be critical and is effectively addressed as follows. By [56, Corollary 3.2], if for two successive values of* ϵ*:* $\epsilon^1 > \epsilon^2$*, the corresponding solutions of the* ϵ*-perturbed quadratic programs (4.37):* u^1 *and* u^2 *are equal, then under certain assumptions,* $u = u^1 = u^2$ *is the least 2-norm solution of the dual linear program (4.32). This result can be implemented computationally by using an* ϵ*, which when decreased by some factor yields the same solution to (4.32).*

We turn now to our computational results.

4.3.3 Numerical Experience

In order to show that our algorithm can achieve very significant feature suppression, our numerical tests and comparisons were carried out on a dataset with the high dimensional input space and a moderate number of data points. On the other hand, in order to show that our proposed algorithm has a computational speed comparable to that of other fast methods, we also performed experiments on more conventional datasets where the dimensionality of the input space is considerably smaller than the number of data points.

Because of the simplicity of our algorithm, we give below a simple MATLAB implementation of the algorithm without the Armijo stepsize, which does not seem to be needed in most applications. Although this is merely an empirical observation in the present case, it considerably simplifies our MATLAB Code 4.1. However, it has also been shown [66, Theorem 3.6] that under a well conditioned assumption, not generally satisfied here, the proposed Newton method indeed terminates in a finite number of steps without an Armijo stepsize. Note that this version of the algorithm is intended for cases where the number of data point m is smaller that the number of features n, i.e. when $m \ll n$ since the speed of the algorithm depends on m in a cubic fashion.

Code 4.1 NLPSVM Code

```
function [w,gamma]=nlpsvm(A,d,nu,delta)
%NLPSV: linear and nonlinear classification
%       without Armijo
%INPUT: A, D, nu, delta. OUTPUT=w, gamma.
%[w,gamma]=nlpsvm(A,d,nu,delta)
```

```
epsi=10^(-1);alpha=10^3;tol=10^(-3);imax=50;
[m,n]=size(A);en=ones(n,1);em=ones(m,1);
u=ones(m,1);%initial point
iter=0;g=1;
epsi=epsi*em;nu=nu*em;
DA=spdiags(d,0,m,m)*A;
while (norm(g)>tol) & (iter<imax)
  iter=iter+1;
  du=d.*u;Adu=A'*du;
  pp=max(Adu-en,0);np=max(-Adu-en,0);
  dd=sum(du)*d;unu=max(u-nu,0);uu=max(-u,0);
  g=-epsi+(d.*(A*pp))-(d.*(A*np))+dd+unu-alpha*uu;
  E=spdiags(sqrt(sign(np)+sign(pp)),0,n,n);
  H=[DA*E d];
  F=delta+sign(unu)+alpha*sign(uu);
  F=spdiags(F,0,m,m);
  di=-((H*H'+F)\g);
  u=u+di;
end
du=d.*u;Adu=A'*du;
pp=max(Adu-en,0);np=max(-Adu-en,0);
w=1/epsi(1)*(pp-np);
gamma=-(1/epsi(1))*sum(du);
return
```

We further note that the MATLAB code above not only works for a linear classifier, but also for a nonlinear classifier as well [65, Equations (1), (10)]. In the nonlinear case, the matrix $K(A, A')D$ is used as input instead of A, and the pair (v, γ), is returned instead of (w, γ). The nonlinear separating surface is then given by (4.37) as:

$$K(x, A')Dv - \gamma = 0. \tag{4.56}$$

Our first numerical testing and comparisons were carried out on the high dimensional Multiple Myeloma dataset available at: http://lambertlab.uams.edu/publicdata.htm, and processed by by David Page and his colleagues [83]. The structure of this dataset with very large n and $(m \ll n)$ results from the DNA microarray dataset used. Hence, feature selection is very desirable in such high dimensional problems. Other tests and comparisons were also carried out on six moderately dimensioned, publicly available datasets [77, 79] and are described in Section 4.3.3.

Multiple Myeloma Dataset

Multiple Myeloma is cancer of the plasma cell. The plasma cell normally produces antibodies that destroy foreign bodies such as bacteria. As a product of the Myeloma disease the plasma cells get out of control and produce a tumor. These tumors can grow in several sites, usually in the soft middle part of bone, the bone marrow. When these tumors appear in multiples sites they are called Multiple Myeloma. A detailed description of the process used to obtain the data can be found in [83].

Description of the Dataset

The data consists of 105 data points, 74 of the points representing newly-diagnosed multiple Myeloma patients while 31 points represent 31 healthy donors. Each data point represents measurements taken from 7008 genes using plasma cells samples from the patients. For each one of the 7008 genes there are two measurements. One measurement is called Absolute Call (AC) and takes on one of three nominal values: A (Absent), M (Marginal) or P (Present). The other measurement, the average difference (AD), is a floating point number that can be either positive or negative. Since each one of the 7008 AC features takes on nominal values from the set $\{A, M, P\}$, a real valued representation is needed to utilize our classifier which requires an input of real numbers. Thus, each nominal value is mapped into a three dimensional binary vector depending on the value that is being represented. This simple and widely used "1 of N" mapping scheme for converting nominal attributes into real-valued attributes is illustrated in Figure 3.4. Once this simple conversion is applied to the dataset, the AC feature space is transformed from a 7008-dimensional space with nominal values A, M, P into a $7008 \times 3 = 21024$ real-valued dimensional space. Adding the numerical AD feature for each of the 7008 genes results in each data point being transformed to a point in R^{28032}, with 21024 coming from the AC transformation mentioned above and 7008 from the AD values. This makes this dataset very interesting for our method, since a main objective of this paper is to show that our proposed algorithm does a remarkable job of suppressing features especially for datasets in a very high dimensional input space.

Numerical Comparisons

Performance of our Newton Linear Programming SVM (**NLPSVM**) algorithm on the Myeloma dataset, in terms of feature selection and generalization ability, is first compared with two publicly available SVM solvers: LSVM [71] and NSVM [34]. Reported times for LSVM here differ from the ones reported in [71] because the calculation time for the matrix H of (4.35) is considered as input time in [71], whereas here it is counted as part of the computational time. The other algorithm included in our comparisons, consists of solving the linear programing formulation (4.46) employing the widely used commercial solver CPLEX 6.5 [48]. We call this approach CPLEX SVM. Termination criteria for all methods, with the exception of CPLEX SVM, was set to $tol = 0.001$, which is the default for LSVM. For CPLEX SVM the termination criterion used was the default supplied in the commercial package. We outline some of the results of our comparative testing.

- All three methods tested: NSVM, NLPSVM and CPLEX SVM obtained 100% *leave-one-out correctness (looc)*. The following tuning procedure was employed for each of the 105 folds:
  - A random tuning set of the the size of 10% of the training data was chosen and separated from the training set.
  - Several SVMs were trained on the remaining 90% of the training data using values of ν equal to 2^i with $i = -12, \ldots, 0, \ldots, 12$. Values of the parameter δ tried were 10^j with $j = -3, \ldots, 0, \ldots, 3$. This made the search space for the pair (ν, δ) a grid of dimension 25×7.
  - Values of ν and δ that gave the best SVM correctness on the tuning set were chosen.

  - A final SVM was trained using the chosen values of ν, δ and all the training data. The resulting SVM was tested on the testing data.

- The remaining parameters were set to the following values: $\epsilon = 10^{-1}, \alpha = 10^{3}, tol = 10^{-3}, imax = 50$

- Our NLPSVM algorithm outperformed all others in the feature selection task, and it obtained 100% looc using only 7 features out of 28032 original features. The closest contender was CPLEX SVM which required four times as many features. This is quite a significant result because using a small number of features is critical for interpretation of the classification results by biologists.

- The average cpu time required by our algorithm for the leave-one-out correctness (*looc*) computations was 75.16 seconds per fold and total time for 105 folds was 7891.80 seconds. This outperformed CPLEX SVM both in cpu time and number of features used. CPLEX SVM had a cpu time of 108.28 per fold, a total time of 11369.40 seconds and used 28 features. However, NLPSVM was considerably slower than the NSVM which had a cpu time of 4.20 seconds average per fold and total *looc* time of 441.00 seconds. Also, the NSVM classifier required 6554 features, more than any classifier obtained by all other methods.

- LSVM failed and reported an out of memory error.

These results are summarized in Table 4.3.3 below.

Tests on Six Other Datasets

In this section we exhibit the effectiveness of NLPSVM in performing feature selection while maintaining accuracy and cpu time comparable to those of other methods that do

Data Set $m \times n$ ($points \times dimensions$)	NSVM[34] looc Time (Sec.) Features	CPLEX SVM[48] looc Time (Sec.) Features	LSVM[71] looc Time (Sec.) Features	NLPSVM looc Time (Sec.) Features
Myeloma 105×28032	**100.0%** **441.00** 6554	**100.0%** 11369.40 28	oom oom oom	**100%** 7891.80 **7**

Table 4.4: NSVM [34], CPLEX SVM [48], LSVM [71] & NLPSVM : *leave-one-out correctness (looc)*, total running times and number of features used by a linear classifier for the Myeloma dataset. Best results are in bold. oom stands for "out of memory".

not perform feature selection. We tested our algorithm on six publicly available datasets. Five from the UCI Machine Learning Repository [77]: Ionosphere, Cleveland Heart, Pima Indians, BUPA Liver and Housing. The sixth dataset is the Galaxy Dim dataset available at [79]. The dimensionality and size of each dataset is given in Table 4.3.3.

Numerical Comparisons Using a Linear Classifier

In this set of experiments we used a linear classifier to compare our method NLPSVM with LSVM, NSVM and CPLEX SVM on the six datasets mentioned above. Because $m \gg n$ for these datasets, it was preferable to use the Sherman-Morrison-Woodbury identity [40] to calculate the direction d_i in the Newton iteration (4.54) and solve an $(n+1) \times (n+1)$ linear system of equations instead of an $m \times m$ linear system of equations. For this purpose define:

$$\begin{aligned} &E^2 := diag(|A'Du| - e)_*, \\ &H := D[AE \;\; e]. \\ &\text{and } \; F := diag((u - \nu e)_* + \alpha(-u)_*) + \delta I. \end{aligned} \tag{4.57}$$

Then, it follows from (4.51) that:

$$\partial^2 f(u) + \delta I = HH' + F,$$

which is the matrix whose inverse is needed in the Newton iteration (4.54).

Applying the Sherman-Morrison-Woodbury identity we have:

$$(HH' + F)^{-1} = F^{-1} - F^{-1}H(I + H'F^{-1}H)^{-1}H'F^{-1}$$

Note that the inverse F^{-1} of F is trivial to calculate since F is a diagonal matrix.

This simple but effective algebraic manipulation makes our algorithm very fast even when $m \gg n$ but n is relatively small.

The values for the parameters ν and δ were again calculated using the same tuning procedure given in Section 4.3.3. The values of the remaining parameters were the same as those used in Section 4.3.3. As shown in Table 4.3.3, the correctness of the four methods was very similar, the execution time including ten-fold cross validation for NSVM was less for all the datasets tested. However, all solutions obtained by NSVM depended on **all** the original input features. In contrast, NLPSVM performed comparably to LSVM, was always faster than CPLEX SVM but used the least number of features on all the datasets compared to all other methods tested.

Numerical Comparisons Using a Nonlinear Classifier

In order to show that our algorithm can also be used to find nonlinear classifiers, we chose three datasets from the UCI Machine Learning Repository for which it is known that a nonlinear classifier performs better that a linear classifier. We used NSVM, LSVM , CPLEX SVM and our proposed algorithm NLPSVM in order to find a nonlinear classifier based on the Gaussian kernel:

$$(K(A,B))_{ij} = \varepsilon^{-\mu\|A_i{}' - B_{\cdot j}\|^2}, \quad i = 1, \ldots, m,\ j = 1, \ldots, k. \tag{4.58}$$

where $A \in R^{m\times n}$, $B \in R^{n\times k}$ and μ is a positive constant. The value of μ in the Gaussian kernel and the value of ν in all the algorithms were chosen by tuning on the values 2^i with i an integer ranging from -12 to 12 following the same procedure described in Section 4.3.3. The value of δ in NLPSVM was obtained also by tuning on the values 10^j with $j = -3, \ldots, 0, \ldots, 3$. The value of the parameter ϵ in this case was set to 10^{-1}. The values of the remaining parameters were the same as in Section 4.3.3. Because the nonlinear kernel matrix is square and since NLPSVM, NSVM and LSVM perform better on rectangular matrices, we also used a rectangular kernel formulation as described in the Reduced SVM (RSVM) [52]. This resulted in as good or better correctness and much faster running times. The size of the random sample used to calculate the rectangular kernel was 10% of the size of the original dataset in all cases. We refer to these variations of NSVM,LSVM, CPLEX SVM and NLPSVM as Reduced NSVM, Reduced LSVM, Reduced CPLEX SVM and Reduced NLPSVM respectively. The results are summarized in Table 4.3.3 for these nonlinear classifiers.

