

 Open access • Journal Article • DOI:10.1109/72.822516

A fast iterative nearest point algorithm for support vector machine classifier design

— [Source link](#) 

S. Sathiya Keerthi, Shirish Shevade, Chiranjib Bhattacharyya, K.R.K. Murthy





Institutions: National University of Singapore

Published on: 01 Jan 2000 - IEEE Transactions on Neural Networks (IEEE)

Topics: Sequential minimal optimization, Support vector machine, Iterative method and Quadratic programming

Related papers:

- [The Nature of Statistical Learning Theory](#)
- [Fast training of support vector machines using sequential minimal optimization](#)
- [Statistical learning theory](#)
- [Support-Vector Networks](#)
- [Duality and Geometry in SVM Classifiers](#)

Share this paper:    

View more about this paper here: <https://typeset.io/papers/a-fast-iterative-nearest-point-algorithm-for-support-vector-1qlgpdf8n>

A Fast Iterative Nearest Point Algorithm for Support Vector Machine Classifier Design

S. S. Keerthi, S. K. Shevade, C. Bhattacharyya, and K. R. K. Murthy

Abstract—In this paper we give a new fast iterative algorithm for support vector machine (SVM) classifier design. The basic problem treated is one that does not allow classification violations. The problem is converted to a problem of computing the nearest point between two convex polytopes. The suitability of two classical nearest point algorithms, due to Gilbert, and Mitchell *et al.*, is studied. Ideas from both these algorithms are combined and modified to derive our fast algorithm. For problems which require classification violations to be allowed, the violations are quadratically penalized and an idea due to Cortes and Vapnik and Frieß is used to convert it to a problem in which there are no classification violations. Comparative computational evaluation of our algorithm against powerful SVM methods such as Platt's sequential minimal optimization shows that our algorithm is very competitive.

Index Terms—Classification, nearest point algorithm, quadratic programming, support vector machine.

I. INTRODUCTION

THE last few years have seen the rise of support vector machines (SVM's) [27] as powerful tools for solving classification and regression problems [5]. A variety of algorithms for solving these problems has emerged. Traditional quadratic programming algorithms [13] and modifications such as the chunking algorithm [26] that make use of the fact that the number of support vectors is usually a small percentage of the total training set have been tried. These algorithms require enormous matrix storage and do expensive matrix operations. To overcome these problems, recently fast iterative algorithms that are also easy to implement have been suggested [20], [12], [19], [8], [23]; Platt's SMO algorithm [20] is an important example. Such algorithms are bound to widely increase the popularity of SVM's among practitioners. This paper makes another contribution in this direction. Transforming a particular SVM classification problem formulation into a problem of computing the nearest point between two convex polytopes in the hidden feature space, we give a fast iterative nearest point algorithm for SVM classifier design that is competitive with the SMO algorithm. Like SMO, our algorithm also is quite straightforward to implement. A pseudocode for our algorithm can be found in [14]. Using this pseudocode an actual running

code can be developed in short time. A Fortran code can be obtained free from the authors for any noncommercial purpose.

The basic problem addressed in this paper is the two category classification problem. Throughout this paper we will use x to denote the input vector of the support vector machine and z to denote the feature space vector which is related to x by a transformation, $z = \phi(x)$. As in all SVM designs, we do not assume ϕ to be known; all computations will be done using only the Kernel function, $K(x, \hat{x}) = \phi(x) \cdot \phi(\hat{x})$, where " \cdot " denotes inner product in the z space.

The simplest SVM formulation is one which does not allow classification violations. Let $\{x_i\}_{i=1}^m$ be a training set of input vectors. Let I = the index set for class 1 and J = the index set for class 2. We assume $I \neq \emptyset$, $J \neq \emptyset$, $I \cup J = \{1, \dots, m\}$ and $I \cap J = \emptyset$. Let us define $z_i = \phi(x_i)$. The SVM design problem without violations is

$$\begin{aligned} \min & \frac{1}{2} \|w\|^2 \\ \text{s.t. } & w \cdot z_i + b \geq 1 \quad \forall i \in I; \quad w \cdot z_j + b \leq -1 \quad \forall j \in J. \end{aligned} \quad (\text{SVM-NV})$$

Let us make the following assumption.

Assumption A1: There exists a (w, b) pair for which the constraints of SVM-NV are satisfied.

If this assumption holds then SVM-NV has an optimal solution, which turns out to be unique. Let $H_+ = \{z : w \cdot z + b = 1\}$ and $H_- = \{z : w \cdot z + b = -1\}$, the bounding hyperplanes separating the two classes. M , the margin between them is given by $M = 2/\|w\|$. Thus SVM-NV consists of finding the pair of parallel hyperplanes that has the maximum margin among all pairs that separate the two classes.

To deal with data which are linearly inseparable in the z -space, and also for the purpose of improving generalization, there is a need to have a problem formulation in which classification violations are allowed. The popular approach for doing this is to allow violations in the satisfaction of the constraints in SVM-NV, and penalize such violations *linearly* in the objective function

$$\begin{aligned} \min & \frac{1}{2} \|w\|^2 + C \sum_k \xi_k \\ \text{s.t. } & w \cdot z_i + b \geq 1 - \xi_i \quad \forall i \in I; \quad w \cdot z_j + b \leq -1 + \xi_j \\ & \forall j \in J; \quad \xi_k \geq 0 \quad \forall k \in I \cup J. \end{aligned} \quad (\text{SVM-VL})$$

(Throughout, we will use k and/or l whenever the indexes run over all elements of $I \cup J$.) Here C is a positive, inverse regularization constant that is chosen to give the correct relative

Manuscript received May 6, 1999; revised November 13, 1999.

S. Keerthi is with the Department of Mechanical and Production Engineering, National University of Singapore, Singapore 119260 (e-mail: mpeesk@guppy.mpe.nus.edu.sg).

S. K. Shevade, C. Bhattacharyya, and K. R. K. Murthy are with the Department of Computer Science and Automation, Indian Institute of Science, Bangalore 560012, India (e-mail: shirish@csa.iisc.ernet.in).

Publisher Item Identifier S 1045-9227(00)01194-2.

weighting between margin maximization and classification violation. Using the Wolfe duality theory [5], [7] SVM-VL can be transformed to the following equivalent dual problem:

$$\begin{aligned} \max \quad & \sum_k \alpha_k - \frac{1}{2} \sum_k \sum_l \alpha_k \alpha_l y_k y_l z_k \cdot z_l \\ \text{s.t.} \quad & 0 \leq \alpha_k \leq C \quad \forall k \\ & \sum_k \alpha_k y_k = 0 \end{aligned} \quad (\text{SVM-VL-DUAL})$$

where $y_i = 1, i \in I$ and $y_j = -1, j \in J$. It is computationally easy to handle this problem since it is directly based on $z_k \cdot z_l$ (kernel) calculations. Platt's SMO algorithm, as well as others are ways of solving SVM-VL-DUAL. SMO is particularly very simple and, at the same time, impressively fast. In very simple terms, it is an iterative scheme in which only two (appropriately chosen) α_i variables are adjusted at any given time so as to improve the value of the objective function. The need for adjusting at least two variables at a time is caused by the presence of the equality constraint.

Remark 1: It is useful to point out that the Wolfe dual of SVM-NV is same as SVM-VL-DUAL, with C set to ∞ . Therefore, any algorithm designed for solving SVM-VL-DUAL can be easily used to solve the dual of SVM-NV, and thereby solve SVM-NV.

Using a suggestion made by Cortes and Vapnik [6], in a recent paper Frieß [9] has explored the use of a sum of squared violations in the cost function

$$\begin{aligned} \min \quad & \frac{1}{2} \|w\|^2 + \frac{\tilde{C}}{2} \sum_k \xi_k^2 \\ \text{s.t.} \quad & w \cdot z_i + b \geq 1 - \xi_i \quad \forall i \in I; \\ & w \cdot z_i + b \leq -1 + \xi_i \quad \forall i \in J. \end{aligned} \quad (\text{SVM-VQ})$$

Preliminary experiments by Frieß have shown this formulation to be promising. Unlike SVM-VL, here there is no need to include nonnegativity constraints on ξ_k for the following reason. Suppose, at the optimal solution of SVM-VQ, ξ_k is negative for some k . Then, by resetting $\xi_k = 0$, we can remain feasible and also strictly decrease the cost. Thus, negative values of ξ_k cannot occur at the optimal solution.

Remark 2: As pointed out by Frieß [9], a very nice property of SVM-VQ is that by doing a simple transformation it can be converted into an instance of SVM-NV. Let e_k denote the m dimensional vector in which the k th component is one and all other components are zero. Define

$$\begin{aligned} \tilde{w} &= \left(\frac{w}{\sqrt{\tilde{C}}} \right); \quad b = \tilde{b}; \quad \tilde{z}_i = \left(\frac{z_i}{\sqrt{\tilde{C}}} \right), \quad i \in I; \\ \tilde{z}_j &= \left(-\frac{z_j}{\sqrt{\tilde{C}}} \right), \quad j \in J. \end{aligned} \quad (1)$$

Then it is easy to see that SVM-VQ transforms to an instance of SVM-NV. (Use $\tilde{w}, \tilde{b}, \tilde{z}$ instead of w, b, z .) Note that, because of the presence of ξ_k variables, SVM-VQ's feasible space is always nonempty and so the resulting SVM-NV is automatically feasible. Also note that, if K denotes the Kernel function in the

SVM-VQ problem, then \tilde{K} , the Kernel function for the transformed SVM-NV problem is given by

$$\tilde{K}(x_k, x_l) = K(x_k, x_l) + \frac{1}{\tilde{C}} \delta_{kl}$$

where δ_{kl} is one if $k = l$ and zero otherwise. Thus, for any pair of training vectors (x_k, x_l) the modified kernel function is easily computed.

Our main aim in this paper is to give a fast algorithm for solving SVM-NV. The main idea consists of transforming SVM-NV into a problem of computing the nearest point between two convex polytopes and then using a carefully chosen nearest point algorithm to solve it. Because of Remark 2, our algorithm can also be easily used to solve SVM-VQ. By Remark 1, algorithms such as SMO can also be used to solve SVM-VQ. In empirical testing however, we have found that our algorithm is more efficient for solving SVM-VQ, than SMO used in this way. Even when the "typical" computational cost of solving SVM-VL by SMO is compared with that of solving SVM-VQ by our algorithm, we find that our algorithm is competitive.

