A Fast Iterative Nearest Point Algorithm for Support Vector Machine Classifier Design[1]

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

ssk@csa.iisc.ernet.in shirish@csa.iisc.ernet.in cbchiru@csa.iisc.ernet.in murthy@csa.iisc.ernet.in

Technical Report TR-ISL-99-03

Intelligent Systems Lab
Dept. of Computer Science & Automation
Indian Institute of Science
Bangalore - 560 012
India
Ph: (91)-(80)-3092907

[1] A revised version of this report is under preparation for submission to a journal. We welcome any comments and suggestions for improving this report.

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, Dem'yanov and Malozemov, 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 Friess is used to convert it to a problem in which there are no classification violations. Comparitive computational evaluation of our algorithm against powerful SVM methods such as Platt's Sequential Minimal Optimization shows that our algorithm is very competitive.

1. Introduction.

The last few years have seen the rise of Support Vector Machines (SVMs)[23] as powerful tools for solving classification and regression problems[4]. Recently, fast iterative algorithms that are also easy to implement have been suggested[18, 11, 17, 7, 21]; Platt's SMO algorithm [18] is an important example. Such algorithms are bound to widely increase the popularity of SVMs 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. Though not as easy as SMO, our algorithm also is quite straightforward to implement. A pseudocode is included in appendix B. Using this pseudocode an actual running code can be developed within a few hours. A Fortran code implemented by us can be obtained from us for any non-commercial 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 only using 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: $I \neq \emptyset$, $J \neq \emptyset$, $I \cup J = \{1, ..., m\}$ and $I \cap J = \emptyset$. Let us define: $z_i = \phi(x_i)$. The SVM design problem without violations is:

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

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:

$$M = \frac{2}{\|w\|}$$

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 to allow violations in the satisfaction of the constraints in SVM-NV, and penalize such violations *linearly* in the objective function:

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

(Throughout, we will use k and/or l whenever the indices run over all elements of $I \cup J$.) Here C is a positive, inverse regularization constant that is chosen to give the correct relative weighting between margin maximization and classification violation. Using Wolfe duality theory[4,6] SVM-VL can be transformed to the following equivalent dual problem:

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

where $y_k = 1, k \in I$ and $y_k = -1, k \in J$. It is computationally easy to handle this problem since it is directly based on $z_i \cdot z_j$ (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. □

In a recent paper Friess[8] has suggested the use a sum of squared violations in the cost function:

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

Preliminary experiments by Friess have shown this formulation to be promising. Unlike SVM-VL, here there is no need to include non-negativity 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[8], 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 1 and all other components are 0. Define:

$$\tilde{w} = \begin{pmatrix} w \\ \sqrt{\tilde{C}}\xi \end{pmatrix}; \quad b = \tilde{b}; \quad \tilde{z}_i = \begin{pmatrix} z_i \\ \frac{1}{\sqrt{\tilde{C}}}e_i \end{pmatrix}, i \in I; \quad \tilde{z}_j = \begin{pmatrix} z_j \\ -\frac{1}{\sqrt{\tilde{C}}}e_j \end{pmatrix}, j \in J. \qquad (1)$$

Then it is easy to see that SVM-V 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 non-empty 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

$$\delta_{kl} = \begin{cases} 1 & \text{if} \quad k = l \\ 0 & \text{otherwise} \end{cases}$$

Thus, for any pair of training vectors, (x_k, x_l), the modified kernel function is very easily computed. $\square$

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 (details are given in section 8) 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.

Recently Friess et.al., [7] (deriving inspiration from the Adatron algorithm given by Anlauf and Biehl for designing Hopfield nets) and Mangasarian and Musicant[17] (using a successive overrelaxation idea) independently 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.[7] applied the Adatron algorithm to solve SVM-NV, Friess[8] applied the same to SVM-VQ, while Mangasarian and Musicant[17] applied the successive overrlaxation scheme to solve SVM-VL. However, the basic algorithmic idea used by them are nearly identical: 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.

In spite of all the computational simplicity brought about by the inclusion of the $b^2/2$ term, our testing shows that these simplified algorithms do not perform as well as algorithms which solve problems without the $b^2/2$ term. Also, since the effect of including the $b^2/2$ term on SVM performance such as generalization etc., is yet to be fully understood, the value of including the $b^2/2$ just for the sake of having some extra implementational ease is questionable.

The paper is organized as follows. In section 2 we reformulate SVM-NV as a problem of computing the nearest point between two convex polytopes. In section 3 we discuss optimality criteria for this nearest point problem. Section 4 discusses important practical issues concerning approximate stopping of dual solutions. It points out the danger of simply stopping the dual solution using dual objective function accuracy, and 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 5 and 6. Two classical algorithms for doing nearest point solution, due to Gilbert[9] and Mitchell et.al.[16], are combined and modified to derive our fast algorithm. Section 7 concerns the simplification of this algorithm to treat the case corresponding to the inclusion of $b^2/2$ in the objective functions of SVM-VQ and SVM-NV. A detailed comparitive computational testing of existing algorithms against our algorithms is taken up in section 8. The two appendices give all details needed for implementing our main algorithm.

2. Reformulation of SVM-NV as a Nearest Point Problem.

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

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

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:

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

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

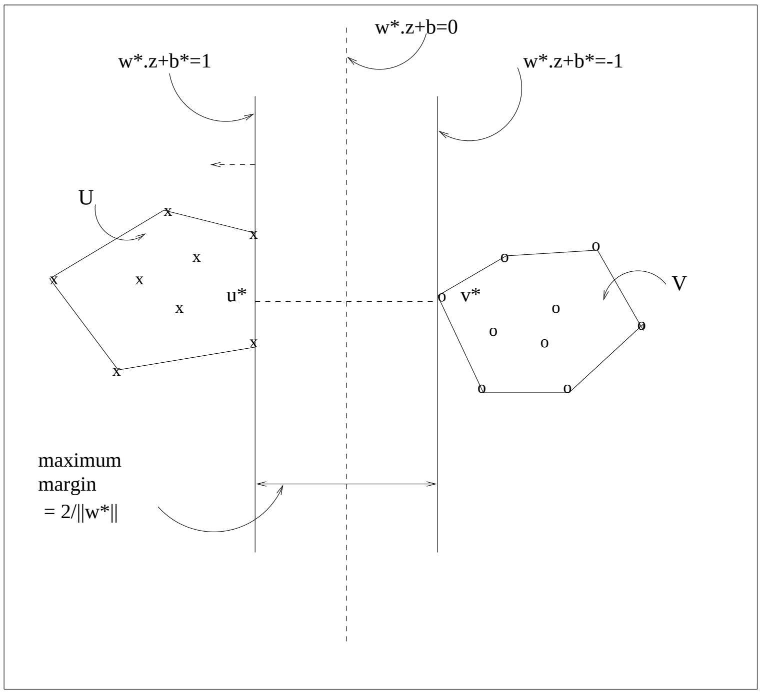

Figure 1: Among all pairs of parallel hyperplanes that separate the two classes, the pair with the largest margin is the one which has $(u^\star - v^\star)$ as the normal direction, where $(u^\star, v^\star)$ is a pair of closest points of U and V. Note that $w^\star = \delta(u^\star - v^\star)$ for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

of NPP algebraically as:

$$\begin{array}{lll} u = \sum_{i \in I} \beta_i z_i; & \beta_i \geq 0, i \in I; & \sum_{i \in I} \beta_i = 1 \\ v = \sum_{j \in J} \beta_j z_j; & \beta_j \geq 0, j \in J; & \sum_{j \in J} \beta_j = 1 \end{array} \tag{2}$$

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

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

The following theorem states this relationship formally.

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

A direct, geometrically intuitive proof is given by Sancheti and Keerthi[20] with reference to a geometrical problem in Robotics. Later, Bennett[2] 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 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[4,6] we first transform SVM-NV to the following equivalent dual problem:

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

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 $\sum_{i \in I} \alpha_i = \sum_{i \in J} \alpha_i$, we can introduce a new variable, λ and rewrite $\sum_k \alpha_k y_k = 0$ as two constraints:

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

If we also define

$$\beta_k = \frac{\alpha_k}{\lambda} \quad \forall\, k \tag{4}$$

then SVM-NV-DUAL can be rewritten as

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

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

$$\lambda^\star = \frac{2}{\sum_k \sum_l \beta_k \beta_l y_k y_l z_k \cdot z_l} \tag{6}$$

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

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

This problem is equivalent to the problem:

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

If we define a matrix P whose columns are $y_1 z_1, ..., y_m z_m$ 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[4] actually yields

$$w^\star = \sum_k \alpha_k^\star y_k z_k$$

Using this, (4) and (6) we can immediately verify the expression for $w^\star$ in (3). The expression for $b^\star$ 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. □

3. 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*, $h_P : R^n \to R$, by

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

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

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

Now consider the case where P is a convex polytope: $Z = \{\hat{z}_1, ..., \hat{z}_r\}$ and $P = \text{co}Z$. It is well-known[14] 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:

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

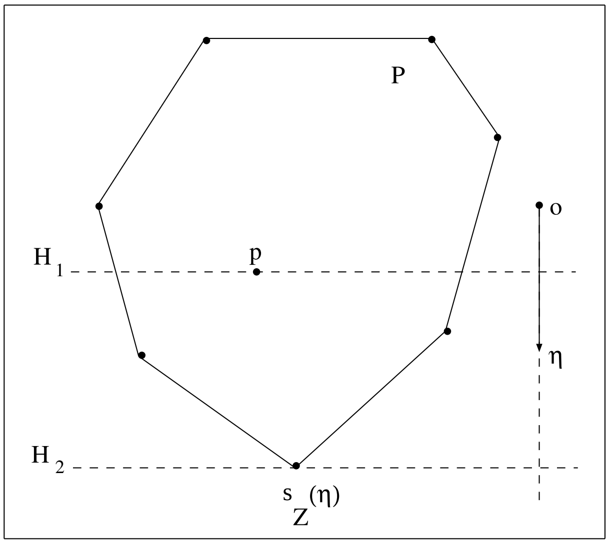

Figure 2: Definition of support properties. $s_Z(\eta)$ is the extreme vertex of P in the direction η. The distance between the hyperplanes, H_1 and H_2 is equal to $g_P(\eta, p)/\|\eta\|$.

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, $g_P : R^n \times P \to R$ by

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

By (9) it follows that

$$g_P(\eta, p) \geq 0 \;\; \forall \;\; \eta \in R^n, \; p \in P \tag{12}$$

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

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

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

$$g(u, v) = g_U(v - u, u) + g_V(u - v, v) \tag{14}$$

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. (a) $g(u, v) \geq 0$. (b) 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 $\|\tilde{u} - v\| < \|u - v\|$. (c) 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 $\|u - \tilde{v}\| < \|u - v\|$. (d) (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 $z \cdot v \geq z \cdot \hat{v}$. 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 $g(u, v) = 0$. □

4. 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. We will show using an example that simply stopping when the dual approximate solution has reached the dual optimal solution within a good accuracy can be dangerous.

Consider the example shown in Figure 3. Let us take it that Class 2 has a single point, which is the origin, o. Thus $v^\star = o$. Let $u^\star$ be on the line segment joining a and b, which are the support

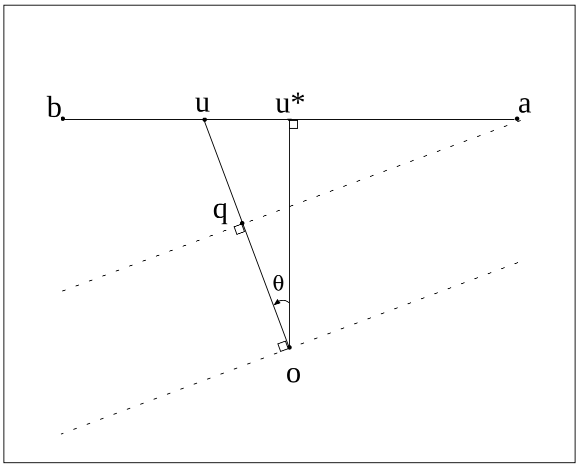

Figure 3: Example to illustrate the danger of stopping with only good dual solution accuaracy.

vectors from Class 1 (i.e., those vectors in $\{z_i\}$ which have positive multipliers in a representation of $u^\star$). Let us use the notation, $\overline{xy}$ to denote the distance between the two points, x and y. Let $\rho^\star = \overline{ou^\star}$, the optimal distance between the two classes, and, $d = \overline{oa}$. Suppose it turns out that d is very much larger than $\rho^\star$. (This condition is quite typical of most classification problems.) Suppose we are solving NPP and we stop when we reach an approximate solution, u that satisfies $\overline{ou} \leq (1+\epsilon)\overline{ou^\star}$. Let us assume that ϵ is much smaller than 1. If we assume that the support vectors are identified correctly (most algorithms, in fact do this well) then u lies on the line segment joining the two support vectors, a and b. Let u be as shown in Figure 3.

Consider triangle $ouu^\star$.

$$\overline{uu^\star} = \sqrt{(\overline{ou})^2 - (\overline{ou^\star})^2} = \sqrt{(2\epsilon + \epsilon^2)}\rho^\star \approx \sqrt{2\epsilon}\rho^\star$$

Then $\sin\theta = \overline{uu^\star}/\overline{ou} = \sqrt{2\epsilon + \epsilon^2}/(1+\epsilon) \approx \sqrt{2\epsilon}$. Now consider triangle $ou^\star a$. $\overline{au^\star} = \sqrt{(\overline{ao})^2 - (\overline{ou^\star})^2} = \sqrt{d^2 - (\rho^\star)^2} \approx d$. Also, $\overline{au} = \overline{au^\star} + \overline{uu^\star} \approx d$. Let q be the point of intersection of the line joining o and u with the line which has $\vec{ou}$ as the normal direction and passes through a. Clearly $\overline{oq}$ is M, the margin corresponding to the approximate solution, u. Note that $M^\star$, the optimal margin for the problem, is equal to $\rho^\star$. To get an estimate of M consider triangle uqa. $\overline{uq} = \overline{au}\sin\theta \approx d\sqrt{2\epsilon}$. Then,

$$M = \overline{ou} - \overline{uq} \approx \rho^\star\sqrt{1+\epsilon} - d\sqrt{2\epsilon} \approx (1 + 0.5\epsilon)\rho^\star - d\sqrt{2\epsilon}$$

From this we finally get

$$\frac{(M^\star - M)}{M^\star} = \frac{(\rho^\star - M)}{\rho^\star} \approx \frac{d}{\rho^\star}\sqrt{2\epsilon} - 0.5\epsilon$$

For small ϵ, it is the first term on the right hand side that dominates. Thus, when $d/\rho^\star$ is large, the error in the margin can be very large! A small deviation, u from $u^\star$ thus leads to a large change in the margin, M from $M^\star$, simply because of the reason that a is much farther from o than $u^\star$! What is danegerous is that, even with a small ϵ, M can end-up being negative and hence, at the end of the algorithm we cannot even generate a feasible (w, b) pair for SVM-NV using the approximate dual solution. Of course, when ϵ is made very very small, these problems disappear. However, a key question still remains to be answered: how do we systematically stop the dual solution algorithm in order to produce a feasible primal solution that has a specified closeness to the optimal primal solution? We will next show that the g function that we introduced in (14) greatly aids in doing such a guaranteed stopping.

