To appear in: *International Conference on Machine Learning* (ICML-02), 2002

Anytime Interval-Valued Outputs for Kernel Machines: Fast Support Vector Machine Classification via Distance Geometry

Dennis DeCoste DECOSTE@AIG.JPL.NASA.GOV
Jet Propulsion Laboratory / California Institute of Technology, 4800 Oak Grove Drive, Pasadena, CA 91109

Abstract

Classifying M query examples using a support vector machine containing L support vectors traditionally requires exactly $M \cdot L$ kernel computations. We introduce a computational geometry method for which classification cost becomes roughly proportional to each query's difficulty (e.g. distance from the discriminant hyperplane). It produces exactly the same classifications, while typically requiring vastly fewer kernel computations. Related "reduced set" methods (e.g. (Burges, 1996)) similarly lower the effective L, but provide neither proportionality with difficulty nor guaranteed preservation of classifications. Experiments on UCI and NASA data illustrate 2 to 64-fold speedups, across both SVMs and kernel Fisher discriminants.

1. Introduction

Support vector machines (SVMs) and other kernel methods have shown much recent promise (Schölkopf & Smola, 2002). However, wide-spread use remains hindered, largely, by query-time costs often much higher than others, such as decision trees, neural networks, and nearest-neighbors using indexing trees.

1.1 The Problem

Classifying a query example involves L kernel evaluations (e.g. dot products), one with each of the L support vectors (SVs). L is typically a large fraction (e.g. 5% - 50%) of the number (ℓ) of training examples. For example, an SVM has recently achieved the lowest error rates on the MNIST benchmark digit recognition task (DeCoste & Schölkopf, 2002), but classifies much slower than the previous best (a neural network), due to many SVs for each digit recognizer (around 20,000).

Particularly troubling is that traditional SVM classification costs are *identical* for each query example, *even* for "easy" examples that other methods (e.g. decision trees) can classify much faster than harder examples.

This paper introduces a computational geometry method which directly addresses this problem, reducing the number of kernel evaluations needed to classify a query example (by as much as a factor of L under favorable conditions). So that the central idea of our approach is not lost in the mathematical details dominating this paper, we summarize the main idea below.

1.2 The Solution: The Main Idea

An SVM defines a discriminant hyperplane in (kernel) feature space. Any hyperplane can be defined by any two points (call them `P` and `N`) that are: equi-distant from any point on the hyperplane, on opposite sides, and connected by a line orthogonal to the hyperplane. For any query point (`Q`), SVM classification determines `Q`'s side of the hyperplane by checking whether `Q` is closer to `P` (positive) or to `N` (negative). As this paper shows, traditional computation of the SVM output for `Q` is equivalent to finding the exact Euclidean distances from `Q` to `P` (i.e. d_{QP}) and to `N` (d_{QN}), but this is typically expensive (being proportional to number of SVs in input space that define the two points `P` and `N` in kernel feature space). More specifically, we show that the output for `Q` is proportional to d^2_{QN}-d^2_{QP}.

For faster classification, this paper develops an efficient method to compute *bounds* on d^2_{QN}-d^2_{QP}, using only a subset of (k) SVs. This involves $k+3$ points in the kernel feature space: k SVs ($S_1 \dots S_k$), `N`, `P`, and `Q`. In Euclidean geometry, any $k+3$ points can be exactly embedded in $k+2$ dimensions *if* all $(k+3)^2$ distances among those points are known. All distances not involving `Q` (i.e. among all S_i,`P`,`N`) are precomputed once. At query time we compute the distances between `Q` and the k S_i, but we refrain from the expensive computations of d_{QP} and d_{QN}. We carefully set up the embedding procedure such that d_{QP} and d_{QN} being unknown leads to only 3 unknown embedding coordinates, all for `Q` (Q_x,Q_y,Q_z). It turns out that `Q` is further constrained to fall on the surface of a sphere with computable radius R (i.e. $Q_x^2 + Q_y^2 + Q_z^2 = R^2$). By solving variables Q_x, Q_y, and Q_z (in an efficient closed form) for the extrema of the equation for d_{QN}^2-d_{QP}^2, we find min and max bounds on the SVM output.

1.3 Incremental Anytime Classification

We achieve efficient *anytime* classification by computing one new distance (d_{QS_k}) at each step k and incrementally tightening the bounds on d_{QN}^2-d_{QP}^2 from step k–1 accordingly. From each new d_{QS_k} we compute one new embedding coordinate (Q_w), as the embedding space grows from $k+1$ to $k+2$ dimensions between steps $k-1$ and k. Our bounds monotonically converge toward the exact SVM output as steps proceed, since the uncertainty (i.e. R^2) shrinks by Q_w^2 at each step. The embeddings of the S_i, P, and N are all precomputed and all Q's coordinates except Q_x, Q_y, and Q_z persist to the next step. For each new step we re-solve for Q_x,Q_y,Q_z to find (tighter) extrema of d_{QN}^2-d_{QP}^2.

Once the bounding interval is clearly on one side of zero, we know the classification ("positive" or "negative") for Q that the original SVM would produce. Geometrically, the 3D sphere of uncertainty in Q's location is initially (at step $k = 1$) large and cross-sected by the SVM's discriminant hyperplane, and then monotonically shrinks (and its center drifts with respect to the hyperplane's embedding) as k increases, until the sphere is completely on one side.

By ordering the sequence $S_1, S_2, \ldots$ by "informativeness" (using a form of eigenvector analysis), the k needed to classify each Q becomes roughly proportional to inherent difficulty (e.g. distance to the hyperplane).

1.4 Roadmap

The rest of this paper essentially works out the details of the above idea, introducing four key concepts: why we focus on d_{QN}^2-d_{QP}^2 (Section 2), how we embed from distances (Section 3), how we efficiently bound d_{QN}^2-d_{QP}^2 (Section 4), and how we generate good sequences (Section 5). We then discuss related work (Section 6), explore empirical comparisons (Section 7), and summarize significance and future work (Section 8).