Again, as in the linear case the correctness of the four methods was similar on all the datasets, the execution time including ten-fold cross validation for NSVM was less for all the datasets tested, but with non-sparse solutions. NLPSVM performance was fast when a reduced rectangular kernel was used and it obtained very sparse solutions that resulted in nonlinear kernel classifiers that are easier to store and to evaluate.

Data Set $m \times n$ (*points* × *dimensions*)	NSVM Train Test Time (Sec.) Features	CPLEX SVM Train Test Time (Sec.) Features	LSVM Train Test Time (Sec.) Features	NLPSVM Train Test Time (Sec.) Features
Ionosphere 351 × 34	**92.9%** **88.9%** **0.91** 34	90.9 % 88.3 % 3.2 17.7	**92.9%** **88.9%** 1.49 34	90.7% 88.0% 2.4 **11.2**
BUPA Liver 345 × 6	70.3% **70.2%** **0.24** 6	**71.2%** 69.9% 5.17 6	70.3% **70.2%** 0.92 6	70.6% 68.8% 1.13 **4.8**
Pima Indians 768 × 8	**77.7%** **77.2%** **0.55** 8	76.8% 77.0% 3.94 5	**77.7%** **77.2%** 2.30 8	76.8% 77.1% 1.07 **4.9**
Cleveland Heart 297 × 13	**87.2%** **86.6%** **0.14** 13	85.9% 85.5% 1.08 7.5	**87.2%** **86.6%** 0.31 13	86.5% 85.9% 0.55 **7.1**
Housing 506 × 13	**87.7%** **86.8%** **0.69** 13	**87.7%** 85.0% 2.54 10.9	**87.7%** **86.8%** 1.53 13	87.0% 85.2% 1.91 **6.5**
Galaxy Dim 4192 × 14	94.0% 94.2% **6.67** 14	**94.7%** **94.7%** 29.82 5	94.0% 94.2% 71.56 14	94.4% 94.6% 8.90 **3.4**

Table 4.5: NSVM [34], CPLEX SVM [48], LSVM [71] & NLPSVM: Training correctness, ten-fold testing correctness, ten-fold training times and number of features needed using a LINEAR classifier. All parameters ν, δ chosen by tuning. For each algorithm a reduced kernel version was also tested. Best results are in bold. Training and testing correctness and number of features are all averages over ten folds, while time is the total time over ten folds.

Algorithm	Data Set $m \times n$ ($points \times dimensions$)	Ionosphere 351×34	BUPA Liver 345×6	Cleveland Heart 297×13
NSVM	Train	96.1	75.7	87.6
	Test	95.0	73.1	86.8
	Time (Sec.)	23.27	25.54	17.51
	$Card(v)$	351	345	297
Reduced NSVM	Train	96.1	**76.4**	86.8
	Test	94.5	73.9	**87.1**
	Time (Sec.)	**0.88**	**0.67**	**0.53**
	$Card(v)$	35	35	30
LSVM	Train	96.1	75.7	87.6
	Test	95.0	73.1	86.8
	Time (Sec.)	23.76	27.01	12.98
	$Card(v)$	351	345	297
Reduced LSVM	Train	96.1	75.1	87.1
	Test	94.5	73.1	86.2
	Time (Sec.)	2.09	1.81	1.09
	$Card(v)$	35	35	30
NLPSVM	Train	94.4	75.4	86.9
	Test	93.5	**73.9**	86.2
	Time (Sec.)	195.31	187.91	70.47
	$Card(v)$	22.3	32.7	50.1
Reduced NLPSVM	Train	94.4	74.5	85.9
	Test	95.1	**73.9**	86.5
	Time (Sec.)	2.65	6.82	5.17
	$Card(v)$	**14.6**	**16.4**	**12.3**
CPLEX SVM	Train	**99.2**	**76.4**	**87.8**
	Test	**96.1**	73.6	86.2
	Time (Sec.)	34.8	34.48	18.37
	$Card(v)$	36.1	26.2	14.1
Reduced CPLEX SVM	Train	98.7	**76.4**	87.0
	Test	95.5	**73.9**	85.6
	Time (Sec.)	3.08	4.42	2.47
	$Card(v)$	26.9	18.7	12.6

Table 4.6: NSVM [34], LSVM [71], NLPSVM, CPLEX SVM [48] and Reduced [52] NSVM, LSVM, NLPSVM, CPLEX SVM: Training correctness, ten-fold testing correctness, ten-fold training times and cardinality of v ($Card(v)$) using a NONLINEAR classifier. Best results are in bold. Training and testing correctness and and cardinality of v are all averages over ten folds, while time is the total time over ten folds.

Chapter 5

Semi-Supervised Support Vector Machines for Unlabeled Data Classification

In semi-supervised learning, where only part of the two-class data is labeled, a SVM algorithm can be also utilized with the exception that the algorithm assigns the unlabeled data to one of two classes in such a way as to achieve separation by two bounding planes and maximizing the margin between the planes.

Bennett and Demiriz [3], who treat datasets which are already partially labeled, formulate the semi-supervised support vector machine (S^3VM) as a mixed integer program (MIP). Their formulation requires the introduction of a binary variable for each unlabeled data point in the training set. This makes the problem difficult to solve for large unlabeled data. State-of-the-art software does not handle easily problems with much more than 50 unlabeled data points. To overcome this difficulty we propose here a formulation that can handle large unlabeled datasets (with a thousand points) and solve the semi-supervised problem in a considerably shorter time. Our new approach consists of formulating the problem as a concave minimization problem which is solved by a successive linear approximation algorithm [61]. Such an approach has been successfully used on a number of machine learning, data mining and other problems [9, 12, 61, 62]. We term

our approach a concave semi-supervised support vector machine (VS^{3}VM).

For classifying unlabeled data, which is our principal aim here, we will make use of the k-median clustering algorithm [12] in combination with the proposed VS^{3}VM as follows. The k-median algorithm is used to select a small representative percentage, 5% to 10%, to be labeled by an expert or an oracle in order to be used as labeled data, together with the remaining part of the data, that remains unlabeled, in VS^{3}VM. Such an approach which can accommodate large datasets produces an improvement as high as 20.4% over a randomly chosen set labeled by an expert and used as a training set in a linear support vector machine. In addition, even if the **entire** dataset is labeled by an expert and classified by a linear support vector machine, our clustering concave minimization approach, using only 5% to 10% of the data as labeled data, can come within an average of 5.1%, in test set correctness, to an SVM trained on the entire dataset labeled by an expert.

When a clustering procedure is combined with VS^{3}VM, as described above, we term the resulting algorithm as clustered VS^{3}VM (CVS^{3}VM).

5.1 Concave Semi-supervised SVM (VS^{3}VM)

We consider here the dataset consisting of m labeled points and p unlabeled points all in R^n. The m labeled points are represented by the matrix $A \in R^{m\times n}$ and p unlabeled points in R^n represented by the matrix $B \in R^{p\times n}$. The labels for A are given by an $m \times m$ diagonal matrix D of ± 1. Bennett and Demiriz [3] formulate the semi-supervised linear support vector machine for generating the separating plane $x'w = \gamma$ for this problem as

follows:

$$\begin{array}{rcl} \min\limits_{w,\gamma,y,z,r,s} \nu e^{'} y + e^{'} z + \mu e^{'} \min\{r,s\} & & \\ \text{s.t.} \quad D(Aw - e\gamma) + y & \geq & e \\ -z \leq \quad w & \leq & z \\ Bw - e\gamma + r & \geq & e \\ -Bw + e\gamma + s & \geq & e \\ y \geq 0, r \geq 0, s \geq 0, & & \end{array} \tag{5.1}$$

where ν and μ are positive parameters. The first two terms of the objective function together with the first two constrains and $y \geq 0$, correspond to a linear SVM (5.6, below) [9, Equation (13)] which attempts to classify the labeled part of the dataset represented by the matrix A. The last term in the objective function together with the remaining constraints assign each row of the matrix B, representing unlabeled data, to class $+1$ or -1, whichever generates a lower misclassification error: $\min\{r, s\}$. The parameters μ, ν are positive numbers that weight the different terms of the objective function and are chosen as described in Section 5.4. Bennett and Demiriz [3] formulate this problem as a mixed integer program (MIP) by assigning a binary decision variable to each row of the unlabeled matrix B. However only relatively small unlabeled datasets (e.g. 50 points [3]) can be handled by this MIP formulation which sometimes fails due to excessive branching [3]. However, if some local search procedure is combined with the MIP formulation together with the clustering techniques proposed in this paper, the MIP approach can conceivably be considerably improved.

We propose here instead a concave minimization procedure, consisting of solving a finite number (typically 5 to 7) of linear programs, which terminate at a point satisfying

a necessary optimality for problem (5.1). The approach is based on the finite successive linear approximation algorithm for minimizing a concave function on a polyhedral set [62, Algorithm 1] and is justified by the fact that nonlinear term $\min\{r, s\}$ in the objective function of (5.1) is concave because it is the minimum of two linear functions. The algorithm consists of linearizing the nonlinear term $\min\{r, s\}$ around the current iterate (r^i, s^i) by taking a supporting plane (generalization of a tangent plane for non-differentiable concave functions) approximation of the function at that point and solving the resulting linear program. This leads to the following finitely terminating successive linear approximation algorithm based on [62, Algorithm 1].

Algorithm 5.1.1 VS^{3}VM *Successive Linear Approximation for S^3VM (5.1) Choose positive values for the parameters μ, ν. Start with a random $(r^0, s^0) \geq 0$. Having (r^i, s^i) determine $(w^{i+1}, \gamma^{i+1}, y^{i+1}, z^{i+1}, r^{i+1}, s^{i+1})$ by solving the linear program:*

$$\min_{w,\gamma,y,z,r,s} \nu e^{'} y + e^{'} z + \mu \partial(e^{'} \min\{r^i, s^i\}) \begin{bmatrix} r - r^i \\ s - s^i \end{bmatrix}$$

$$\begin{array}{rcl} s.t. \quad D(Aw - e\gamma) + y & \geq & e \\ -z \leq \quad w & \leq & z \\ Bw - e\gamma + r & \geq & e \\ -Bw + e\gamma + s & \geq & e \\ y \geq 0, r \geq 0, s \geq 0. & & \end{array} \tag{5.2}$$

where the supergradient $\partial(e^{'} \min\{r^i, s^i\})$ of $e' \min\{r, s\}$ is defined below in (5.4). Stop

when the following necessary optimality condition holds:

$$\begin{aligned} &\nu e^{'}(y^{i+1}-y^{i})+e^{'}(z^{i+1}-z^{i}) \\ &+\mu\partial(e^{'}\min\{r^{i},s^{i}\})^{\prime}\begin{bmatrix} r^{i+1}-r^{i} \\ s^{i+1}-s^{i} \end{bmatrix}=0. \end{aligned} \tag{5.3}$$

For a concave function $f : R^n \longrightarrow R$ the supergradient $\partial(f(x))$ of f at x is a vector in R^n satisfying:

$$f(y)-f(x)\leq \partial f(x)(y-x),$$

for all $y \in R^n$. The supergradient reduces to the ordinary gradient $\nabla f(x)$, when f is differentiable at x [85, 88]. The set of all supergradients at a point x is called the superdifferential.

In our case $e^{'}\min\{r,s\} : R^{2p} \longrightarrow R$ is a non-differentiable concave function and its supergradient is given by:

$$\partial(e^{'}\min\{r,s\})=\sum_{j=1}^{p}\begin{cases} \begin{pmatrix} I_j \\ 0_p \end{pmatrix} & \text{if} \quad r_j<s_j \\ (1-\lambda)\begin{pmatrix} I_j \\ 0_p \end{pmatrix}+\lambda\begin{pmatrix} 0_p \\ I_j \end{pmatrix} & \text{if} \quad r_j=s_j \\ \begin{pmatrix} 0_p \\ I_j \end{pmatrix} & \text{if} \quad r_j>s_j \end{cases} \tag{5.4}$$

Here $0_p \in R^p$ is a column vector of zeros, $I_j \in R^p$ is the jth column of the identity matrix I, and $\lambda \in (0, 1)$. In all our computations we set $\lambda = 0.5$.