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 $\hat{u} \to u^\star$ and $\hat{v} \to v^\star$, where $(u^\star, v^\star)$ is a solution of NPP. By part (d) of Theorem 2 and the fact that g is a continuous function, $g(\hat{u}, \hat{v}) \to 0$. Also, $\|\hat{z}\| \to \|z^\star\|$ where $\hat{z} = \hat{u} - \hat{v}$ and $z^\star = u^\star - v^\star$. By assumption A1, $\|\hat{z}\| \geq \|z^\star\| > 0$. Thus we also have $g(\hat{u}, \hat{v})/\|\hat{z}\|^2 \to 0$. 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 $\|w^\star\| \geq (1 - \epsilon)\|\hat{w}\|$ where $(w^\star, b^\star)$ 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

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

Figure 4 geometrically depicts the situation. Let $\hat{M}$ be the margin corresponding to the direction, $\hat{z}$. Clearly,

$$\hat{M} = \frac{-h_U(-\hat{z}) - h_V(\hat{z})}{\|\hat{z}\|} \tag{16}$$

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

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

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

$$\hat{w} = \gamma\hat{z} \tag{18}$$

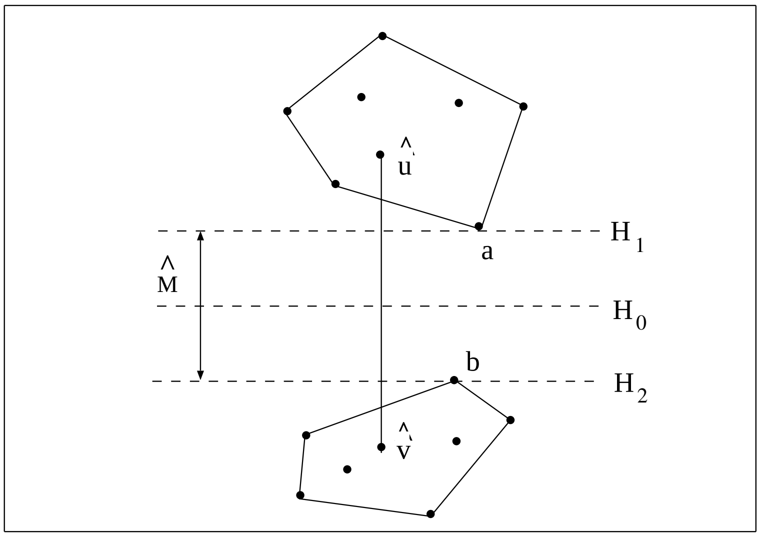

Figure 4: A situation satisfying (15). Here: $a = s_U(-\hat{z})$; $b = s_V(\hat{z})$; $H_1 : \hat{z} \cdot z = -h_U(-\hat{z})$; $H_2 : \hat{z} \cdot z = h_V(\hat{z})$; and $H_0 : \hat{z} \cdot z = 0.5(h_V(\hat{z}) - h_U(-\hat{z}))$.

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

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

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

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

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

$$\frac{\|w^\star\|}{\|\hat{w}\|} = \frac{\hat{M}}{M^\star} = \frac{-h_U(-\hat{z}) - h_V(\hat{z})}{\|\hat{z}\|\|z^\star\|} = \frac{-h_U(-\hat{z}) - h_V(\hat{z})}{\|\hat{z}\|^2} \cdot \frac{\|\hat{z}\|}{\|z^\star\|} \geq (1-\epsilon)$$

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

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

Thus we have proved the following important result.

Theorem 3. Let $(u^\star, v^\star)$ be a solution of NPP, $z^\star = u^\star - v^\star$, and $(w^\star, b^\star)$ 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

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

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

$$\frac{\|w^\star\|^2}{\|\hat{w}\|^2} = (\frac{\|w^\star\|}{\|\hat{w}\|})^2 \geq (1-\epsilon)^2 \geq (1-2\epsilon)$$

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

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

5. Iterative Algorithms for NPP.

NPP has been well studied in the literature, and a number of good algorithms have been given for it[9,16,25,1,13]. Best general purpose algorithms for NPP such as Wolfe's algorithm[25] 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.[2] 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.

5.1. Gilbert's Algorithm.

Gilbert's algorithm[9] 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[25,10]. 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[14], i.e.,

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

[2] It is interesting to point out that the active set method used by Kaufman[12] to solve SVMs is *identical* to Wolfe's algorithm when both are used to solve SVM-NV.

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

$$\min\{\|z\| : z \in Z\} \tag{MNP}$$

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, $Z = \mathrm{co}W$, where

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

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,

$$u = \sum_{i\in I} \beta_i z_i, \quad \sum_{i\in I} \beta_i = 1, \quad \beta_i \geq 0 \; \forall\, i \in I,$$

$$v = \sum_{j\in J} \beta_j z_j, \quad \sum_{j\in J} \beta_j = 1, \quad \beta_j \geq 0 \; \forall\, j \in J$$

Then write $z = u - v$ as

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

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:

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

$$s_Z(\eta) = s_U(\eta) - s_V(-\eta) \tag{23}$$

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 3. 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. (a) $g_Z(-z, z) \geq 0$. (b) If $\bar{z}$ is any point in Z such that $\|z\|^2 - z.\bar{z} > 0$, 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 $g_Z(-z, z) = 0$. □

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 $z = z^\star$, the solution of MNP. On the other hand, if $g_Z(-z, z) > 0$, then $\bar{z} = s_Z(-z)$ satisfies $\|z\|^2 - z.\bar{z} > 0$. By part (b) 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 $z^\star = z$; 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 appendix A. Figure 5 shows a few iterations of Gilbert's algorithm on a two dimensional example. Gilbert showed that, if the algorithm does not stop with $z^\star$ at step 1 within a finite number of iterations, then $z \to z^\star$ asymptotically, as the number of iterations goes to infinity. He also showed that the algorithm has a linear rate of convergence.

We now discuss how Gilbert's algorithm can be adapted to solve an NPP that arises from an SVM classifier design problem. Step 0 can be easily implemented by choosing any $i \in I$, $j \in J$ and setting $z = z_i - z_j$. In a typical usage of Gilbert's algorithm, z is actually maintained in memory. This is possible for linear SVM design, where the x space and z space coincide. However, in a nonlinear SVM design problem that is based on kernel functions, z is located either in a very large dimensional space or in an infinite dimensional space. In such a case, instead of maintaining z one has to maintain a convex hull representation of z:

$$z = \sum_t \gamma_t w_t, \quad \sum_t \gamma_t = 1, \quad \gamma_t \geq 0 \; \forall \; t$$

where $\{w_t\} \subset W$ (see (21)). Only the $\{\gamma_t\}$ and some details associated with each w_t need to be stored and maintained. Let us now explain what these details are. It is easy to note that new w_t points arise only from $\bar{z}$ in step 2. Thus each w_t has the representation, $w_t = z_{i(t)} - z_{j(t)}$, where

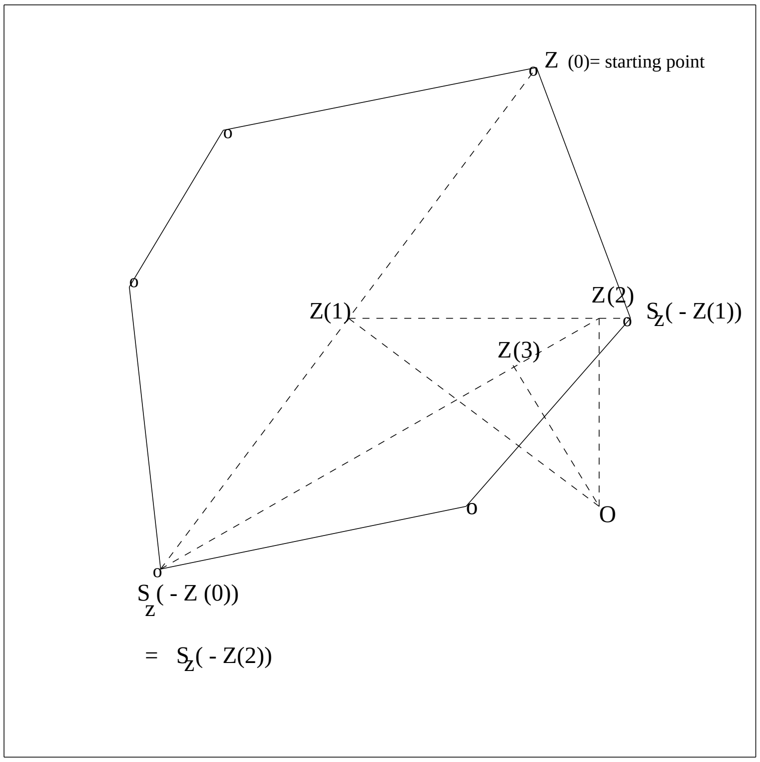

Figure 5: A few iterations of Gilbert's algorithm on a two dimensional example.

$i(t) \in I$ and $j(t) \in J$. The pair, $(i(t), j(t))$ carries full information about w_t and hence it is sufficient to store this detail for each t.

Step 1 can be implemented using (22) and (23). To do the computations efficiently, it is useful to maintain a cache of the inner product, $e_s = z \cdot z_s$, for all those indices, $s \in \{1, \ldots, m\}$ which are needed in the definition of $\{w_t\}$. Let S denote the set of such indices, i.e., $S = \{i(t)\}_t \cup \{j(t)\}_t$. In step 2, $\tilde{z}$ takes the form, $\tilde{z} = (1-\lambda)z + \lambda\bar{z}$ where $\bar{z} = z_i - z_j$ and $i \in I$, $j \in J$ are determined in step 1. At the end of step 2, the inner products cache, $\{e_s\}$ is updated using the following sequence of steps. (1) If $i \notin S$, set $S := S \cup \{i\}$, compute

$$e_i = z \cdot z_i = \sum_t \gamma_t w_t \cdot z_i = \sum_t \gamma_t (z_{i(t)} \cdot z_i - z_{j(t)} \cdot z_i)$$

and include it in the cache. (2) Repeat the above step for j. (3) Then for each $s \in S$ set: $e_s := (1-\lambda)e_s + \lambda(z_i \cdot z_s - z_j \cdot z_s)$. Then the γ_t are updated by the following sequence of steps. (1) Set: $\gamma_t := (1-\lambda)\gamma_t \;\forall\, t$. (2) If $z_i - z_j$ is equal to some already existing w_t then make a further modification to γ_t (only for that t): $\gamma_t := \gamma_t + \lambda$. (3) On the other hand, if $z_i - z_j$ is not equal to any of the already existing w_t then include $z_i - z_j$ as a new member of $\{w_t\}$ and set its γ_t to λ. Note that the updatings involve two training vectors (z_i and z_j) and require $2n_t$ kernel evaluations where n_t is the number of t indices in the representation of z. Since n_t is large (hundreds or thousands) for most problems, the cost of computing the kernels and doing the updating of cache and γ_t is usually much higher than the cost of doing all other operations (such as minimizing norm on a line segment). A similar comment can be made for all algorithms considered in this paper.

At each iteration of the algorithm, a pair of points, $u \in U$ and $v \in V$ that yield $z = u - v$ is always available:

$$u = \sum_t \gamma_t z_{i(t)} \qquad v = \sum_t \gamma_t z_{j(t)}$$

Typically the condition, $g_Z(-z, z) = 0$ does not occur at a finite k. However, since $z \to z^\star$ as the number of iterations goes to infinity, we have $g_Z(-z, z) \to 0$. It is easy to use (14) to see that

$$g_Z(-z, z) = g(u, v)$$

Because of this, the algorithm can be terminated (after a finite number of iterations) when

$$g_Z(-z, z) \leq \epsilon \|z\|^2$$

is satisfied, leading to the determination of an approximate solution of SVM-NV that satisfies the bounds given in Theorem 3.

We implemented Gilbert's algorithm using all the above ideas and tested its performance on a variety of classification problems. While the algorithm always makes rapid movement towards 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., $z^\star \cdot w_t > z^\star \cdot z^\star$), then the algorithm is slow in driving the corresponding γ_t to 0. Because of these reasons, Gilbert's algorithm, by itself, is not very efficient for solving the NPPs that arise in SVM classifier design.

5.2. Mitchell-Dem'yanov-Malozemov Algorithm.

Many years after Gilbert's work, Mitchell, Dem'yanov and Malozemov[16] 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 $z = z^\star$, 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 $z = z^\star$ and hence the above condition automatically holds. However, at an approximate point, $z \neq z^\star$, the situation is different. The point, z could be very close to $z^\star$ and yet, there could very well exist one or more w_t such that

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

(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 Figure 6),

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

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

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 Figure 6. MDM algorithm tries to crush the total slab towards 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

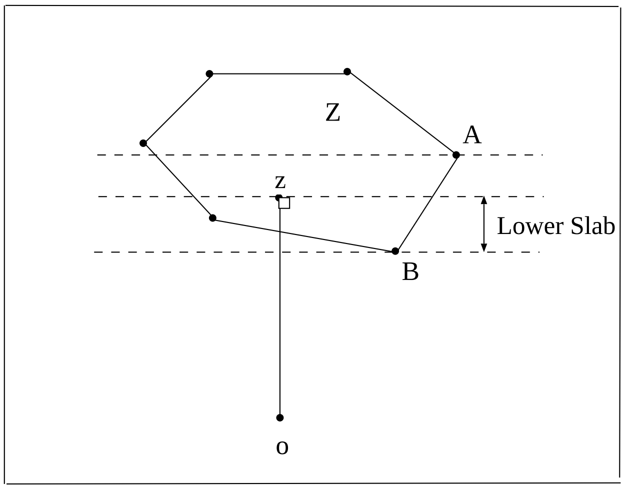

Figure 6: Definition of Δ. Here: $A = w_{tmin}$; $B = s_Z(-z)$; and $\Delta(z) = (-z \cdot B) - (-z \cdot A)$.

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 $tmin$ be an index such that

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

Let $d = s_Z(-z) - w_{tmin}$. 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 $z \neq z^\star$. Since, in the representation for z, γ_{tmin} decreases during this movement, λ has to be limited to the interval defined by $0 \leq \lambda \leq \gamma_{tmin}$. Define $\bar{z} = z + \gamma_{tmin} d$. So, the basic iteration of MDM algorithm consists of finding the point of minimum norm on the line segment joining z and $\bar{z}$.

The structure of MDM algorithm is essentially the same as Gilbert's algorithm and so we feel there is no need to state it formally. Also, all the remarks that we made for Gilbert's algorithm concerning its adaptation to the solution of NPPs arising in SVM classifier design hold good for MDM algorithm too. The details are only slightly more complicated because of the fact that the definition of $\bar{z}$ involves 4 members of $\{z_i\}$. In particular, note that the updating of the $\{e_s\}$ requires $4n_t$ kernel evaluations in each iteration, where n_t is the number of t indices in the representation of z. We omit giving full details of the updates so as to avoid repetition.

We implemented and tested MDM algorithm. It works faster than Gilbert's algorithm, especially in the end stages when z approaches $z^\star$. 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 full details of the final algorithm incorporating both observations, in section 6 and the two appendices.

5.3. 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 $tmin$ and the updating of γ_t and of the inner products cache, e_s. 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 $tmin$, $tmax$ 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 Figure 7. 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 $s_Z(-z) = w_{tmax}$ for some index, $tmax$. (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 0 without affecting the representation of z in anyway.) The points w_{tmin} and w_{tmax} are respectively shown as A and B in the figure. The point, $z+\gamma_{tmin}(w_{tmax}-w_{tmin})$ (i.e., the $\bar{z}$ of MDM algorithm) is shown as C.

Idea 1. At step 2 of each iteration of MDM algorithm, take $\tilde{z}$ to be the point of minimum

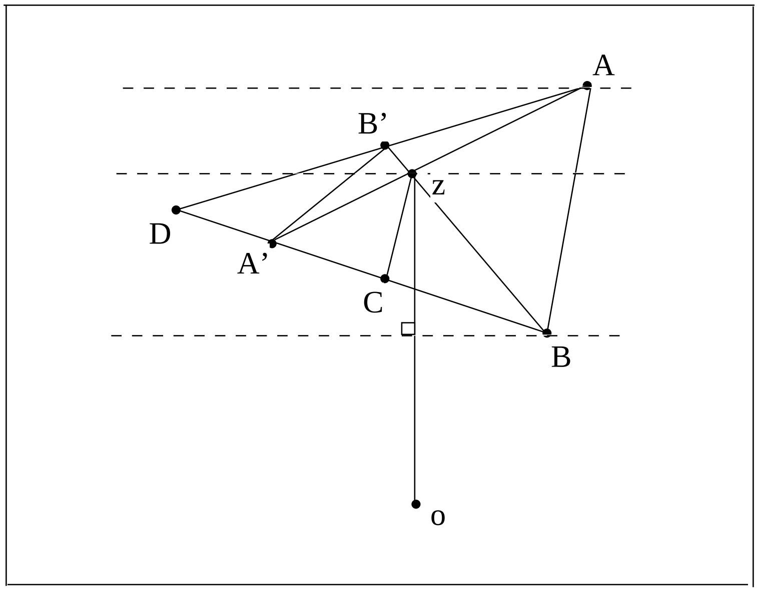

Figure 7: Illustrating the various ideas for improvement. Here: $A = w_{tmin}$; $B = s_Z(-z) = w_{tmax}$; zB = Gilbert line segment; zC = MDM line segment (zC is parallel to AB); T_1 = triangle zCB; T_2 = triangle $zA'B$; Q_3 = quadrilateral $ABA'B'$; and, T_4 = triangle DAB.

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 appendix A. 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 cache, e_s 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 re-normalization. Let us define: $A' = z(tmin)$; see Figure 7. Since

$$C = z + \gamma_{tmin}(w_{tmax} - w_{tmin}) = (1 - \gamma_{tmin})A' + \gamma_{tmin} B$$

it follows that C lies on the line segment joining A' and w_{tmax}.

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 $B' = z(tmax)$ 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 indices, $tmin$ and $tmax$ from the representation of z and then doing a re-normalization, i.e.,

$$D = \frac{1}{(1 - \gamma_{tmin} - \gamma_{tmax})}(z - \gamma_{tmin} w_{tmin} - \gamma_{tmax} w_{tmax})$$

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 worser! (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.