2. SVM Binary Classifier

This section summaries essential SVM terminology and then shows that SVM outputs are proportional to d_{QN}^2-d_{QP}^2, which is fundamental to our approach.

Given an ℓ-by-d data matrix (X), an ℓ-by-1 target labels vector (y), a kernel function (K), and a regularization scalar parameter (C), training a binary SVM classifier traditionally consists of the following Quadratic Programming (QP) dual formulation:

$$\begin{array}{ll} \textit{minimize:} & \frac{1}{2}\sum_{i,j=1}^{\ell} \alpha_i\alpha_j y_i y_j K(x_i,x_j) - \sum_{i=1}^{\ell}\alpha_i \\ \textit{subject to:} & 0 \le \alpha_i \le C,\ \sum_{i=1}^{\ell}\alpha_i y_i = 0, \end{array}$$

where ℓ is the number of training examples and y_i is the label (+1 for positive example, -1 for negative) for the i-th training example (x_i).

The kernel implicitly projects any two examples from their d-dimensional vectors in input space (x_i, x_j) into (possibly infinite) feature space vectors ($\phi(x_i)$, $\phi(x_j)$) and returns their dot product in that feature space:

$$K_{ij} \equiv K(x_i,x_j) \equiv \phi(x_i)\cdot\phi(x_j). \quad (1)$$

By not explicitly computing the coordinates of the projected vectors, kernels provide large non-linear feature spaces while avoiding the curse of dimensionality.

Popular kernels (with parameters a, p, σ) are: [1]

$$\begin{array}{ll} \text{linear:} & K(u,v) = u\cdot v \equiv \sum_{i=1}^{d} u_i v_i, \\ \text{polynomial:} & K(u,v) = (u\cdot v + a)^p, \\ \text{RBF:} & K(u,v) = \exp(\frac{-||u-v||^2}{2\sigma^2}), \\ \text{normalized:} & K(u,v) := K(u,v)K(u,u)^{-\frac{1}{2}}K(v,v)^{-\frac{1}{2}}. \end{array}$$

2.1 SVM Outputs: Standard Formulation

The SVM classification, $F(x)$, given any example x and the vector α (of length ℓ) determined by the above optimization, is traditionally computed as:

$$F(x) = \text{sign}(G(x)), \quad G(x) = \sum_{\alpha_i \ne 0} \alpha_i y_i K(x,x_i) - b. \quad (2)$$

Let SV^+ (SV^-) denote the set of positive (negative) *support vector* examples (for which $0 < \alpha_i \le C$). Similarly, define corresponding "in-bounds" subsets IN^+ and IN^-, for which $0 < \alpha_i < C$. [2]

2.2 Kernel Distances

The (Euclidean) distance between examples x_i and x_j in the feature space of the kernel is, via (1):

$$d_{ij} \equiv \sqrt{||\phi(x_i)-\phi(x_j)||^2} = \sqrt{K_{ii} - 2K_{ij} + K_{jj}}. \quad (3)$$

For any two kernel points U,V defined from sets (U,V) of input space (d-dim) vectors and weight vectors β,β':

$$\mathtt{U} = \sum_{u_i\in U}\beta_i\phi(u_i), \quad \mathtt{V} = \sum_{v_i\in V}\beta_i'\phi(v_i), \quad d_{\mathtt{UV}}^2 = ||\mathtt{U}-\mathtt{V}||^2 \quad (4)$$

defines kernel distance, computable from kernels as:

$$d_{\mathtt{UV}}^2 = \sum_{i,j}\beta_i\beta_j K(u_i,u_j) - 2\sum_{i,j}\beta_i\beta_j' K(u_i,v_j) + \sum_{i,j}\beta_i'\beta_j' K(v_i,v_j) \quad (5)$$

[1] Where 2-norm is defined as $||u-v||^2 \equiv u\cdot u - 2u\cdot v + v\cdot v$.

[2] Bias b is chosen midway between values of G (without the -b term) over $x_i \in IN^+$ and values of G over $x_i \in IN^-$.

2.3 SVM Outputs: Via Kernel Distances

An SVM defines a linear discriminant hyperplane in kernel feature space. Any hyperplane can be defined as all points equi-distant from two points such that the hyperplane is orthogonal to the line connecting those two points. Two special points defining the SVM hyperplane, in terms of α and SV from training, are:

$$\mathtt{P} \equiv \sum_{x_i \in SV^+} \alpha_i^+ \phi(x_i) \ , \quad \mathtt{N} \equiv \sum_{x_i \in SV^-} \alpha_i^- \phi(x_i). \qquad (6)$$

In practice, normalizing α^+ and α^- to each sum to 1 is useful. [3] Using the SVM constraint $\sum_{i=1}^{\ell} \alpha_i y_i = 0$ (i.e. $\sum_i \alpha_i^+ = \sum_j \alpha_j^-$) , normalization is simply:

$$s = \sum_i \alpha_i^+, \qquad \alpha_i^+ := \frac{1}{s}\alpha_i^+, \qquad \alpha_i^- := \frac{1}{s}\alpha_i^-. \qquad (7)$$

The SVM classification output $F(x)$ can be computed via the following alternative (to (2)) definition of $G(x)$:

$$G(x) = \frac{1}{2} \left[\ g(x) \ s \quad - \quad \frac{c}{s} \ \right] \ - \ b, \qquad (8)$$

$$\boxed{g(x) = d^2_{\mathtt{QN}} - d^2_{\mathtt{QP}}.} \qquad (9)$$

where $\mathtt{Q} = \phi(x)$ is the point in kernel space to which given query example x (implicitly) projects,

$$c = \sum_{x_i, x_j \in SV^-} \alpha_i \alpha_j K(x_i, x_j) - \sum_{x_i, x_j \in SV^+} \alpha_i \alpha_j K(x_i, x_j) \quad (10)$$

is a constant (precomputed *before* normalizing alphas), and s is the normalization factor from (7).