By [62, Theorem 3] Algorithm 5.1.1 terminates after a finite number of linear programs at a point satisfying the necessary optimality condition (5.3) for problem (5.1).

Our numerical experiments showed that instead of a random starting point $(r^0, s^0) \geq 0$, a much better starting point for the Algorithm 2.1 can be obtained by solving the following linear program:

$$\begin{array}{rcl} \min\limits_{w,\gamma,y,z,r,s} \nu e'y + e'z + \dfrac{\mu}{2}(e'(r+s)) & & \\ \text{s.t.} \quad D(Aw - e\gamma) + y & \geq & e \\ -z \leq \quad w & \leq & z \\ Bw - e\gamma + r & \geq & e \\ -Bw + e\gamma + s & \geq & e \\ y \geq 0, r \geq 0, s \geq 0, & & \end{array} \tag{5.5}$$

which corresponds to the linear program (5.2) with a supergradient of $e' \min\{r, s\}$ evaluated at $r = s$.

We turn our attention now to a combination of the S^{3}VM with the k-median clustering algorithm that will enable us to handle unlabeled data.

5.2 Clustering + VS^{3}VM (CVS^{3}VM) for Unlabeled Data

To handle unlabeled data we make use of the k-median clustering algorithm [12], which de-emphasizes outliers, in order to form a small training set (5% to 10% of the data) by

choosing among the unlabeled data a "representative" subset to be labeled by an expert. This also constitutes a means for handling large unlabeled datasets such as those that occur in data mining in which case relatively few points can be labeled through expensive or time consuming services of an expert. The clustering approach can also be used as part of an incremental algorithm where only a small percentage of incoming data is chosen by the k-median algorithm to be labeled.

Our approach here will consist of the following: For a given percentage of the data, select a "good" subset to label and give the resulting labeled-unlabeled dataset to the VS^{3}VM Algorithm 5.1.1. We describe now the "selection" procedure of the above approach, which will be carried out by using the k-median clustering algorithm [12] described in the section below.

5.3 The k-median Clustering Algorithm

Consider a set of t data points in R^n represented by a general matrix $H \in R^{t\times n}$. We first find k cluster centers for the data such that the sum of distances between each point and the closest cluster center $C_l, l = 1, \ldots, k$ is minimized. The idea then is to treat points within a certain distance from these k cluster centers as representative points of that cluster, and hence of the overall dataset, and have them labeled by an expert. These points generate the matrix A of the semi-supervised Algorithm 5.1.1 S^{3}VM. The rest of the points remain unlabeled for use in S^{3}VM as the matrix B. In order to achieve this we use the simple and finite k-median clustering algorithm of [12] given below. When the k-median clustering algorithm is applied as described to select labeled data for the VS^{3}VM Algorithm 5.1.1, the algorithm is referred to as the **CVS^{3}VM Algorithm**.

Algorithm 5.3.1 k-Median Algorithm *Given* $C_1^j, \ldots, C_k^j$ *at iteration* j*, compute* $C_1^{j+1}, \ldots, C_k^{j+1}$ *by the following two steps:*

(a) Cluster Assignment: *For each* H_i'*,* $i = 1, \ldots t$*, determine* $\ell(i)$ *such that* $C_{\ell(i)}^j$ *is closest to* H_i' *in the one norm.*

(b) Cluster Center Update: *For* $\ell = 1, \ldots, k$ *choose* C_ℓ^{j+1} *as a median of all* H_i' *assigned to* C_ℓ^j*.*

Stop when $C_\ell^{j+1} = C_\ell^j$*.*

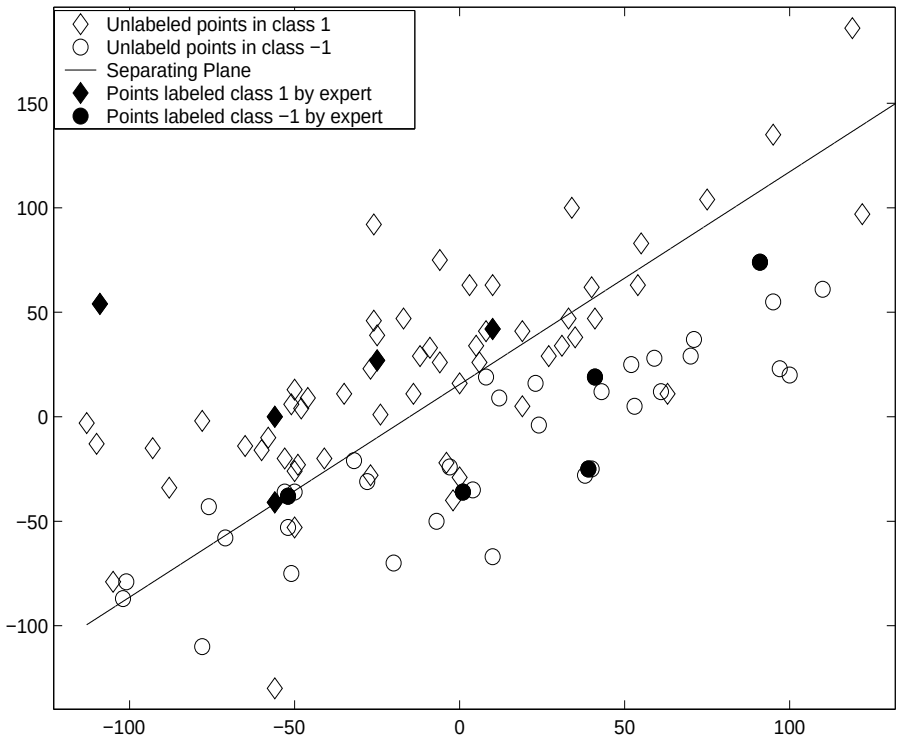

Figure 5.1: CVS^{3}VM for Unlabeled Data: Example showing 10% of a dataset, as solid points, whose labels, diamonds and circles, are unknown to the k-median algorithm which selects them to be labeled by an expert and then are used as labeled data in VS^{3}VM Algorithm 5.1.1. The remaining 90% points are used as unlabeled data by VS^{3}VM. **Resulting separating plane correctly classifies 81% of the data**

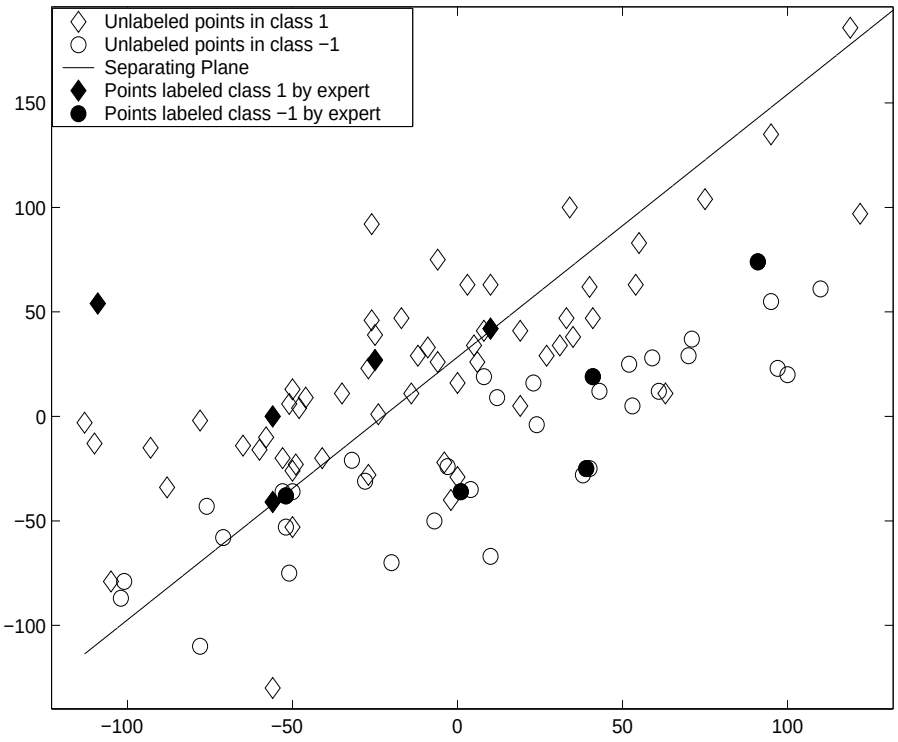

Figure 5.2: Cluster + SVM for Unlabeled Data: Example showing 10% of a dataset, as solid points, whose labels, diamonds and circles, are unknown to the k-median algorithm which selects them to be labeled by an expert and then are used as labeled data in a linear SVM (5.6). The resulting separating plane correctly classifies 72% of the data.

The choice of k in the k-median algorithm above depends on the size of the original dataset and is typically chosen so that a certain desired total percentage, say 5% to 10%, of the dataset falls within a desired distance from a closest cluster center.

To show that the clustered choice of data labeled by an expert in combination with a semi-supervised SVM is the most effective way for handling unlabeled data, we compared CVS^{3}VM with other plausible approaches as follows.

1. **Total Set + SVM: Total Set labeled by expert + Linear SVM**

   We solve here the linear SVM:

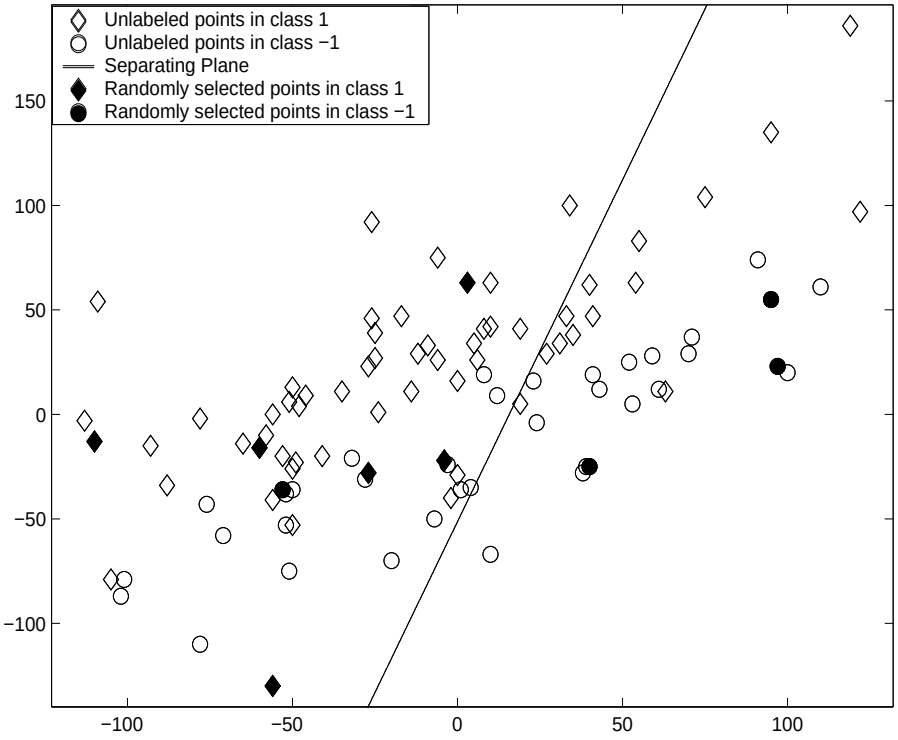

Figure 5.3: Random + SVM for Unlabeled Data: Example showing 10% of a dataset, as solid points, which are randomly selected and labeled by an expert, then used as training set in a linear SVM (5.6). Resulting separating plane correctly classifies 69% of the data.

$$\begin{array}{rcl} \min\limits_{w,\gamma,y,z} \nu e'y + e'z & & \\ \text{s.t.} \quad D(Aw - e\gamma) + y & \geq & e \\ -z \leq \qquad w \qquad & \leq & z \\ y \geq 0. & & \end{array} \tag{5.6}$$

Thus training is done here on a completely labeled data set which is equivalent to (5.1) with an empty B. In contrast, although CS^{3}VM is trained on just a 5% to 10% of the data that is labeled by an expert, its test correctness is close to that obtained by a linear SVM using **all** the dataset labeled by an expert as shown by our numerical tests.