6. A Fast Sequential 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:

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

$$k \in J \Rightarrow \quad z \cdot z_k - z \cdot v \geq \frac{\epsilon}{2}\|z\|^2 \tag{27}$$

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 (A.8) we get an appreciable decrease in the objective function of NPP:

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

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 3) we get:

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

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 Sequential 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

$$z_k \in \tilde{U} \quad \text{if} \quad k \in I, \qquad z_k \in \tilde{V} \quad \text{if} \quad k \in J \tag{30}$$

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 indices (one each from I and J) to define $\bar{z}$. In a similar spirit, by appropriately choosing $\tilde{U}$ and $\tilde{V}$, the various ideas of section 5.3 can be extended while involving only two indices. Before we discuss a detailed algorithm, we prove convergence of the generic algorithm.

Theorem 5. The generic sequential algorithm terminates in step 1 with a (u, v) satisfying (29), after a finite number of iterations. $\square$

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. $\Box$

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:

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

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

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

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 imin and jmin as follows (see Figure 8):

$$\begin{aligned} \text{imin} &= \arg\min\{-z \cdot z_i : i \in \hat{I}, \beta_i > 0\} \\ \text{jmin} &= \arg\min\{z \cdot z_j : j \in \hat{J}, \beta_j > 0\} \end{aligned} \tag{32}$$

(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 indices 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 $\{\text{imin}, \text{jmin}\}$ and then doing operations along ideas similar to those in section 5.3 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 kmin as follows: kmin = imin if $-z \cdot z_{\text{imin}} + z \cdot u < z \cdot z_{\text{jmin}} - z \cdot v$; else kmin = jmin. Then step 2 is carried out using one of the following four cases, depending on k and kmin. See Figure 9 for a geometric description of the cases.

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

$$C' = \frac{1}{(1 - \beta_{\text{kmin}})}(u - \beta_{\text{kmin}} z_{\text{kmin}}) = u + \mu(u - z_{\text{kmin}}) \tag{33}$$

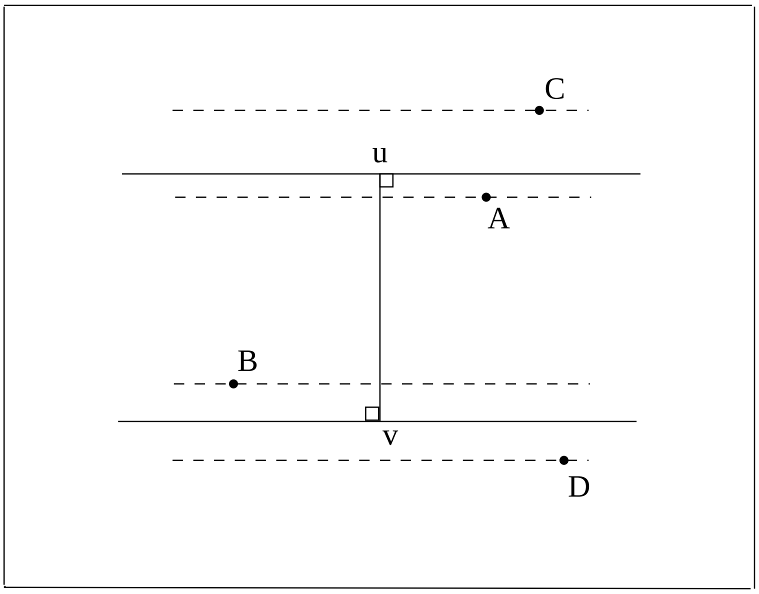

Figure 8: Illustration of imin, jmin, imax and jmax. Here $C = z_{\text{imin}}$, $D = z_{\text{jmin}}$, $A = z_{\text{imax}}$ and $B = z_{\text{jmax}}$.

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

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

$$D' = \frac{1}{(1-\beta_{\text{kmin}})}(v - \beta_{\text{kmin}} z_{\text{kmin}}) = v + \mu(v - z_{\text{kmin}}) \tag{34}$$

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

Case 3. $k \in I$, kmin $\in J$. 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. $k \in J$, kmin $\in I$. 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. However, 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 non-support 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

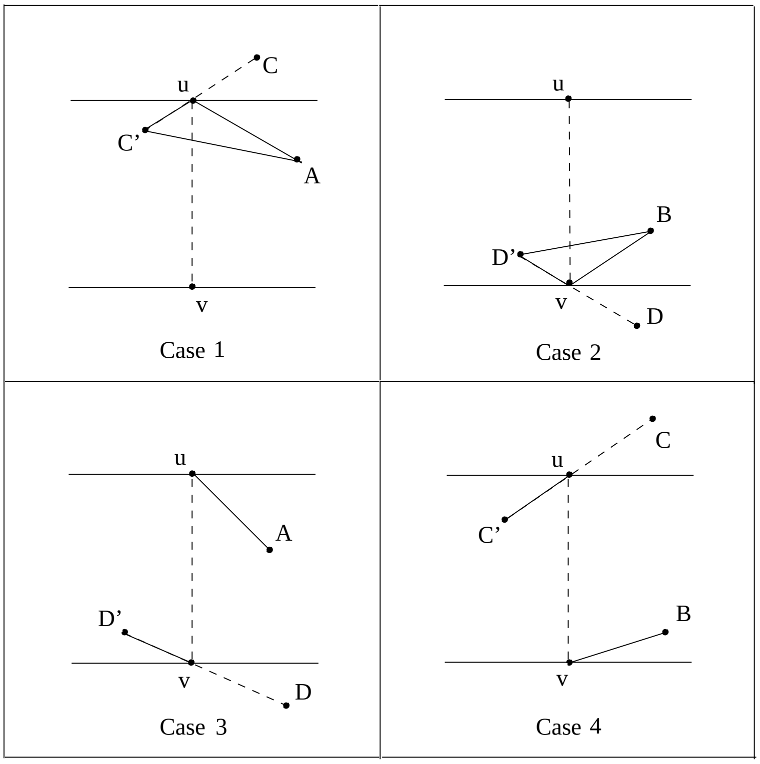

Figure 9: A geometric description of the 4 cases. For Cases 1 and 3, $A = z_k$; for Cases 2 and 4, $B = z_k$.

(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 imin 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 non-support 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 indices 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 imax, jmax and kmax as follows.

$$\begin{aligned} \text{imax} &= \arg\max\{-z \cdot z_i : i \in \hat{I}\} \\ \text{jmax} &= \arg\max\{z \cdot z_j : j \in \hat{J}\} \end{aligned} \tag{35}$$

$$\text{kmax} = \begin{cases} \text{imax} & \text{if} \quad -z \cdot z_{\text{imax}} + z \cdot u > z \cdot z_{\text{jmax}} - z \cdot v \\ \text{jmax} & \text{otherwise} \end{cases} \tag{36}$$

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 kmax and doing one iteration using $k = \text{kmax}$. These iterations are continued until the approximate optimality criteria are satisfied over the support vector set, i.e., when

$$\begin{aligned} g_U^{\text{supp}} &= -z \cdot z_{\text{imax}} + z \cdot u \leq \tfrac{\epsilon}{2}\|z\|^2 \\ g_V^{\text{supp}} &= z \cdot z_{\text{jmax}} - z \cdot v \leq \tfrac{\epsilon}{2}\|z\|^2 \end{aligned} \tag{37}$$

In the early stages of the algorithm (say a few of the TypeI-then-TypeII rounds) a "nearly accurate" set of support vector indices 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 $\mathcal{A}$.

The main ideas relating to Type II iterations that we have described above can also be used in Platt's SMO algorithm. We have done this and have also done some initial testing. From this limited testing, it appears that these ideas do not cause a significant difference to the SMO algorithm. On the other hand, these ideas make a big difference for our algorithm, i.e., implementing Type II by sequentially running k over all support vector indices many times progresses much much slower compared to the implementation based on kmax.

Full details concerning the actual implementation of the full algorithm is adequately explained in the two appendices of this report. So we will omit the details here.

7. A Simplification Using the $\mathbf{b^2}$ term.

In this section we consider a simpler SVM formulation suggested recently by Friess et. al.[7] and Friess[8]. Suppose the term, $b^2/2$ is added to the cost functions of SVM-NV and SVM-VQ that were formulated in section 1. Then by appropriate rearrangement these problems can be written as a special SVM-NV problem in which b is absent. Let us take SVM-VQ to illustrate this point. Similar to (1) we can define

$$\tilde{w} = \begin{pmatrix} w \\ \sqrt{\tilde{C}}\xi \\ b \end{pmatrix}; \quad \tilde{z}_i = \begin{pmatrix} z_i \\ \frac{1}{\sqrt{\tilde{C}}} e_i \\ 1 \end{pmatrix}, i \in I; \quad \tilde{z}_j = \begin{pmatrix} z_j \\ -\frac{1}{\sqrt{\tilde{C}}} e_j \\ 1 \end{pmatrix}, j \in J. \tag{35}$$

Then it is easy to see that SVM-V transforms to *an instance* of the following problem:

$$\begin{aligned} \min \quad & \tfrac{1}{2}\|w\|^2 \\ \text{s.t.} \quad & w \cdot z_i \geq 1 \qquad \forall i \in I \\ & w \cdot z_j \leq -1 \quad \forall j \in J \end{aligned} \tag{SVM − NV − Nob}$$

As in the discussion below equation (1), if K denotes the Kernel function in the original problem, then $\tilde{K}$, the Kernel function for the transformed problem is given by

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

In the case of SVM-NV, when $b^2/2$ is added to the objective function, the transformation to an instance of SVM-NV-Nob is easier than the above steps; we only have to leave out ξ.

The rest of this section will correspond to the notations in SVM-NV-Nob. Let us assume the following.

Assumption A2. The feasible space of SVM-NV-Nob is non-empty. □

As noted in Remark 2, SVM-VQ always has non-empty feasible space and hence, when it is transformed to SVM-NV-Nob, it also has a non-empty feasible space and the above assumption holds. Let $W = \{z_i : i \in I\} \cup \{-z_i : i \in J\}$ and $Z = \text{co}W$. It is also easy to show that this problem is equivalent to:

$$\min \|z\| \quad \text{s.t.} \;\; z \in Z \tag{MNP}$$

Parallel to Theorem 1 we can derive the following result.

Theorem 6. $w^\star$ solves SVM-NV-Nob if and only if $z^\star$ solves MNP and

$$w^\star = \frac{1}{\|z^\star\|^2} z^\star \tag{38}$$

As in section 2, the following Wolfe dual problem of SVM-NV-Nob can be used to understand this result:

$$\max \sum_k \alpha_k - \frac{1}{2}\sum_k \sum_l \alpha_k \alpha_l y_k y_l z_k \cdot z_l \;\; \text{s.t.} \;\; \alpha_k \geq 0 \quad \forall k \tag{39}$$

Introduce $\lambda = \sum_k \alpha_k$,

$$\beta_k = \frac{\alpha_k}{\lambda} \quad \forall k \tag{40}$$

to rewrite (39) as

$$\begin{aligned} \max \quad & \lambda - \tfrac{\lambda^2}{2} \textstyle\sum_k \sum_l \beta_k \beta_l y_k y_l z_k \cdot z_l \\ \text{s.t.} \quad & \beta_k \geq 0 \;\; \forall k; \;\; \textstyle\sum_k \beta_k = 1. \end{aligned} \tag{41}$$

Optimizing λ for fixed β gives

$$\lambda^\star = \frac{1}{\sum_k \sum_l \beta_k \beta_l y_k y_l z_k \cdot z_l} \tag{42}$$

Substituting this in (41) yields the equivalent problem

$$\begin{array}{ll} \min & \frac{1}{2} \sum_k \sum_l \beta_k \beta_l y_k y_l z_k \cdot z_l \\ \text{s.t.} & \beta_k \geq 0 \ \ \forall k; \ \ \sum_k \beta_k = 1. \end{array}$$

which is equivalent to MNP if we observe that $P\beta \in \text{co}\{y_1 z_1, ..., y_m z_m\}$ where P is as in the discussion below Theorem 1. Since $w^\star = \sum_k \alpha_k^\star y_k z_k$, where $\alpha^\star$ denotes a solution of (39), we can use (40) and (42) to easily derive (38).

Remark 4. Unlike (21), here the set W that defines Z is directly prescribed in the space where U and V are located, and that too, in a very simple fashion. Therefore, Gilbert's algorithm, MDM algorithm, and its modifications apply directly without need for Minkowski set difference transformations. Thus their implementation becomes much simpler than that described in section 6. Since the ideas are very straight forward, we do not go into the details. □

8. Computational Comparisons.

In this section we compare the performance of our algorithm(s) against other iterative algorithms for SVM classifier design. Since the objective of this paper is to design fast algorithms, our focus in this section is only on studying computational efficiency. Whether or not the various SVM formulations tried in this paper lead to good generalizing solutions is a different issue that is not dealt with here; it is important, however, to do a separate study of this issue.

The following 6 methods were chosen for comparison.

1. **SMO_L :** Platt's SMO was applied to SVM-VL.

2. **SMO_Q :** SVM-VQ was converted to SVM-NV, which was then solved using SMO.

3. **NPA :** SVM-VQ was converted to SVM-NV, which was then solved by the near point algorithm of section 6.

4. **SOR :** Mangasarian and Musicant's SOR was applied to SVM-VL-BSQ, which is the modification of SVM-VL obtained by adding the cost, $b^2/2$ in the objective function.

5. **MNA :** SVM-VQ-BSQ, the modification of SVM-VQ obtained by adding the cost, $b^2/2$ in the objective function, was converted to SVM-NV-Nob and then solved using the minimum norm algorithm that we briefly outlined at the end of section 7.

6. **SOR_Q :** The same problem solved by MNA is solved using SOR. As we already mentioned in section 1, this way of using SOR is identical to Friess' Adatron algorithm[8]. However we have preferred to call it as SOR_Q since the heuristics (using two types of loops, sorting support vectors etc.) that cause significant improvements are basically taken from Mangasarian and Musicant's paper[17].

We implemented all these methods in Fortran and ran them using $g77$ on a 200 MHz Pentium machine. In the case of SMO and SOR, they were implemented exactly along the lines suggested by Platt[18] and Mangasarian and Musicant[17].

When testing an algorithm, it is important to specify clearly what criteria are used to stop the algorithm. When comparing different algorithms, this is even more important; also, in this case, it is crucial to select stopping criteria that are as uniform as possible. In our testing we have used stopping criteria which were suggested by Platt[18] and Joachims[11]. In each of the *transformed formulations* used by the six methods listed above, we have a pair of parallel hyperplanes that form the "boundaries" for the two classes. Let us call these hyperplanes as B_1 and B_2. The definitions of B_1 and B_2 differs for different transformed formulations. Let us define:

$$w = \sum_k \alpha_k y_k z_k, \quad \bar{u} = \sum_{i \in I} \beta_i z_i, \quad \bar{v} = \sum_{j \in J} \beta_j z_j, \quad \bar{z} = \sum_k \beta_k y_k z_k$$

Using these, Table 1 defines B_1 and B_2 for the six methods. Note that the notations in the last two columns correspond to the transformed formulation. For instance, for NPA, the z_k's used in the definitions correspond to those in SVM-NV, which are actually the $\tilde{z}_k$'s defined in (1).

Let D denote the distance between the two hyperplanes, B_1 and B_2. Given a relative tolerance, ϵ, let us form symmetric bands, each of thickness, ϵD, using the four shifted hyperplanes, H_1, H_2, H_3 and H_4, as illustrated in Figure 10. Let us simply refer to a variable used in the problems (α_k or β_k) as a *multiplier*. As we know, all multipliers have to satisfy non-negativity constraints. Further, in the case of SMO_L and SOR, the multipliers also have to satisfy the upper bound constraint: $\alpha_k \leq C$; for notational purposes we will take $C = \infty$ for all other methods. At a

Method	Problem addressed	Transformed Formulation	Iteration Variables	Eqns of B_1 and B_2 in z-space	D
SMO_L	SVM-VL	No Transformation	α, b	$w \cdot z + b = \pm 1$	$2/\|w\|$
SMO_Q	SVM-VQ	SVM-NV	α, b	$w \cdot z + b = \pm 1$	$2/\|w\|$
NPA	SVM-VQ	SVM-NV	β	$\bar{z} \cdot z = \bar{z} \cdot \bar{u}$, $\bar{z} \cdot z = \bar{z} \cdot \bar{v}$	$\|\bar{z}\|$
SOR	SVM-VL-BSQ	SVM-VL without b	α	$w \cdot z = \pm 1$	$2/\|w\|$
MNA	SVM-VQ-BSQ	SVM-NV-Nob	β	$\bar{z} \cdot z = \pm \bar{z} \cdot \bar{z}$	$2\|\bar{z}\|$
SOR_Q	SVM-VQ-BSQ	SVM-NV-Nob	α	$w \cdot z = \pm 1$	$2/\|w\|$

Table 1: Definitions of B_1, B_2 and D.

particular instance we say that the k-th multiplier *violates* the stopping criteria if *any one* of the following four conditions is met.