To verify (8), compare restatements of (2):

$$G(x) = \sum_{x_i \in SV^+} \alpha_i K(x, x_i) \ - \sum_{x_i \in SV^-} \alpha_i K(x, x_i) \ - b \quad (11)$$

and of (9) (via (5)):
$g(x) = K(x,x) - 2\sum_{x_i \in SV^-} \alpha_i^- K(x, x_i) + \sum_{x_i, x_j \in SV^-} \alpha_i^- \alpha_j^- K(x_i, x_j) - [K(x,x) - 2\sum_{x_i \in SV^+} \alpha_i^+ K(x, x_i) + \sum_{x_i, x_j \in SV^+} \alpha_i^+ \alpha_j^+ K(x_i, x_j)]$.

3. Embedding from Distances

This section simply states our straight-forward general embedding procedure. The next section uses this to embed P, N and other points from high-dimensional kernel space, in which kernel distances are computable (from (5)), into low-dimensional subspaces, in order to efficiently infer sufficient bounds on d^2_{QN}-d^2_{QP}. We selected these specific equations to support the most efficient incremental embedding as dimension grows.

Given all k^2 distances $(d_{i,j})$ among any k points $(X_1 \ldots X_k)$, one can always embed them into $k-1$ dimensions. The next section exploits the upper-triangular form of our embedding matrix V (Table 1).

Table 1. Embedding matrix V.

	X_1	X_2	X_3	X_4	X_5	$\cdots$	X_k
1	0	$V_{2,1}$	$V_{3,1}$	$V_{4,1}$	$V_{5,1}$	$\cdots$	$V_{k,1}$
2	0	0	$V_{3,2}$	$V_{4,2}$	$V_{5,2}$	$\cdots$	$V_{k,2}$
3	0	0	0	$V_{4,3}$	$V_{5,3}$	$\cdots$	$V_{k,3}$
4	0	0	0	0	$V_{5,4}$	$\cdots$	$V_{k,4}$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
$k-1$	0	0	0	0	0	$\cdots$	$V_{k,k-1}$

Using distance equations of the form:

$$d^2_{ij} = (V_{i,1} - V_{j,1})^2 + \cdots + (V_{i,k-1} - V_{j,k-1})^2, \quad (12)$$

we compute V via the equations of Table 2. [4]

Table 2. Embedding equations.

$$V_{2,1} = \sqrt{d^2_{2,1}}$$

$$V_{3,1} = \frac{1}{2V_{2,1}}[V^2_{2,1} + d^2_{3,1} - d^2_{3,2}], \ V_{3,2} = \sqrt{d^2_{3,1} - V^2_{3,1}}$$

$$\begin{aligned} V_{4,1} &= \frac{1}{2V_{2,1}}[V^2_{2,1} + d^2_{4,1} - d^2_{4,2}] \\ V_{4,2} &= \frac{1}{2V_{3,2}}[V^2_{3,2} + d^2_{4,2} - d^2_{4,3} + \\ &\qquad (V_{4,1} - V^2_{3,1}) - (V_{4,1} - V_{2,1})^2 \ ] \\ V_{4,3} &= \sqrt{d^2_{4,1} - V^2_{4,1} - V^2_{4,2}} \end{aligned}$$

$\cdots$

$$\begin{aligned} V_{i,j} &= 0 && \text{if } j \geq i \\ V_{i,j} &= \sqrt{d^2_{i,1} - V^2_{i,1} - V^2_{i,2} - \ldots - V_{i,j-1}} && \text{if } j = i-1 \\ V_{i,j} &= \frac{1}{2V_{j+1,j}} \ [\ V^2_{j+1,j} \ + \ d^2_{i,j} \ - \ d^2_{i,j+1} \ + \\ &\quad (V_{i,1} - V_{i-1,1})^2 \ - \ (V_{i,1} - V_{i-2,1})^2 \ + \\ &\quad (V_{i,2} - V_{i-1,2})^2 \ - \ (V_{i,2} - V_{i-2,2})^2 \ + \ \ldots \ + \\ &\quad (V_{i,j-1} - V_{i-1,j-1})^2 \ - \ (V_{i,j-1} - V_{i-2,j-1})^2 \ ] \end{aligned}$$

4. Computing Bounded Outputs

This section focuses on the heart of our approach: bounding $g(x)$. Given bounds $g_L(x) \leq g(x) \leq g_H(x)$ for query x, we bound $G(x)$ and $F(x)$ (via (2),(8)):

$$\boxed{F_L = \text{sign}(G_L(x)), \ G_L(x) = \tfrac{1}{2}[g_L(x)s - \tfrac{c}{s}] - b} \quad (13)$$

$$\boxed{F_H = \text{sign}(G_H(x)), G_H(x) = \tfrac{1}{2}[g_H(x)s - \tfrac{c}{s}] - b} \quad (14)$$

where $G_L(x) \leq G(x) \leq G_H(x)$. When $F_L = F_H$, we know the SVM's classification of x (i.e. "+" or "-").

Bounding $g(x)$ involves three stages:

[3]This ensures that neither P nor N are outside the convex hull of the examples of their respective classes, helps keep distances smaller, and improves numeric stability. After normalization, P and N are the closest points from the two convex hulls (Bennett & Bredensteiner, 2000).

[4]Standard Cholesky factorization $(K = G'G)$ of kernel matrix $K_{ij} = \phi(X_i) \cdot \phi(X_j)$ gives an alternative (k-by-k) embedding (G). However, Table 2's embedding somewhat simplifies the computations of Section 4 and also makes incremental embedding (used in Section 4.4) explicit.

Table 3. $k+3$ point embedding matrix at step k.