2. **Random + SVM: Random choice for labeling by expert + Linear SVM**

   Here, 5%-10% of the data to be labeled is chosen randomly and used as training data in the linear SVM algorithm. The information on the remaining unlabeled data is not considered since we are applying a supervised learning approach to the labeled data only. Because of the random choice of the training data, we performed this experiment 10 times in order to obtain a more consistent result. As was expected, this approach gave the worst performance.

3. **Clustering + SVM: Clustering choice of data + Linear SVM**

   In this case no information on the unlabeled data is used. However, since the k-median clustering algorithm is used to choose the data to be labeled, a "smarter" choice of the data to be labeled is made. An improvement on test set correctness over Random + SVM is obtained.

4. **CVS^{3}VM: Clustered choice of the data + VS^{3}VM**

   This is the principal proposed algorithm of this paper. Pick a small percentage of the unlabeled data by clustering to be labeled, then use this labeled data with the the remaining unlabeled data in the Concave Semi-supervised SVM (VS^{3}VM).

5. **Cluster + S^{3}VM: Clustering choice of data + S^{3}VM (MIP)**

   This case is similar to CVS^{3}VM except that instead of solving a concave minimization problem we solve here a Mixed Integer problem (MIP) as proposed in [3].

We now illustrate how CVS^{3}VM works on a simple 2-dimensional example of 100 data points consisting of diamond and circular shapes depicted in Figure 5.1, created by the

NDC (normally distributed clusters) generator [78]. The labels (diamond and circular shapes) are made known to VS^{3}VM as follows. In Figure 5.1 the solid shapes are the 10% of the original unlabeled dataset that are selected by the k-median algorithm to be labeled data and given to VS^{3}VM while the remaining 90% of the original data shown as hollow shapes are treated as unlabeled data by VS^{3}VM. The resulting separating plane shown in Figure 5.1 correctly classifies 81% of the data. If we now drop the unlabeled data from the training part of the problem and revert to a linear SVM (5.6) trained on data chosen by a k-median algorithm, we obtain the separating plane shown in Figure 5.2 which correctly classifies a lower percentage of the data: 72%. Finally, if we use the SVM (5.6) on a randomly chosen set of points that are labeled and depicted as solid points, we obtain the separating plane shown in Figure 5.3 with a still lower correctness of 69%.

5.4 Numerical Testing

Our numerical testing was carried out on five publicly available labeled datasets. Unlabeled data was simulated by dropping labels from some or all the points in a given dataset. Four of the datasets are from the UCI Machine Learning Repository [77], and one of them was created by the NDC (normally distributed clusters) generator [78]. Table 5.4 shows the number of points of each dataset and the dimensionality of the space they are in. All the matrix manipulations were carried out using MATLAB [75]. Linear programs were solved by calling the state-of-the-art CPLEX solver [21] from MATLAB and GAMS [15].

Test 5.4.1 *The following five experiments were performed:*

(i) **Total Set Linear SVM**

Linear SVM (5.6) trained on a completely labeled dataset.

(ii) **Random + Linear SVM** *A linear SVM for which a random 5% to 10% subset of the data is selected as the training set to be labeled by an expert.*

(iii) **Cluster + Linear SVM** *A linear SVM is used together with the k-median Clustering Algorithm 5.3.1 for selection of a 5%-10% subset of data points to be labeled and used as a training set.*

(vi) **CVS^{3}VM** *Algorithm VS^{3}VM 5.1.1 with k-median Clustering Algorithm 5.3.1 for selection of a 5%-10% subset of data points to be labeled by an expert with the rest of the data remaining unlabeled in Algorithm VS^{3}VM 5.1.1.*

(v) **Cluster + S^{3}VM (MIP)** *Use Algorithm S^{3}VM [3] (MIP formulation) with k-median Clustering Algorithm 5.3.1 for selection of a 5%-10% subset of data points to be labeled by an expert with the rest of the data remaining unlabeled.*

When the k-median algorithm was used in order to choose the 5% (10%) subset of the data, the value for k was approximately set to 5% (10%) of the total number of points in the whole dataset. For example for the Ionosphere dataset of 351 points, we chose 10% of the data to be labeled, thus $k = 35$. The labeled training set was chosen as the set of points within a certain distance from the cluster centers of the k-median algorithm so as to total to 10% of the original unlabeled dataset. For CVS^{3}VM, we performed a variation of the standard *tenfold cross-validation*. Once we obtained our 10% labeled data to be used as training set, we divided the remaining 90% of data points into 10 folds, so that each fold contained 9% of the original dataset. We then used nine of these

10 folds (81% of the original points) as unlabeled training data and the remaining 9% as a testing set. We repeated this procedure ten times choosing a different fold for testing and the remaining folds as a unlabeled training data every time. The 10% of the labeled data was fixed for all the ten problems corresponding to each fold. See Figure 5.4 for a graphical depiction of this procedure. When comparison with a random choice of labeled data was made, we repeated the latter process ten times and reported the average.

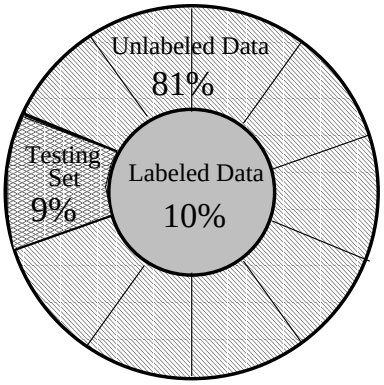

Figure 5.4: CVS^{3}VM tenfold Cross Validation

The parameter ν appearing in the formulation of the S^{3}VM problem (5.1) is determined by a formula proposed in [3]: $\nu = \frac{10^3}{m+p}$, where m is the number of the labeled points, and p the number of the remaining unlabeled points. The parameter μ in (5.1) was adjusted by tuning. When the SVM was used on the complete dataset as labeled data, the parameter ν was also adjusted by tuning.

Table 5.4 gives tenfold testing set correctness for 5 totally unlabeled datasets for the five algorithms described in Test 5.4.1: Total Set + Linear SVM, Random + Linear SVM, Cluster + Linear SVM, CVS^{3}VM and Cluster + S^{3}VM (MIP). The improvement and the p-values shown in the table are relative to the Random + Linear SVM approach.

CVS^{3}VM had the highest test set correctness and the relative improvement, over Random + Linear SVM, was as high as as 20.4% with a p-value of 0.02. Also, CVS^{3}M

test set correctness using as little as 5% to 10% of the data as labeled data, was on average within 5.1% of that for a linear SVM using **all** the data as a labeled training dataset.

When S^3VM-MIP terminated before reaching the 10,000-seconds time limit, it was much slower than CVS^3M. For example, in the Heart dataset the total time spent by S^3VM-MIP on tenfold cross validation was 462.2 seconds, while the time used by VS^3VM was 14.9 seconds in total. The corresponding times for the Housing dataset were 1193.2 seconds for S^3VM-MIP and 32.9 for VS^3VM.

Data Set points×dim.	Total Set SVM Test	R + SVM Test	C + SVM Test Improvement* p-value*	CVS^{3}VM Test Improvement* p-value*	C + S^{3}VM(MIP) Test Improvement* p-value*
NDC Set 1000×32	72.6%	58.0%	60.0% 3.4% 0.38	**67.0%** 15.5% 0.01	Failed† - -
Heart 297×13	83.2%	69.0%	73.3% 6.2% 0.01	**76.0%** 10.1% 0.01	**76.0%** 10.1% 0.01
Housing 506×13	86.2%	70.0%	73.3% 4.7% 0.18	**81.4%** 16.2% 0.05	81.0% 15.7% 0.08
Ionosphere 351×34	87.1%	78.3%	78.5% 0.25% 0.93	**84.2%** 7.5% 0.05	Failed† - -
Sonar 208×60	77.4%	64.0%	74.6% 16.6% 0.04	**77.1%** 20.4% 0.02	Failed† - -

Table 5.1: Tenfold test set correctness of the experiments described in Test 5.4.1 on 5 public datasets. **Bold figures denote highest test set correctness.**

(i) **Total Set SVM :** Entire dataset labeled by an expert and used by a linear SVM (5.6).

(ii) **R + SVM:** Small randomly chosen subset labeled by an expert and used by a linear SVM (5.6).

(iii) **C + SVM:** Small subset of the data chosen by clustering, labeled by an expert and used by a linear SVM (5.6).

(iv) **CVS^{3}VM:** Small subset of the data chosen by clustering, labeled by an expert and VS^{3}VM (5.1) solved the concave minimization algorithm, Algorithm 5.1.1.

(v) **C + S^{3}VM:** Same as 4 except a Mixed Integer Program (MIP) is used instead of the concave minimization algorithm, Algorithm 5.1.1.

* The relative improvement and the the p-value are calculated with respect to **Random + SVM**.

† Failure was declared when total time exceeded 10,000 seconds.

Chapter 6

Finite Newton Method for Lagrangian Support Vector Machine Classification

In this chapter we establish a finitely terminating Newton method for the unconstrained minimization of a strongly convex, piecewise quadratic function, which underlies a linear or nonlinear kernel classifier [19, 22, 65, 92, 103]. Our piecewise quadratic function is the implicit Lagrangian, first proposed in [74] and utilized in [71] for a highly effective iterative scheme, the Lagrangian Support Vector Machine (LSVM). In order to handle problems with very large dimensional input spaces, we use here instead a fast finite Newton method for finding the unconstrained unique global minimum solution of the implicit Lagrangian associated with the classification problem. The solution is obtained by solving a system of linear equations, a finite number of times.

6.1 Strongly Convex Quadratic Programming SVM Formulation

In many essentially equivalent formulations of the classification problem [33, 35, 52, 53], the square of 2-norm of the slack variable y is minimized with weight $\frac{\nu}{2}$ instead of the 1-norm of y as in (3.1). In addition the distance between the planes (2.2) is measured in the $(n+1)$-dimensional space of $(w,\gamma) \in R^{n+1}$, that is $\frac{2}{\|(w,\gamma)\|}$. Measuring the margin in this $(n+1)$-dimensional space instead of R^n induces strong convexity and has little or no effect in general on the problem as was shown in [68] experimentally. Thus using twice the reciprocal squared of the margin instead, yields our modified SVM problem as follows:

$$\begin{array}{rll} \min\limits_{(w,\gamma,y)\in R^{n+1+m}} & \frac{\nu}{2}y'y + \frac{1}{2}(w'w+\gamma^2) & \\ \text{s.t.} \qquad D(Aw-e\gamma)+y & \geq & e \\ y & \geq & 0. \end{array} \tag{6.1}$$

It has been shown computationally [71] that this reformulation (6.1) of the conventional support vector machine formulation (2.1) often yields similar results to (2.1). The dual of this problem is [59]:

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

The variables (w,γ) of the primal problem which determine the separating surface (2.4) are recovered directly from the solution of the dual (6.2) above by the relations:

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

We immediately note that the matrix appearing in the dual objective function is positive definite. We simplify the formulation of the dual problem (6.2) by defining two matrices as follows:

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

With these definitions the dual problem (6.2) becomes

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

To develop the nonlinear kernel classifier we use again the notation described in the introduction. As described before in previous chapters, the linear separating surface (2.4) is replaced by the nonlinear surface:

$$K(x', A')Du = \gamma, \tag{6.6}$$

where u is the solution of the dual problem (6.2) with the linear kernel AA' replaced by the nonlinear kernel product $K(A, A')K(A, A')'$ [65, Equation (8.10)], that is:

$$\min_{0 \leq u \in R^m} \frac{1}{2}u'(\frac{I}{\nu} + D(K(A, A')K(A, A')' + ee')D)u - e'u. \tag{6.7}$$

This leads to a redefinition of the matrix Q of (6.5) as follows:

$$H = D[K(A, A') \quad -e], \qquad Q = \frac{I}{\nu} + HH'. \tag{6.8}$$

Note that the nonlinear separating surface (6.6) degenerates to the linear one (2.4) if we let $K(A, A') = AA'$ and make use of (6.3).

We describe now a general framework for generating a fast and effective algorithm for solving the quadratic program (6.5) by solving a system of linear equations a finite number of times.