- $y_k = 1$, multiplier is positive and the resulting SVM output is above H_1.
- $y_k = -1$, multiplier is positive and the resulting SVM output is below H_4.
- $y_k = 1$, multiplier is lower than C and the resulting output is below H_2.
- $y_k = -1$, multiplier is lower than C and the resulting output is above H_3.

An algorithm is stopped when there is no multiplier that violates the stopping criteria. *The value, $\epsilon = 0.001$ was used for all experiments.* In the case of NPA, note that, for the same ϵ, the stopping criteria just described is tighter than those suggested in section 4 based on the g function, i.e., (15). For the sake of uniform comparison, we have used the tighter criteria mentioned above.

The following standard problems were used in our testing: Wisconsin Breast Cancer data[24,3]; Two Spirals data[22]; Checkers data[12]; UCI Adult data[19]; and Web page classification data[19, 11]. Except for Checkers data, for which we created a random set of points on a 4×4 checkers grid (see Figure 11), all other data sets were downloaded from the sites mentioned in the above references and were used in full for training, i.e., no division into training/validation/test sets was made. 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 and Web data. We have used only the data from the first, fourth and

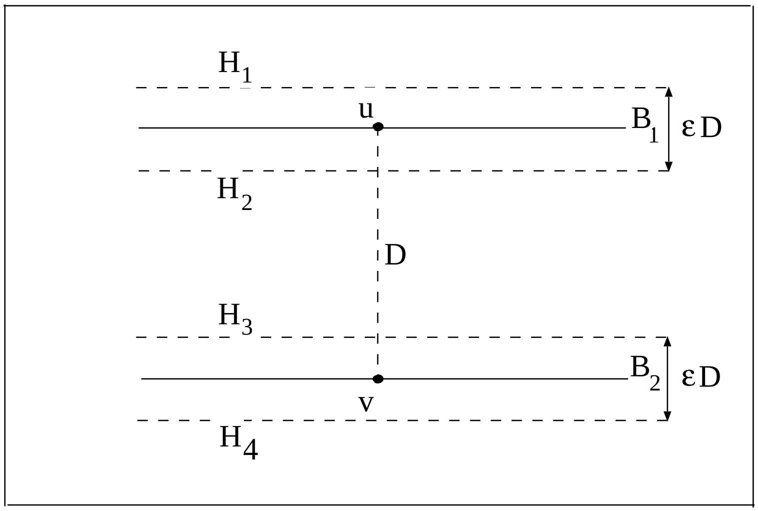

Figure 10: Illustration of the notations used in the stopping criteria.

seventh stages. The gaussian kernel,

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

was used in all experiments. The σ^2 values employed, together with n, the dimension of the input, and, m, the number of training points, are given in Table 2. The σ^2 values given in this table were chosen as follows. For the Adult and Web data the σ^2 values are the same as those used by Platt in his experiments on SMO; for other data, we chose σ^2 suitably to get good generalization.

Mangasarian and Musicant pointed out that the final solutions obtained with and without the $b^2/2$ term included in the cost function of an SVM formulation are typically close. We also have observed this in our experiments. Given this, the methods, SMO_Q, NPA, MNA and SOR_Q can be directly compared; similarly, SMO_L and SOR can be directly compared. To make a cross comparison between these two groups of 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 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 all 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, good performance of a method in a range of C or $\tilde{C}$ values is important. Whether or not a particular method has such a

Data Set	σ^2	n	m
Wisconsin Breast Cancer	4.0	9	683
Two Spirals	0.5	2	195
Checkers	0.5	2	465
Adult-1	10.0	123	1605
Adult-4	10.0	123	4781
Adult-7	10.0	123	16100
Web-1	10.0	300	2477
Web-4	10.0	300	7366
Web-7	10.0	300	24692

Table 2: Data Set Properties.

performance gets revealed clearly in our experiments.

As we have argued earlier in this paper, the cost of updating the caches is the dominant part of the total computational cost. Therefore, the total number of kernel evaluations is a very good indicator of the computing cost. Since such a measure is pretty much independent of the computing environment used, it is easy for others developing new algorithms to compare their methods against the ones investigated here, without actually running these methods again. In Tables 3-20 we have given the total number of kernel evaluations and the numbers of support, upper bound vectors used by various runs. (The number of upper bound vectors applies only for SMO_L and SOR.) For SOR_Q, the values are not given in many rows. This is because computing times involved in these runs were very large and so SOR_Q was not run till completion. In the case of Adult-7 and Web-7 data we compared only SMO and NPA. Figures 13-23 graphically show the variation of computing cost as a function of the solution space margin, M.

Figure 12 shows the classification region formed by the SVM designed using NPA, for the Checkers problem. Equations (18)-(20) were used to form the primal solution and the resulting classification region. It is clear that these equations lead to good results. The Checkers problem is separable in the z-space; but it is difficult to solve. For the no violations case, NPA gave best results, as evident from Figure 17 (low M values).

The following conclusions emerge from the computational results.

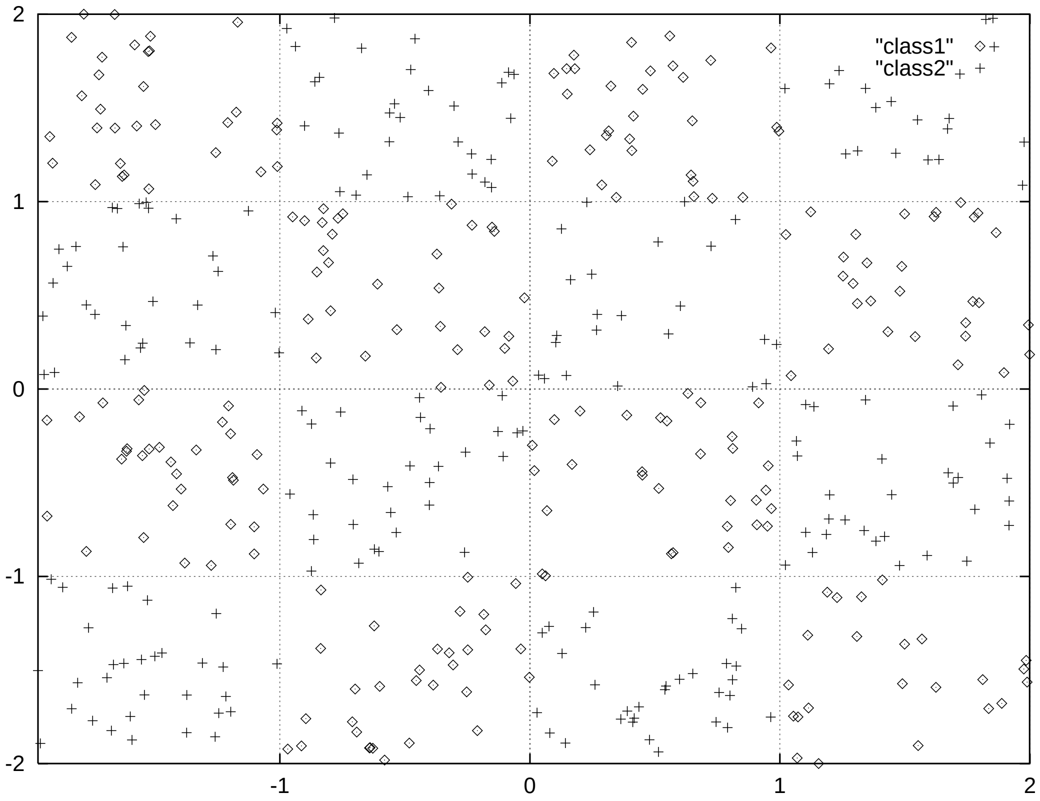

Figure 11: Data used for the Checkers problem.

1. For the SVM problem formulation which linearly penalizes classification violations, SMO is clearly better than SOR.

2. For the SVM problem formulation which quadratically penalizes classification violations, NPA is clearly the best.

3. SOR_Q (Adatron) is a very slow method.

4. SOR and MNA, the other two methods based on inclusion of $b^2/2$ term in the cost function, perform reasonably well, but not as fast as SMO_L and NPA. Also, their computational cost increases as C, $\tilde{C}$ are increased to large values.

5. When SMO_L and NPA are compared over a number of C, $\tilde{C}$ values, they perform quite closely, on the average. The runs on Adult and Web data point to the fact that both these methods scale equally well as a function of data size. At large C, $\tilde{C}$ values, NPA seems to have an edge over SMO_L on most datasets, which indicates that it is the better method for solving SVM-NV. Overall these two methods give the best performance.

6. The linear violation formulation results in a smaller number of support vectors when compared to the quadratic violation formulation. While the difference is very little in the Wisconsin Breast Cancer, Two Spirals, and Checkers datasets, the difference is very prominent in Adult and Web datasets, especially at small C, $\tilde{C}$ values. If such a large difference does occur in a particular problem, then the linear formulation does have a great advantage when using the SVM classifier for classification after the design process is over.

Conclusion.

In this paper we have given a new, fast algorithm for SVM classifier design using a carefully devised nearest point algorithm. The performance of this algorithm on a number of benchmark problems is very encouraging. Though not as simple as the SMO algorithm, our algorithm also is quite straightforward to implement, as indicated by the pseudocode in appendix B. Our nearest point formulation as well as the algorithm are special to classification problems and cannot be used for SVM regression.

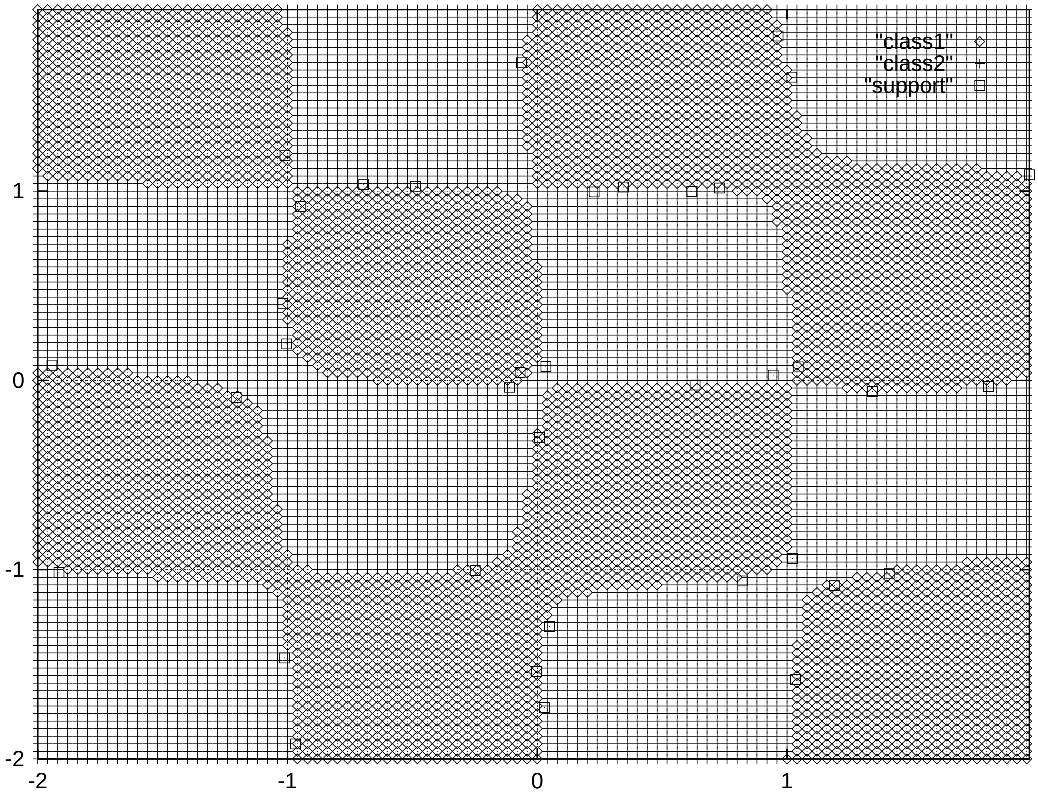

Figure 12: The classification regions formed by the NPA design for the Checkers problem for $\tilde{C} = \infty$. There are 37 support vectors.

References

[1] R.O. Barr, An efficient computational procedure for a generalized quadratic programming problem, *SIAM J. Control*, Vol.7, No.3, 1969, pp.415-429.

[2] R. Bennett and E.J. Bredensteiner, Geometry in learning, Tech. Report, Department of Mathematical Sciences, Rennselaer Polytechnic Institute, New York, 1996.

[3] R. Bennett and O.L. Mangasarian, Robust linear programming discrimination of two linearly inseparable sets, *Optimization Methods and Software*, Vol.1, 1992, pp.23-34.

[4] C.J.C. Burges, A tutorial on support vector machines for pattern recognition, *Data Mining and Knowledge Discovery*, Vol.2, Number 2, 1998.

[5] C. Cortes and V. Vapnik, Support Vector Networks, *Machine Learning*, Vol.20, 1995, pp.273-297.

[6] R. Fletcher, *Practical Methods of Optimizations.* John Wiley and Sons, 2nd Edition, 1987.

[7] T.T. Friess, N. Cristianini, and C. Campbell, The kernel adatron algorithm: a fast and simple learning procedure for support vector machines, *Proc. 15th International Conference on Machine Learning*, Morgan Kaufman Publishers, 1998.

[8] T.T. Friess, Support vector networks: The kernel adatron with bias and soft-margin, Tech. Report, The University of Sheffield, Dept. of Automatic Control and Systems Engineering, Sheffield, England, 1998.

[9] E.G. Gilbert, Minimizing the quadratic form on a convex set, *SIAM J. Control*, Vol.4, 1966, pp.61-79.

[10] 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. Robotics and Automation*, Vol.4, 1988, pp.193-203.

[11] T. Joachims, Making large-scale support vector machine learning practical, in B. Schölkopf, C. Burges, A. Smola. *Advances in Kernel Methods: Support Vector Machines*, MIT Press, Cambridge, MA, December 1998.

[12] L. Kaufman, Solving the quadratic programming problem arising in support vector classification, in B. Schölkopf, C. Burges, A. Smola. *Advances in Kernel Methods: Support Vector Machines*, MIT Press, Cambridge, MA, December 1998.

[13] C.L. Lawson and R.J. Hanson, *Solving least squares problems.* Prentice Hall, 1974.

[14] S.R. Lay, *Convex sets and their applications.* John Wiley and Sons, 1982.

[15] V.J. Lumelsky, On fast computation of distance between line segments, *Information Processing Letters*, Vol.21, 1985, pp.55-61.

[16] B.F. Mitchell, V.F. Dem'yanov, and V.N. Malozemov, Finding the point of a polyhedron closest to the origin, *SIAM J. Control*, Vol.12, 1974, pp.19-26.

[17] O.L. Mangasarian and D.R. Musicant, Successive overrelaxation for support vector machines, Tech. Report, Computer Sciences Dept., University of Wisconsin, Madison, WI, USA, 1998.

[18] J.C. Platt, Fast training of support vector machines using sequential minimal optimization, in B. Schölkopf, C. Burges, A. Smola. *Advances in Kernel Methods: Support Vector Machines*, MIT Press, Cambridge, MA, December 1998.

[19] J.C. Platt, Adult and Web Datasets. *http://www.research.microsoft.com/˜jplatt*

[20] N.K. Sancheti and S.S. Keerthi, Computation of certain measures of proximity between convex polytopes: A complexity viewpoint, *Proc. of IEEE International Conference on Robotics and Automation*, Nice, France, 1992, pp.2508-2513.

[21] A.J. Smola and B. Schölkopf, A tutorial on support vector regression, *NeuroCOLT Technical Report TR-1998-030*, Royal Holloway College, London, UK, 1998.

[22] Two Spirals Data.
ftp://ftp.boltz.cs.cmu.edu/pub/neural-bench/bench/two-spirals-v1.0.tar.gz

[23] V. Vapnik, *The Nature of Statistical Learning Theory.* Springer-Verlag, New York, 1995.

[24] Wisconsin Breast Cancer Data.
ftp://128.195.1.46/pub/machine-learning-databases/breast-cancer-wisconsin/

[25] P.Wolfe, Finding the nearest point in a polytope, *Mathematical Programming*, Vol.11, 1976, pp.128-149.