	S_1	$\cdots$	$S_{k-1} \equiv A$	$S_k \equiv B$	N	P	Q
1	0	$\cdots$	A_1	B_1	N_1	P_1	Q_1
$\vdots$	$\vdots$	$\ddots$	$\cdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$v \equiv k-2$	0	$\cdots$	A_v	B_v	N_v	P_v	Q_v
$w \equiv k-1$	0	$\cdots$	0	B_w	N_w	P_w	Q_w
$x \equiv k$	0	$\cdots$	0	0	N_x	P_x	$Q_x = ?$
$y \equiv k+1$	0	$\cdots$	0	0	0	P_y	$Q_y = ?$
$z \equiv k+2$	0	$\cdots$	0	0	0	0	$Q_z = ?$

4.1 Stage 1: embed k SVs, `N`, `P` (pre-query)

Given k SVs $(S_1, \ldots, S_k)$, `N`, and `P`, we precompute (once, before any queries) all distances between those $k+2$ points (using (5)) and their embedding into $k+1$ dimensions (using the procedure of Section 3).

We discuss how the k SVs are selected in Section 5.

4.2 Stage 2: embed x into `Q` (query-time)

Similarly, at query-time we embed x by computing (via (3)) the k distances between x and each of the SVs $(S_1, \ldots, S_k)$. We reapply the procedure of Section 3, with the embedding dimension now extended by 1 (to account for the one new point `Q`). All S_i,`N`, and `P` points retain their old embeddings (with zeros for their one new (k+2)-th coordinate).

4.3 Stage 3: bound $g(x)$ (closed-form optimize)

Table 3 illustrates the embedding matrix resulting from stages 1 and 2. As shown, we introduce labels "w,x,y,z" for the final four coordinates $(k-1, \ldots, k+2)$.

Three coordinates for `Q` (Q_x,Q_y,Q_x) are unknown, because their embedding equations rely on the two expensive distances (d_{QN} and d_{QP}) that we wish to avoid computing. However, they are constrained by the embedding equation for case "j=i-1" for $V_{i,j}$ in Table 2:

$$Q_z = \sqrt{d^2_{QS_1} - Q_1^2 - \ldots - Q_w^2 - Q_x^2 - Q_y^2}.$$

Reformulating:

$$Q_x^2 + Q_y^2 + Q_z^2 = R^2, \tag{15}$$

$$\boxed{R^2 = d^2_{QS_1} - Q_1^2 - \ldots - Q_w^2.} \tag{16}$$

This means that `Q`'s location uncertainty is restricted to a 3D sphere of radius R. The min and max bounds of $g(x)$ occur at the two poles of this sphere, with the polar axis being the sphere's diameter line which is orthogonal to the SVM's hyperplane in our embedding.

To bound $g(x)$, we first rewrite $g(x)$ in terms of Q_x,Q_y,Q_z via the definition of Euclidean distances:

$$g(x) = d^2_{QN} - d^2_{QP} \quad \equiv \quad g^- - g^+ \quad \equiv \quad g, \tag{17}$$

$$g^- = (Q_1 - N_1)^2 + \cdots + (Q_w - N_w)^2 + (Q_x - N_x)^2 + Q_y^2 + Q_z^2 \tag{18}$$

$$g^+ = (Q_1 - P_1)^2 + \cdots + (Q_w - P_w)^2 + (Q_x - P_x)^2 + (Q_y - P_y)^2 + Q_z^2 \tag{19}$$

To find bounds, we simply need to find the values (Q_{x*},Q_{y*},Q_{z*}) which give the extrema of g. We do this by setting the derivative of g with respect to each variable to zero, as follows.

Since only Q_x,Q_y, and Q_z can vary, all derivatives of g have the following form:

$$\delta g^- = \delta \ [(Q_x^2 - 2N_x Q_x + Q_y^2 + Q_z^2] \tag{20}$$

$$\delta g^+ = \delta \ [(Q_x^2 - 2P_x Q_x + Q_y^2 - 2P_y Q_y + Q_z^2] \tag{21}$$

$$\delta g = \delta g^- - \delta g^+ = \delta \ [2P_y \ Q_y - 2(N_x - P_x) \ Q_x] \tag{22}$$

Plug $Q_y = \text{sign}(Q_y)\sqrt{R^2 - Q_z^2 - Q_x^2}$ (from (15)) into (22):

$$\frac{\delta g}{\delta Q_x} = \frac{\delta}{\delta Q_x}[2P_y \ \text{sign}(Q_y)(R^2 - Q_z^2 - Q_x^2)^{\frac{1}{2}} - 2(N_x - P_x)Q_x] \tag{23}$$

For $\frac{\delta g}{\delta Q_x} = 0$, (23) gives:

$$N_x - P_x = -P_y \ \text{sign}(Q_y) \ (R^2 - Q_z^2 - Q_x^2)^{-\frac{1}{2}} \ Q_x \tag{24}$$

Squaring both sides of (24) and solving for Q_x yields:

$$Q_{x*} = \pm \sqrt{\frac{(N_x - P_x)^2 \ (R^2 - Q_{z*}^2)}{(N_x - P_x)^2 \ + \ P_y^2}} \tag{25}$$

With Q_{x*} fixed, similarly reuse (22),(15) to find Q_{z*}:

$$\frac{\delta g}{\delta Q_z} = \frac{\delta}{\delta Q_z}[2P_y \ \text{sign}(Q_y)(R^2 - Q_z^2 - Q_x^2)^{\frac{1}{2}} - 2(N_x - P_x)Q_x] \tag{26}$$

$$\frac{\delta g}{\delta Q_z} = 0 = -2P_y \ \text{sign}(Q_y) \ (R^2 - Q_z^2 - Q_x^2)^{-\frac{1}{2}} \ Q_z \tag{27}$$

Clearly, (27) holds exactly when $Q_{z*} = 0$. [5] Combining with (25) and (15) yields:

$$\boxed{Q_{x*} = \pm \frac{R(N_x - P_x)}{\sqrt{(N_x - P_x)^2 + P_y^2}}, \ Q_{y*} = \pm\sqrt{R^2 - Q_{x*}^2}, \ Q_{z*} = 0.} \tag{28}$$

Finally, plugging the four combinations of the signs of Q_{x*} and Q_{y*} into the equation for $g(x)$ (i.e. (17)–(19)) tells us the min ($g_L(x)$) and max ($g_H(x)$) bounds.

