

---

# Anytime Query-Tuned Kernel Machine Classifiers Via Cholesky Factorization

---

Dennis DeCoste

Machine Learning Systems Group

Jet Propulsion Laboratory / California Institute of Technology

4800 Oak Grove Drive, Pasadena, CA 91109, USA

*dennis.decoste@jpl.nasa.gov*

## Abstract

We recently demonstrated 2 to 64-fold query-time speedups of Support Vector Machine and Kernel Fisher classifiers via a new computational geometry method for anytime output bounds (DeCoste, 2002). This new paper refines our approach in two key ways. First, we introduce a simple linear algebra formulation based on Cholesky factorization, yielding simpler equations and lower computational overhead. Second, this new formulation suggests new methods for achieving additional speedups, including tuning on query samples. We demonstrate effectiveness on benchmark datasets.

## 1 Introduction

Support vector machines (SVMs) and other kernel methods have shown much recent promise (Schölkopf & Smola, 2002). However, wide-spread use on large-scale tasks remains hindered by query-time costs often much higher than others, such as decision trees and neural networks. For example, an SVM recently achieved the lowest error rates on the MNIST benchmark digit recognition task (DeCoste & Schölkopf, 2002), but classified much more slowly than the previous best (a neural network), due to many SVs for each digit recognizer (around 20,000). It is also troubling that classification costs are *identical* for each query example, *even* for “easy” examples that other methods (e.g. decision trees) can classify relatively quickly.

We recently demonstrated 2 to 64-fold query-time speedups of Support Vector Machine and Kernel Fisher Discriminant classifiers based on a new computational geometry method for anytime output bounds (DeCoste, 2002). Unlike related approximation methods such as “reduced sets” (e.g. (Burges, 1996; Schölkopf et al., 1999; Schölkopf et al., 1998; Burges & Schölkopf, 1997; Romdhani et al., 2001)), our approach guarantees preservation of *all* classifications of the original kernel machine. Furthermore, unlike related “exact simplification” methods (Downs et al., 2001) we also achieve “proportionality to difficulty” — our classification time tends to be inversely proportional to a query’s distance from the discriminant hyperplane.

This new paper improves upon (DeCoste, 2002) in two key ways. First, Section 3 introduces a simple linear algebra formulation based on Cholesky factorization,

giving simpler equations and lower computational overhead. Second, Section 4 develops new methods that exploit this formulation, giving additional speedups, including tuning for query samples. Section 5 demonstrates on benchmark datasets.

## 2 Brief Review of Kernel Machines

This section reviews key kernel machine terminology. For concreteness, but without loss of generality, we do so in terms of one common case (binary SVMs).

Given a  $d$ -by- $n$  data matrix ( $X$ ), an  $n$ -by-1 labels vector ( $y$ ), a kernel function ( $K$ ), and a regularization scalar ( $C$ ), a binary SVM classifier is trained by optimizing an  $n$ -by-1 weighting vector  $\alpha$  to satisfy the Quadratic Programming (QP) dual form:

$$\begin{aligned} \text{minimize:} \quad & \frac{1}{2} \sum_{i,j=1}^n \alpha_i \alpha_j y_i y_j K(X_i, X_j) - \sum_{i=1}^n \alpha_i \\ \text{subject to:} \quad & 0 \leq \alpha_i \leq C, \quad \sum_{i=1}^n \alpha_i y_i = 0, \end{aligned}$$

where  $n$  is the number of training examples and  $y_i$  is the label (+1 for positive example, -1 for negative) for the  $i$ -th  $d$ -dimensional training example ( $X_i$ ).

The kernel avoids the curse of dimensionality by implicitly projecting any two  $d$ -dimensional example 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:

$$K(X_i, X_j) \equiv \Phi(X_i) \cdot \Phi(X_j). \quad (1)$$

Popular kernels (with model selection parameters  $a, p, \sigma$ ) include:<sup>1</sup>

$$\begin{aligned} \text{linear:} \quad & K(u, v) = u \cdot v \equiv u^T v \equiv \sum_{i=1}^d u_i v_i, \\ \text{polynomial:} \quad & K(u, v) = (u \cdot v + a)^p, \\ \text{RBF:} \quad & K(u, v) = \exp\left(-\frac{\|u-v\|^2}{2\sigma^2}\right), \\ \text{normalized:} \quad & K(u, v) = G(u, v)G(u, u)^{-\frac{1}{2}}G(v, v)^{-\frac{1}{2}}, \text{ for any given kernel } G. \end{aligned}$$

The output  $f(x)$  on any query example  $x$ , for any kernel machine with trained weighting vector  $\beta$ , is defined (for suitable scalar bias  $b$  also determined during training) as a simple dot product in kernel feature space:

$$f(x) = W \cdot Q - b, \quad W = \sum_{i=1}^n \beta_i \Phi(X_i), \quad Q \equiv \Phi(x). \quad (2)$$

Assume (without loss of generality) that only the first  $m$  ( $m \leq n$ ) columns of  $X$  have non-zero  $\beta_i$ . Output  $f(x)$  is traditionally computed in  $\mathcal{O}(m)$  time via:

$$f(x) = \sum_{i=1}^m \beta_i \Phi(X_i) \cdot \Phi(x) - b = \sum_{i=1}^m \beta_i K(X_i, x) - b \quad (3)$$

For example, a binary SVM (of  $m$  SVs) has  $\beta_i = y_i \alpha_i$  and classifies  $x$  as  $\text{sign}(f(x))$ .