Appendix A. *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

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

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

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

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

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

Furthermore, suppose the following condition holds:

$$\bar{z}\cdot z < \|z\|^2 - \tilde{\epsilon} \tag{A.4}$$

Let us consider two cases.

<u>*Case 1.* $\|\bar{z}\|^2 \leq \bar{z}\cdot z$.</u> In this case, we have, by (A.3) and (A.4),

$$\|z\|^2 - \|\tilde{z}\|^2 = \|z\|^2 - \|\bar{z}\|^2 \geq \|z\|^2 - \bar{z}\cdot z > \tilde{\epsilon} \tag{A.5}$$

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

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

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

$$\|z\|^2 - \|\tilde{z}\|^2 > \frac{\tilde{\epsilon}^2}{L^2} \tag{A.7}$$

Combining the two cases we get

$$\|z\|^2 - \|\tilde{z}\|^2 > \min\{\tilde{\epsilon}, \frac{\tilde{\epsilon}^2}{L^2}\} \tag{A.8}$$

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 0 and solving for λ_1 and λ_2 yields

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

where: $e_{11} = \|Q-P\|^2$, $e_{22} = \|R-P\|^2$, $e_{12} = (Q-P)\cdot(R-P)$, $den = e_{11}e_{22} - e_{12}^2$, $f_1 = (P-Q)\cdot P$, 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 $P\cdot P - \lambda_2 f_1 - \lambda_3 f_2$.

Appendix B. *Pseudo Code for the NPP Algorithm.*

Procedure Line_Segment $(d_{11}, d_{22}, d_{12}, d, \lambda)$

Purpose: Computes details associated with x, the point of least norm on the line segment joining x_1 and x_2. Given $d_{11} = \|x_1\|^2$, $d_{22} = \|x_2\|^2$ and $d_{12} = x_1 \cdot x_2$ on input, the procedure returns: $d = \|x\|^2$; λ satisfying $0 \le \lambda \le 1$ such that $x = x_1 + \lambda(x_2 - x_1)$.

Steps:

% Initially set the first vertex.

$d = d_{11}$, $\lambda = 0$

% Check if second vertex improves d.

if $(d_{22} < d)$ then

$d = d_{22}$, $\lambda = 1$

end if

% Check if the interior of line segment improves d.

$den = d_{11} + d_{22} - 2d_{12}$

if ($den \leq 0$) return

$num = d_{11}d_{22} - d_{12}^2$

$\tilde{d} = num/den$

$\tilde{\lambda} = (d_{11} - d_{12})/den$

if ($0 < \tilde{\lambda} < 1$ and $\tilde{d} < d$) then

$d = \tilde{d}$, $\lambda = \tilde{\lambda}$

end if

return

Procedure Triangle $(d_{11}, d_{22}, d_{33}, d_{12}, d_{13}, d_{23}, d, \lambda_1, \lambda_2, \lambda_3)$

Purpose: Computes details associated with x, the point of least norm on the triangle formed by the convex hull of three points, x_1, x_2 and x_3. Given $d_{ij} = x_i \cdot x_j$ for all $1 \leq i, j \leq 3$ on input, the procedure returns: $d = \|x\|^2$; λ_1, λ_2, λ_3 satisfying $0 \leq \lambda_i \leq 1$ forall i, $\lambda_1 + \lambda_2 + \lambda_3 = 1$, such that $x = \lambda_1 x_1 + \lambda_2 x_2 + \lambda_3 x_3$.

Steps:

%Initially start with the line segment joining x_1 and x_2.

call Line_Segment$(d_{11}, d_{22}, d_{12}, d, \lambda_2)$

$\lambda_1 = 1 - \lambda_2$, $\lambda_3 = 0$

% Check if the line segment joining x_2 and x_3 improves d.

call Line_Segment$(d_{22}, d_{33}, d_{23}, \tilde{d}, \tilde{\lambda}_3)$

if ($\tilde{d} < d$) then

$d = \tilde{d}$, $\lambda_1 = 0$, $\lambda_2 = 1 - \tilde{\lambda}_3$, $\lambda_3 = \tilde{\lambda}_3$

end if

% Check if the line segment joining x_3 and x_1 improves d.

call Line_Segment$(d_{33}, d_{11}, d_{13}, \tilde{d}, \tilde{\lambda}_1)$

if $(\tilde{d} < d)$ then

$d = \tilde{d}$, $\lambda_1 = \tilde{\lambda}_1$, $\lambda_2 = 0$, $\lambda_3 = 1 - \tilde{\lambda}_1$

end if

% Check if the interior of the triangle improves d.

$e_{11} = d_{22} + d_{11} - 2d_{12}$, $e_{22} = d_{33} + d_{11} - 2d_{13}$, $e_{12} = d_{23} - d_{12} - d_{13} + d_{11}$

$den = e_{11}e_{22} - e_{12}^2$

if $(den \leq 0)$ return

$f_1 = d_{11} - d_{12}$, $f_2 = d_{11} - d_{13}$

$\tilde{\lambda}_2 = (e_{22}f_1 - e_{12}f_2)/den$, $\tilde{\lambda}_3 = (-e_{12}f_1 + e_{11}f_2)/den$, $\tilde{\lambda}_1 = 1 - \tilde{\lambda}_2 - \tilde{\lambda}_3$

$\tilde{d} = d_{11} - \tilde{\lambda}_2 f_1 - \tilde{\lambda}_3 f_2$

if $(\tilde{\lambda}_1 > 0$, $\tilde{\lambda}_2 > 0$, $\tilde{\lambda}_3 > 0$ and $\tilde{d} < d)$ then

$d = \tilde{d}$, $\lambda_1 = \tilde{\lambda}_1$, $\lambda_2 = \tilde{\lambda}_2$, $\lambda_3 = \tilde{\lambda}_3$

end if

return

Procedure Two_Lines $(r, s_1, s_2, d_1, d_2, dold, d, \lambda_1, \lambda_2, flag)$

Purpose: Let L_1 be the line segment joining A and B, and, L_2 be the line segment joining C and D. This procedure computes details associated with $u \in L_1$ and $v \in L_2$ the pair of points which minimize the distance between the two lines. The inputs are: $r = (B - A) \cdot (D - C)$; $s_1 = (B - A) \cdot (C - A)$, $s_2 = (D - C) \cdot (C - A)$, $d_1 = \|B - A\|^2$, $d_2 = \|D - C\|^2$ and $dold = \|C - A\|^2$. On return, λ_1, λ_2 and d are such that: $u = A + \lambda_1(B - A)$; $v = C + \lambda_2(D - C)$ and $d = \|u - v\|^2$. If there is ill-conditioning, and no reasonable computation is possible, $flag$ is set to 0; otherwise it is set to 1.

Steps:

% This is a simplified version of Lumelsky's method[15].

If $d_1 \leq 0$ or $d_2 \leq 0$ set $flag = 0$ and return.

$den = d_1 d_2 - r^2$. If $den \leq 0$ set $flag = 0$ and return.

$\lambda_1 = (s_1 d_2 - s_2 r)/den$

if $(\lambda_1 < 0)$ then

$\lambda_1 = 0$

else if $(\lambda_1 > 1)$ then

$\lambda_1 = 1$

end if

$\lambda_2 = (\lambda_1 r - s_2)/d_2$

if $(\lambda_2 < 0)$ then

$\lambda_2 = 0$

else if $(\lambda_2 > 1)$ then

$\lambda_2 = 1$

else

$d = dold + d_1\lambda_1^2 + d_2\lambda_2^2 - 2r\lambda_1\lambda_2 - 2s_2\lambda_2 - 2s_1\lambda_1$

$flag = 1$

return

end if

$\lambda_1 = (\lambda_2 r + s_1)/d_1$

if $(\lambda_1 < 0)$ then

$\lambda_1 = 0$

else if $(\lambda_1 > 1)$ then

$\lambda_1 = 1$

end if

$d = dold + d_1\lambda_1^2 + d_2\lambda_2^2 - 2r\lambda_1\lambda_2 - 2s_2\lambda_2 - 2s_1\lambda_1$

$flag = 1$

return

Procedure Gilbert_Step (kmax, success)

Purpose: Given an index, kmax that violates the optimality criteria, this procedure does one modified Gilbert step. If there is a failure due to some ill-conditioned situation, the procedure returns success $= 0$; otherwise it returns success $= 1$. All required variables and caches are updated if success $= 1$. This procedure is called from *Take_Step*, but very rarely, say just once or twice during the entire algorithm.

Steps:

% Do two cases depending on kmax.

if $y_{\text{kmax}} = 1$ then

imax = kmax

$d_{11} = \delta_{zz}$

dimax = kernel(imax,imax)

$d_{22} = \text{dimax} + \delta_{vv} - 2e_v(imax)$

$d_{12} = e_u(\text{imax}) - e_v(\text{imax}) - \delta_{uv} + \delta_{vv}$

call Line_Segment$(d_{11}, d_{22}, d_{12}, d, \lambda)$

if $(d \geq \delta_{zz})$ then

success=0

return

end if

success=1, $\delta_{zz} = d$

$\delta_{uu} = (1-\lambda)^2\delta_{uu} + \lambda^2\text{dimax} + \lambda(1-\lambda)e_u(\text{imax})$

$\delta_{uv} = (1-\lambda)\delta_{uv} + \lambda e_v(\text{imax})$

$e_u(k) = (1-\lambda)e_u(k) + \lambda\text{kernel}(\text{imax}, k) \quad \forall \;\; k \in \hat{I} \cup \hat{J}$

$\beta_i = (1-\lambda)\beta_i \;\; \forall \;\; i \in \hat{I}$

$\beta_{\text{imax}} = \beta_{\text{imax}} + \lambda$

else

jmax = kmax

$d_{11} = \delta_{zz}$

djmax = kernel(jmax,jmax)

$d_{22} = \text{djmax} + \delta_{uu} - 2e_u(jmax)$

$d_{12} = e_v(\text{jmax}) - e_u(\text{jmax}) - \delta_{uv} + \delta_{uu}$

call Line_Segment$(d_{11}, d_{22}, d_{12}, d, \lambda)$

if $(d \geq \delta_{zz})$ then

success=0

return

end if

success=1, $\delta_{zz} = d$

$\delta_{vv} = (1-\lambda)^2\delta_{vv} + \lambda^2 \text{djmax} + \lambda(1-\lambda)e_v(\text{jmax})$

$\delta_{uv} = (1-\lambda)\delta_{uv} + \lambda e_u(\text{jmax})$

$e_v(k) = (1-\lambda)e_v(k) + \lambda \text{kernel}(\text{jmax}, k) \quad \forall \ k \in \hat{I} \cup \hat{J}$

$\beta_j = (1-\lambda)\beta_j \ \ \forall \ j \in \hat{J}$

$\beta_{\text{jmax}} = \beta_{\text{jmax}} + \lambda$

end if

return

Procedure Take_Step (kmax, success)

Purpose: Given an index, kmax that violates the optimality criteria, this procedure chooses kmin and does one iteration. If there is a failure due to some ill-conditioned situation, the procedure returns success = 0; otherwise it returns success = 1. All required variables and caches are updated if success = 1.

Steps:

% Compute kmin.

$zdu = \delta_{uu} - \delta_{uv}$, $zdv = \delta_{uv} - \delta_{vv}$

imin = $\arg\min\{zdu - e_u(i) + e_v(i) : i \in \hat{I}, \beta_i > 0\}$

jmin = $\arg\min\{-zdv + e_u(j) - e_v(j) : j \in \hat{J}, \beta_j > 0\}$

if $(zdu - e_u(\text{imin}) + e_v(\text{imin}) < -zdv + e_u(\text{jmin}) - e_v(\text{jmin}))$ then

kmin = imin

else

kmin = jmin

end if

% Do a modified Gilbert step if $\beta_{\text{kmin}} = 1$.

if $(\beta_{\text{kmin}} = 1)$ then

call Gilbert_Step(kmax, success)

return

end if

$\mu = \beta_{\text{kmin}}/(1-\beta_{\text{kmin}})$

$\psi = \text{kernel}(\text{kmin}, \text{kmax})$

% There are four cases to consider.

% Case 1 ...

if $y_{\text{kmax}} = 1$ and $y_{\text{kmin}} = 1$ then

imax = kmax, imin = kmin

$d_{11} = \delta_{zz}$

$t_1 = \delta_{uu} - e_u(\text{imin}) - \delta_{uv} + e_v(\text{imin})$

$d_{22} = \delta_{zz} + \mu^2(\delta_{uu} + \text{kernel}(\text{imin}, \text{imin}) - 2e_u(\text{imin})) + 2\mu t_1$

$d_{33} = \text{kernel}(\text{imax}, \text{imax}) + \delta_{vv} - 2e_v(\text{imax})$

$d_{12} = \delta_{zz} + \mu t_1$

$d_{13} = e_u(\text{imax}) - \delta_{uv} - e_v(\text{imax}) + \delta_{vv}$

$d_{23} = d_{13} + \mu(e_u(\text{imax}) - \delta_{uv} - \psi + e_v(\text{imin}))$

call Triangle $(d_{11}, d_{22}, d_{33}, d_{12}, d_{13}, d_{23}, d, \lambda_1, \lambda_2, \lambda_3)$

if $(d \geq \delta_{zz})$ then

success = 0

return

end if

success = 1, $\delta_{zz} = d$

$\bar\lambda_1 = \lambda_1 + \lambda_2 + \lambda_2\mu$, $\bar\lambda_2 = -\lambda_2\mu$, $\bar\lambda_3 = \lambda_3$

$\delta_{uu} = \bar\lambda_1^2\delta_{uu} + \bar\lambda_2^2\text{kernel}(\text{imin}, \text{imin}) + \bar\lambda_3^2\text{kernel}(\text{imax}, \text{imax}) + 2\bar\lambda_1\bar\lambda_2 e_u(\text{imin}) +$

$2\bar\lambda_1\bar\lambda_3 e_u(\text{imax}) + 2\bar\lambda_2\bar\lambda_3\psi$

$\delta_{uv} = \bar\lambda_1\delta_{uv} + \bar\lambda_2 e_v(\text{imin}) + \bar\lambda_3 e_v(\text{imax})$

$e_u(k) = \bar\lambda_1 e_u(k) + \bar\lambda_2\text{kernel}(\text{imin}, k) + \bar\lambda_3\text{kernel}(\text{imax}, k)\ \forall\ k \in \hat I \cup \hat J$

$\beta_i = \bar\lambda_1\beta_i\ \forall\ i \in \hat I$

$\beta_{\text{imax}} = \beta_{\text{imax}} + \bar\lambda_3$

if $(\lambda_1 = 0)$ then

remove imin from $\hat I$

else

$\beta_{\text{imin}} = \beta_{\text{imin}} + \bar\lambda_2$

end if

% Case 2 ...

else if $y_{\text{kmax}} = -1$ and $y_{\text{kmin}} = -1$ then

jmax = kmax, jmin = kmin

$d_{11} = \delta_{zz}$

$t_1 = \delta_{vv} - e_v(\text{jmin}) - \delta_{uv} + e_u(\text{jmin})$

$d_{22} = \delta_{zz} + \mu^2(\delta_{vv} + \text{kernel}(\text{jmin}, \text{jmin}) - 2e_v(\text{jmin})) + 2\mu t_1$

$d_{33} = \text{kernel}(\text{jmax}, \text{jmax}) + \delta_{uu} - 2e_u(\text{jmax})$

$d_{12} = \delta_{zz} + \mu t_1$

$d_{13} = e_v(\text{jmax}) - \delta_{uv} - e_u(\text{jmax}) + \delta_{uu}$

$d_{23} = d_{13} + \mu(e_u(\text{jmin}) - \delta_{uv} - \psi + e_v(\text{jmax}))$

call Triangle $(d_{11}, d_{22}, d_{33}, d_{12}, d_{13}, d_{23}, d, \lambda_1, \lambda_2, \lambda_3)$

if $(d \geq \delta_{zz})$ then

success = 0

return

end if

success = 1, $\delta_{zz} = d$

$\bar{\lambda}_1 = \lambda_1 + \lambda_2 + \lambda_2\mu$, $\bar{\lambda}_2 = -\lambda_2\mu$, $\bar{\lambda}_3 = \lambda_3$

$\delta_{vv} = \bar{\lambda}_1^2\delta_{vv} + \bar{\lambda}_2^2\text{kernel}(\text{jmin}, \text{jmin}) + \bar{\lambda}_3^2\text{kernel}(\text{jmax}, \text{jmax}) + 2\bar{\lambda}_1\bar{\lambda}_2 e_v(\text{jmin}) +$

$+2\bar{\lambda}_1\bar{\lambda}_3 e_v(\text{jmax}) + 2\bar{\lambda}_2\bar{\lambda}_3\psi$