[5] Which is not surprising, since Q_z^2 terms cancel in (17)–(19); extrema occur when the quantity of R is distributed (via (15)) among all terms involving only Q_x and Q_y.

4.4 Incremental Refinement of Bounds

To provide efficient anytime bounding, we proceed in steps ($k = 1, \ldots, L$), incrementally updating or precomputing quantities (if not involving Q) whenever possible. At each step k, we compute one new distance (d_{QS_k}, using (3)). All coordinates for S_i, P, N for step k are precomputed and we need only update a single coordinate ($Q_w \equiv V_{\mathtt{Q},k-1}$) when going to step k, via: $Q_w = \frac{1}{2B_w} [\ B_w^2 + d_{QA}^2 - d_{QB}^2 + (Q_1 - B_1)^2 - (Q_1 - A_1)^2 + \ldots + (Q_v - B_v)^2 - (Q_v - A_v)^2\ ]$.

We incrementally update R^2 (with $R_{(1)}^2 = d_{QS_1}^2$) via:

$$R_{(k+1)}^2 = R_{(k)}^2 - Q_w^2, \tag{29}$$

incrementally update g (with $g_{w(0)} = 0$) using:

$$g_{w(k)} = g_{w(k-1)} + (Q_w - N_w)^2 - (Q_w - P_w)^2, \tag{30}$$

and compute new extrema via:

$$g_*(x) = g_{w(k)} + (Q_{x*} - N_x)^2 - (Q_{x*} - P_x)^2 + Q_{y*}^2 - (Q_{y*} - P_y)^2 \tag{31}$$

4.5 Complexity Analysis

Despite a fair amount of mathematical detail, the computation required per query-time step is typically only a little more than a kernel computation. [6]

The only costs which are not $O(1)$ per step k are: computing d_{QS_k} ($O(d)$) and computing Q_w ($O(k-2)$).

After k steps, the cumulative cost (per query) would be $O(d \cdot k + \frac{1}{2}k^2)$, whereas traditional SVM ouputs cost $O(d \cdot L)$. Thus, if step $k > \min(L, \sqrt{d \cdot L})$ occurs for a query before $F_L = F_H$, it becomes advisable to drop out and classify that query with the full SVM. Fortunately, empirical evidence (e.g. Section 7) suggests that for many queries and data sets this is seldom required. Limiting steps to that threshold also keeps precomputed space cost below $O(d \cdot L)$, the size of the SVM. This complexity is particularly well-suited for large $L > d$, as is common in large SVM applications.

4.6 Probabilities

Our computational geometry approach also provides a promising basis for obtaining the probability p^+ (p^-) at any step k that the full SVM will classify Q as positive (negative). *Assuming* that Q is equally likely to be anywhere on the surface area of the sphere that our method restricts it to, p^+ (p^-) is essentially: the surface area of the "spherical cap" which is on the P (N) side of the discriminant hyperplane, divided by the total surface area – which is known to be a linear relation (Weisstein, 1998). Specifically:

$$p^+ = \frac{\max(G_H, 0)}{G_H + \max(-G_L, 0)}, p^- = \frac{\max(-G_L, 0)}{\max(G_H, 0) - G_L}. \tag{32}$$

Section 7 gives evidence that (32) can give useful approximations, seemingly much better than random guessing whenever $F_L = F_H$ is not yet true. However, further research is required to understand exactly under what conditions the assumption of queries projecting uniformly over the sphere surface is actually reasonable. We suspect that the promising performance of (32) in our experiments to date arises from sequence S_i orderings (i.e. as in Section 5) putting first those SVs capturing the most variance in the kernel matrix.

5. Finding Good Sequences (S_i)

The main focus of this paper is on introducing our efficient $G(x)$ bounding mechanism. However, demonstrating its effectiveness requires some means for finding useful S_i sequences. We have found that Sparse Greedy Matrix Approximation (SGMA) (Smola & Schölkopf, 2000) suffices for this purpose. [7]

SGMA provides a form of column (basis function) selection, via greedy eigenvector analysis. Given L d-dimensional vectors (e.g. SVs in our case) and a desired subset size k, SGMA efficiently determines which k of those L candidate columns gives a partial L-by-k kernel matrix which best approximates the full L-by-L matrix. In our application, this results in an (approximately) ordered sequence of the k most "informative" SVs (roughly orthogonal in kernel feature space).

Since SGMA does not consider the distribution of the future query set, nor our true cost function (i.e. the number of steps k required to achieve $F_L = F_H$), developing even better S_i sequence generators is likely to yield even better classification speedups, and thus is a very worthy future research direction.

6. Related Work

Our approach is reminiscent of recent distance geometry methods to find molecular structures from atomic distances (e.g. (Yoon et al., 2000)). However, they are designed for 3D embeddings and distribute uncertainty among all points. In contrast, we optimize for our special interest in quantity d_{QN}^2-d_{QP}^2 per se.

[6] To minimize overhead per step, we employ various coding tricks, including computing and checking bounds only every few (e.g. 10) steps. Good task-specific skip sizes can be pre-determined by trials over training examples.

[7] Specifically, we use "Algorithm 10.1" in (Schölkopf & Smola, 2002), including their trick to limit to 59 candidates at each greedy step (with high probability of near-optimal results), thus only a constant slower than random ordering.

Our method also shares similarities with fast multi-dimensional scaling (MDS) (Faloutsos & Lin, 1995), which finds low-dimensional embeddings roughly faithful to known distances among all points. However, all *our* embeddings are *exactly* faithful to the given distances, *except* for `Q`'s coordinates Q_x, Q_y, and Q_z. Our analog to MDS's "stress" puts *all* approximation error in the single quantity d_{QN}^2-d_{QP}^2 and finds its bounding *extrema*, whereas MDS would tend to find its *mean*.