6.2 Implicit Lagrangian Formulation

The implicit Lagrangian formulation [71, Equation (17)], [92, Section 10.6.2] consists of replacing the nonnegativity constrained quadratic minimization problem (6.5) by the equivalent *unconstrained* piecewise quadratic minimization of the implicit Lagrangian $L(u)$:

$$\min_{u \in R^m} L(u) = \min_{u \in R^m} \frac{1}{2}u'Qu - e'u + \frac{1}{2\alpha}(\|(-\alpha u + Qu - e)_+\|^2 - \|Qu - e\|^2), \tag{6.9}$$

where α is a sufficiently large but finite positive parameter, and the plus function $(\cdot)_+$, defined in the Introduction, replaces negative components of a vector by zeros. Reformulation of the constrained problem (6.5) as an unconstrained problem (6.9) is based on ideas [74] of converting the optimality conditions of (6.5) to an unconstrained minimization problem as follows. Because the Lagrange multipliers of the constraints $u \geq 0$ of (6.5) turn out to be components of the gradient $Qu - e$ of the objective function, these components of the gradient can be used as Lagrange multipliers in an Augmented Lagrangian [6, 89] formulation of (6.5) which leads precisely to the unconstrained formulation (6.9). Our finite Newton method consists of applying Newton's method to this unconstrained minimization problem and showing that it terminates in a finite number of steps at the global minimum. The gradient of $L(u)$ is:

$$\begin{aligned} \nabla L(u) &= (Qu - e) + \tfrac{1}{\alpha}(Q - \alpha I)((Q - \alpha I)u - e)_+ - \tfrac{1}{\alpha}Q(Qu - e) \\ &= \tfrac{(\alpha I - Q)}{\alpha}((Qu - e) - ((Q - \alpha I)u - e)_+). \end{aligned} \tag{6.10}$$

In order to apply the Newton method we need the $m \times m$ Hessian matrix of second partial derivatives of $L(u)$, which does not exist in the ordinary sense because its gradient, $\nabla L(u)$, is not differentiable. However, a generalized Hessian of $L(u)$ in the sense of [28, 43, 66]

exists and is defined as the following $m \times m$ matrix:

$$\partial^2 L(u) = \frac{(\alpha I - Q)}{\alpha}(Q + diag((Q - \alpha I)u - e)_*(\alpha I - Q)), \tag{6.11}$$

where, as defined in the Introduction, $diag(\cdot)_*$ denotes a diagonal matrix and $(\cdot)_*$ denotes the step function. Our basic Newton step consists of solving the system of m linear equations:

$$\nabla L(u^i) + \partial^2 L(u^i)(u^{i+1} - u^i) = 0, \tag{6.12}$$

for the unknown $m \times 1$ vector u^{i+1} given a current iterate u^i. We will show in the next section that this iteration coupled with a stepsize, terminates at the global minimum solution to the problem. For that we need the positive definiteness of $\partial^2 L(u)$, which we establish now under the very simple condition that Q is positive definite and that $\alpha > ||Q||$ as follows.

Proposition 6.2.1 Positive Definiteness of the Generalized Hessian *Let Q be an arbitrary $m \times m$ symmetric positive definite matrix and let $\alpha > ||Q||$. Then:*

(i) The generalized Hessian matrix $\partial^2 L(u)$, defined by (6.11), is positive definite.

(ii) The generally non-symmetric matrix factor $P(u)$ of $\partial^2 L(u) = \frac{(\alpha I - Q)}{\alpha} P(u)$:

$$P(u) := Q + diag((Q - \alpha I)u - e)_*(\alpha I - Q), \tag{6.13}$$

is positive definite.

Proof

(i) Since $\alpha > ||Q||$, it follows that $\alpha I - Q$ is positive definite. Hence the product $(\alpha I - Q)Q$ of the two positive definite matrices $\alpha I - Q$ and Q is positive definite

[82, Theorem 6.2.1]. Define the diagonal matrix:

$$E(u) := diag((Q - \alpha I)u - e)_*. \tag{6.14}$$

Since, by the definition of the step function $(\cdot)_*$, each element of $E(u)$ is in the interval $[0, 1]$, it follows that:

$$(\alpha I - Q)E(u)(\alpha I - Q) = (E(u)^{\frac{1}{2}}(\alpha I - Q))^2 \tag{6.15}$$

is positive semidefinite. Hence the generalized Hessian:

$$\partial^2 L(u) = \frac{(\alpha I - Q)}{\alpha} Q + \frac{(\alpha I - Q)}{\alpha} E(u)(\alpha I - Q), \tag{6.16}$$

the sum of a positive definite and a positive semidefinite matrix is positive definite.

(ii) Since:

$$P(u) = \alpha(\alpha I - Q)^{-1}\partial^2 L(u), \tag{6.17}$$

is the product of two positive definite matrices, it follows again by [82, Theorem 6.2.1] that $P(u)$ is positive definite. □

We note now that, since both $\nabla L(u)$ and $\partial^2 L(u)$ contain the multiplicative factor $\frac{(\alpha I - Q)}{\alpha}$ which is positive definite, it follows that the Newton iteration (6.12) can be simplified to:

$$h(u^i) + \partial h(u^i)(u^{i+1} - u^i) = 0, \tag{6.18}$$

where

$$h(u) := (Qu - e) - ((Q - \alpha I)u - e)_+ = (\frac{\alpha I - Q}{\alpha})^{-1}\nabla L(u), \tag{6.19}$$

and

$$\partial h(u) := Q + E(u)(\alpha I - Q) = P(u) = (\frac{\alpha I - Q}{\alpha})^{-1}\partial^2 L(u). \tag{6.20}$$

The simpler iteration (6.18) will be used in our implementation instead of the equivalent iteration (6.12).

Another useful tool in our implementation will be the Sherman-Morrison-Woodbury identity [40] when we are classifying large datasets with a *linear* classifier. For such problems we have, with Q defined by (6.4) and $E(u)$ replaced by E for notational simplicity:

$$\begin{aligned} \partial h(u) &= \alpha E + (I - E)Q = \alpha E + \tfrac{I-E}{\nu} + (I - E)HH' \\ &= F + (I - E)HH' = F(I + F^{-1}(I - E)HH') = F(I + SHH'), \end{aligned} \tag{6.21}$$

where F and S are defined as the following positive and nonnegative diagonal matrices respectively:

$$F := \alpha E + \frac{I - E}{\nu}, \;\; S := F^{-1}(I - E). \tag{6.22}$$

By using a special case of the Sherman-Morrison-Woodbury identity [40]:

$$(I + KH')^{-1} = I - K(I + H'K)^{-1}H', \tag{6.23}$$

on the last expression of (6.21), with $K := SH$, we have:

$$\partial h(u)^{-1} = (I + SHH')^{-1}F^{-1} = (I - SH(I + H'SH)^{-1}H')F^{-1}, \tag{6.24}$$

where we need to invert the $(n+1) \times (n+1)$ matrix $(I + H'SH)$ instead of the potentially much larger $m \times m$ matrix $(I + SHH')$. This will be the case whenever we generate a linear classifier for problems with $m >> n$.

We turn now to details of the Newton algorithm and its finite termination properties.

6.3 Finite Newton Classification Method

We first state our Newton algorithm for solving the piecewise quadratic minimization problem (6.9) for an arbitrary positive definite Q using the simplified iteration (6.18)

together with an Armijo stepsize [2, 53] defined below in order to guarantee finite termination from any starting point.

Algorithm 6.3.1 Newton Algorithm for (6.9) *Let $h(u)$ and $\partial h(u)$ be defined by (6.19) and (6.20). Start with any $u^0 \in R^m$. For $i = 0, 1, \ldots$:*

(i) Stop if $h(u^i - \partial h(u^i)^{-1} h(u^i)) = 0$.

(ii) $u^{i+1} = u^i - \lambda_i \partial h(u^i)^{-1} h(u^i) = u^i + \lambda_i d^i$,

where $\lambda_i = \max\{1, \frac{1}{2}, \frac{1}{4}, \ldots\}$ is the Armijo stepsize such that:

$$L(u^i) - L(u^i + \lambda_i d^i) \geq -\delta \lambda_i \nabla L(u^i)' d^i, \tag{6.25}$$

for some $\delta \in (0, \frac{1}{2})$, and d^i is the Newton direction:

$$d^i = -\partial h(u^i)^{-1} h(u^i), \tag{6.26}$$

obtained by solving (6.18).

(iii) $i = i + 1$. Go to (i).

We state and prove now our finite termination result for this Newton algorithm. A possible intuitive justification of the finite termination is that as the iterates converge to the unique global solution, the correct quadratic surfaces on whose intersection the solution lies are correctly identified, and hence a single Newton step captures that solution.

Theorem 6.3.2 Finite Termination of Newton Algorithm *For the symmetric positive definite matrix Q defined by (6.4) or (6.8), the sequence $\{u^i\}$ of Algorithm 6.3.1 terminates at the global minimum solution $\bar{u}$ of (6.9) and hence that of (6.5) provided $\alpha > \|Q\|$.*

Proof That the sequence $\{u^i\}$ converges to the global solution $\bar{u}$, for which $h(\bar{u}) = 0$, follows from standard results (e.g.[60, Theorem 2.1, Example 2.1(ii), Example 2.4(iv)]) of unconstrained minimization of a strongly convex function using a Newton direction with an Armijo stepsize. This is exactly what is done in our Algorithm 6.3.1 above. We now establish finite termination of the sequence $\{u^i\}$ at $\bar{u}$. Our Newton iteration (6.18) can be written as:

$$(Qu^i - e) - ((Q - \alpha I)u^i - e)_+ + (Q + E(u^i)(\alpha I - Q))(u^{i+1} - u^i) = 0, \tag{6.27}$$

which we rewrite by subtracting from it the equality $h(\bar{u}) = 0$ satisfied by the solution $\bar{u}$:

$$(Q\bar{u} - e) - ((Q - \alpha I)\bar{u} - e)_+ = h(\bar{u}) = 0. \tag{6.28}$$

This results in the equivalent iteration:

$$\begin{array}{c} ((Q - \alpha I)\bar{u} - e)_+ - ((Q - \alpha I)u^i - e)_+ \\ -E(u^i)(Q - \alpha I)(u^{i+1} - u^i) + Q(u^{i+1} - \bar{u}) = 0. \end{array} \tag{6.29}$$

We show now that this Newton iteration is satisfied uniquely (since $\partial h(u^i)$ is nonsingular) by $u^{i+1} = \bar{u}$ when u^i is sufficiently close to $\bar{u}$ and hence the Newton iteration terminates at $\bar{u}$ at step (i) of Algorithm 6.3.1. Setting $u^{i+1} = \bar{u}$ in (6.29) and canceling the last term $Q(\bar{u} - \bar{u}) = 0$, gives:

$$((Q - \alpha I)\bar{u} - e)_+ - ((Q - \alpha I)u^i - e)_+ - E(u^i)(Q - \alpha I)(\bar{u} - u^i) = 0. \tag{6.30}$$

We verify now that that this equation is indeed satisfied when u^i is sufficiently close to $\bar{u}$ by looking at each component j, $j = 1, \ldots, m$ of the equation (6.30). We consider the the following nine possible combinations determined by the vector function $r(u)$ appearing in (6.30) defined as:

$$r_j(u) := ((Q - \alpha I)u - e)_j, \quad j = 1, \ldots, m. \tag{6.31}$$

Noting that every element of the diagonal matrix $E(u)$ defined by (6.14), that is $E_{jj}(u) = (r_j(u))_*$, $j = 1, \ldots, m$, is in the interval $[0, 1]$, we have:

(i) $r_j(\bar{u}) > 0,\ r_j(u^i) > 0$:

$$(Q_j - \alpha I_j)\bar{u} - 1 - (Q_j - \alpha I_j)u^i + 1 - 1 \cdot (Q_j - \alpha I_j)(\bar{u} - u^i) = 0$$

(ii) $r_j(\bar{u}) > 0,\ r_j(u^i) = 0$: Cannot occur when u^i is sufficiently close to $\bar{u}$.

(iii) $r_j(\bar{u}) > 0,\ r_j(u^i) < 0$: Cannot occur when u^i is sufficiently close to $\bar{u}$.