$\delta_{uv} = \bar{\lambda}_1\delta_{uv} + \bar{\lambda}_2 e_u(\text{jmin}) + \bar{\lambda}_3 e_u(\text{jmax})$

$e_v(k) = \bar{\lambda}_1 e_v(k) + \bar{\lambda}_2\text{kernel}(\text{jmin}, k) + \bar{\lambda}_3\text{kernel}(\text{jmax}, k)\ \forall\ k \in \hat{I} \cup \hat{J}$

$\beta_j = \bar{\lambda}_1\beta_j\ \forall\ j \in \hat{J}$

$\beta_{\text{jmax}} = \beta_{\text{jmax}} + \bar{\lambda}_3$

if $(\lambda_1 = 0)$ then

remove jmin from $\hat{J}$

else

$\beta_{\text{jmin}} = \beta_{\text{jmin}} + \bar{\lambda}_2$

end if

% Case 3 ...

else if $y_{\text{kmax}} = 1$ and $y_{\text{kmin}} = -1$ then

imax = kmax, jmin = kmin

$r = \mu(e_v(\text{imax}) - \psi - \delta_{uv} + e_u(\text{jmin}))$

$s_1 = (e_v(\text{imax}) - e_u(\text{imax}) - \delta_{uv} + \delta_{uu})$

$s_2 = \mu(\delta_{vv} - \delta_{uv} - e_v(\text{jmin}) + e_u(\text{jmin}))$

$d_1 = \text{kernel}(\text{imax}, \text{imax}) + \delta_{uu} - 2e_u(\text{imax})$

$d_2 = \mu^2(\delta_{vv} + \text{kernel}(\text{jmin}, \text{jmin}) - 2e_v(\text{jmin}))$

$dold = \delta_{zz}$

call Two_Lines $(r, s_1, s_2, d_1, d_2, dold, d, \lambda_1, \lambda_2, flag)$

if $(flag = 0$ or $d \geq \delta_{zz})$ then

success = 0

return

end if

success = 1, $\delta_{zz} = d$

$\delta_{uu} = (1 - \lambda_1)^2\delta_{uu} + \lambda_1^2\text{kernel}(\text{imax}, \text{imax}) + 2(1 - \lambda_1)\lambda_1 e_u(\text{imax})$

$\delta_{vv} = (1 + \lambda_2\mu)^2\delta_{vv} + (\lambda_2\mu)^2\text{kernel}(\text{jmin}, \text{jmin}) - 2(1 + \lambda_2\mu)\lambda_2\mu e_v(\text{jmin})$

$\delta_{uv} = (1 - \lambda_1)(1 + \lambda_2\mu)\delta_{uv} - (1 - \lambda_1)\lambda_2\mu e_u(\text{jmin}) +$

$(1 + \lambda_2\mu)\lambda_1 e_v(\text{imax}) - \lambda_1\lambda_2\mu\psi$

$e_u(k) = (1 - \lambda_1)e_u(k) + \lambda_1\text{kernel}(\text{imax}, k) \ \forall\ k \in \hat{I} \cup \hat{J}$

$e_v(k) = (1 + \lambda_2\mu)e_v(k) - \lambda_2\mu\text{kernel}(\text{jmin}, k) \ \forall\ k \in \hat{I} \cup \hat{J}$

$\beta_i = (1 - \lambda_1)\beta_i \ \forall\ i \in \hat{I}$

$\beta_j = (1 + \lambda_2\mu)\beta_j \ \forall\ j \in \hat{J}$

$\beta_{\text{imax}} = \beta_{\text{imax}} + \lambda_1$

if $(\lambda_2 = 1)$ then

remove jmin from $\hat{J}$

else

$\beta_{\text{jmin}} = \beta_{\text{jmin}} - \lambda_2\mu$

end if

% Case 4 ...

else if $y_{\text{kmax}} = -1$ and $y_{\text{kmin}} = 1$ then jmax = kmax, imin = kmin

$r = \mu(e_u(\text{jmax}) - \psi - \delta_{uv} + e_v(\text{imin}))$

$s_1 = (e_u(\text{jmax}) - e_v(\text{jmax}) - \delta_{uv} + \delta_{vv})$

$s_2 = \mu(\delta_{uu} - \delta_{uv} - e_u(\text{imin}) + e_v(\text{imin}))$

$d_1 = \text{kernel}(\text{jmax}, \text{jmax}) + \delta_{vv} - 2e_v(\text{jmax})$

$d_2 = \mu^2(\delta_{uu} + \text{kernel}(\text{imin}, \text{imin}) - 2e_u(\text{imin}))$

$dold = \delta_{zz}$

call Two_Lines $(r, s_1, s_2, d_1, d_2, dold, d, \lambda_1, \lambda_2, flag)$

if $(flag = 0$ or $d \geq \delta_{zz})$ then

    success = 0

    return

end if

success = 1, $\delta_{zz} = d$

$\delta_{vv} = (1 - \lambda_1)^2\delta_{vv} + \lambda_1^2\text{kernel}(\text{jmax}, \text{jmax}) + 2(1 - \lambda_1)\lambda_1 e_v(\text{jmax})$

$\delta_{uu} = (1 + \lambda_2\mu)^2\delta_{uu} + (\lambda_2\mu)^2\text{kernel}(\text{imin}, \text{imin}) - 2(1 + \lambda_2\mu)\lambda_2\mu e_u(\text{imin})$

$\delta_{uv} = (1 - \lambda_1)(1 + \lambda_2\mu)\delta_{uv} - (1 - \lambda_1)\lambda_2\mu e_v(\text{imin}) +$

    $(1 + \lambda_2\mu)\lambda_1 e_u(\text{jmax}) - \lambda_1\lambda_2\mu\psi$

$e_v(k) = (1 - \lambda_1)e_v(k) + \lambda_1\text{kernel}(\text{jmax}, k)\ \forall\ k \in \hat{I} \cup \hat{J}$

$e_u(k) = (1 + \lambda_2\mu)e_u(k) - \lambda_2\mu\text{kernel}(\text{imin}, k)\ \forall\ k \in \hat{I} \cup \hat{J}$

$\beta_j = (1 - \lambda_1)\beta_j\ \forall\ j \in \hat{J}$

$\beta_i = (1 + \lambda_2\mu)\beta_i\ \forall\ i \in \hat{I}$

$\beta_{\text{jmax}} = \beta_{\text{jmax}} + \lambda_1$

if $(\lambda_2 = 1)$ then

    remove imin from $\hat{I}$

else

    $\beta_{\text{imin}} = \beta_{\text{imin}} - \lambda_2\mu$

end if

end if

return

Main Routine

Comments: As in the paper, I and J respectively indicate the index sets for the two classes. In the implementation this is best handled by means of a binary vector, y: $y_k = 1$ indicating $k \in I$ and $y_k = -1$ indicating $k \in J$. The user has to give the kernel function: kernel(i,j) denotes the kernel function applied to the i-th and j-th input vectors. The lists of support vectors, $\hat{I}$ and $\hat{J}$ can be easily represented by maintaining a single integer array, alind: alind(k) = 1 means $k \in \hat{I} \cup \hat{J}$ and alind(k) = 0 means otherwise. Whether k is in $\hat{I}$ or $\hat{J}$ can be determined using y. *Memory:* Apart from the integer arrays, y and alind mentioned above, the implementation needs three real arrays, β, e_u and e_v; each of these arrays have size m, where m denotes the number of training vectors. Of course, since we have directly prescribed the kernel function, we have not made any mention of the memory needed for the input training vectors. These vectors can be handled according to the problem's need, such as real/integer, dense/sparse etc.

Steps:

Initially choose any $i \in I$, $j \in J$ and set:

$\hat{I} = \{i\}$, $\hat{J} = \{j\}$, $\beta_i = 1$, $\beta_j = 1$, $n_{supp} = 2$

$\delta_{uu} = \text{kernel}(i,i)$, $\delta_{vv} = \text{kernel}(j,j)$, $\delta_{uv} = \text{kernel}(i,j)$, $\delta_{zz} = \delta_{uu} + \delta_{vv} - 2\delta_{uv}$

$e_u(i) = \delta_{uu}$, $e_u(j) = \delta_{uv}$, $e_v(i) = \delta_{uv}$, $e_v(j) = \delta_{vv}$,

Examine_NonSV=1, SV_Optimality=1, NonSV_Optimality=0

Num_Type1_Updates=0, Num_Type2_Updates=0

While (SV_Optimality=0 or NonSV_Optimality=0)

{

if any $k \in \hat{I} \cup \hat{J}$ exists with $\beta_k \leq 0$, remove k from $\hat{I} \cup \hat{J}$

if (Examine_NonSV=1) then

Num_Type1_Updates = 0, NonSV_Optimality=1, Examine_NonSV=0

For each $k \notin \hat{I} \cup \hat{J}$ do:

{

$e_u(k) = \sum_{i \in \hat{I}} \beta_i \text{kernel}(i,k)$, $e_v(k) = \sum_{j \in \hat{J}} \beta_j \text{kernel}(j,k)$

$zdk = e_u(k) - e_v(k)$, $zdu = \delta_{uu} - \delta_{uv}$, $zdv = \delta_{uv} - \delta_{vv}$

if ($k \in I$ and $zdu - zdk \geq 0.5\epsilon\delta_{zz}$) or ($k \in J$ and $zdk - zdv \geq 0.5\epsilon\delta_{zz}$) then

NonSV_Optimality=0

$\beta_k = 0$

if ($y_k = 1$) set $\hat{I} = \hat{I} \cup \{k\}$; else, set $\hat{J} = \hat{J} \cup \{k\}$

call Take_Step (k, success)

if (success = 0) then

if ($y_k = 1$) remove k from $\hat{I}$; else, remove k from $\hat{J}$

else

Num_Type1_Updates = Num_Type1_Updates + 1

end if

end if

}

else

Examine_NonSV=1

$n_{supp}^{old} = n_{supp}$; $n_{supp} = |\hat{I} \cup \hat{J}|$

if ($|n_{supp}^{old} - n_{supp}| \geq 0.05 n_{supp}$) then

Max_Updates= m

else

Max_Updates= $10m$

end if

Loop_Completed = 0, Num_Type2_Updates = 0

While (Loop_Completed = 0)

{

$zdu = \delta_{uu} - \delta_{uv}$, $zdv = \delta_{uv} - \delta_{vv}$

imax = $\arg\max\{zdu - zdi : zdi = e_u(i) - e_v(i), i \in \hat{I}\}$

jmax = $\arg\max\{zdj - zdv : zdj = e_u(j) - e_v(j), j \in \hat{J}\}$

$g_U = (zdu - e_u(\text{imax}) + e_v(\text{imax}))$, $g_V = (e_u(\text{jmax}) - e_v(\text{jmax}) - zdv)$

if $g_U \leq 0.5\epsilon\delta_{zz}$ and $g_V \leq 0.5\epsilon\delta_{zz}$ then

Loop_Completed = 1, SV_Optimality = 1

```
else
  if (g_U ≥ g_V) then set kmax = imax; else, set kmax = jmax
  call Take_Step (kmax, success)
  if (success = 0) then
    Loop_Completed = 1, SV_Optimality = 0
  else
    Num_Type2_Updates = Num_Type2_Updates + 1
  end if
  if (Num_Type2_Updates > Max_Updates) then
    Loop_Completed = 1, SV_Optimality = 0
  end if
end if
}
end if
if (Num_Type1_Updates = 0 and Num_Type2_Updates = 0) then
  Optimality criteria not satisfied. Algorithm is unable to improve solution anymore.
  Perhaps ε is too small. Evaluate g = g_U + g_V over the entire training set to
  decide accuracy of the current solution; see Section 4 of the paper.
end if
}
```

Inference: For doing inference using the designed parameters, set $h_U(-\hat{z}) = e_v(\text{imax}) - e_u(\text{imax})$, $h_V(\hat{z}) = e_u(\text{jmax}) - e_v(\text{jmax})$, compute γ and $\hat{b}$ using (19) and (20) and use the sign of $[\gamma \sum_{k \in \hat{I} \cup \hat{J}} y_k \beta_k \text{kernel}(x, x_k)] + \hat{b}$ to decide if a given x belongs to Class 1 or Class 2.

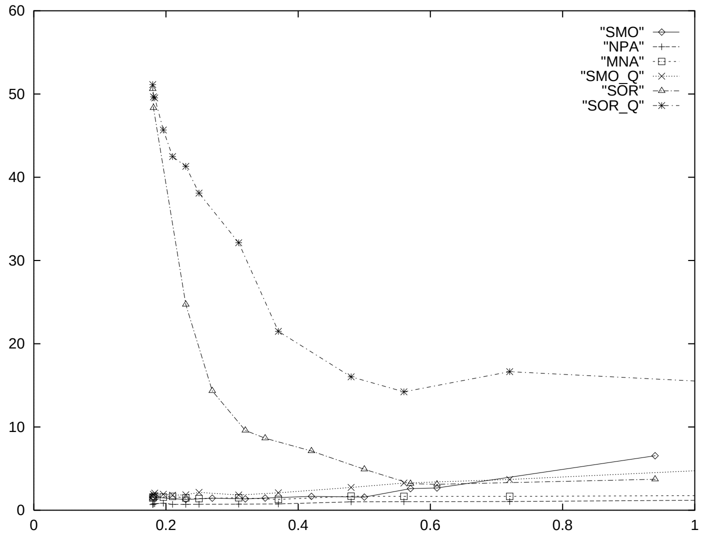

Figure 13: Wisconsion Breast Cancer data: Number of Kernel Evaluations $\times$ 10^{-6} (vertical axis) shown as a function of M (horizontal axis)

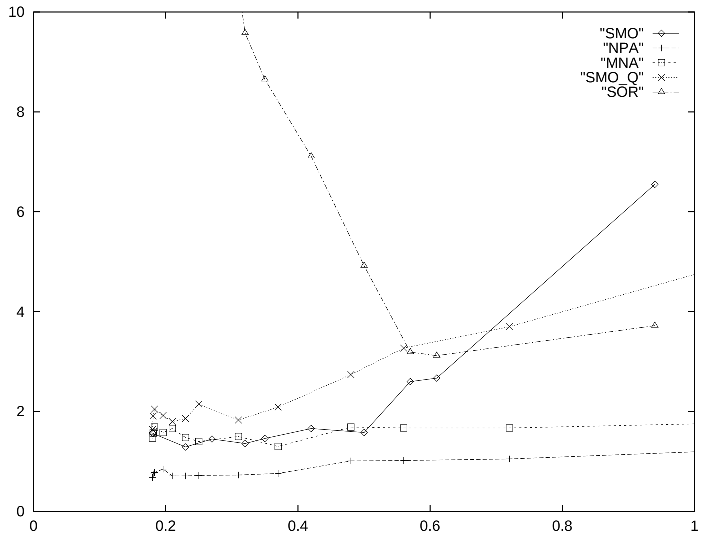

Figure 14: Figure 13 redrawn after leaving out SOR_Q so as to compare other methods clearly.

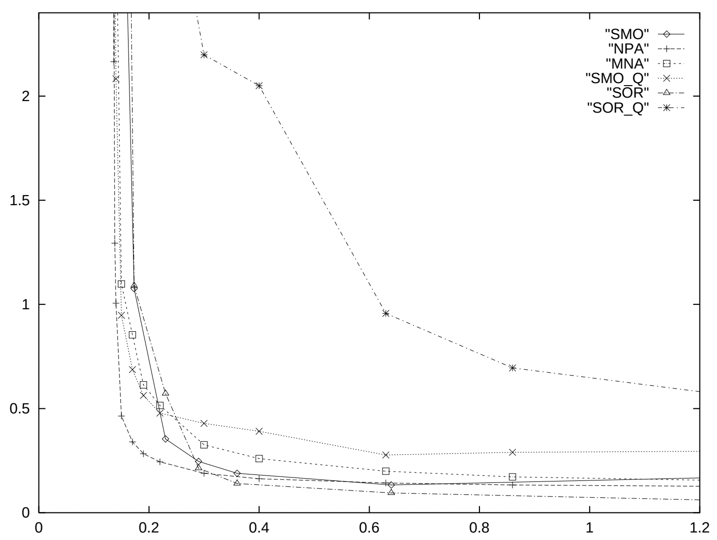

Figure 15: Two Spirals data: Number of Kernel Evaluations $\times 10^{-6}$ (vertical axis) shown as a function of M (horizontal axis)

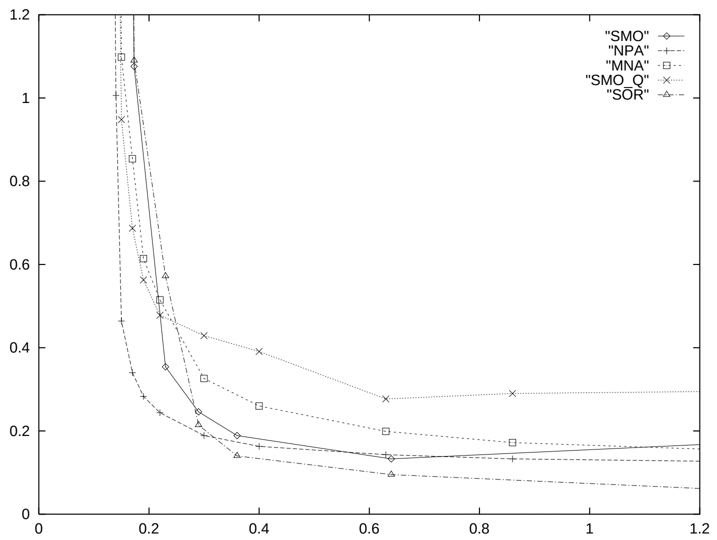

Figure 16: Figure 15 redrawn after leaving out SOR_Q so as to compare other methods clearly.