6.1 Special Cases Subsumed by Our Approach

For the very special case of a linear kernel, it is well-known that classification requires a single dot-product with a single d-dimensional vector w:

$$G(x) = w \cdot x - b, \tag{33}$$

which can be precomputed from all the SVs as follows:

$$w = w^+ - w^-, \quad w^+ = \sum_{x_i \in SV^+} \alpha_i x_i, \quad w^- = \sum_{x_i \in SV^-} \alpha_i x_i. \tag{34}$$

As shown in Section 7, our approach exploits such "weight folding", if w^+ and w^- are explicitly included in the S_i sequence. The key advantage is that a non-linear SVM which happens to give *almost*-linear discriminants in input space can put these weight vectors early in its S_i sequence, often classifying nearly as quickly as a linear SVM that uses (33). [8]

As Section 7 also shows, the speedups due to *exact simplification* methods (Downs et al., 2001), in which the L-by-L kernel matrix has rank less than L, essentially falls out as a special case as well, due to our use of SGMA SV ordering.

6.2 Reduced Set Methods

High classification costs arise from large sets SV^+ and SV^-. Using two points ($\widehat{\mathtt{P}}$,$\widehat{\mathtt{N}}$) approximating `P`,`N` with much smaller sets of d-dimensional vectors can lower cost proportionally. Let (with $\mathtt{S} \equiv \mathtt{P} - \mathtt{N}$ and $\widehat{\mathtt{S}} \equiv \widehat{\mathtt{P}} - \widehat{\mathtt{N}}$):

$$\widehat{\mathtt{P}} \equiv \sum_{z_i \in Z^+} \beta_i^+ \phi(z_i), \quad \widehat{\mathtt{N}} \equiv \sum_{z_i \in Z^-} \beta_i^- \phi(z_i), \quad \widehat{\mathtt{S}} \equiv \sum_{z_i \in Z} \beta_i \phi(z_i). \tag{35}$$

Whereas we find bounds $G_L(x) \le G(x) \le G_H(x)$, *reduced set* methods (e.g. (Burges, 1996; Schölkopf et al., 1999; Romdhani et al., 2001)) employ approximations to give *estimates*:

$$G(x) \approx \widetilde{G}_{\widehat{\mathtt{P}}\widehat{\mathtt{N}}}(x) = \sum_{z_i \in Z^+} \beta_i^+ K(x, z_i) - \sum_{z_i \in Z^-} \beta_i^- K(x, z_i) - b \tag{36}$$

$$G(x) \approx \widetilde{G}_{\widehat{\mathtt{S}}}(x) = \sum_{z_i \in Z} \beta_i K(x, z_i) - b \tag{37}$$

Reduced set methods essentially find β^+, Z^+ minimizing $d_{\mathtt{P}\widehat{\mathtt{P}}}^2 = ||\mathtt{P} - \widehat{\mathtt{P}}||^2$ β^-, Z^- minimizing $d_{\mathtt{N}\widehat{\mathtt{N}}}^2 = ||\mathtt{N} - \widehat{\mathtt{N}}||^2$, or β, Z minimizing $d_{\mathtt{S}\widehat{\mathtt{S}}}^2 = ||\mathtt{S} - \widehat{\mathtt{S}}||^2$, via costly nonlinear global optimization. Given fixed Z, (Schölkopf et al., 1999) shows that direct inversion optimizes β:

$$\beta \;=\; (K^z)^{-1} \;\; K^{zx} \;\; \alpha, \tag{38}$$

where K^z is a matrix with elements $K_{ij}^z = \phi(z_i) \cdot \phi(z_j)$ and K^{zx} has $K_{ij}^{zx} = \phi(z_i) \cdot \phi(x_j)$, $\forall z_i \in Z, \forall x_j \in SV$.

Although never explored in previous reduced set work, reduced set output estimations $\widetilde{G}$(x) could be naively bounded as well, using $g_L(x) = (d_{\mathtt{Q}\widehat{\mathtt{N}}} - d_{\mathtt{N}\widehat{\mathtt{N}}})^2 - (d_{\mathtt{Q}\widehat{\mathtt{P}}} + d_{\mathtt{P}\widehat{\mathtt{P}}})^2$ and $g_H(x) = (d_{\mathtt{Q}\widehat{\mathtt{N}}} + d_{\mathtt{N}\widehat{\mathtt{N}}})^2 - (d_{\mathtt{Q}\widehat{\mathtt{P}}} - d_{\mathtt{P}\widehat{\mathtt{P}}})^2$, based on triangular inequality constraints among `N`,$\widehat{\mathtt{N}}$,`Q` and among `P`,$\widehat{\mathtt{P}}$,`Q`, to infer bounds $G_L(x)$ and $G_H(x)$ using our equations (13) and (14). However, very small approximation errors (e.g. $d_{\mathtt{P}\widehat{\mathtt{P}}}$ and $d_{\mathtt{N}\widehat{\mathtt{N}}}$) would be necessary to get $F_L(x) = F_H(x)$.

Alternatively, we could embed just those two approximation points (e.g. $S_1 = \widehat{\mathtt{N}}$, $S_2 = \widehat{\mathtt{P}}$, k=2), giving a simple five-point ($\widehat{\mathtt{N}}$,$\widehat{\mathtt{P}}$,`N`,`P`,`Q`) special-case embedding. In that case, our bounding method would achieve the best possible bounds on $G(x)$, regardless of approximation errors per se. It would take advantage of any favorable geometric arrangement among the five points, not depending on how far they are from each other per se (e.g. as naive triangular inequalities bounds would).

However, we have found that using many simple embedding points S_i (each with $\beta_i = 1$) performs better than using fewer complex embedding points such as $\widehat{\mathtt{P}}$ and $\widehat{\mathtt{N}}$ (i.e. each defined in terms of many vectors z_i and β_i's) Nevertheless, the idea of at least some of the S_i in our embedding point sequence being defined in terms of multiple z_i vectors (and/or non-uniform β_i's) seems worthy of future research.