Frieß *et al.* [8] (deriving inspiration from the Adatron algorithm given by Anlauf and Biehl [1] for designing Hopfield nets) and, later, Mangasarian and Musicant [19] (using a successive overrelaxation idea) suggested the inclusion of the extra term $b^2/2$ in the objective functions of the various formulations. This is done with computational simplicity in mind. When $b^2/2$ is added to the objective functions of the primal SVM problems, i.e., SVM-NV, SVM-VL and SVM-VQ, it turns out that the only equality constraint in the dual problems, i.e., $\sum_i \alpha_i y_i = 0$, gets eliminated. Therefore, it is easy to give an iterative algorithm for improving the dual objective function simply by adjusting a single α_i at a time, as opposed to the adjustment of two such variables required by SMO. Frieß *et al.* [8] applied their kernel Adatron algorithm to solve SVM-NV, Frieß [9] applied the same to SVM-VQ, while Mangasarian and Musicant [19] applied the successive overrelaxation scheme to solve SVM-VL. The basic algorithmic ideas used by them is as follows: choose one α_i , determine a unconstrained step size for α_i as if there are no bounds on α_i , and then clip the step size so that the updated α_i satisfies all its bounds.

If the term $b^2/2$ is included in the objective functions of SVM-VQ and SVM-NV, then these problems can be easily transformed to a problem of computing the nearest point of a single convex polytope from the origin, a problem that is much simpler than finding the nearest distance between two convex polytopes. Our nearest point algorithm mentioned earlier simplifies considerably for this simpler problem [14]. Our testing, described in [14] shows that these simplified algorithms do not perform as well as algorithms which solve problems without the $b^2/2$ term. However, for problems where the entire kernel matrix can be computed and stored in memory they can still be very useful.

This paper is organized as follows. In Section II we reformulate SVM-NV as a problem of computing the nearest point between two convex polytopes. In Section III we discuss optimality criteria for this nearest point problem. Section IV derives a simple check for stopping nearest point algorithms so

as to get guaranteed accuracy for the solution of SVM-NV. The main algorithm of the paper is derived in Sections V and VI. Two classical algorithms for doing nearest point solution, due to Gilbert [10] and Mitchell *et al.* [18], are combined and modified to derive our fast algorithm. A comparative computational testing of our algorithm against Platt's SMO algorithm is taken up in Section VII. We conclude with some closing remarks in Section VIII.

II. REFORMULATION OF SVM-NV AS A NEAREST POINT PROBLEM

Given a set S we use $\text{co}S$ to denote the convex hull of S , i.e., $\text{co}S$ is the set of all convex combinations of elements of S

$$\text{co}S = \left\{ \sum_{k=1}^l \beta_k s_k : s_k \in S, \beta_k \geq 0, \sum_{k=1}^l \beta_k = 1 \right\}.$$

Let $U = \text{co}\{z_i : i \in I\}$ and $V = \text{co}\{z_i : i \in J\}$ where $\{z_k\}$, I and J are as in the definition of SVM-NV. Since I and J are finite sets, U and V are convex polytopes. Consider the following generic problem of computing the minimum distance between U and V

$$\min \|u - v\| \quad \text{s.t. } u \in U, \quad v \in V. \quad (\text{NPP})$$

It can be easily noted that the solution of NPP may not be unique. We can rewrite the constraints of NPP algebraically as

$$\begin{aligned} u &= \sum_{i \in I} \beta_i z_i; \quad \beta_i \geq 0, \quad i \in I; \quad \sum_{i \in I} \beta_i = 1 \\ v &= \sum_{j \in J} \beta_j z_j; \quad \beta_j \geq 0, \quad j \in J; \quad \sum_{j \in J} \beta_j = 1. \end{aligned} \quad (2)$$

The equivalence of the problems SVM-NV and NPP can be easily understood by studying the geometry shown in Fig. 1. Assumption A1 is equivalent to assuming that U and V are nonintersecting. Thus A1 implies that the optimal cost of NPP is positive. If (w^*, b^*) denotes the solution of SVM-NV and (u^*, v^*) denotes a solution of NPP, then by using the facts that maximum margin $= 2/\|w^*\| = \|u^* - v^*\|$ and $w^* = \delta(u^* - v^*)$ for some δ , we can easily derive the following relationship between the solutions of SVM-NV and NPP:

$$w^* = \frac{2}{\|u^* - v^*\|^2} (u^* - v^*); \quad b^* = \frac{\|v^*\|^2 - \|u^*\|^2}{\|u^* - v^*\|^2}. \quad (3)$$

The following theorem states this relationship formally.

Theorem 1: (w^*, b^*) solves SVM-NV if and only if there exist $u^* \in U$ and $v^* \in V$ such that (u^*, v^*) solves NPP and (3) holds.

A direct, geometrically intuitive proof is given by Sancheti and Keerthi [22] with reference to a geometrical problem in robotics. Later, Bennett [3] proved a somewhat close result in the context of learning algorithms. Here we only give a discussion that follows the traditional Wolfe-Dual approach employed in the SVM literature. The main reason for doing this is

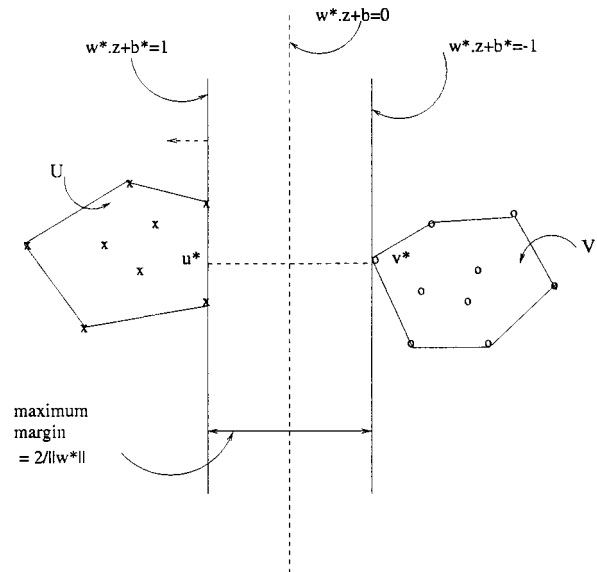


Fig. 1. Among all pairs of parallel hyperplanes that separate the two classes, the pair with the largest margin is the one which has (u^*, v^*) as the normal direction, where (u^*, v^*) is a pair of closest points of U and V . Note that $w^* = \delta(u^* - v^*)$ for δ chosen such that $\|u^* - v^*\| = 2/\|w^*\|$.

to show the relationships of NPP and the variables in it with the Wolfe-Dual of SVM-NV and the variables there.

Using Wolfe duality theory [5], [7] we first transform SVM-NV to the following equivalent dual problem:

$$\begin{aligned} \max \quad & \sum_k \alpha_k - \frac{1}{2} \sum_k \sum_l \alpha_k \alpha_l y_k y_l z_k \cdot z_l \\ \text{s.t.} \quad & \alpha_k \geq 0 \quad \forall k; \quad \sum_k \alpha_k y_k = 0 \end{aligned} \quad (\text{SVM-NV-DUAL})$$

where, as before, $y_i = 1, i \in I$ and $y_j = -1, j \in J$. Since $\sum_k \alpha_k y_k = 0$ implies that $\sum_{i \in I} \alpha_i = \sum_{j \in J} \alpha_j$, we can introduce a new variable, λ and rewrite $\sum_k \alpha_k y_k = 0$ as two constraints

$$\sum_{i \in I} \alpha_i = \lambda, \quad \sum_{j \in J} \alpha_j = \lambda.$$

If we also define

$$\beta_k = \frac{\alpha_k}{\lambda} \quad \forall k \quad (4)$$

then SVM-NV-DUAL can be rewritten as

$$\begin{aligned} \max \quad & 2\lambda - \frac{\lambda^2}{2} \sum_k \sum_l \beta_k \beta_l y_k y_l z_k \cdot z_l \\ \text{s.t.} \quad & \beta_k \geq 0 \quad \forall k; \quad \sum_{i \in I} \beta_i = 1; \quad \sum_{j \in J} \beta_j = 1. \end{aligned} \quad (5)$$

In this formulation it is convenient to first optimize λ keeping β constant, and then optimize β on the outer loop. Optimizing with respect to λ yields

$$\lambda^* = \frac{2}{\sum_k \sum_l \beta_k \beta_l y_k y_l z_k \cdot z_l}. \quad (6)$$

Substituting this in (5) and simplifying, we see that (5) becomes equivalent to

$$\begin{aligned} \max & 2 \left/ \left(\sum_k \sum_l \beta_k \beta_l y_k y_l z_k \cdot z_l \right) \right. \\ \text{s.t. } & \beta_k \geq 0 \quad \forall k; \quad \sum_{i \in I} \beta_i = 1; \quad \sum_{j \in J} \beta_j = 1. \end{aligned} \quad (7)$$

This problem is equivalent to the problem

$$\begin{aligned} \min & \frac{1}{2} \sum_k \sum_l \beta_k \beta_l y_k y_l z_k \cdot z_l \\ \text{s.t. } & \beta_k \geq 0 \quad \forall k; \quad \sum_{i \in I} \beta_i = 1; \quad \sum_{j \in J} \beta_j = 1. \end{aligned} \quad (8)$$

If we define a matrix P whose columns are $y_1 z_1, \dots, y_m z_m$ then it is easy to note that

$$\sum_k \sum_l \beta_k \beta_l y_k y_l z_k \cdot z_l = \|P\beta\|^2.$$

If β satisfies the constraints of (8), then $P\beta = u - v$ where $u \in U$ and $v \in V$. Thus (8) is equivalent to NPP.

Wolfe duality theory [5] actually yields

$$w^* = \sum_k \alpha_k^* y_k z_k.$$

Using this, (4) and (6) we can immediately verify the expression for w^* in (3). The expression for b^* can be easily understood from geometry.

Remark 3: The above discussion also points out an important fact: there is a simple redundancy in the SVM-NV-DUAL formulation which gets removed in the NPP formulation. Note that NPP has two equality constraints and SVM-NV-DUAL has only one, while both have the same number of variables. Therefore, when quadratic programming algorithms are applied to SVM-NV-DUAL and NPP separately they work quite differently, even when started from the same “equivalent” starting points.