(iv) $r_j(\bar{u}) = 0,\ r_j(u^i) > 0$:

$$0 - (Q_j - \alpha I_j)u^i + 1 - 1 \cdot (Q_j - \alpha I_j)(\bar{u} - u^i) = 1 - Q_j\bar{u} + \alpha\bar{u}_j = -r_j(\bar{u}) = 0$$

(v) $r_j(\bar{u}) = 0,\ r_j(u^i) = 0$:

$$0 + 0 - [0, 1] \cdot ((Q_j - \alpha I_j)(\bar{u} - u^i) - e + e) = [0, 1] \cdot (r_j(\bar{u}) - r_j(u^i)) = 0$$

(vi) $r_j(\bar{u}) = 0,\ r_j(u^i) < 0$:

$$0 + 0 - 0 \cdot (Q_j - \alpha I_j)(\bar{u} - u^i) = 0$$

(vii) $r_j(\bar{u}) < 0,\ r_j(u^i) > 0$: Cannot occur when u^i is sufficiently close to $\bar{u}$.

(viii) $r_j(\bar{u}) < 0,\ r_j(u^i) = 0$: Cannot occur when u^i is sufficiently close to $\bar{u}$.

(ix) $r_j(\bar{u}) < 0,\ r_j(u^i) < 0$:

$$0 + 0 - 0 \cdot (Q_j - \alpha I_j)(\bar{u} - u^i) = 0$$

Consequently for u^i is sufficiently close to $\bar{u}$, the Newton iteration is uniquely satisfied by $\bar{u}$, and hence, terminates at $\bar{u}$. $\square$

We turn now to our computational results.

6.4 Numerical Experience

Because of the simplicity of our algorithm, we give below a simple MATLAB implementation of the algorithm without the Armijo stepsize, which does not seem to be needed in most applications. Although this is merely an empirical observation in the present case, it considerably simplifies our MATLAB Code 6.4.1. However, it has also been shown [66, Theorem 3.6] that under a well conditioned assumption, not generally satisfied here, the proposed Newton method indeed terminates in a finite number of steps without an Armijo stepsize.

We further note that the provided MATLAB code 6.4.1 not only works for a linear classifier, but also for a nonlinear classifier as well [65, Equations (1), (10)]. In the nonlinear case, the matrix $K(A, A')$ is used as input instead of A, and the pair $(\hat{u}, \gamma)$, where $\hat{u} = K(A, A')Du$, is returned instead of (w, γ). The nonlinear separating surface is then given by (6.6) as:

$$K(x, A')\hat{u} - \gamma = 0. \tag{6.32}$$

Our numerical testing and comparisons were carried out on the high dimensional Multiple Myeloma dataset available at:

$$http://lambertlab.uams.edu/publicdata.htm,$$

and processed by David Page and his colleagues [83]. Further tests and comparisons were also carried out on six moderately dimensioned, publicly available datasets [77, 79].

We describe our tests and comparisons now.

Code 6.4.1 NSVM: Finite Newton LSVM Code

```
function [w,gamma]=nsvm(A,d,nu);
% NSVM:linear and nonlinear classification without Armijo
% INPUT: A, D, nu. OUTPUT: w, gamma
% [w, gamma] = nsvm(A,d,nu);
[m,n]=size(A);iter=0;
u=zeros(m,1);e=ones(m,1);
H=[diag(d)*A -d]; Q=speye(m)/nu+H*H';
alpha=1.1*((1/nu)+(norm(H',2)^2));
hu=-max(((Q*u-e)-alpha*u),0)+Q*u-e;
while norm(hu)>10^(-5)
    iter=iter+1
    star=sign(max(((Q-alpha*eye(m))*u-e),0));
    dhu=sparse((eye(m)-diag(star))*Q+alpha*diag(star));
    delta=dhu\hu;
    unew=u-delta;
    u=unew;
    hu=-max(((Q*u-e)-alpha*u),0)+Q*u-e;
end

 w=A'*(d.*u);gamma=-sum(d.*u);

return
```

6.4.1 Multiple Myeloma Dataset

In this section we performed experiments using the multiple Myeloma dataset that is described in detail in Section 4.3.3.

As described in 4.3.3 this dataset is transformed in a dataset with 105 points in R^{28032}. This makes this dataset very interesting for our method, since a main objective of this paper is to show that our proposed algorithm can quickly classify points in very high dimensional spaces.

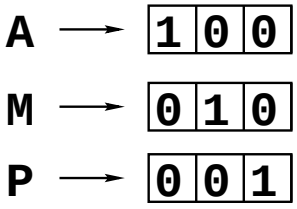

Figure 6.1: Real-valued representation of the AC features set $\{A, M, P\}$.

Numerical Comparisons

Performance of our Newton SVM (**NSVM**) algorithm on the Myeloma dataset, in terms of speed and generalization ability, is first compared with two publicly available SVM solvers: LSVM [71] and SVM^{light} version 5.0 [50]. The comparison with LSVM was carried out because both LSVM and NSVM minimize the same unconstrained differentiable convex implicit Lagrangian function (6.9) but using entirely different techniques. Reported times for LSVM here differ from the ones reported in [71] because the calculation time for the matrix H of (6.4) is considered as input time in [71], whereas here it is counted as part of the computational time. The other algorithm included in our comparisons, SVM^{light}, solves a different optimization problem with a classification error

measured using the 1-norm instead of the 2-norm. This solver was included in our experiments because it is widely cited in the literature and is often used as a benchmark for SVM classification algorithms. Termination criteria for all methods was set to 0.001 which is the default for SVM^{light}. We outline some of the results of our comparative testing.

- Both NSVM and SVM^{light} obtained 100% *leave-one-out correctness (looc)*. However, we note that SVM^{light} did not perform as well with the default value of its parameter C which determines the trade-off between empirical and generalization errors. In order to find an optimal value for both ν (NSVM and LSVM) and C (SVM^{light}) the following tuning procedure was employed on each fold:
  - A random tuning set of the the size of 10% of the training data was chosen and separated from the training set.
  - Several SVMs were trained on the remaining 90% of the training data using values for C or ν equal to 2^i where $i = -12, \ldots, 0, \ldots, 12$.
  - The value of C or ν that gave the best SVM correctness on the tuning set was chosen.
  - A final SVM was trained using the chosen value of C and ν and all the training data. The resulting SVM was tested on the testing data.
- The average cpu time required by our algorithm for the leave-one-out correctness (*looc*) computations was 4.11 seconds per case and total time for all cases was 432.40 seconds. This outperformed the SVM^{light} cpu times of 27.83 seconds average per case and total *looc* time of 2922.15 seconds.
- LSVM failed and reported an out of memory error.

These results are summarized in Table 6.4.1 below.

Data Set $m \times n$ ($points \times dimensions$)	NSVM looc Time (Sec.)	SVM^{light} looc Time (Sec.)	LSVM looc Time (Sec.)
Myeloma 105×28032	**100.0%** **432.40**	**100.0%** 2922.15	oom oom

Table 6.1: NSVM, SVM^{light} & LSVM: *leave-one-out correctness (looc)* and total running times using a linear classifier for the Myeloma dataset. Best results are in bold. oom stands for "out of memory".

6.4.2 Six Publicly Available Datasets

Even though our algorithm is primarily intended for datasets with very high dimensional input space and a moderate number of points, it also performed very well on more conventional datasets where the opposite is true. To exhibit this fact we tested our algorithm on six publicly available datasets. Five from the UCI Machine Learning Repository [77]: Ionosphere, Cleveland Heart, Pima Indians, BUPA Liver and Housing. The sixth dataset is the Galaxy Dim dataset available at [79]. The dimensionality and size of each dataset is given in Table 6.4.3.

6.4.3 Numerical Comparisons Using a Linear Classifier

In this set of experiments we used a linear classifier to compare our method NSVM with LSVM and SVM^{light} on the six datasets mentioned above. Because $m >> n$ for these datasets, it was preferable to use (6.24) in solving the Newton iteration (6.18) and inverting an $(n+1) \times (n+1)$ matrix instead of an $m \times m$ matrix. The complexity of the

original NSVM for a linear kernel is $O(km^3)+O(m^2n)$, where k is the number of iterations of NSVM and the term $O(m^2n)$ reflects the time for computing the matrix Q of of (6.4). By using the Sherman-Morrison-Woodbury formula, the complexity changes to $O(kn^3)+O(m^2n)$, which is obviously preferable when $m >> n$. However, the explicit calculation of the matrix Q of equation (6.4) which is of the order $O(m^2n)$ is very time consuming when m is large. Hence, instead of explicitly calculating Q and then performing the matrix-vector product Qu, needed in computing $h(u)$, we calculate:

$$Qu = (\frac{I}{\nu} + HH')u = \frac{1}{\nu}u + H(H'u), \tag{6.33}$$

where $H'u$ is calculated first then $H(H'u)$ is calculated next. This simple but effective algebraic manipulation changes the complexity of the algorithm to be linear in m that is, $O(kn^3) + O(mn)$. This makes our algorithm very fast even when $m >> n$ but n is relatively small.

The values for the parameters C and ν were again calculated using the same tuning procedure explained in section 6.4.1

As shown in Table 6.4.3, the correctness of the three methods was very similar, but the execution time including ten-fold cross validation for NSVM was less for all the datasets tested.

6.4.4 Numerical Comparisons Using a Nonlinear Classifier

In order to show that our algorithm can also be used to find nonlinear classifiers, we chose three datasets from the UCI Machine Learning Repository for which it is known that a nonlinear classifier performs better that a linear classifier. We used NSVM, LSVM and SVM^{light} in order to find a nonlinear classifier based on the Gaussian kernel 1.1. The

Data Set $m \times n$ (*points* × *dimensions*)	NSVM Train Test Time (Sec.)	SVM^{light} Train Test Time (Sec.)	LSVM Train Test Time (Sec.)
Ionosphere 351 × 34	**93.2%** **89.8%** **0.95**	92.0 % 88.3 % 2.3	**93.2%** **89.8%** 1.49
BUPA Liver 345 × 6	**70.3%** **70.2%** **0.19**	70.1% 69.3% 5.17	**70.3%** **70.2%** 0.61
Pima Indians 768 × 8	**77.7%** 77.0% **0.48**	77.4% **77.1%** 3.87	**77.7%** **77.0%** 2.04
Cleveland Heart 297 × 13	**87.3%** **86.3%** **0.31**	87.1% 85.9% 0.88	**87.3%** **86.3%** 0.83
Housing 506 × 13	87.2% **86.6%** **0.58**	**87.6%** 85.8% 5.57	87.2% **86.6%** 1.53
Galaxy Dim 4192 × 14	**95.0%** **95.3%** **7.16**	91.3% 91.2% 15.94	**95.0%** **95.3%** 76.67

Table 6.2: NSVM, SVM^{light} & LSVM: Training correctness, ten-fold testing correctness and ten-fold training times using a LINEAR classifier. NSVM and LSVM parameter ν and SVM^{light} parameter C, all chosen by tuning. Best results are in bold.

value of μ in the Gaussian kernel and the value of ν in NSVM and LSVM and C in SVM^{light} were chosen all by tuning from the set of values 2^i with i an integer ranging from -12 to 12 following the same procedure described in section 6.4.1. Because the nonlinear kernel matrix is square and since both NSVM and LSVM perform better on rectangular matrices, we also used a rectangular kernel formulation as described in the Reduced SVM (RSVM) [52]. This resulted in as good or better correctness and much faster running times. The size of the random sample used to calculate the rectangular kernel was 10% of the size of the original dataset in all cases. We refer to these variations

of NSVM and LSVM as Reduced NSVM and Reduced LSVM respectively. The results are summarized in Table 6.4.4 for these nonlinear classifiers.

Data Set $m \times n$ $(points \times dimensions)$	Ionosphere 351×34	BUPA Liver 345×6	Cleveland Heart 297×13
NSVM			
Train	**96.1**	75.7	**87.6**
Test	95.0	73.1	86.8
Time (Sec.)	23.27	25.54	17.51
Reduced NSVM			
Train	**96.1**	76.4	86.8
Test	94.5	73.9	**87.1**
Time (Sec.)	**0.88**	**0.67**	**0.53**
SVM^{light}			
Train	94.4	**77.2**	87.1
Test	**96.0**	**74.2**	85.9
Time (Sec.)	2.42	2.74	0.74
LSVM			
Train	**96.1**	75.7	**87.6**
Test	95.0	73.1	86.8
Time (Sec.)	23.76	27.01	12.98
Reduced LSVM			
Train	**96.1**	75.1	87.1
Test	94.5	73.1	86.2
Time (Sec.)	2.09	1.81	1.09

Table 6.3: NSVM, Reduced NSVM, SVM^{light}, LSVM & Reduced LSVM: Training correctness, ten-fold testing correctness and ten-fold training times using a NONLINEAR classifier. Best results are in bold.