7. Experiments

We checked our approach on two UCI datasets (Blake & Merz, 1998): `Sonar` (to test relatively high d (high kernel costs)) and `Haberman` (to contrast with related experiments (Downs et al., 2001)). We also report MISR cloud classification results motivating us. [9] We confirmed $G_L(x) \le G(x) \le G_H(x)$ always held. [10]

[8] Specifically, for a non-linear SVM, we first train a linear SVM and use its folded w^+ and w^- as the first points (S_1,S_2) in the sequence for bounding non-linear SVMs.

[9] See `http://www-misr.jpl.nasa.gov/`.

[10] All SVMs trained with SMO (Platt, 1999). Bounded and exact classification used the same efficient (MATLAB) matrix operations on the same Sun 450Mhz Ultra60.

Table 4. Results summary.

		Sonar	Haberman	MISR
1	d	60	3	222
2	ℓ^+, ℓ^-	97, 111	81, 225	1000, 1000
3	kernel (norm)	$(u\cdot v{+}1)^2$	$(u\cdot v{+}1)^3$	RBF σ=1
4	C	1	1000	0.1
5	L SVs $(\alpha = C)$	165 (153)	180 (150)	513 (434)
6	$\|SV^+\|, \|SV^-\|$	81, 84	79, 101	268, 245
	$F_L = F_H$ SVM classification:			
	train data: (ℓ)	208	306	2000
21	k min,max	3, 111	2, 12	3, 302
22	k mean,median	26.2, 19.5	4.5, 4	109.2, 69.5
	w/o w^+,w^- :			
23	k min,max	1, 110	1, 10	1, 184
24	k mean,median	28.1, 23	4.5, 5	134.9, 133
	test data: (M)	43 x10000	126 x10000	10000
31	k min,max	4, 21	2, 8	8, 302
32	k mean,median	**11.2**, 10	**4.2**, 5	**123.9**, 78
	w/o w^+,w^- :			
33	k min,max	4, 24	1, 6	55, 208
34	k mean,median	16.7, 16	4.3, 5	142.8, 141
	$F_L = F_H$ (w^+, w^-) vs exact SVM on test data:			
41	exact SVM, secs	45.7	89.6	8.7
	$F_L = F_H$, secs	5.7	4.7	4.0
42	time speedup	8.0	18.9	2.2
43	L/k mean speedup	**14.7**	**42.9**	**4.1**
	KFD classification (on train data, with w^+, w^-):			
51	$\|\alpha^+\|, \|\alpha^-\|$	97, 111	149, 157	920, 1080
52	k min,max	3, 187	1, 25	3, 302
53	k mean,median	88.6, 93	4.8, 4	279.6, 302
54	ℓ/k mean speedup	**2.3**	**63.8**	**7.2**

Table 4 summarizes some of our results. Rows labelled 21-22 indicate the min, max, and average steps k required for $F_L = F_H$ using all training data as queries. Rows 23-24 report the same for when w^+,w^- (recall Equation 34) are not used as S_1,S_2. Rows 31-34 similarly report both cases for a second (test) dataset. For the small Sonar and Haberman, this second set is the non-SV training examples, demonstrating that examples farther from the discriminant hyperplane often require much smaller k. Our Haberman result betters (Downs et al., 2001)'s, whose exactly-simplified SVM needed 18 SVs: our mean steps needed is 4.5 (row 22).

Rows 41-43 report speedups of our bounding approach over exact SVM computations for the test sets, both in terms of time ratios and of average k versus numbers of SVs (L), confirming that our current implementation overhead per step is a small constant (about 2). [11]

Rows 51-54 indicate that our bounds also speedup kernel Fisher discriminant (KFD) classifiers. [12]

[11] We duplicated the small UCI test sets by a factor of 10,000 to get stable time measurements.

[12] We trained as in (Mika et al., 2001), but interpreted all resulting negative (positive) α's as defining our N (P) points (for which all α_i are positive). Speeding up KFDs is especially useful, since they seldom have any zero α's.

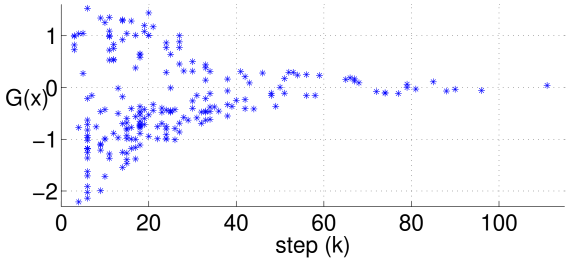

Figure 1. $G(x)$ vs min k giving $F_L(x) = F_H(x)$ (Sonar).

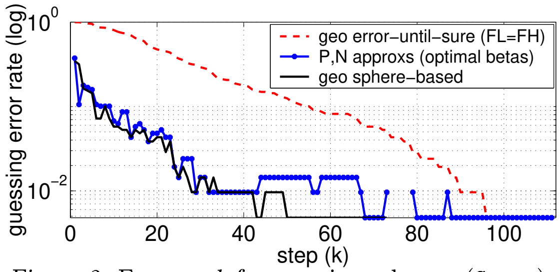

Figure 2. Error vs k for guessing schemes (Sonar).

Figure 1 demonstrates "proportionality to difficulty" for Sonar (rows 21-22). This scatterplot shows that query x with SVM output $G(x)$ far from zero requires less work. For training data, $-1 \le G(x) \le 1$ iff x is a SV; so, max(k)=21 for non-SVs (row 31) can be seen from this scatterplot. The proportion of queries for which $F_L{=}F_H$ by any step k can also be roughly seen.