III. OPTIMALITY CRITERIA FOR NPP

First let us give some definitions concerning support properties of a convex polytope. For a given compact set P , let us define the *support function*, $h_P : R^n \rightarrow R$, by

$$h_P(\eta) = \max\{\eta \cdot z : z \in P\} \quad (9)$$

See Fig. 2. We use $s_P(\eta)$ to denote any one solution of (9), i.e., $s_P(\eta)$ satisfies

$$h_P(\eta) = s_P(\eta) \cdot \eta \quad \text{and } s_P(\eta) \in P. \quad (10)$$

Now consider the case where P is a convex polytope: $Z = \{\hat{z}_1, \dots, \hat{z}_r\}$ and $P = \text{co } Z$. It is well known [16] that the maximum of a linear function over a convex polytope is attained by an extreme point. This means that $h_P = h_Z$ and $s_P = s_Z$. Therefore h_P and s_P can be determined by a simple enumeration of inner products

$$\begin{aligned} h_P(\eta) &= h_Z(\eta) = \max\{\eta \cdot \hat{z}_k : k = 1, \dots, r\} \\ s_P(\eta) &= s_Z(\eta) = \hat{z}_l \quad \text{where } \eta \cdot \hat{z}_l = h_Z(\eta). \end{aligned} \quad (11)$$

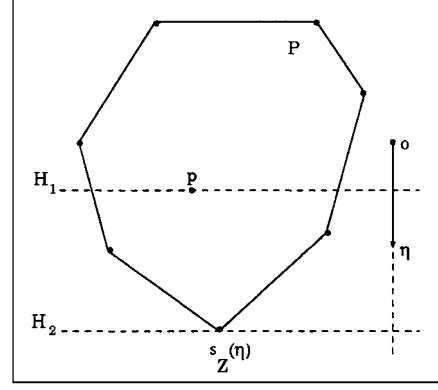


Fig. 2. Definition of support properties. $s_Z(\eta)$ is the extreme vertex of P in the direction η . The distance between the hyperplanes, H_1 and H_2 is equal to $g_P(\eta, p)/\|\eta\|$.

Thus (11) provides a simple procedure for evaluating the support properties of the convex polytope, $P = \text{co } Z$. Let us also define the function, $g_P : R^n \times P \rightarrow R$ by

$$g_P(\eta, p) = h_P(\eta) - \eta \cdot p.$$

By (9) it follows that

$$g_P(\eta, p) \geq 0 \quad \forall \eta \in R^n, \quad p \in P. \quad (12)$$

We now adapt these general definitions to derive an optimality criterion for NPP. A simple geometrical analysis shows that a pair, $(u, v) \in U \times V$ solves NPP if and only if

$$h_U(-z) = -z \cdot u \quad \text{and} \quad h_V(z) = z \cdot v$$

where $z = u - v$; in other words, (u, v) solves NPP if and only if $u = s_U(-z)$ and $v = s_V(z)$. Equivalently, (u, v) is optimal if and only if

$$g_U(v - u, u) = 0 \quad \text{and} \quad g_V(u - v, v) = 0. \quad (13)$$

Let us define the function, $g : U \times V \rightarrow R$ by

$$g(u, v) = g_U(v - u, u) + g_V(u - v, v). \quad (14)$$

Since g_U and g_V are nonnegative functions, it also follows that (u, v) is optimal if and only if $g(u, v) = 0$. The following theorem states the above results (and related ones) formally.

Theorem 2: Suppose $u \in U, v \in V$ and $z = u - v$. Then the following hold. 1) $g(u, v) \geq 0$. 2) if $\bar{u} \in U$ is a point that satisfies $z \cdot \bar{u} < z \cdot u$, then there is a point \tilde{u} on the line segment, $\text{co}\{u, \bar{u}\}$ such that $\|\tilde{u} - v\| < \|u - v\|$. 3) if $\bar{v} \in V$ is a point that satisfies $z \cdot \bar{v} < z \cdot v$, then there is a point \tilde{v} on the line segment, $\text{co}\{v, \bar{v}\}$ such that $\|u - \tilde{v}\| < \|u - v\|$. 4) (u, v) solves NPP if and only if $g(u, v) = 0$.

Proof: a) This follows from (12) and (14).

b) Define a real variable t and a function, $\phi(t) = \|u + t(\bar{u} - u) - v\|^2$. ϕ describes the variation of $\|\hat{u} - v\|^2$ as a generic point \hat{u} varies from u to \bar{u} over the line segment joining them. Since $\phi'(0) = 2z \cdot (\bar{u} - u) < 0$, the result follows.

c) The proof is along similar lines as that of b).

d) First let $g(u, v) = 0$. Let $\hat{u} \in U, \hat{v} \in V$. Since $g_U(-z, u) = 0$ and $g_V(z, v) = 0$, we have $z \cdot u \leq z \cdot \hat{u}$ and

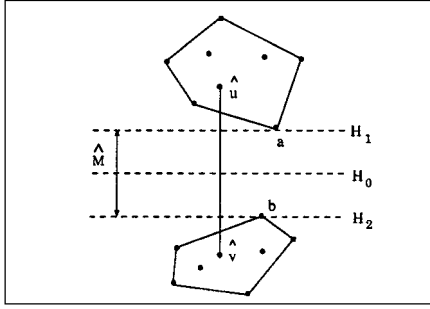


Fig. 3. A situation satisfying (15). Here: $a = s_U(-\hat{z})$; $b = s_V(\hat{z})$; $H_1 : \hat{z} \cdot z = -h_U(-\hat{z})$; $H_2 : \hat{z} \cdot z = h_V(\hat{z})$; and $H_0 : \hat{z} \cdot z = 0.5(h_V(\hat{z}) - h_U(-\hat{z}))$.

$z \cdot v \geq z \cdot \hat{v}$. Subtracting the two inequalities we get $\|z\|^2 \leq z \cdot \hat{z}$ where $\hat{z} = \hat{u} - \hat{v}$. Now

$$\|z\|^2 \leq \|z\|^2 + \|z - \hat{z}\|^2 = \|\hat{z}\|^2 + 2(\|z\|^2 - \hat{z} \cdot z) \leq \|\hat{z}\|^2$$

and hence (u, v) is optimal for NPP. On the other hand, if (u, v) is optimal for NPP, we can set $\bar{u} = s_U(-z)$, $\bar{v} = s_V(z)$ and use results a) and b) to show that $g(u, v) = 0$. \square

IV. STOPPING CRITERIA FOR NPP ALGORITHMS

When a numerical algorithm is employed to solve any of the problems mentioned earlier, approximate stopping criteria need to be employed. In SVM design a dual formulation (such as SVM-NV-DUAL and NPP) is solved instead of the primal (such as SVM-NV) because of computational ease. However, it has to be kept in mind that it is the primal solution that is of final interest and that care is needed to ensure that the numerical algorithm has reached an approximate primal solution with a guaranteed specified closeness to the optimal primal solution. It is easy to give an example [14] to show that simply stopping when the dual approximate solution has reached the dual optimal solution within a good accuracy can be dangerous, as far as the solution of SVM-NV is concerned.

Suppose we have an algorithm for NPP that iteratively improves its approximate solution, $\hat{u} \in U$, $\hat{v} \in V$ in such a way that $\hat{u} \rightarrow u^*$ and $\hat{v} \rightarrow v^*$, where (u^*, v^*) is a solution of NPP. Recall the function g defined in (14). By part (d) of Theorem 2 and the fact that g is a continuous function $g(\hat{u}, \hat{v}) \rightarrow 0$. Also, $\|\hat{z}\| \rightarrow \|z^*\|$ where $\hat{z} = \hat{u} - \hat{v}$ and $z^* = u^* - v^*$. By assumption A1, $\|\hat{z}\| \geq \|z^*\| > 0$. Thus we also have $g(\hat{u}, \hat{v})/\|\hat{z}\|^2 \rightarrow 0$. Suppose ϵ is a specified accuracy parameter satisfying $0 < \epsilon < 1$, and that it is desired to stop the algorithm when we have found a (\hat{w}, \hat{b}) pair that is feasible to SVM-NV and $\|w^*\| \geq (1 - \epsilon)\|\hat{w}\|$ where (w^*, b^*) is the optimal solution of SVM-NV. We will show that this is possible if we stop the dual algorithm when (\hat{u}, \hat{v}) satisfies

$$g(\hat{u}, \hat{v}) \leq \epsilon \|\hat{z}\|^2. \quad (15)$$

Fig. 3 geometrically depicts the situation. Let \hat{m} be the margin corresponding to the direction, \hat{z} . Clearly

$$\hat{m} = \frac{-h_U(-\hat{z}) - h_V(\hat{z})}{\|\hat{z}\|}. \quad (16)$$

Note that, since $g(\hat{u}, \hat{v}) = \|\hat{z}\|^2 + h_U(-\hat{z}) + h_V(\hat{z}) \leq \epsilon \|\hat{z}\|^2$, we have

$$-h_U(-\hat{z}) - h_V(\hat{z}) \geq (1 - \epsilon)\|\hat{z}\|^2. \quad (17)$$

Thus, $\hat{M} \geq (1 - \epsilon)\|\hat{z}\| > 0$. The determination of \hat{w} that is consistent with the feasibility of SVM-NV can be easily found by setting

$$\hat{w} = \gamma \hat{z} \quad (18)$$

and choosing γ such that $\hat{M} = 2/\|\hat{w}\|$. Using (16) we get

$$\gamma = \frac{2}{(-h_U(-\hat{z}) - h_V(\hat{z}))}. \quad (19)$$

Since the equation of the central separating hyperplane, H_0 has to be of the form, $\hat{w} \cdot z + \hat{b} = 0$, we can also easily get the expression for \hat{b} as

$$\hat{b} = \frac{h_V(\hat{z}) - h_U(-\hat{z})}{h_V(\hat{z}) + h_U(-\hat{z})}. \quad (20)$$

Using (16) and (17) and the fact that $m^* = \|z^*\|$, we get

$$\begin{aligned} \frac{\|w^*\|}{\|\hat{w}\|} &= \frac{\hat{m}}{m^*} = \frac{-h_U(-\hat{z}) - h_V(\hat{z})}{\|\hat{z}\| \|z^*\|} \\ &= \frac{-h_U(-\hat{z}) - h_V(\hat{z})}{\|\hat{z}\|^2} \cdot \frac{\|\hat{z}\|}{\|z^*\|} \geq (1 - \epsilon). \end{aligned}$$

To derive a bound for $\|z^*\|/\|\hat{z}\|$, note that $\|z^*\| = M^* \geq \hat{M} = (-h_U(-\hat{z}) - h_V(\hat{z}))/\|\hat{z}\|$. Then

$$\frac{\|z^*\|}{\|\hat{z}\|} \geq \frac{-h_U(-\hat{z}) - h_V(\hat{z})}{\|\hat{z}\|^2} \geq (1 - \epsilon).$$

Thus we have proved the following important result.