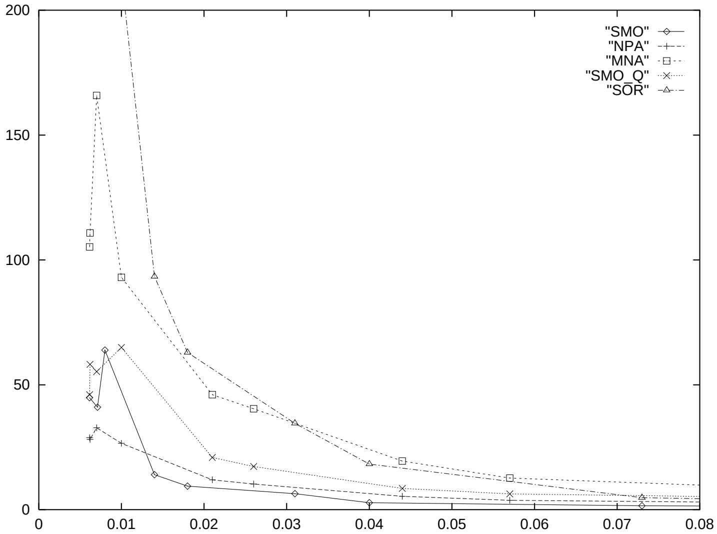

Figure 17: Checkers data: Number of Kernel Evaluations $\times 10^{-6}$ (vertical axis) shown as a function of M (horizontal axis)

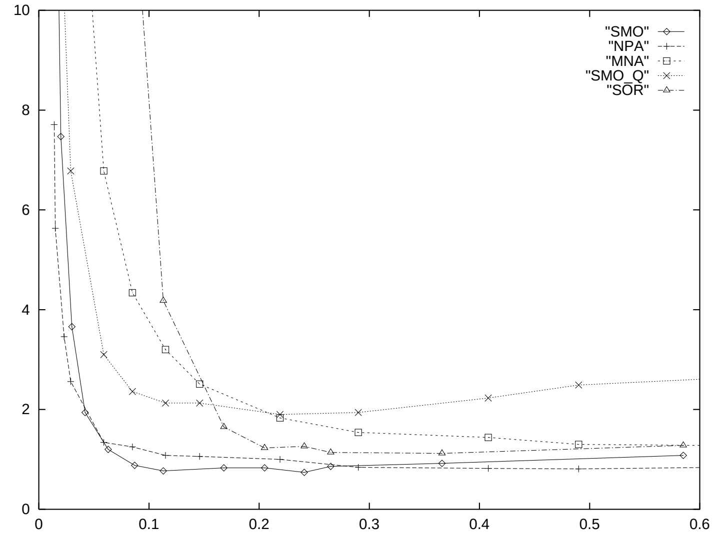

Figure 18: Adult-1 data: Number of Kernel Evaluations $\times 10^{-7}$ (vertical axis) shown as a function of M (horizontal axis)

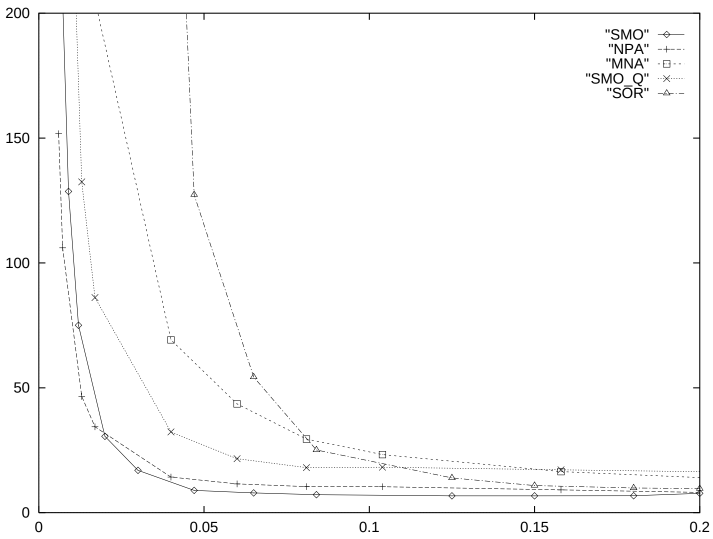

Figure 19: Adult-4 data: Number of Kernel Evaluations $\times\ 10^{-7}$ (vertical axis) shown as a function of M (horizontal axis)

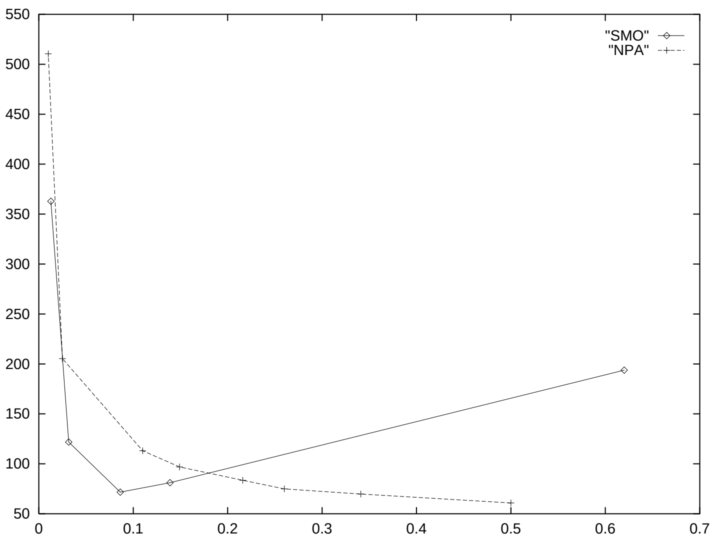

Figure 20: Adult-7 data: Number of Kernel Evaluations $\times\ 10^{-7}$ (vertical axis) shown as a function of M (horizontal axis)

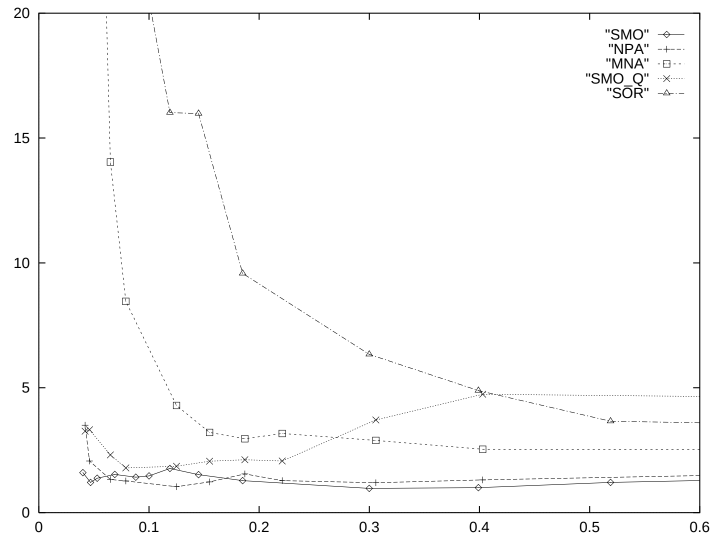

Figure 21: Web-1 data: Number of Kernel Evaluations $\times 10^{-7}$ (vertical axis) shown as a function of M (horizontal axis)

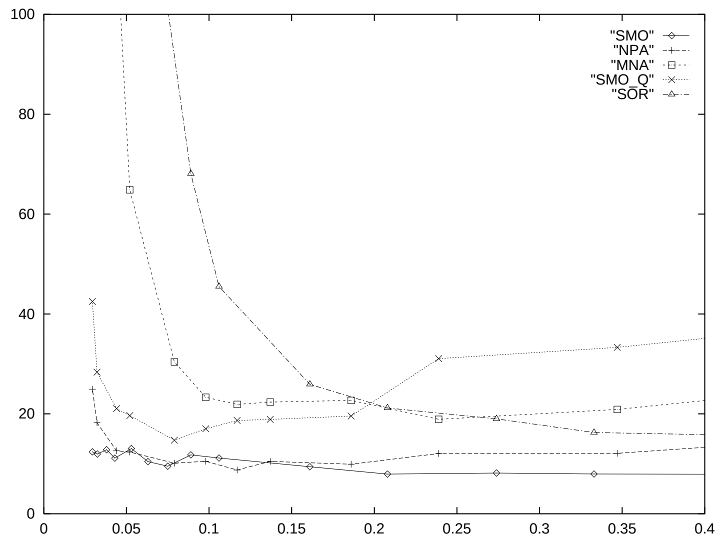

Figure 22: Web-4 data: Number of Kernel Evaluations $\times 10^{-7}$ (vertical axis) shown as a function of M (horizontal axis)

M	C	$\tilde{C}$	SMO	SMO_Q	NPA	SOR	MNA	SOR_Q
1.13		0.03		5.23	1.21		1.79	15.00
0.94	0.02		6.55			3.72		
0.72		0.1		3.70	1.04		1.67	16.65
0.61	0.04		2.67			3.12		
0.57	0.06		2.60			3.19		
0.56		0.2		3.27	0.99		1.67	14.23
0.5	0.1		1.58			4.92		
0.48		0.3		2.74	1.31		1.69	16.03
0.42	0.2		1.66			7.11		
0.37		0.6		2.09	0.74		1.30	21.49
0.35	0.4		1.46			8.65		
0.32	0.5		1.36			9.58		
0.31		1.0		1.83	0.85		1.50	32.13
0.27	0.7		1.45			14.34		
0.25		2.0		2.15	0.59		1.40	38.10
0.23	1.0		1.29			24.72		
0.23		3.0		1.86	0.71		1.48	41.31
0.21		5.0		1.80	0.73		1.66	42.49
0.196		10.0		1.92	0.85		1.58	45.67
0.183		50.0		2.05	0.78		1.69	49.53
0.181	2.0		1.56			48.33		
0.181		10^2		1.91	0.78		1.57	49.66
0.18	3.0		1.57			50.62		
0.18		5×10^2		1.64	0.69		1.47	51.14

Table 3: Wisconsin Breast Cancer data: Number of Kernel evaluations $\times 10^{-6}$

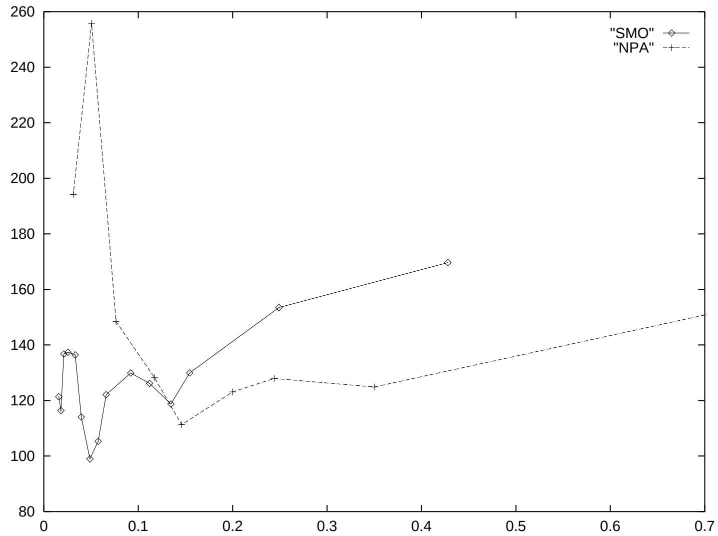

Figure 23: Web-7 data: Number of Kernel Evaluations $\times 10^{-7}$ (vertical axis) shown as a function of M (horizontal axis)

M	C	$\tilde{C}$	SMO	SMO_Q	NPA	SOR	MNA	SOR_Q
1.13		0.03		652	652		652	652
0.94	0.02		5, 476			10, 462		
0.72		0.1		505	505		505	505
0.61	0.04		136, 311			56, 384		
0.57	0.06		234, 169			199, 202		
0.56		0.2		473	473		473	473
0.5	0.1		234, 131			235, 129		
0.48		0.3		431	427		424	425
0.42	0.2		250, 88			250, 87		
0.37		0.6		360	360		360	360
0.35	0.4		252, 63			254, 64		
0.32	0.5		256, 54			254, 54		
0.31		1.0		352	352		352	352
0.27	0.7		254, 49			255, 49		
0.25		2.0		330	330		330	330
0.23	1.0		267, 31			267, 31		
0.23		3.0		327	327		327	327
0.21		5.0		317	319		318	317
0.196		10.0		311	311		311	311
0.183		50.0		303	303		305	303
0.181	2.0		296, 3			292, 3		
0.181		10^2		303	303		302	300
0.18	3.0		298, 0			295, 0		
0.18		5×10^2		301	298		301	295

Table 4: Wisconsin Breast Cancer data: Number of support, upper bound vectors

M	C	$\tilde{C}$	SMO	SMO_Q	NPA	SOR	MNA	SOR_Q
6.38	0.02		0.172			0.057		
4.47		0.03		0.311	0.104		0.123	0.210
3.19	0.04		0.191			0.057		
2.13	0.06		0.172			0.057		
1.52		0.1		0.299	0.121		0.142	0.474
1.28	0.1		0.172			0.057		
0.86		0.2		0.290	0.134		0.172	0.695
0.64	0.2		0.133			0.095		
0.63		0.3		0.277	0.143		0.199	0.957
0.40		0.6		0.391	0.162		0.260	2.050
0.36	0.4		0.189			0.140		
0.30		1.0		0.429	0.188		0.326	2.198
0.29	0.5		0.246			0.214		
0.23	0.7		0.354			0.572		
0.22		2.0		0.478	0.238		0.515	3.362
0.19		3.0		0.563	0.276		0.614	2.554
0.173	1.0		1.076			1.090		
0.17		5.0		0.687	0.340		0.854	2.465
0.15		10.0		0.948	0.452		1.098	3.644
0.14		50.0		2.083	1.094		2.982	6.729
0.138	2.0		4.994			9.960		
0.138		10^2		2.640	1.300		5.204	6.396
0.1367	3.0		2.824			8.132		
0.1364		5×10^2		3.466	1.540		9.083	8.611
0.1362		10^3		3.311	2.21		11.024	7.662
0.136	10.0		2.711			7.610		

Table 5: Two Spirals Data: Number of Kernel evaluations $\times 10^{-6}$

M	C	$\tilde{C}$	SMO	SMO_Q	NPA	SOR	MNA	SOR_Q
6.38	0.02		2, 193			0, 195		
4.47		0.03		195	195		195	195
3.19	0.04		2, 192			0, 195		
2.13	0.06		2, 193			0, 195		
1.52		0.1		195	195		195	195
1.28	0.1		2, 193			0, 195		
0.86		0.2		195	195		195	195
0.64	0.2		2, 192			2, 192		
0.63		0.3		195	195		195	195
0.40		0.6		195	195		195	195
0.36	0.4		6, 184			5, 185		
0.30		1.0		195	195		195	195
0.29	0.5		16, 174			15, 174		
0.23	0.7		31, 155			30, 156		
0.22		2.0		195	195		195	195
0.19		3.0		194	194		194	194
0.173	1.0		63, 120			61, 122		
0.17		5.0		188	188		188	188
0.15		10.0		185	185		185	185
0.14		50.0		181	181		181	181
0.138	2.0		170, 4			172, 4		
0.138		10^2		181	181		181	181
0.1367	3.0		171, 2			174, 2		
0.1364		5×10^2		178	179		178	178
0.1362		10^3		176	178		176	176
0.136	10.0		172, 0			174, 0		