Chapter 7

Conclusion

7.1 Proximal Support Vector Machine Classification

We have proposed an extremely simple procedure for generating linear and nonlinear classifiers based on proximity to one of two parallel planes that are pushed as far apart as possible. This procedure, a proximal support vector machine (PSVM), requires nothing more sophisticated than solving a simple nonsingular system of linear equations, for either a linear or nonlinear classifier. In contrast, standard SVM classifiers require a more costly solution of a linear or quadratic program. For a linear classifier, all that is needed by PSVM is the inversion of a small matrix of the order of the input space dimension, typically of the order of 100 or less, even if there are millions of data points to classify. For a nonlinear classifier, a linear system of equations of the order of the number of data points needs to be solved. This allows us to easily classify datasets with as many as a few thousand of points. For larger datasets, data selection and reduction methods such as [32, 37, 52] can be utilized as indicated by some of our numerical results. Our computational results demonstrate that PSVM classifiers obtain test set correctness statistically comparable to that of standard of SVM classifiers at a fraction of the time, sometimes an order of magnitude less.

Taking advantage of the properties of PSVM, we have proposed a very fast and simple incremental SVM classifier based on proximity of each class of points to one

of two parallel planes that are pushed apart to improve generalization. The principal features of the proposed algorithm are its ability to retire old data and add new data very easily, its effectiveness and speed in handling massive datasets incrementally, and its ability to compress large datasets in a compression ratio of order $\frac{mn}{n^2} = \frac{m}{n}$, where m, the number of points, can be of the order of millions, and n, the input dimension, is of the order 10 to a 100. These features coupled with its simplicity will hopefully make it a useful classification tool.

We also extended the PSVM algorithm for use in generating linear and nonlinear multicategory classifiers. The one-from-the-rest approach is based on proximity of each class to one of two parallel planes that are pushed as far apart as possible. This procedure, a multicategory proximal support vector machine (MPSVM) with balancing and Newton refinement, requires nothing more sophisticated than solving k simple systems of linear equations, for either a linear or nonlinear classifier, where k is the number of classes. In contrast, standard one-from-the-rest support vector machine classifiers require the more costly solution of k linear or quadratic programs. For a linear classifier, all that is needed by MPSVM is the solution of k nonsingular systems of linear equations of the order of the input space dimension, typically of 100 or less, even if there are millions of data points to classify. For a nonlinear classifier, a reduction method using rectangular kernels such as [52] is utilized and k linear systems of the order of as small as 15% of the data points are solved. Our computational results demonstrate that MPSVM classifiers obtain test set correctness comparable to that of standard one-from-the-rest SVM classifiers at a fraction of the time, often orders of magnitude less.

We have also proposed a novel Newton refinement algorithm that can improve classification accuracy for any two-class kernel classifier. This refinement is very fast, since

it is a minimization problem in only two variables and is easy to implement. Future research plans include applying this refinement to other linear and nonlinear kernel-based classification algorithms. We have also addressed the problem of unbalanced datasets, which often occurs in one-from-rest classification approaches, by applying a very simple balanced version of PSVM together with a Newton refinement.

7.2 Knowledge Based Support Vector Machines

We have proposed an efficient procedure for incorporating prior knowledge in the form of polyhedral knowledge sets into a linear support vector machine classifier either in combination with a given dataset or based solely on the knowledge sets. This novel and promising approach for handling prior knowledge is worthy of further study, with special focus on ways to handle and simplify the combinatorial nature of incorporating prior knowledge into linear inequalities.

Potential future applications of this approach are problems where training data may not be easily available, but expert knowledge may be readily available in the form of knowledge sets. This would correspond to solving our knowledge based linear program (3.15) with $\nu = 0$. A typical example of this type is breast cancer prognosis [54, 55] where knowledge sets by themselves generated a linear classifier as accurate as any classifier based on data points. This is a new way of incorporating prior knowledge into powerful support vector machine classifiers. Another important avenue to pursue is that of knowledge sets characterized by convex but nonpolyhedral sets.

We have also extended knowledge based SVMs to nonlinear kernel classifiers. The classifier is obtained using a linear programming formulation with any nonlinear symmetric kernel where no positive definiteness (Mercer) condition is assumed. The formulation

works equally well with or without conventional datasets. We note that unlike the linear kernel case with prior knowledge [36], described in Section 3.1, where the absence of conventional datasets was handled by deleting some constraints from a linear programming formulation, here arbitrary representative points from the knowledge sets are utilized to play the role of such datasets, that is A and D in (3.36). The issues associated with sampling the knowledge sets, in situations where there are no conventional data points, also constitute an interesting topic for future research.

7.3 Sparse Classifiers: Data and Feature Selection

We have proposed a minimal support vector machine that extracts a minimum number of points from a given dataset in order to define a separating surface that classifies the dataset into two categories, based on this minimal subset of the data only. This minimal property, in the spirit of Occam's Razor [7], is not only useful in classifying very large datasets using only a fraction of the data, but also maintains or improves generalization over other classifiers that use a considerably higher number of data points in order to determine the separating surface.

In Section 4.1, we extended the idea to nonlinear classifiers. We have addressed one of the serious computational difficulties associated with such problems when we attempt to use a nonlinear kernel classifier on a large training dataset. Such problems result in the unwieldy explicit dependence of the nonlinear classifier on almost all the entries of a huge dataset. By utilizing a leave-one-out error bound, we have proposed an algorithm, based on solving a few linear programs, that generates an accurate nonlinear kernel classifier that typically depends on less than 10% of the original data. With the exception of the multiple class USPS dataset, the nonlinear separator is equally or more accurate than

classifiers using the full dataset and is much faster to evaluate, making it suitable for fast on-line decision making. This allows us to tackle nonlinear classification problems that up to now were very difficult to solve. The fact that our formulation also reduces the number of data points needed if we have to re-solve the problem suggests promising new applications, such as incremental classification of massive datasets where only a small fraction of the data is kept before merging it into incrementally arriving data.

At the end of this chapter, we presented a fast and finitely terminating Newton method for feature selection. When nonlinear kernels are used, the algorithm performs feature selection in a high dimensional space of the dual variable, resulting in a nonlinear kernel classifier that depends on a small number of kernel functions. This makes the method an excellent choice for classification when feature selection or a fast nonlinear kernel classifier is required, as in the case of online decision making such as fraud or intrusion detection.

The NLPSVM algorithm requires only a linear equation solver, which makes it simple, fast and easily accessible. In addition, NLPSVM can be applied very effectively to classification problems in very large dimensional input spaces, which is often the case in the analysis of gene expression microarray data. NLPSVM can also be used effectively for classifying large datasets in smaller dimensional input space. As such, NLPSVM is a versatile, stand-alone algorithm for classification which we hope is a valuable addition to the tools of data mining and machine learning.

7.4 Semi-Supervised Support Vector Machines for Unlabeled Data Classification

We have proposed a concave formulation for the semi-supervised SVM problem and given a fast finite linear programming based formulation for its solution. Unlike a mixed integer formulation, our concave minimization Algorithm 5.1.1 can handle large datasets that are mostly unlabeled. Numerical tests show the potential of the concave semi-supervised support vector machine algorithm as an efficient and viable tool for handling large, totally unlabeled datasets. This is carried out by selecting a small portion of the unlabeled dataset by clustering, labeling it by an expert, and using the concave minimization algorithm VS^{3}VM. Future directions include application of VS^{3}VM to incremental data mining, where a small portion of the dataset is labeled incrementally as new data becomes available, as well as multi-category unlabeled data classification.

7.5 Finite Newton Method for Lagrangian Support Vector Machine Classification

We presented a fast, stand-alone and finitely terminating Newton method for solving classification problems. The method is simple, fast and can be applied to problems with a very large dimensional input space, which is often the case for problems related to analysis of gene expression microarray data. Even though the method is intended to be applied to problems with very large dimensional input space, it showed excellent performance in other problems as well. Computational testing on a variety of real-world test problems demonstrate the effectiveness of the proposed method.

7.6 Summary

In this thesis, we have applied mathematical programming techniques to propose algorithms that address several support vector machines issues such as the following.

- **Speed:** We proposed two very fast stand-alone algorithms that require no specialized software, proximal SVM (PSVM) and Newton Lagrangian SVM (NSVM).We also extended the PSVM idea to multiclass problems.

- **Scalability:** We proposed a variation of PSVM that can handle very large datasets in an incremental fashion. The proposed incremental PSVM algorithm was used to classify a synthetic 1-billion points dataset in ten-dimensional input space in about 2 hours and 26 minutes.

- **Data dependence and sparse representation:** We proposed two concave minimization based algorithms, linear MSVM for support vector reduction and nonlinear MSVM for both kernel data dependence reduction and support vector reduction. We also proposed a Newton approach to solve the 1-norm SVM that leads to very sparse solutions and does not require specialized software.

- **Use of unlabeled data:** A concave minimization approach was proposed to solve a semi-supervised support vector machine. In this formulation, part of the problem data is unlabeled. Our formulation performed much faster and handled much larger problems than integer programming formulations that solve the same problem.

- **Knowledge incorporation:** A novel idea was proposed to incorporate knowledge in the form of polyhedral sets into a linear or nonlinear SVM formulation which resulted in improved classifiers and the ability to base classifiers partially or

completely on prior knowledge.

Bibliography

[1] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen. *LAPACK User's Guide*. SIAM, Philadelphia, Pennsylvania, third edition, 1999. http://www.netlib.org/lapack/.

[2] L. Armijo. Minimization of functions having Lipschitz-continuous first partial derivatives. *Pacific Journal of Mathematics*, 16:1–3, 1966.

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

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

[5] K. P. Bennett and O. L. Mangasarian. Multicategory separation via linear programming. *Optimization Methods and Software*, 3:27–39, 1993.

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

[7] A. Blumer, A. Ehrenfeucht, D. Haussler, and M. K. Warmuth. Occam's razor. *Information Processing Letters*, 24:377–380, 1987.

[8] L. Bottou, C. Cortes, J. Denker, H. Drucker, I. Guyon, L. Jackel, Y. LeCun, U. Muller, E. Sackinger, P. Simard, and V. Vapnik. Comparison of classifier methods: A case study in handwriting digit recognition. In *International Conference on Pattern Recognition*, pages 77–87. IEEE Computer Society Press, 1994.

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

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

[11] P. S. Bradley, O. L. Mangasarian, and J. B. Rosen. Parsimonious least norm approximation. *Computational Optimization and Applications*, 11(1):5–21, October 1998. ftp://ftp.cs.wisc.edu/math-prog/tech-reports/97-03.ps.

[12] 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.

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

[14] E. J. Bredensteiner and K. P. Bennett. Multicategory classification by support vector machines. *Computational Optimization and Applications*, 12:53–79, 1999.

[15] A. Brooke, D. Kendrick, and A. Meeraus. *GAMS: A User's Guide*. The Scientific Press, South San Francisco, CA, 1988.

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

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

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

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

[20] S. Cost and S. Salzberg. A weighted nearest neighbor algorithm for learning with symbolic features. *Machine Learning*, 10:57–58, 1993.

[21] CPLEX Optimization Inc., Incline Village, Nevada. *Using the CPLEX(TM) Linear Optimizer and CPLEX(TM) Mixed Integer Optimizer (Version 2.0)*, 1992.

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

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

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

[25] Sanjoy Dasgupta. Experiments with random projection. In *Uncertainty in Artificial Intelligence: Proceedings of the Sixteenth Conference (UAI-2000)*, pages 143–151, San Francisco, CA, 2000. Morgan Kaufmann Publishers.

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

[27] 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.

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

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

[30] A. V. Fiacco and G. P. McCormick. *Nonlinear Programming: Sequential Unconstrained Minimization Techniques*. John Wiley & Sons, New York, NY, 1968.

[31] S. Fine and K. Scheinberg. Efficient svm training using low-rank kernel representations. *Journal of Machine Learning Research*, 2:243–264, 2001.