Theorem 3: Let (u^*, v^*) be a solution of NPP, $z^* = u^* - v^*$, and (w^*, b^*) be the optimal solution of SVM-NV. Let $0 < \epsilon < 1$. Suppose $\hat{u} \in U$, $\hat{v} \in V$, $\hat{z} = \hat{u} - \hat{v}$ and (15) holds. Then (\hat{w}, \hat{b}) as defined by (18)–(20) is feasible for SVM-NV and

$$\min \left(\frac{\|w^*\|}{\|\hat{w}\|}, \frac{\|\hat{M}\|}{\|M^*\|}, \frac{\|z^*\|}{\|\hat{z}\|} \right) \geq (1 - \epsilon).$$

If one is particularly interested in getting a bound on the cost function of SVM-NV then it can be easily obtained

$$\frac{\|w^*\|^2}{\|\hat{w}\|^2} = \left(\frac{\|w^*\|}{\|\hat{w}\|} \right)^2 \geq (1 - \epsilon)^2 \geq (1 - 2\epsilon).$$

A similar bound can be obtained for $\|z^*\|^2/\|\hat{z}\|^2$.

All these discussions point to the important fact that (15) can be used to effectively terminate a numerical algorithm for solving NPP.

V. ITERATIVE ALGORITHMS FOR NPP

NPP has been well studied in the literature, and a number of good algorithms have been given for it [10], [18], [29], [2], [15]. Best general-purpose algorithms for NPP such as Wolfe's algorithm [29] terminate within a finite number of steps; however they require expensive matrix storage and matrix operations in each step that makes them unsuitable for use in large

SVM design. (It is interesting to point out that the active set method used by Kaufman [13] to solve SVM's is *identical* to Wolfe's algorithm when both are used to solve SVM-NV.) Iterative algorithms that need minimal memory (i.e., memory size needed is linear in m , the number of training vectors), but which only reach the solution asymptotically as the number of iterations goes to infinity, seem to be better suited for SVM design. In this section we will take up some such algorithms for investigation.

A. Gilbert's Algorithm

Gilbert's algorithm [10] was one of the first algorithms suggested for solving NPP. It was originally devised to solve certain optimal control problems. Later its modifications have found good use in pattern recognition and robotics [29], [11]. In this section we briefly describe the algorithm and point to how it can be adapted for SVM classifier design.

Let $\{z_k\}$, I , J , U , and V be as in the definition of NPP. Let Z denote the Minkowski set difference of U and V [16], i.e.,

$$Z = U \ominus V = \{u - v : u \in U, v \in V\}.$$

Clearly, NPP is equivalent to the following minimum norm problem:

$$\min\{\|z\| : z \in Z\}. \quad (\text{MNP})$$

Unlike NPP this problem has a unique solution. MNP is very much like NPP and seems like a simpler problem than NPP, but it is only superficially so. Z is the convex polytope, $Z = \text{co } W$, where

$$W = \{z_i - z_j : i \in I, j \in J\} \quad (21)$$

a set with $m_1 m_2$ points ($m_1 = |I|$, $m_2 = |J|$). To see this, take two general points, $u \in U$ and $v \in V$ with the representation

$$\begin{aligned} u &= \sum_{i \in I} \beta_i z_i, \quad \sum_{i \in I} \beta_i = 1, \quad \beta_i \geq 0 \quad \forall i \in I \\ v &= \sum_{j \in J} \beta_j z_j, \quad \sum_{j \in J} \beta_j = 1, \quad \beta_j \geq 0 \quad \forall j \in J. \end{aligned}$$

Then write $z = u - v$ as

$$\begin{aligned} z &= \sum_{i \in I} \beta_i z_i - \sum_{j \in J} \beta_j z_j \\ &= \left(\sum_{j \in J} \beta_j \right) \sum_{i \in I} \beta_i z_i - \left(\sum_{i \in I} \beta_i \right) \sum_{j \in J} \beta_j z_j \\ &= \sum_{i \in I} \sum_{j \in J} \beta_i \beta_j (z_i - z_j) \end{aligned}$$

and note the fact that $\sum_{i \in I} \sum_{j \in J} \beta_i \beta_j = (\sum_{i \in I} \beta_i) (\sum_{j \in J} \beta_j) = 1$. In general it is possible for Z to have $O(m_1 m_2)$ vertices. Since $m_1 m_2$ can become very large, it is definitely not a good idea to form W and then define Z using it.

Gilbert's algorithm is a method of solving MNP that does not require the explicit formation of W . Its steps require only the evaluations of the support properties, h_Z and s_Z . Fortunately these functions can be computed efficiently

$$\begin{aligned} h_Z(\eta) &= h_W(\eta) = \max_{i \in I, j \in J} \eta \cdot (z_i - z_j) \\ &= \max_{i \in I} (\eta \cdot z_i) + \max_{j \in J} (-\eta \cdot z_j) \\ &= h_U(\eta) + h_V(-\eta). \end{aligned} \quad (22)$$

$$s_Z(\eta) = s_U(\eta) - s_V(-\eta). \quad (23)$$

Thus, for a given η the computation of $h_Z(\eta)$ and $s_Z(\eta)$ requires only $O(m_1 + m_2)$ time.

An optimality criterion for MNP can be easily derived as in Section III. This is stated in the following theorem. We will omit its proof since it is very much along the lines of proof of Theorem 2. The g function of (12), as applied to Z plays a key role.

Theorem 4: Suppose $z \in Z$. Then the following hold. 1) $g_Z(-z, z) \geq 0$. 2) If \bar{z} is any point in Z such that $\|z\|^2 - z \cdot \bar{z} > 0$, there is a point \tilde{z} in the line segment, $\text{co}\{z, \bar{z}\}$ satisfying $\|\tilde{z}\| < \|z\|$. (c) z solves MNP if and only if $g_Z(-z, z) = 0$. \square

Gilbert's algorithm is based on the results in the above theorem. Suppose we start with some $z \in Z$. If $g_Z(-z, z) = 0$ then $z = z^*$, the solution of MNP. On the other hand, if $g_Z(-z, z) > 0$, then $\bar{z} = s_Z(-z)$ satisfies $\|z\|^2 - z \cdot \bar{z} > 0$. By part 2) of the theorem, then, searching on the line segment connecting z to \bar{z} yields a point whose norm is smaller than $\|z\|$. These observations yield Gilbert's algorithm.

Gilbert's Algorithm for Solving MNP

0) Choose $z \in Z$.

1) Compute $h_Z(-z)$ and $g_Z(-z, z)$. If $g_Z(-z, z) = 0$ stop with $z^* = z$; else, set $\bar{z} = s_Z(-z)$.

2) Compute \tilde{z} , the point on the line segment joining z and \bar{z} which has least norm, set $z = \tilde{z}$ and go back to Step 1).

An efficient algorithm for computing the point of least norm on the line segment joining two points is given in the Appendix. Gilbert showed that, if the algorithm does not stop with z^* at Step 1) within a finite number of iterations, then $z \rightarrow z^*$ asymptotically, as the number of iterations goes to infinity.

To adapt Gilbert's algorithm for SVM design it is important to note that z can be represented as $z = \sum_t \gamma_t w_t$, where $\sum_t \gamma_t = 1$, $\gamma_t \geq 0 \forall t$ and $w_t \in W$, i.e., w_t has the form $w_t = z_{i(t)} - z_{j(t)}$. Hence only $\{\gamma_t\}$ and $\{(i(t), j(t))\}$ need to be stored and maintained in order to represent z . Maintaining and updating cache for the inner products, $z \cdot z_{i(t)}$ and $z \cdot z_{j(t)}$ improves efficiency. More details are given in [14].

We implemented Gilbert's algorithm and tested its performance on a variety of classification problems. While the algorithm always makes rapid movement toward the solution during its initial iterations, on many problems it was very slow as it approached the final solution. Also, suppose there is a w_t which gets picked up by the algorithm during its initial stages, but which is not needed at all in representing the final solution (i.e., $z^* \cdot w_t > z^* \cdot z^*$), then the algorithm is slow in driving the corresponding γ_t to zero. Because of these reasons, Gilbert's algorithm, by itself, is not very efficient for solving the NPP's that arise in SVM classifier design.

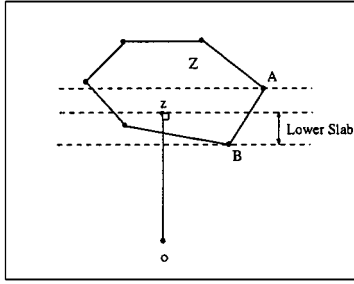


Fig. 4. Definition of Δ . Here: $A = w_{t \min}$; $B = s_Z(-z)$; and $\Delta(z) = (-z \cdot B) - (-z \cdot A)$.

B. Mitchell–Dem'yanov–Malozemov Algorithm

Many years after Gilbert's work, Mitchell *et al.* [18] independently suggested a new algorithm for MNP. We will refer to their algorithm as the MDM algorithm. Unlike Gilbert's algorithm, MDM algorithm fundamentally uses the representation, $z = \sum_t \gamma_t w_t$ in its basic operation. When $z = z^*$, it is clear that $z \cdot w_t = z \cdot z$ for all t which have $\gamma_t > 0$. If $g_Z(-z, z) = 0$ then $z = z^*$ and hence the above condition automatically holds. However, at an approximate point, $z \neq z^*$, the situation is different. The point, z could be very close to z^* and yet, there could very well exist one or more w_t such that

$$z \cdot w_t \gg z \cdot z \quad \text{and} \quad \gamma_t > 0 \quad (24)$$

(though γ_t is small). For efficiency of representation as well as algorithm performance, it is a good idea to eliminate such t from the representation of z . With this in mind, define the function (see Fig. 4)

$$\Delta(z) = h_Z(-z) - \min_{t: \gamma_t > 0} (-z \cdot w_t). \quad (25)$$

Clearly, $\Delta(z) \geq 0$, and, $\Delta(z) = 0$ if and only $g_Z(-z, z) = 0$, i.e., $z = z^*$.

At each iteration, MDM algorithm attempts to decrease $\Delta(z)$ whereas Gilbert's algorithm only tries to decrease $g_Z(-z, z)$. This can be seen geometrically in Fig. 4. MDM algorithm tries to crush the total slab toward zero while Gilbert's algorithm only attempts to push the lower slab to zero. This is the essential difference between the two algorithms. Note that, if there is a t satisfying (24) then MDM algorithm has a much better potential to quickly eliminate such t from the representation.

Let us now describe the main step of MDM algorithm. Let t_{\min} be an index such that

$$-z \cdot w_{t_{\min}} = \min_{t: \gamma_t > 0} (-z \cdot w_t)$$

Let $d = s_Z(-z) - w_{t_{\min}}$. With the reduction of $\Delta(z)$ in mind, MDM algorithm looks for improvement of norm value along the direction d at z . For $\lambda > 0$, let $z = z + \lambda d$ and $\phi(\lambda) = \|z\|^2$. Since $\phi'(0) = 2z \cdot d = -2\Delta(z)$, it is easy to see that moving along d will strictly decrease $\|z\|$ if $z \neq z^*$. Since, in the representation for z , $\gamma_{t_{\min}}$ decreases during this movement, λ has to be limited to the interval defined by $0 \leq \lambda \leq \gamma_{t_{\min}}$. Define $\bar{z} = z + \gamma_{t_{\min}} d$. So, the basic iteration of MDM algorithm consists of finding the point of minimum norm on the line segment joining z and \bar{z} .