Table 6: Two Spirals Data: Number of support, upper bound vectors

M	C	$\tilde{C}$	SMO	SMO_Q	NPA	SOR	MNA	SOR_Q
0.180	1		1.310			1.387		
0.104		10		4.224	2.530		6.889	451.747
0.092	5		1.336			3.639		
0.073	10		1.538			4.774		
0.057		50		6.320	3.542		12.668	518.311
0.044		10^2		8.484	5.015		19.443	
0.040	50		2.791			18.251		
0.031	10^2		6.384			34.534		
0.026		5×10^2		17.266	9.804		40.375	
0.021		10^3		20.876	11.684		46.040	
0.018	5×10^2		9.422			62.935		
0.014	10^3		14.016			93.374		
0.010		10^4		64.950	20.117		93.031	
0.008	5×10^3		63.880			274.164		
0.0071	10^4		41.005			294.070		
0.007		10^5		55.279	31.128		165.860	
0.0062		10^6		58.156	24.609		110.790	
0.00615		10^7		46.017	24.373		105.215	
0.00614	5×10^4		44.847			317.870		

Table 7: Checkers Data: Number of Kernel evaluations $\times\ 10^{-6}$

M	C	$\tilde{C}$	SMO	SMO_Q	NPA	SOR	MNA	SOR_Q
0.180	1		21, 295			21, 295		
0.104		10		242	242		242	242
0.092	5		30, 182			30, 182		
0.073	10		31, 142			30, 142		
0.057		50		143	143		143	143
0.044		10^2		131	131		131	
0.040	50		34, 80			34, 80		
0.031	10^2		36, 59			36, 59		
0.026		5×10^2		88	88		88	
0.021		10^3		73	73		73	
0.018	5×10^2		36, 28			37, 28		
0.014	10^3		36, 19			36, 19		
0.010		10^4		50	50		50	
0.008	5×10^3		40, 5			40, 5		
0.0071	10^4		36, 3			36, 3		
0.007		10^5		39	39		39	
0.0062		10^6		38	38		38	
0.00615		10^7		37	37		37	
0.00614	5×10^4		37, 0			37, 0		

Table 8: Checkers Data: Number of support, upper bound vectors

M	C	$\tilde{C}$	SMO	SMO_Q	NPA	SOR	MNA	SOR_Q
1.116		0.03		2.64	0.74		0.97	32.71
0.660		0.1		2.67	0.81		1.27	37.97
0.585	0.1		1.08			1.28		
0.490		0.2		2.49	0.85		1.30	47.92
0.408		0.3		2.23	0.79		1.44	
0.366	0.2		0.92			1.12		
0.290		0.6		1.94	0.89		1.54	
0.265	0.4		0.86			1.14		
0.241	0.5		0.74			1.26		
0.219		1.0		1.90	0.96		1.83	
0.205	0.7		0.83			1.23		
0.168	1.0		0.83			1.65		
0.146		2.0		2.13	1.06		2.51	
0.115		3.0		2.13	1.15		3.20	
0.113	2.0		0.77			4.18		
0.087	3.0		0.88					
0.085		5.0		2.36	1.19		4.34	
0.063	5.0		1.20					
0.059		10.0		3.10	1.41		6.78	
0.042	10.0		1.94					
0.030	20.0		3.66					
0.029		50.0		6.78	2.62		16.01	
0.023		10^2		10.20	3.37		22.43	
0.020	50.0		7.47			32.86		
0.017	10^2		12.01			38.26		
0.015	2×10^2		17.08					
0.015		5×10^2		18.77	5.69		43.80	
0.014		10^3		24.25	6.60		60.66	
0.012	5×10^2		23.91					

Table 9: Adult-1 Data: Number of Kernel evaluations $\times 10^{-7}$

M	C	$\tilde{C}$	SMO	SMO_Q	NPA	SOR	MNA	SOR_Q
1.116		0.03		1602	1602		1602	1602
0.660		0.1		1378	1377		1383	1382
0.585	0.1		55, 764			42, 776		
0.490		0.2		1246	1246		1249	1249
0.408		0.3		1191	1191		1192	
0.366	0.2		49, 734			46, 735		
0.290		0.6		1118	1118		1119	
0.265	0.4		66, 670			65, 672		
0.241	0.5		73, 648			72, 651		
0.219		1.0		1077	1077		1077	
0.205	0.7		85, 625			86, 625		
0.168	1.0		107, 584			105, 586		
0.146		2.0		1034	1032		1033	
0.115		3.0		997	996		996	
0.113	2.0		161, 517			159, 519		
0.087	3.0		211, 467					
0.085		5.0		951	950		951	
0.063	5.0		271, 412					
0.059		10.0		898	898		898	
0.042	10.0		370, 324					
0.030	20.0		475, 219					
0.029		50.0		776	776		776	
0.023		10^2		736	734		735	
0.020	50.0		567, 107			626, 48		
0.017	10^2		615, 53			636, 29		
0.015	2×10^2		636, 39					
0.015		5×10^2		687	687		687	
0.014		10^3		677	677		678	
0.012	5×10^2		637, 24					

Table 10: Adult-1 Data: Number of support, upper bound vectors

M	C	$\tilde{C}$	SMO	SMO_Q	NPA	SOR	MNA	SOR_Q
0.728		0.03		22.51	5.88		9.12	
0.469		0.1		19.70	6.22		10.10	
0.360		0.2		17.19	6.84		10.97	
0.316	0.1		10.86			11.22		
0.301		0.3		18.15	7.04		11.38	
0.254	0.2		8.14			10.53		
0.212		0.6		16.19	8.26		13.36	
0.200	0.4		7.74			9.62		
0.180	0.5		6.77			9.86		
0.158		1.0		17.13	8.96		16.43	
0.150	0.7		6.72			10.83		
0.125	1.0		6.73			13.93		
0.104		2.0		18.21	9.88		23.23	
0.084	2.0		7.17			25.10		
0.081		3.0		18.02	9.87		29.47	
0.065	3.0		7.92			54.31		
0.060		5.0		21.59	11.51		43.56	
0.047	5.0		8.93			127.33		
0.040		10.0		32.32	15.06		69.19	
0.030	10.0		16.93			645.80		
0.020	20.0		30.56					
0.017		50.0		86.13	35.55		205.15	
0.013		10^2		132.45	46.08		311.96	
0.012	50.0		75.05					
0.009	10^2		128.62					
0.0072		5×10^2		365.69	102.33		773.20	
0.007	2×10^2		214.95					
0.006		10^3		544.91	146.89		1171.51	
0.0056	5×10^2		419.61					

Table 11: Adult-4 Data: Number of Kernel evaluations $\times 10^{-7}$

M	C	$\tilde{C}$	SMO	SMO_Q	NPA	SOR	MNA	SOR_Q
0.728		0.03		4111	4111		4121	
0.469		0.1		3570	3569		3578	
0.360		0.2		3385	3383		3385	
0.316	0.1		68, 2136			66, 2141		
0.301		0.3		3306	3305		3307	
0.254	0.2		99,1953			94, 1960		
0.212		0.6		3198	3199		3200	
0.200	0.4		135, 1813			130, 1816		
0.180	0.5		157, 1772			158, 1772		
0.158		1.0		3116	3116		3117	
0.150	0.7		196, 1713			187, 1714		
0.125	1.0		233, 1653			230, 1654		
0.104		2.0		2999	2998		3000	
0.084	2.0		348, 1524			347, 1525		
0.081		3.0		2912	2915		2914	
0.065	3.0		413, 1459			404, 1462		
0.060		5.0		2823	2824		2824	
0.047	5.0		540, 1349			534, 1351		
0.040		10.0		2679	2680		2678	
0.030	10.0		743, 1154			741, 1156		
0.020	20.0		1006, 944					
0.017		50.0		2368	2371		2370	
0.013		10^2		2240	2242		2242	
0.012	50.0		1278, 667					
0.009	10^2		1468, 464					
0.0072		5×10^2		2032	2034		2034	
0.007	2×10^2		1581, 309					
0.006		10^3		1967	1968		1968	
0.0056	5×10^2		1661, 190					

Table 12: Adult-4 Data: Number of support, upper bound vectors

M	C	$\tilde{C}$	SMO	NPA
0.62	0.01		193.72	
0.50		0.03		60.83
0.341		0.1		69.77
0.26		0.2		74.88
0.216		0.3		83.56
0.1491		0.6		96.83
0.1389	0.4		81.10	
0.11		1.0		113.10
0.0863	1.0		71.62	
0.0317	5.0		121.80	
0.025		10.0		205.28
0.0128	20.0		362.83	
0.01		50.0		510.50

Table 13: Adult-7 Data: Number of Kernel evaluations $\times\ 10^{-7}$

M	C	$\tilde{C}$	SMO	NPA
0.62	0.01		37, 7818	
0.50		0.03		11950
0.341		0.1		10975
0.26		0.2		10652
0.216		0.3		10530
0.1491		0.6		10320
0.1389	0.4		355, 5659	
0.11		1.0		10181
0.0863	1.0		559, 5376	
0.0317	5.0		1321, 4658	
0.025		10.0		9343
0.0128	20.0		2438, 3728	
0.01		50.0		8393

Table 14: Adult-7 Data: Number of support, upper bound vectors

M	C	$\tilde{C}$	SMO	SMO_Q	NPA	SOR	MNA	SOR_Q
2.943		0.03		2.72	1.42		2.41	
2.592	0.1		0.86			1.77		
1.296	0.2		1.05			2.30		
1.267		0.1		4.20	1.52		2.54	
0.789		0.2		5.16	1.51		2.52	
0.648	0.4		1.33			3.56		
0.608		0.3		4.65	1.42		2.53	
0.519	0.5		1.21			3.66		
0.403		0.6		4.74	1.45		2.54	
0.399	0.7		1.00			4.89		
0.306		1.0		3.71	1.41		2.89	
0.300	1.0		0.97			6.34		
0.221		2.0		2.07	1.27		3.17	
0.187		3.0		2.12	1.20		2.96	
0.185	2.0		1.28			9.58		
0.155		5.0		2.06	1.13		3.21	
0.145	3.0		1.52			15.98		
0.125		10.0		1.86	1.08		4.29	
0.119	5.0		1.77			16.02		
0.100	10.0		1.47			20.52		
0.088	20.0		1.42			25.38		
0.079		50.0		1.79	1.36		8.46	
0.069	50.0		1.53			33.74		
0.065		10^2		2.31	1.38		14.04	
0.053	10^2		1.38			24.64		
0.047	2×10^2		1.21			31.12		
0.046		5×10^2		3.32	1.89		44.87	
0.042		10^3		3.26	2.61		86.92	
0.040	5×10^2		1.60			24.74		

Table 15: Web-1 Data: Number of Kernel evaluations $\times 10^{-7}$

M	C	$\tilde{C}$	SMO	SMO_Q	NPA	SOR	MNA	SOR_Q
2.943		0.03		2477	2477		2477	
2.592	0.1		284, 92			299, 97		
1.296	0.2		307, 93			303, 93		
1.267		0.1		2329	2332		2341	
0.789		0.2		2002	1998		2021	
0.648	0.4		308, 94			321, 92		
0.608		0.3		1774	1772		1793	
0.519	0.5		319, 94			325, 93		
0.403		0.6		1411	1411		1419	
0.399	0.7		331, 89			339, 91		
0.306		1.0		1190	1187		1192	
0.300	1.0		346, 87			355, 85		
0.221		2.0		995	994		1001	
0.187		3.0		913	917		922	
0.185	2.0		372, 72			377, 72		
0.155		5.0		800	800		811	
0.145	3.0		381, 60			383, 62		
0.125		10.0		707	705		709	
0.119	5.0		374, 46			367, 44		
0.100	10.0		418, 28			394, 34		
0.088	20.0		459, 24			388, 29		
0.079		50.0		596	597		596	
0.069	50.0		430, 18			366, 25		
0.065		10^2		549	550		550	
0.053	10^2		427, 17			346, 18		
0.047	2×10^2		377, 15			341, 15		
0.046		5×10^2		493	492		497	
0.042		10^3		479	479		485	
0.040	5×10^2		397, 11			326, 15		

Table 16: Web-1 Data: Number of support, upper bound vectors

M	C	$\tilde{C}$	SMO	SMO_Q	NPA	SOR	MNA	SOR_Q
1.537		0.03		45.60	13.34		21.84	
1.267	0.1		7.28			12.04		
0.676		0.1		46.44	12.42		24.53	
0.633	0.2		7.79			14.42		
0.440		0.2		36.49	12.37		24.03	
0.347		0.3		33.31	11.78		20.89	
0.333	0.4		7.97			16.27		
0.274	0.5		8.15			19.00		
0.239		0.6		31.06	13.53		18.91	
0.208	0.7		7.94			21.19		
0.186		1.0		19.57	10.07		22.69	
0.161	1.0		9.42			25.91		
0.137		2.0		18.90	11.56		22.36	
0.117		3.0		18.69	9.80		21.91	
0.106	2.0		11.17			45.51		
0.098		5.0		17.02	10.47		23.33	
0.089	3.0		11.79			68.07		
0.079		10.0		14.73	10.05		30.40	
0.075	5.0		9.53			101.09		
0.063	10.0		10.45			149.19		
0.053	20.0		13.06			250.34		
0.052		50.0		19.66	13.43		64.85	
0.044		10^2		21.05	17.48		115.67	
0.043	50.0		11.13			242.75		
0.038	10^2		12.82			233.53		
0.0324	2×10^2		11.96			212.41		
0.0322		5×10^2		28.36	18.20		380.20	
0.0294		10^3		42.50	19.71		640.84	
0.0294	5×10^2		12.41			272.77		

Table 17: Web-4 Data: Number of Kernel evaluations $\times 10^{-7}$

M	C	$\tilde{C}$	SMO	SMO_Q	NPA	SOR	MNA	SOR_Q
1.537		0.03		7010	7012		7050	
1.267	0.1		428, 328			411, 333		
0.676		0.1		5314	5308		5366	
0.633	0.2		463, 325			427, 332		
0.440		0.2		4330	4316		4373	
0.347		0.3		3813	3803		3819	
0.333	0.4		488, 317			462, 326		
0.274	0.5		499, 317			480, 321		
0.239		0.6		3127	3124		3124	
0.208	0.7		523, 308			503, 309		
0.186		1.0		2743	2742		2750	
0.161	1.0		578, 282			542, 289		
0.137		2.0		2330	2321		2344	
0.117		3.0		2140	2142		2145	
0.106	2.0		681, 216			658, 221		
0.098		5.0		1989	1995		1997	
0.089	3.0		829, 178			691, 186		
0.079		10.0		1812	1815		1816	
0.075	5.0		760, 132			760, 137		
0.063	10.0		918, 85			772, 101		
0.053	20.0		971, 66			771, 67		
0.052		50.0		1576	1579		1580	
0.044		10^2		1486	1495		1493	
0.043	50.0		948, 50			737, 66		
0.038	10^2		957, 44			740, 57		
0.0324	2×10^2		893, 42			701, 55		
0.0322		5×10^2		1335	1331		1341	
0.0294		10^3		1318	1303		1324	
0.0294	5×10^2		816, 40			709, 49		

Table 18: Web-4 Data: Number of support, upper bound vectors

M	C	$\tilde{C}$	SMO	NPA
0.7		0.03		150.79
0.428	0.1		169.66	
0.35		0.1		124.85
0.249	0.2		153.46	
0.244		0.2		127.93
0.2		0.3		123.09
0.1545	0.4		129.93	
0.146		0.6		111.32
0.1346	0.5		118.74	
0.1173		1.0		128.19
0.112	0.7		126.14	
0.092	1.0		129.90	
0.0765		3.0		148.47
0.066	2.0		122.05	
0.0576	3.0		105.32	
0.0505		10.0		255.76
0.0488	5.0		99.0	
0.0397	10.0		114.03	
0.0332	20.0		136.40	
0.0313		50.0		194.21
0.0256	50.0		137.40	
0.02124	100.0		136.73	
0.01812	200.0		116.39	
0.01593	500.0		121.36	

Table 19: Web-7 Data: Number of Kernel evaluations $\times 10^{-7}$

M	C	$\tilde{C}$	SMO	NPA
0.7		0.03		17551
0.428	0.1		744, 1286	
0.35		0.1		12192
0.249	0.2		888, 1235	
0.244		0.2		10083
0.2		0.3		9192
0.1545	0.4		1050, 1125	
0.146		0.6		7868
0.1346	0.5		1023, 1100	
0.1173		1.0		7070
0.112	0.7		1100, 1008	
0.092	1.0		1337, 914	
0.0765		3.0		6010
0.066	2.0		1587, 687	
0.0576	3.0		1779, 560	
0.0505		10.0		5167
0.0488	5.0		2132, 437	
0.0397	10.0		2072, 323	
0.0332	20.0		2177, 252	
0.0313		50.0		4469
0.0256	50.0		2177, 210	
0.02124	100.0		2112, 184	
0.01812	200.0		2109, 172	
0.01593	500.0		2061, 164	

Table 20: Web-7 Data: Number of support, upper bound vectors