[32] G. Fung and O. L. Mangasarian. Data selection for support vector machine classification. In R. Ramakrishnan and S. Stolfo, editors, *Proceedings KDD-2000: Knowledge Discovery and Data Mining, August 20-23, 2000, Boston, MA*, pages 64–70, New York, 2000. Asscociation for Computing Machinery. ftp://ftp.cs.wisc.edu/pub/dmi/tech-reports/00-02.ps.

[33] 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. Asscociation for Computing Machinery. ftp://ftp.cs.wisc.edu/pub/dmi/tech-reports/01-02.ps.

[34] G. Fung and O. L. Mangasarian. Finite Newton method for Lagrangian support vector machine classification. Technical Report 02-01, Data Mining Institute, Computer Sciences Department, University of Wisconsin, Madison, Wisconsin, February 2002. ftp://ftp.cs.wisc.edu/pub/dmi/tech-reports/02-01.ps. Neurocomputing, to appear.

[35] G. Fung and O. L. Mangasarian. Incremental support vector machine classification. In H. Mannila R. Grossman and R. Motwani, editors, *Proceedings of the 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.

[36] G. Fung, O. L. Mangasarian, and J. Shavlik. Knowledge-based support vector machine classifiers. Technical Report 01-09, Data Mining Institute, Computer Sciences Department, University of Wisconsin, Madison, Wisconsin, November 2001. ftp://ftp.cs.wisc.edu/pub/dmi/tech-reports/01-09.ps, NIPS 2002 Proceedings, to appear.

[37] 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.

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

[39] T. Van Gestel, J. Suykens, G. Lanckriet, A. Lambrechts, B. De Moor, and J. Vandewalle. Multiclass ls-svms: moderated outputs and coding-decoding schemes. *Neural Processing Letters*. to appear.

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

[41] F. J. Gonzalez-Castano and R. R. Meyer. Projection support vector machines. Technical Report 00-05, Computer Sciences Department, University of Wisconsin, Madison, WI, November 2000.

[42] J. Gracke, M. Griebel, and M. Thess. Data mining with sparse grids. Technical report, Institut für Angrwandte Mathematik, Universitaät Bonn, Bonn, Germany, 2000. http://wissrech.iam.uni-bonn.de/research/projects/garcke/sparsemining.html.

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

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

[46] A. E. Hoerl and R. W. Kennard. Biased estimation for nonorthogonal problems. *Technometrics*, 12:55–67, 1952.

[47] C.-W. Hsu and C.-J. Lin. A comparison on methods for Multi-Class support vector machines, 2001. http://www.csie.ntu.edu.tw/ cjlin/papers.html.

[48] ILOG CPLEX Division, 889 Alder Avenue, Incline Village, Nevada. *CPLEX Optimizer*. `http://www.cplex.com/`.

[49] T. S. Jaakkola and D. Haussler. Probabilistic kernel regression models. In *Proceedings of the 1999 Conference on AI and Statistics*, San Mateo, CA, 1999. Morgan Kaufmann.

[50] 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.

[51] C. Kanzow, H. Qi, and L. Qi. On the minimum norm solution of linear programs. Preprint, University of Hamburg, Hamburg, 2001. http://www.math.uni-hamburg.de/home/kanzow/paper.html. Journal of Optimization Theory and Applications, to appear.

[52] Y.-J. Lee and O. L. Mangasarian. RSVM: Reduced support vector machines. Technical Report 00-07, Data Mining Institute, Computer Sciences Department, University of Wisconsin, Madison, Wisconsin, July 2000. Proceedings of the First SIAM International Conference on Data Mining, Chicago, April 5-7, 2001, CD-ROM Proceedings. ftp://ftp.cs.wisc.edu/pub/dmi/tech-reports/00-07.ps.

[53] Y.-J. Lee and O. L. Mangasarian. SSVM: A smooth support vector machine. *Computational Optimization and Applications*, 20:5–22, 2001. Data Mining Institute, University of Wisconsin, Technical Report 99-03. ftp://ftp.cs.wisc.edu/pub/dmi/tech-reports/99-03.ps.

[54] Y.-J. Lee, O. L. Mangasarian, and W. H. Wolberg. Breast cancer survival and chemotherapy: a support vector machine analysis. Technical Report 99-10, Data Mining Institute, Computer Sciences Department, University of Wisconsin, Madison, Wisconsin, December 1999. DIMACS Series in Discrete Mathematics and Theoretical Computer Science, American Mathematical Society, Volume 55, 2000, 1-10. ftp://ftp.cs.wisc.edu/pub/dmi/tech-reports/99-10.ps.

[55] 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.

[56] S. Lucidi. A new result in the theory and computation of the least-norm solution of a linear program. *Journal of Optimization Theory and Applications*, 55:103–117, 1987.

[57] O. L. Mangasarian. Linear and nonlinear separation of patterns by linear programming. *Operations Research*, 13:444–452, 1965.

[58] O. L. Mangasarian. Normal solutions of linear programs. *Mathematical Programming Study*, 22:206–216, 1984.

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

[60] 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.

[61] O. L. Mangasarian. Machine learning via polyhedral concave minimization. In H. Fischer, B. Riedmueller, and S. Schaeffler, editors, *Applied Mathematics and Parallel Computing - Festschrift for Klaus Ritter*, pages 175–188. Physica-Verlag A Springer-Verlag Company, Heidelberg, 1996. ftp://ftp.cs.wisc.edu/math-prog/tech-reports/95-20.ps.

[62] O. L. Mangasarian. Solution of general linear complementarity problems via nondifferentiable concave minimization. *Acta Mathematica Vietnamica*, 22(1):199–205, 1997. ftp://ftp.cs.wisc.edu/math-prog/tech-reports/96-10.ps.

[63] 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.

[64] O. L. Mangasarian. Minimum-support solutions of polyhedral concave programs. *Optimization*, 45:149–162, 1999. ftp://ftp.cs.wisc.edu/math-prog/tech-reports/97-05.ps.

[65] 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.

[66] O. L. Mangasarian. A finite Newton method for classification problems. Technical Report 01-11, Data Mining Institute, Computer Sciences Department, University of Wisconsin, Madison, Wisconsin, December 2001. ftp://ftp.cs.wisc.edu/pub/dmi/tech-reports/01-11.ps.*Optimization Methods and Software* 17, 2002, 913-929.

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

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

[69] O. L. Mangasarian and D. R. Musicant. Active support vector machine classification. In Todd K. Leen, Thomas G. Dietterich, and Volker Tresp, editors, *Advances in Neural Information Processing Systems 13*, pages 577–583, Cambridge, MA, 2001. MIT Press. ftp://ftp.cs.wisc.edu/pub/dmi/tech-reports/00-04.ps.

[70] O. L. Mangasarian and D. R. Musicant. Data discrimination via nonlinear generalized support vector machines. In M. C. Ferris, O. L. Mangasarian, and J.-S. Pang, editors, *Complementarity: Applications, Algorithms and Extensions*, pages 233–251, Dordrecht, Netherlands, January 2001. Kluwer Academic Publishers. ftp://ftp.cs.wisc.edu/math-prog/tech-reports/99-03.ps.

[71] 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.

[72] 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.

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

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

[75] MATLAB. *User's Guide*. The MathWorks, Inc., Natick, MA 01760, 1994-2001. http://www.mathworks.com.

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

[77] P. M. Murphy and D. W. Aha. UCI machine learning repository, 1992. www.ics.uci.edu/∼mlearn/MLRepository.html.

[78] D. R. Musicant. NDC: normally distributed clustered datasets, 1998. www.cs.wisc.edu/∼musicant/data/ndc/.

[79] 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.

[80] M. C. O'Neill. Escherchia coli promoters: I. concensus as it relates to spacing class, specificity, repeat substructure, and three dimensional organization. *Journal of Biological Chemistry*, 264:5522–5530, 1989.

[81] M. Opper and O. Winther. Gaussian processes and SVM: Mean field and leave-one-out. In A. Smola, P. Bartlett, B. Schölkopf, and D. Schuurmans, editors, *Advances in Large Margin Classifiers*, pages 311–326, Cambridge, MA, 2000. MIT Press.

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

[83] D. Page, F. Zhan, J. Cussens, M. Waddell, J. Hardin, B. Barlogie, and J. Shaughnessy, Jr. Comparative data mining for microarrays: A case study based on multiple myeloma. Technical Report 1453, Computer Sciences Department, University of Wisconsin, November 2002.

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

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

[86] J. R. Quinlan. *Induction of Decision Trees*, volume 1. 1986.

[87] J. Rissanen. Stochastic complexity and modeling. *Annals of Statistics*, 14:1080–1100, 1986.

[88] R. T. Rockafellar. *Convex Analysis*. Princeton University Press, Princeton, New Jersey, 1970.

[89] R. T. Rockafellar. Augmented Lagrange multiplier functions and duality in nonconvex programming. *SIAM Journal on Control*, 12:268–285, 1974.

[90] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. In D. E. Rumelhart and J. L. McClelland, editors, *Parallel Distributed Processing*, pages 318–362, Cambridge, Massachusetts, 1986. MIT Press.

[91] A. Smola, P. L. Bartlett, B. Schölkopf, and J. Schuurmann (editors). *Advances in Large Margin Classifiers*. MIT Press, Cambridge, MA, 2000.

[92] A. Smola and B. Schölkopf. *Learning with Kernels*. MIT Press, Cambridge, MA, 2002.

[93] A. J. Smola and B. Schölkopf. Sparse greedy matrix approximation for machine learning. In *Proc. 17th International Conf. on Machine Learning*, pages 911–918. Morgan Kaufmann, San Francisco, CA, 2000.

[94] J. A. K. Suykens, L. Lukas, P. Van Dooren, B. De Moor, and J. Vandewalle. Least squares support vector machine classifiers: a large scale algorithm. In *European Conference on Circuit Theory and Design, ECCTD'99*, pages 839–842, Stresa, Italy, 1999.

[95] J. A. K. Suykens, L. Lukas, and J. Vandewalle. Sparse least squares support vector machine classifiers. In *European Symposium on Artificial Neural Networks*, pages 37–42, 24 av. L. Mommaerts, B-1140 Evere, Belgium, 2000. D-Facto publications. http://www.dice.ucl.ac.be/esann/proceedings/index.html.

[96] J. A. K. Suykens and J. Vandewalle. Least squares support vector machine classifiers. *Neural Processing Letters*, 9(3):293–300, 1999.

[97] J. A. K. Suykens and J. Vandewalle. Multiclass least squares support vector machines. In *Proceedings of IJCNN'99*, pages CD–ROM, Washington, DC, 1999.

[98] A. N. Tikhonov and V. Y. Arsenin. *Solutions of Ill–Posed Problems*. John Wiley & Sons, New York, 1977.

[99] M. Tipping. The relevance vector machine. In S. A. Solla, T. K. Leen, and K.-R. Müller, editors, *Advances in Neural Information Processing Systems*, volume 12, pages 652–658, Cambridge, MA, 2000. MIT Press.

[100] G. G. Towell, J. W. Shavlik, and M. Noordewier. Refinement of approximate domain theories by knowledge-based artificial neural networks. In *Proceedings of the Eighth National Conference on Artificial Intelligence (AAAI-90)*, pages 861–866, 1990.

[101] V. Roth V. and V. Steinhage. Nonlinear discriminant analysis using kernel function. In S.A. Solla, T.K. Leen, and K.-R. Mueller, editors, *Advances in Neural Information Processing Systems–NIPS*99*, pages 568–574, 1999.

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

[103] V. N. Vapnik. *The Nature of Statistical Learning Theory*. Springer, New York, second edition, 2000.

[104] V. N. Vapnik and O. Chapelle. Bounds on expectation for SVM. In A. Smola, P. Bartlett, B. Schölkopf, and D. Schuurmans, editors, *Advances in Large Margin Classifiers*, pages 261–280, Cambridge, MA, 2000. MIT Press.

[105] J. Weston and R. Herbrich. Adaptive margin support vector machines. In A. Smola, P. Bartlett, B. Schölkopf, and D. Schuurmans, editors, *Advances in Large Margin Classifiers*, pages 281–295, Cambridge, MA, 2000. MIT Press.

[106] J. Weston and C. Watkins. Multi-class support vector machines. Technical report csd-tr-98-04, Royal Holloway, University of London, Surrey, England, 1998.

[107] 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.

[108] I. H. Witten and E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*. Morgan Kaufmann, San Francisco, CA, 1999.