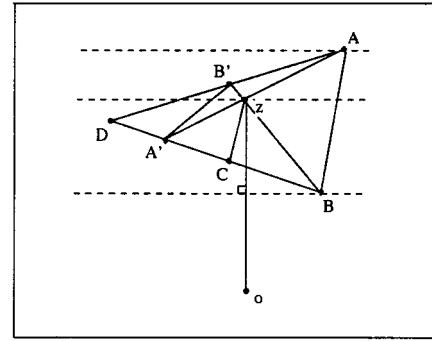


Fig. 5. Illustrating the various ideas for improvement. Here: $A = w_{t \min}$; $B = s_Z(-z) = w_{t \max}$; zB = Gilbert line segment; zC = MDM line segment (zC is parallel to AB); T_1 = triangle zCB ; T_2 = triangle $zA'B$; Q_3 = quadrilateral $ABA'B'$; and, T_4 = triangle DAB .

We implemented and tested MDM algorithm. It works faster than Gilbert's algorithm, especially in the end stages when z approaches z^* . Algorithms which are much faster than MDM algorithm can be designed using the following two observations: (1) it is easy to combine the ideas of Gilbert's algorithm and MDM algorithm into a hybrid algorithm which is faster; and (2) by working directly in the space where U and V are located it is easy to find just two elements of $\{z_k\}$ that are used to modify the z , and at the same time, keep the essence of the hybrid algorithm mentioned above. In the next section we describe the ideas behind the hybrid algorithm. Then we take up details of the final algorithm incorporating both observations in Section VI.

C. A Hybrid Algorithm

A careful look at MDM algorithm shows that the main computational effort of each iteration is associated with the computation of $h_Z(-z)$, the determination of t_{\min} and the updating of γ_t and of the inner products cache. Hence we can quite afford to make the remaining steps of an iteration a little more complex, provided that leads to some gain and does not disturb the determination of t_{\min} , t_{\max} , and the updatings. Since Gilbert's algorithm mainly requires the computation of $h_Z(-z)$ and $s_Z(-z)$, which are anyway computed by the MDM algorithm, it is possible to easily insert the main idea of Gilbert's algorithm into MDM algorithm so as to improve it. A number of ideas emerge from this attempt. We will first describe these ideas and then comment on their goodness.

The situation at a typical iteration is shown in Fig. 5. As before, let us take the representation for z as

$$z = \sum_t \gamma_t w_t.$$

Without loss of generality let us assume that $s_Z(-z) = w_{t_{\max}}$ for some index, t_{\max} . (If $s_Z(-z)$ is not equal to any w_t , we can always include it as one more w_t and set the corresponding γ_t to zero without affecting the representation of z in anyway.) The points $w_{t_{\min}}$ and $w_{t_{\max}}$ are, respectively, shown as A and B in the figure. The point, $z + \gamma_{t_{\min}}(w_{t_{\max}} - w_{t_{\min}})$ (i.e., the \bar{z} of MDM algorithm) is shown as C .

Idea 1: At Step 2 of each iteration of the MDM algorithm, take \tilde{z} to be the point of minimum norm on T_1 , the triangle formed by z , C and B .

An algorithm for computing the point of minimum norm on a triangle is given in the Appendix. Since the line segment connecting z and C (the one on which MDM algorithm finds the minimum point) is a part of T_1 , this idea will locally perform (in terms of decreasing norm) at least as well as MDM algorithm. Note also that, since the minimum norm point on T_1 can be expressed as a linear combination of z , A and B , the cost of updating γ_t and the inner products cache is the same as that of MDM algorithm.

For a given \bar{t} , let us define

$$z(\bar{t}) = \frac{1}{1 - \gamma_{\bar{t}}}(z - \gamma_{\bar{t}}w_{\bar{t}}).$$

Clearly, $z(\bar{t})$ is the point in Z obtained by removing $w_{\bar{t}}$ from the representation of z and doing a renormalization. Let us define $A' = z(t_{\min})$; see Fig. 5. Since

$$C = z + \gamma_{t_{\min}}(w_{t_{\max}} - w_{t_{\min}}) = (1 - \gamma_{t_{\min}})A' + \gamma_{t_{\min}}B$$

it follows that C lies on the line segment joining A' and $w_{t_{\max}}$.

Idea 2: At Step 2 of each iteration of MDM algorithm, take \tilde{z} to be the point of minimum norm on T_2 , the triangle formed by z , A' and B .

Because T_2 contains T_1 , Idea 2 will produce a \tilde{z} whose norm is less than or equal to that produced by Idea 1.

One can carry this idea further to generate two more ideas. Let $B' = z(t_{\max})$ and D be the point of intersection of the line joining A , B' and the line joining B , A' ; it is easily checked that D is the point obtained by removing both the indexes, t_{\min} and t_{\max} from the representation of z and then doing a renormalization, i.e.,

$$D = \frac{1}{(1 - \gamma_{t_{\min}} - \gamma_{t_{\max}})}(z - \gamma_{t_{\min}}w_{t_{\min}} - \gamma_{t_{\max}}w_{t_{\max}}).$$

Let Q_3 denote the quadrilateral formed by the convex hull of $\{A, B, A', B'\}$ and T_4 be the triangle formed by D , A , and B . Clearly, $T_2 \subset Q_3$. Also, along the lines of an earlier argument, it is easy to show that $Q_3 \subset T_4$.

Idea 3: At Step 2 of each iteration of MDM algorithm, take \tilde{z} to be the point of minimum norm on Q_3 .

Idea 4: At Step 2 of each iteration of MDM algorithm, take \tilde{z} to be the point of minimum norm on T_4 .

We implemented and tested all of these ideas. Idea 1 gives a very good overall improvement over MDM algorithm and Idea 2 improves it further, a little more. However, we have found that, in overall performance, Idea 3 performs somewhat worse compared to Idea 2, and Idea 4 performs even worse! (We do not have a clear explanation for these performances.) On the basis of these empirical observations, we recommend Idea 2 to be the best one for use in modifying MDM algorithm.

VI. A FAST ITERATIVE ALGORITHM FOR NPP

In this section we will give an algorithm for NPP directly in the space in which U and V are located. The key idea is motivated by considering Gilbert's algorithm. Let $u \in U$ and $v \in V$ be the approximate solutions at some given iteration and $z = u - v$.

We say that an index, k satisfies condition \mathcal{A} at (u, v) if

$$k \in I \Leftrightarrow -z \cdot z_k + z \cdot u \geq \frac{\epsilon}{2}\|z\|^2 \quad (26)$$

$$k \in J \Leftrightarrow z \cdot z_k - z \cdot v \geq \frac{\epsilon}{2}\|z\|^2. \quad (27)$$

Suppose k satisfies (26). If \tilde{u} is the point on the line segment joining u and z_k that is closest to v , then, by the relation (A.8) given in the Appendix we get an appreciable decrease in the objective function of NPP

$$\|u - v\|^2 - \|\tilde{u} - v\|^2 \geq \min\left(\tilde{\epsilon}, \frac{\tilde{\epsilon}}{L^2}\right) \quad (28)$$

where $\tilde{\epsilon} = 0.5\epsilon\|z\|^2$. A parallel comment can be made if k satisfies (27). On the other hand, if we are unable to find an index k satisfying \mathcal{A} then (recall the definitions from Section III) we get

$$g_U(-z, u) \leq \frac{\epsilon}{2}\|z\|^2, \quad g_V(z, v) \leq \frac{\epsilon}{2}\|z\|^2, \quad g(u, v) \leq \epsilon\|z\|^2 \quad (29)$$

which, by Theorem 3, is a good way of stopping. These observations lead us to the following generic algorithm, which gives ample scope for generating a number of specific algorithms from it. The efficient algorithm that we will describe soon after is one such special instance.

Generic Iterative Algorithm for NPP:

0) Choose $u \in U$, $v \in V$ and set $z = u - v$.

- 1) Find an index k satisfying \mathcal{A} . If such an index cannot be found, stop with the conclusion that the approximate optimality criterion, (29) is satisfied. Else go to Step 2) with the k found.
- 2) Choose two convex polytopes, $\tilde{U} \subset U$ and $\tilde{V} \subset V$ such that $u \in \tilde{U}$, $v \in \tilde{V}$ and

$$z_k \in \tilde{U} \quad \text{if } k \in I, \quad z_k \in \tilde{V} \quad \text{if } k \in J. \quad (30)$$

Compute (\tilde{u}, \tilde{v}) to be a pair of closest points minimizing the distance between \tilde{U} and \tilde{V} . Set $u = \tilde{u}$, $v = \tilde{v}$ and go back to Step 1). \square

Suppose, in Step 2) we do the following. If $k \in I$, choose \tilde{U} as the line segment joining u and z_k , and $\tilde{V} = \{v\}$. Else, if $k \in J$, choose \tilde{V} as the line segment joining v and z_k , and $\tilde{U} = \{u\}$. Then the algorithm is close in spirit to Gilbert's algorithm. Note however, that here only one index, k plays a role in the iteration, whereas Gilbert's algorithm requires two indexes (one each from I and J) to define \tilde{z} . In a similar spirit, by appropriately choosing \tilde{U} and \tilde{V} , the various ideas of Section V-C can be extended while involving only two indexes. Before we discuss a detailed algorithm, we prove convergence of the generic algorithm.

Theorem 5: The generic iterative algorithm terminates in Step 1 with a (u, v) satisfying (29), after a finite number of iterations.

The proof is easy. First, note that (28) holds because of the way \tilde{U} and \tilde{V} are chosen in Step 2). Since, by assumption A1, $\|z\|^2$ is uniformly bounded below by zero, there is a uniform decrease in $\|z\|^2$ at each iteration. Since the minimum distance

between U and V is nonnegative the iterations have to terminate within a finite number of iterations.