Figure 2 contrasts alternative ways to *guess* classifications, earlier than our bounds strictly warrant. "Error" here means with respect to classifications $F(x)$, *not* training targets. The (top) dashed plot ("$F_L = F_H$") gives the worst-case baseline: each x guessing wrong until $F_L(x){=}F_H(x)$. The "geo sphere-based" plot uses our p^+,p^- ((32)): guessing "positive" ("negative") if $p^+ > .5$ ($p^- > .5$). The "P,N approx" plot uses reduced set approximations, to estimate $\widetilde{G}_{\widehat{\mathbb{S}}}$ ((37)) and guess (sign($\widetilde{G}_{\widehat{\mathbb{S}}}(x)$)). For each step, $\widehat{\mathbb{S}} = \{S_1, \ldots, S_k\}$ (first k SGMA sequence SVs) and (38) gives β. [13]

Both "P,N approx" and "geo sphere-based": start better than random guessing (78 and 72 errors respectively at k=1), hit zero error at k=74, and perform similarly in between. [14] The rapid drop in "P,N approx" error is consistent with other guessing performances of reduced sets (e.g. (Romdhani et al., 2001)). However, our p^+, p^- guessing drops steadily (1 error by k=50, none after $k=74$), whereas reduced sets clearly act more eratic. This gives evidence that our bounds are good not only for preventing any errors but also as the basis for guessing that is competitive with alternatives not guaranteeing preservation of classifications.

[13] We find bias b for each k same as before (Section 2.1), except using $\widetilde{G}$ instead of G (but same IN^+ and IN^-).

[14] Note the log scale for improved distinguishability.

8. Conclusions

It is important to appreciate both the immediate and the enabling contributions of this work.

Our new incremental anytime bounding approach immediately yields dramatic speedups of SVM classification, without *any* loss of classification accuracy with respect to the original SVM. Our approach follows the guiding principle of Vapnik's seminal SVM work (Vapnik, 1998): do no more work than required for the task at hand. For SVM training, this means not modelling probability densities for a class discrimination task. For SVM classification, this means not computing exact SVM outputs if bounds (or even just signs) suffice.

Also, our speedups seem applicable not only to SVMs, but any kernel machine of the form $G(x) = \sum_i \beta_i K(x, z_i)$ - b, regardless of the training cost function (e.g. as Section 7 shows for KFDs). This includes kernel regression machines, if interval outputs suffice.

Furthermore, our anytime approach also opens the door to more than "just" classification speedups per se. Under certain assumptions (Section 4.6) our bounds provide a reasonable basis for determining at each step k the probability that the original SVM would classify `Q` as positive (or negative). Such anytime probabilistic approaches could inform better ways to allocate computational resources during large time-critical classification tasks. For example, it could inform methods that try to minimize the expected overall test error rate per unit of classification time spent so far, by prioritizing work on those queries with current probabilities of being classified as positive or negative being roughly equal (e.g. queries having wide output bounds centered near zero). When available classification time expires, each query could classify based on anytime probabilities at that time.

This work opens several exciting directions for future work. One is to find better sequences (of S_i points) for a given query `Q`. Our methods so far (Section 5) greedily order SVs only, and without regard to specific queries. As reduced set work shows, vectors other than training examples can provide more compact approximations. Furthermore, different sequence orderings and/or points will work best for different clusters of queries, motivating future work on finding trees, instead of one fixed sequence for all queries.

Finally, we argue that speeding up classification is fundamental to speeding up other aspects, including training and model selection. For example, we are investigating $G(x)$ bounding *during* SMO training, to speed up KKT condition checks (e..g. $y_i G(x_i) > 1$ iff $\alpha_i = 0$) dominating train times for huge data sets.

Acknowledgements

Dominic Mazzoni, Mike Turmon, Becky Castano, and Michael Burl provided helpful feedback. This research was carried out by the Jet Propulsion Laboratory, California Institute of Technology, under contract with the National Aeronautics and Space Administration.

References

Bennett, K. P., & Bredensteiner, E. J. (2000). Duality and geometry in SVM classifiers. *ICML*.

Blake, C., & Merz, C. (1998). UCI repository of machine learning databases.

Burges, C. (1996). Simplified support vector decision rules. *Intl. Conf. on Machine Learning (ICML)*.

Burges, C., & Schölkopf, B. (1997). Improving the accuracy and speed of support vector machines. *NIPS*.

DeCoste, D., & Schölkopf, B. (2002). Training invariant SVMs. *Machine Learning*, *46*.

Downs, T., Gates, K., & Masters, A. (2001). Exact simplification of support vector solutions. *Journal of Machine Learning Research (JMLR)*, *2*, 293–297.

Faloutsos, C., & Lin, K.-I. (1995). FastMap: A fast algorithm for indexing, data-mining and visualization of traditional and multimedia datasets. *ACM SIGMOD Intl. Conf. on Management of Data*.

Mika, S., Rätsch, G., & Müller, K.-R. (2001). A mathematical programming approach to the kernel Fisher algorithm. *Advances in Neural Information Processing Systems (NIPS) 13*.

Platt, J. (1999). Using sparseness and analytic QP to speed training of support vector machines. *NIPS 11*.

Romdhani, S., Torr, P., Schölkopf, B., & Blake, A. (2001). Computationally efficient face detection. *Intl. Conf. on Computer Vision (ICCV-2001)*.

Schölkopf, B., Mika, S., Burges, C., Knirsch, P., Müller, K.-R., Rätsch, G., & Smola, A. (1999). Input space vs. feature space in kernel-based methods. *IEEE Transactions on Neural Networks*, *10*.

Schölkopf, B., & Smola, A. (2002). *Learning with kernels*. Cambridge, MA: MIT Press.

Smola, A., & Schölkopf, B. (2000). Sparse greedy matrix approximation for machine learning. *ICML*.

Vapnik, V. (1998). *Statistical learning theory*. Wiley.

Weisstein, E. (1998). *The CRC concise encylopedia of mathematics*. CRC Press. See http:// mathworld.wolfram.com/SphericalCap.html.

Yoon, J.-M., Gad, Y., & Wu, Z. (2000). *Mathematical modeling of protein structure using distance geometry* (Technical Report 00-24). University of Houston.