Consider the situation in Step 2) where an index k satisfying \mathcal{A} is available. Suppose the algorithm maintains the following representations for u and v :

$$u = \sum_{i \in \hat{I}} \beta_i z_i, \quad v = \sum_{j \in \hat{J}} \beta_j z_j \quad (31)$$

where $\hat{I} \subset I$, $\hat{J} \subset J$, and

$$\beta_k > 0 \quad \forall k \in \hat{I} \cup \hat{J}, \quad \sum_{i \in \hat{I}} \beta_i = 1, \quad \sum_{j \in \hat{J}} \beta_j = 1.$$

We will refer to a z_k , $k \in \hat{I} \cup \hat{J}$ as a *support vector*, consistent with the terminology adopted in the literature. Let us define $i \min$ and $j \min$ as follows (see Fig. 6):

$$\begin{aligned} i \min &= \arg \min \{-z \cdot z_i : i \in \hat{I}, \beta_i > 0\} \\ j \min &= \arg \min \{z \cdot z_j : j \in \hat{J}, \beta_j > 0\}. \end{aligned} \quad (32)$$

(Though redundant, we have stated the conditions, $\beta_i > 0$ and $\beta_j > 0$ to stress their importance. Also, in a numerical implementation, an index $k \in \hat{I} \cup \hat{J}$ having $\beta_k = 0$ could occur numerically. Unless such indexes are regularly cleaned out of $\hat{I} \cup \hat{J}$, it is best to keep these positivity checks in (32).)

There are a number of ways of combining u , v , z_k and one index from $\{i \min, j \min\}$ and then doing operations along ideas similar to those in Section V-C to create a variety of possibilities for \tilde{U} and \tilde{V} . We have implemented and tested some of the promising ones and empirically arrived at one final choice, which is the only one that we will describe in detail.

First we set $k \min$ as follows: $k \min = i \min$ if $-z \cdot z_{i \min} + z \cdot u < z \cdot z_{j \min} - z \cdot v$; else $k \min = j \min$. Then Step 2) is carried out using one of the following four cases, depending on k and $k \min$. See Fig. 7 for a geometric description of the cases.

Case 1: $k \in I$, $k \min \in I$. Let C' be the point in U obtained by removing $z_{k \min}$ from the representation of u , i.e.,

$$C' = \frac{1}{(1 - \beta_{k \min})} (u - \beta_{k \min} z_{k \min}) = u + \mu(u - z_{k \min}) \quad (33)$$

where $\mu = \beta_{k \min} / (1 - \beta_{k \min})$. Choose: \tilde{U} to be the triangle formed by u , C' and z_k ; and, $\tilde{V} = \{v\}$.

Case 2: $k \in J$, $k \min \in J$. Let D' be the point in V obtained by removing $z_{k \min}$ from the representation of v , i.e.,

$$D' = \frac{1}{(1 - \beta_{k \min})} (v - \beta_{k \min} z_{k \min}) = v + \mu(v - z_{k \min}) \quad (34)$$

where $\mu = \beta_{k \min} / (1 - \beta_{k \min})$. Choose: \tilde{V} to be the triangle formed by v , D' and z_k ; and, $\tilde{U} = \{u\}$.

Case 3: $k \in I$, $k \min \in J$. Choose: \tilde{U} to be the line segment joining u and z_k ; and \tilde{V} to be the line segment joining v and D' where D' is as in (34).

Case 4: $k \in J$, $k \min \in I$. Choose: \tilde{V} to be the line segment joining v and z_k ; and \tilde{U} to be the line segment joining u and C' where C' is as in (33).

This defines the basic operation of the algorithm. Efficient algorithms for doing nearest point computations involving a triangle and line segments can be found in the Appendix and

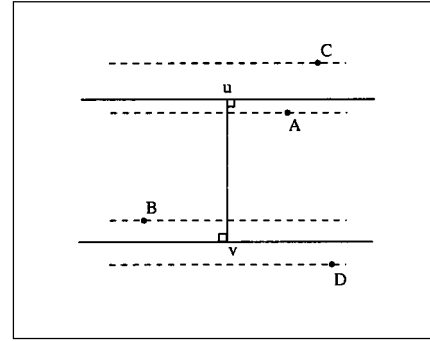


Fig. 6. Illustration of $i \min$, $j \min$, $i \max$, and $j \max$. Here $C = z_{i \min}$, $D = z_{j \min}$, $A = z_{i \max}$ and $B = z_{j \max}$.

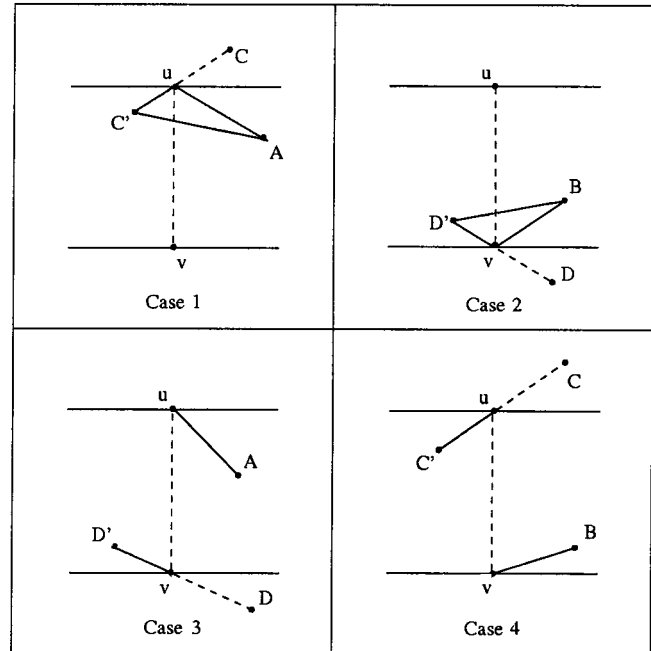


Fig. 7. A geometric description of the four cases. For Cases 1 and 3, $A = z_k$; for Cases 2 and 4, $B = z_k$.

[17]. Tremendous improvement in efficiency can be achieved by doing two things: 1) maintaining and updating caches for some variables and 2) interchanging operations between the support vector and nonsupport vector sets. These ideas are directly inspired from those used by Platt in his SMO algorithm. Let us now go into these details.

First let us discuss the need for maintaining cache for some variables. A look at (26), (27), and (32) shows that $u \cdot z_k$, $v \cdot z_k$, $u \cdot u$, $u \cdot v$, $v \cdot v$ and $z \cdot z$ are important variables. If any of these variables is to be computed from scratch, it is expensive; for example, computation of $u \cdot z_k$ (for a single k) using (31) requires the evaluation of m_1 kernel computations and $O(m_1)$ computation (where $m_1 = |\hat{I}|$, the number of elements in \hat{I}) and, to compute $i \min$ of (32) this has to be repeated m_1 times! Therefore it is important to maintain caches for these variables. Let us define: $e_u(k) = u \cdot z_k$, $e_v(k) = v \cdot z_k$, $\delta_{uu} = u \cdot u$, $\delta_{vv} = v \cdot v$, $\delta_{uv} = u \cdot v$, and $\delta_{zz} = z \cdot z$.

Among these cache variables, e_u and e_v are the only vectors. Since the algorithm spends much of its operation adjusting the β

of the support vectors, it is appropriate to bring in the nonsupport vectors only rarely to check satisfaction of optimality criteria. In that case, it is efficient to maintain $e_u(k)$ and e_v only for $k \in \hat{I} \cup \hat{J}$ and, for other indexes compute these quantities from scratch whenever needed.

As mentioned in the last paragraph, it is important to spend most of the iterations using $k \in \hat{I} \cup \hat{J}$. With this in mind, we define two types of loops. The *Type I* loop goes *once* over all $k \notin \hat{I} \cup \hat{J}$, sequentially, choosing one at a time, checking condition \mathcal{A} , and, if it is satisfied, doing one iteration, i.e., Step 2) of the generic algorithm. The *Type II* loop operates only with $k \in \hat{I} \cup \hat{J}$, doing the iterations many many times until certain criteria are met. Whereas, in Platt's SMO algorithm, the iterations run over one support vector index at a time, we have adopted a different strategy for Type II loop. Let us define i_{\max} , j_{\max} , and k_{\max} as follows:

$$\begin{aligned} i_{\max} &= \arg \max \{-z \cdot z_i : i \in \hat{I}\} \\ j_{\max} &= \arg \max \{z \cdot z_j : j \in \hat{J}\} \end{aligned} \quad (35)$$

$$k_{\max} = \begin{cases} i_{\max}, & \text{if } -z \cdot z_{i_{\max}} + z \cdot u > z \cdot z_{j_{\max}} - z \cdot v \\ j_{\max}, & \text{otherwise.} \end{cases} \quad (36)$$

These can be efficiently computed mainly because of the caching of $e_u(k)$ and $e_v(k)$ for all $k \in \hat{I} \cup \hat{J}$. A basic Type II iteration consists of determining k_{\max} and doing one iteration using $k = k_{\max}$. These iterations are continued until the approximate optimality criteria are satisfied over the support vector set, i.e., when

$$\begin{aligned} g_U^{\text{supp}} &= -z \cdot z_{i_{\max}} + z \cdot u \leq \frac{\epsilon}{2} \|z\|^2 \\ g_V^{\text{supp}} &= z \cdot z_{j_{\max}} - z \cdot v \leq \frac{\epsilon}{2} \|z\|^2. \end{aligned} \quad (37)$$

In the early stages of the algorithm (say a few of the TypeI-then-TypeII rounds) a “nearly accurate” set of support vector indexes get identified. Until that happens, we have found that it is wasteful to spend too much time doing Type II iterations. Therefore we have adopted the following strategy. If, at the point of entry to a Type II loop, the percentage difference of the number of support vectors as compared with the value at the previous entry is greater than 2%, then we limit the number of Type II iterations to m , the total number of training pairs in the problem. This is done to roughly make the cost of Type I and Type II loops equal. In any case, we have found it useful to limit the number of Type II iterations to $10m$. The algorithm is stopped when (37) holds, causing the algorithm to exit Type II loop, and, in the Type I loop that follows there is no k satisfying condition \mathcal{A} .

Full details concerning the actual implementation of the full algorithm and a pseudocode for the algorithm can be found in [14]. Using them it is extremely easy to develop a working code in short time.

VII. COMPUTATIONAL EXPERIMENTS

In this section we empirically evaluate the performance of the nearest point algorithm (NPA) described in the last section.

TABLE I
PROPERTIES OF DATA SETS

Data Set	σ^2	n	m	m_{val}	m_{test}
Checkers	0.5	2	465	155	155
Adult-1	10.0	123	1605	535	535
Adult-4	10.0	123	4781	1594	1594
Adult-7	10.0	123	16100		

Since Platt's SMO algorithm is currently one of the fastest algorithms for SVM design, we compare NPA against SMO. Apart from comparison of computational cost we also compare the two algorithms on how well they generalize.

We implemented both algorithms in Fortran and ran them using *g77* on a 200-MHz Pentium machine. SMO was implemented exactly along the lines suggested by Platt in [20].

Although we have compared the methods on a number of problems, here we only report the (representative) performance on two benchmark problems: Checkers data and UCI Adult data [25], [21]. We created the Checkers data by generating a random set of points on a 4×4 checkers grid [14]. The Adult data set was taken from Platt's web page [21]. In the case of Adult data set, the inputs are represented in a special binary format, as used by Platt in the testing of SMO. To study scaling properties as training data grows, Platt did staged experiments on the Adult data. We have used only the data from the first, fourth and seventh stages. For training we used exactly the sets given by Platt. For validation and testing we used subsets of the large validation and test sets given by Platt. The Gaussian kernel

$$K(x_i, x_j) = \exp(-0.5 \|x_i - x_j\|^2 / \sigma^2)$$

was used in all experiments. The σ^2 values employed, n , the dimension of the input, and m , m_{val} , and m_{test} , the sizes of the training, validation and test sets, are given in Table I. The σ^2 values given in this table were chosen as follows. For the Adult data the σ^2 values are the same as those used by Platt in his experiments on SMO; for the Checkers data, we chose σ^2 suitably to get good generalization.

We first applied the three algorithms on the training sets and compared the computational costs. The algorithms apply to different SVM formulations: SMO solves SVM-VL whereas NPA solves SVM-VQ. Hence, to do a proper cross comparison of the methods, we did the following. Let M denote the final margin obtained by a solution in the solution space. For each data set, we chose two different ranges for C and \tilde{C} values in such a way that they roughly cover the same range of M values. (It may be noted that M has an inverse relationship with C and \tilde{C} .) Then we ran the methods on a bunch of C and \tilde{C} values sampled from those ranges. Such a study is important for another reason, as well. When a particular method is used for SVM design, C or \tilde{C} is usually unknown, and it has to be chosen by trying a number of values and using a validation set. Therefore, fast performance of a method in a range of C or \tilde{C} values is important. Whether or not a particular method has such a performance gets revealed clearly in our experiments. A common stopping criterion was chosen for both methods; see [14] for details.

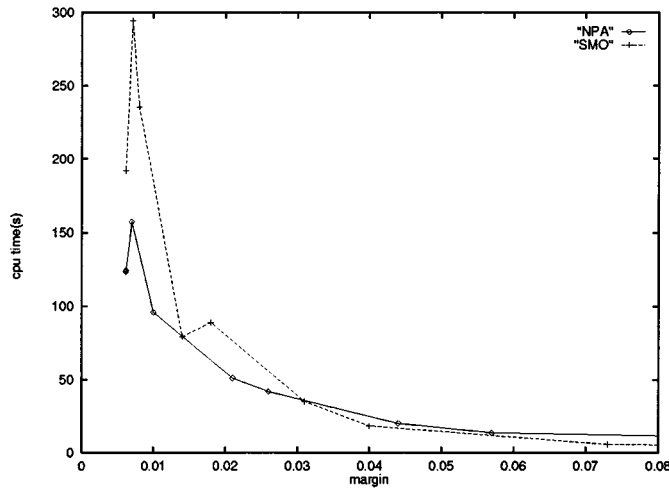


Fig. 8. Checkers data: CPU time (in seconds) shown as a function of the margin M .

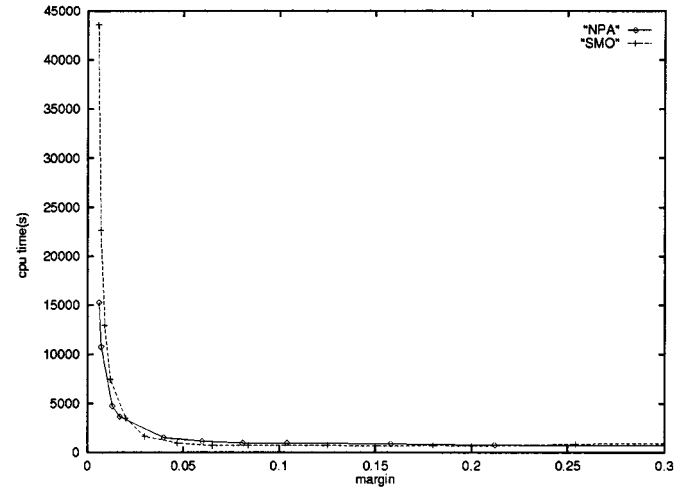


Fig. 10. Adult-4 data: CPU Time (in seconds) shown as a function of the margin M .

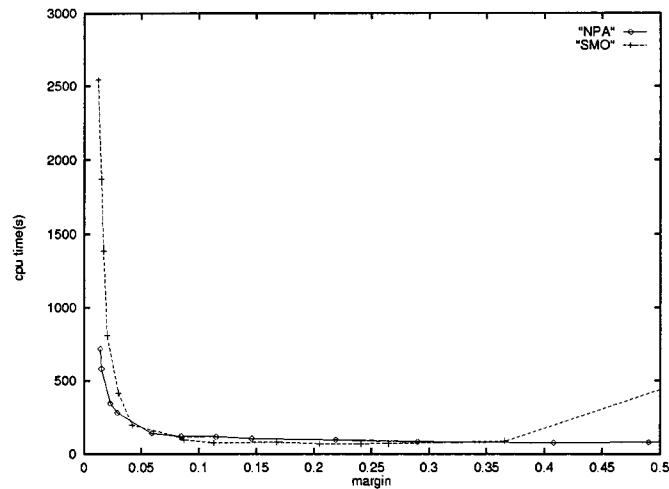


Fig. 9. Adult-1 data: CPU time (in seconds) shown as a function of the margin M .

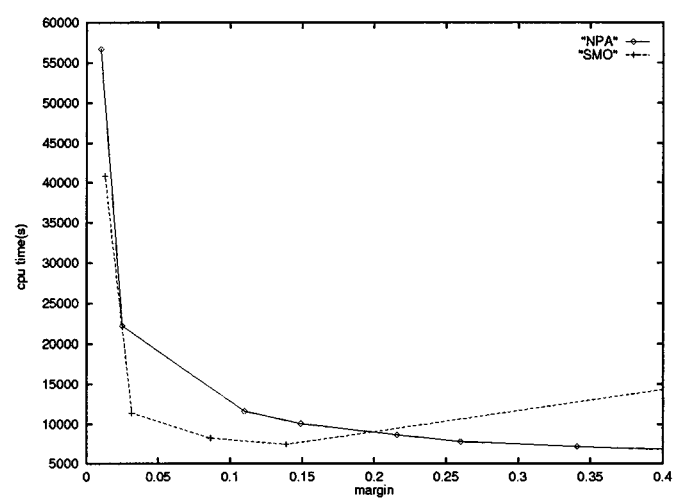


Fig. 11. Adult-7 data: CPU time (in seconds) shown as a function of the margin M .

For the various datasets, Figs. 8–11 show the variation of computing cost (measured as CPU time in seconds) as a function of the margin, M . On an average, SMO and NPA perform quite closely. At small M values (large C , \tilde{C} values), NPA seems to do better than SMO, which indicates that it is a better method for solving SVM-NV. For the higher M values the two algorithms are equally efficient. The performance on the Adult datasets also indicates that NPA scales to large size problems as well as SMO.

Table II gives a rough idea of the variation of the number of support vectors with M for the various combinations of datasets and algorithms. The linear violation formulation (SMO) results in a smaller number of support vectors when compared to the quadratic violation formulation (NPA). While the difference is small for the Checkers dataset, the difference is prominent for the Adult datasets, especially at high M (small C , \tilde{C}) values. If such a difference occurs in a particular problem, then the linear formulation has an advantage when using the SVM classifier for inference after the design process is over.

After training on several C , \tilde{C} values we studied generalization properties of SVM classifiers designed using SMO and

NPA. For each combination of algorithm and dataset, the corresponding validation set was used to choose the best C or \tilde{C} value. (The Adult-7 dataset was not used in these experiments because of the large computing times involved in choosing optimal C and \tilde{C} values using the validation set.) Using the classifier corresponding to this best value the performance (percentage misclassification) on the test set was evaluated. The results are given in Table III. Clearly, the classifiers designed using SMO and NPA give nearly the same performance.

VIII. CONCLUSION

In this paper we have developed a new fast algorithm for designing SVM classifiers using a carefully devised nearest point algorithm. The comparative performance of this algorithm against the SMO algorithm on a number of benchmark problems is excellent. The performance studies done in this paper as well as those done by Frieß [9] indicate that the SVM classifier formulation that quadratically penalizes classification violations is worth considering. Like the SMO algorithm, our

TABLE II

NUMBER OF SUPPORT VECTORS. THE NUMBERS ARE GIVEN AT SOME REPRESENTATIVE M VALUES. FOR SMO THE TWO VALUES IN EACH ITEM ARE THE NUMBER OF α_k 's WHICH ARE IN THE OPEN INTERVAL $(0, C)$ AND THE NUMBER OF α_k 's AT C , RESPECTIVELY. FOR NPA THE VALUE IS THE NUMBER OF NON-ZERO β_k 's

Data Set	SMO			NPA		
	High M	Middle M	Low M	High M	Middle M	Low M
Checkers	21, 295	34, 80	37, 0	242	88	37
Adult-1	55, 764	107, 584	637, 24	1602	1034	677
Adult-4	68, 2136	233, 1653	1661, 190	3385	2999	1967
Adult-7	37, 7818	355, 5659	2438, 3728	11950	10181	8393

TABLE III

PERCENTAGE MISCLASSIFICATION ON THE TEST SET

	Checkers	Adult-1	Adult-4
SMO	5.8	13.8	14.6
NPA	3.9	14.2	13.4

algorithm also is quite straightforward to implement, as indicated by the pseudocode in [14]. Our nearest point formulation as well as the algorithm are special to classification problems and cannot be used for SVM regression.

APPENDIX I

LINE SEGMENT AND TRIANGLE ALGORITHMS

In this Appendix we give two useful algorithms: one for computing the nearest point from the origin to a line segment; and one for computing the nearest point from the origin to a triangle.

Consider the problem of computing \tilde{z} , the nearest point of the line segment joining two points, z and \bar{z} from the origin. Expressing \tilde{z} in terms of a single real variable λ as

$$\tilde{z} = \lambda \bar{z} + (1 - \lambda)z \quad (\text{A.1})$$

and minimizing $\|\tilde{z}\|^2$ with respect to λ , it is easy to obtain the following expression for the optimal λ :

$$\lambda = \begin{cases} 1 & \text{if } \|\tilde{z}\|^2 \leq \bar{z} \cdot z \\ \frac{\|z\|^2 - \bar{z} \cdot z}{\|\bar{z} - z\|^2} & \text{otherwise.} \end{cases} \quad (\text{A.2})$$

It is also easy to derive an expression for the optimal value of $\|\tilde{z}\|^2$

$$\|\tilde{z}\|^2 = \begin{cases} \|\bar{z}\|^2 & \text{if } \|\tilde{z}\|^2 \leq \bar{z} \cdot z \\ \frac{\|z\|^2 \|\bar{z}\|^2 - (\bar{z} \cdot z)^2}{\|\bar{z} - z\|^2} & \text{otherwise.} \end{cases} \quad (\text{A.3})$$

Furthermore, suppose the following condition holds:

$$\bar{z} \cdot z < \|z\|^2 - \tilde{\epsilon}. \quad (\text{A.4})$$

Let us consider two cases.

Case 1) $\frac{\|\bar{z}\|^2 \leq \bar{z} \cdot z}{(\text{A.4})}$ In this case, we have, by (A.3) and

$$\|z\|^2 - \|\tilde{z}\|^2 = \|z\|^2 - \|\bar{z}\|^2 \geq \|z\|^2 - \bar{z} \cdot z > \tilde{\epsilon}. \quad (\text{A.5})$$

Case 2) $\frac{\|\bar{z}\|^2 > \bar{z} \cdot z}{\text{Using (A.3) we get, after some algebra}}$

$$\|z\|^2 - \|\tilde{z}\|^2 = \frac{(\|z\|^2 - \bar{z} \cdot z)^2}{\|z - \bar{z}\|^2}. \quad (\text{A.6})$$

Let $\|z - \bar{z}\| \leq L$ for some finite L . (If z and \bar{z} are points of a polytope then such a bound exists. By (A.4) and (A.6) we get

$$\|z\|^2 - \|\tilde{z}\|^2 > \frac{\tilde{\epsilon}^2}{L^2}. \quad (\text{A.7})$$

Combining the two cases we get

$$\|z\|^2 - \|\tilde{z}\|^2 > \min \left\{ \tilde{\epsilon}, \frac{\tilde{\epsilon}^2}{L^2} \right\}. \quad (\text{A.8})$$

Consider next, the problem of computing the nearest point of a triangle joining three points, P , Q , and R from the origin. We have found it to be numerically robust to compute the minimum distance from the origin to each of the edges of the triangle and the minimum distance from the origin to the interior of the triangle (if such a minimum exists) and then take the best of these four values. The first three distances can be computed using the line segment algorithm we described above. Let us now consider the interior. This is done by computing the minimum distance from the origin to the two dimensional affine space formed by P , Q , and R , and then testing whether the nearest point lies inside the triangle. Setting the gradient of $\|P + (Q - P)\lambda_2 + (R - P)\lambda_3\|^2$ to zero and solving for λ_1 and λ_2 yields

$$\lambda_2 = (e_{22}f_1 - e_{12}f_2)/\text{den}, \quad \lambda_3 = (-e_{12}f_1 + e_{11}f_2)/\text{den} \quad (\text{A.9})$$

where $e_{11} = \|Q - P\|^2$, $e_{22} = \|R - P\|^2$, $e_{12} = (Q - P) \cdot (R - P)$, $\text{den} = e_{11}e_{22} - e_{12}^2$, $f_1 = (P - Q) \cdot P$, and $f_2 = (P - R) \cdot R$. Let $\lambda_1 = 1 - \lambda_2 - \lambda_3$. Clearly, the minimum point to the affine space lies inside the triangle if and only if $\lambda_1 > 0$, $\lambda_2 > 0$, and $\lambda_3 > 0$; in that case, the nearest point is given by $\lambda_1 P + \lambda_2 Q + \lambda_3 R$ and the square of the minimum distance is $P \cdot P - \lambda_2 f_1 - \lambda_3 f_2$.

REFERENCES

- [1] J. K. Anlauf and M. Biehl, "The adatron: An adaptive perceptron algorithm," *Europhys. Lett.*, vol. 10, no. 7, pp. 687–692, 1989.

- [2] R. O. Barr, "An efficient computational procedure for a generalized quadratic programming problem," *SIAM J. Contr.*, vol. 7, no. 3, pp. 415–429, 1969.
- [3] R. Bennett and E. J. Bredensteiner, "Geometry in Learning," Dept. Math. Sci., Rensselaer Polytechnic Inst., Troy, NY, Tech. Rep., 1996.
- [4] R. Bennett and O. L. Mangasarian, "Robust linear programming discrimination of two linearly inseparable sets," *Optimization Methods and Software*, vol. 1, pp. 23–34, 1992.
- [5] C. J. C. Burges, "A tutorial on support vector machines for pattern recognition," *Data Mining and Knowledge Discovery*, vol. 2, no. 2, 1998.
- [6] C. Cortes and V. Vapnik, "Support vector networks," *Machine Learning*, vol. 20, pp. 273–297, 1995.
- [7] R. Fletcher, *Practical Methods of Optimization*, 2nd ed. New York: Wiley, 1987.
- [8] T. T. Frieß, N. Cristianini, and C. Campbell, "The kernel adatron algorithm: A fast and simple learning procedure for support vector machines," in *Proc. 15th Int. Conf. Machine Learning*, San Mateo, CA, 1998.
- [9] T. T. Frieß, "Support Vector Networks: The Kernel Adatron with Bias and Soft-Margin," Univ. Sheffield, Dept. Automat. Contr. Syst. Eng., Sheffield, U.K., Tech. Rep., 1998.
- [10] E. G. Gilbert, "Minimizing the quadratic form on a convex set," *SIAM J. Contr.*, vol. 4, pp. 61–79, 1966.
- [11] E. G. Gilbert, D. W. Johnson, and S. S. Keerthi, "A fast procedure for computing the distance between complex objects in three dimensional space," *IEEE J. Robot. Automat.*, vol. 4, pp. 193–203, 1988.
- [12] T. Joachims, "Making large-scale support vector machine learning practical," in *Advances in Kernel Methods: Support Vector Machines*, B. Schölkopf, C. Burges, and A. Smola, Eds. Cambridge, MA: MIT Press, Dec. 1998.
- [13] L. Kaufman, "Solving the quadratic programming problem arising in support vector classification," in *Advances in Kernel Methods: Support Vector Machines*, B. Schölkopf, C. Burges, and A. Smola, Eds. Cambridge, MA: MIT Press, Dec. 1998.
- [14] S. S. Keerthi, S. K. Shevade, C. Bhattacharyya, and K. R. K. Murthy. (1999, Mar.) A Fast Iterative Nearest Point Algorithm for Support Vector Machine Classifier Design. Intell. Syst. Lab., Dept. Comput. Sci. Automat., Indian Inst. Sci., Bangalore, Karnataka, India. [Online]. Available: <http://guppy.mpe.nus.edu.sg/~mpessk>
- [15] C. L. Lawson and R. J. Hanson, *Solving Least Squares Problems*. Englewood Cliffs, NJ: Prentice-Hall, 1974.
- [16] S. R. Lay, *Convex Sets and Their Applications*. New York: Wiley, 1982.
- [17] V. J. Lumelsky, "On fast computation of distance between line segments," *Inform. Processing Lett.*, vol. 21, pp. 55–61, 1985.
- [18] B. F. Mitchell, V. F. Dem'yanov, and V. N. Malozemov, "Finding the point of a polyhedron closest to the origin," *SIAM J. Contr.*, vol. 12, pp. 19–26, 1974.
- [19] O. L. Mangasarian and D. R. Musicant, "Successive Overrelaxation for Support Vector Machines," Computer Sciences Dept., University of Wisconsin, Madison, WI, Tech. Rep., 1998.
- [20] J. C. Platt, "Fast training of support vector machines using sequential minimal optimization," in *Advances in Kernel Methods: Support Vector Machines*, B. Schölkopf, C. Burges, and A. Smola, Eds. Cambridge, MA: MIT Press, Dec. 1998.
- [21] Adult and Web Datasets, J. C. Platt. [Online]. Available: <http://www.research.microsoft.com/~jplatt>
- [22] N. K. Sancheti and S. S. Keerthi, "Computation of certain measures of proximity between convex polytopes: A complexity viewpoint," in *Proc. IEEE Int. Conf. Robot. Automat.*, Nice, France, 1992, pp. 2508–2513.
- [23] A. J. Smola and B. Schölkopf, "A Tutorial on Support Vector Regression," Royal Holloway College, London, U.K., NeuroCOLT Tech. Rep. TR-1998-030, 1998.
- [24] Two Spirals Data [Online]. Available: <ftp://ftp.boltz.cs.cmu.edu/pub/neural-bench/bench/two-spirals-v1.0.tar.gz>
- [25] UCI Machine Learning Repository [Online]. Available: <http://www.ics.uci.edu/~mllearn/MLRepository.html>
- [26] V. Vapnik, *Estimation of Dependences Based on Empirical Data*. Berlin, Germany: Springer-Verlag, 1982.
- [27] V. Vapnik, *The Nature of Statistical Learning Theory*. New York: Springer-Verlag, 1995.

- [28] Wisconsin Breast Cancer Data [Online]. Available: <ftp://128.195.1.46/pub/machine-learning-databases/breast-cancer-wisconsin/>
- [29] P. Wolfe, "Finding the nearest point in a polytope," *Math. Programming*, vol. 11, pp. 128–149, 1976.



S. S. Keerthi received the bachelor's degree in mechanical engineering from REC Trichy, University of Madras, India, in 1980, the master's degree in mechanical engineering from the University of Missouri, Rolla, in 1982, and the Ph.D. degree in control engineering from the University of Michigan, Ann Arbor, in 1986.

After working for about one year with Applied Dynamics International, Ann Arbor, doing R&D in real-time simulation, he joined the faculty of the Department of Computer Science and Automation,

Indian Institute of Science, Bangalore in April 1987. His academic research covered the following areas: geometric problems in robotics, algorithms for dynamic simulation of multibody systems, nonholonomic motion planning and neural networks. He joined the Control Division of the Department of Mechanical and Production Engineering, National University of Singapore, in May 1999, as Associate Professor. He has published more than 40 papers in leading international journals and conferences.



S. K. Shevade received the M.Sc.(Eng.) degree from the Indian Institute of Science.

He is currently with the Department of Computer Science and Automation, Indian Institute of Science. His research interests include algorithms, neural networks, and pattern recognition.



C. Bhattacharyya received the M.E. degree in electrical engineering from the Indian Institute of Science in 1995. He is currently a Ph.D. student in the Department of Computer Science and Automation, Indian Institute of Science.

His research interests lie in belief networks, mean field theory, and neural networks.



K. R. K. Murthy received the M.E. degree in systems science and automation from Indian Institute of Science in 1995. He is currently a Ph.D. student in the Department of Computer Science and Automation, Indian Institute of Science.

His interests are in information filtering, machine learning, and neural networks.