## 3 Anytime Output Bounds via Cholesky Factorization

This section reformulates the complex computational geometry method of (DeCoste, 2002) as a simpler linear algebra approach having lower computational overhead.

Our basic idea is to efficiently embed the (implicit) feature space points  $Q$  and  $W$  of (2) into *explicit*, but typically short ( $k \ll m$ ),  $k$ -dimensional vectors  $Q(k)$  and  $W(k)$ , such that we can approximate (3) in only  $\mathcal{O}(k)$  time via:

$$f_k(x) = W(k) \cdot Q(k) - b \approx W \cdot Q - b = f(x). \quad (4)$$

<sup>1</sup>Where the 2-norm is defined as  $\|u-v\|^2 \equiv u \cdot u - 2u \cdot v + v \cdot v$ .

However, in contrast to similar “reduced set” approximations (e.g. (Burges, 1996)), we also determine tight *bounds* for any size- $k$  embeddings  $W(k)$  and  $Q(k)$ :

$$L_k(x) \equiv f_k(x) - \text{gap}_k(x) \leq f(x) \leq f_k(x) + \text{gap}_k(x) \equiv H_k(x), \quad (5)$$

by computing (via (23) later) a key quantity  $\text{gap}_k(x) \geq 0$  at query-time, having the convergence property that  $\text{gap}_{k+1}(x) < \text{gap}_k(x)$  until  $\text{gap}_{k+1}(x) = \text{gap}_k(x) = 0$ .

These bounds enable us to aggressively speedup classification, by exploiting “proportionality to difficulty” while preserving *all* classifications of the original kernel machine. In an anytime manner, for each query  $x$ , we will consider progressively larger  $k$  until  $\text{sign}(L_k(x)) = \text{sign}(H_k(x))$ .

### 3.1 Stage 1 (pre-query-time): Embed $W$

In this section we show how to pre-compute once, the embedding of  $W$ , which enables efficient query-time computation of  $\text{gap}_k(x)$  in the following two sections. Assume we are given a matrix  $Z$  representing an ordered sequence of  $n_Z$  ( $n_Z \leq n$ )  $d$ -dimensional column vectors  $Z_i$ . For example, Section 4 discusses how to produce good  $Z$  via a (greedy) ordering of the original training data  $X$ .

First, compute the  $n_Z$ -by- $n_Z$  matrix  $K_{ZZ}$ ,  $n_Z$ -by-1 vector  $K_{ZW}$ , and scalar  $K_{WW}$ :

$$K_{ZZ}(i, j) = \Phi(Z_i) \cdot \Phi(Z_j) = K(Z_i, Z_j) \quad (1 \leq i, j \leq n_Z), \quad (6)$$

$$K_{ZW}(i) = \Phi(Z_i) \cdot \sum_{j=1}^m \frac{\beta_j}{s} \Phi(X_j) = \sum_{j=1}^m \frac{\beta_j}{s} K(Z_i, X_j) \quad (1 \leq i \leq n_Z), \quad (7)$$

$$K_{WW} = \sum_{i=1}^m \sum_{j=1}^m \frac{\beta_i \beta_j}{s^2} \Phi(X_i) \cdot \Phi(X_j) = \frac{1}{s^2} \beta' K_{XX} \beta, \quad K_{XX}(i, j) = K(X_i, X_j), \quad (8)$$

where  $\beta$  are the kernel machine’s trained weights over  $X$  and, for numeric stability, we use normalization scalar:

$$s = \sum_{i=1}^m |\beta_i| \quad (9)$$

(otherwise,  $K_{WW}$  would scale with  $m$  whereas  $K_{ZZ}(i, j)$  do not).

Second, combine them all into one composite kernel matrix:<sup>2</sup>

$$\mathcal{K} = \left[ \begin{array}{c|c} K_{ZZ} & K_{ZW} \\ \hline K_{ZW}' & K_{WW} \end{array} \right], \quad (10)$$

Finally, we embed all  $n_Z + 1$  of these kernel space points (i.e. all  $Z_i \equiv \Phi(Z_i)$  and  $W$ ) into  $n_Z + 1$  dimensions. To compute the  $(n_Z + 1)$ -by- $(n_Z + 1)$  embedding matrix  $V$ , we simply use Cholesky factorization:  $\mathcal{K} = V'V$ .

Table 1 illustrates the resulting upper-triangular embedding matrix (i.e. when  $k=n_Z$ ). Specifically,  $V$  is the upper-leftmost  $(k + 1)$ -by- $(k + 1)$  portion. For notational simplicity, we define  $W \equiv s \cdot V_{n_Z+1}$ , so that  $W_i$  clearly refers to the  $i$ -th coordinate of  $W$ ’s embedding (i.e.  $W_i \equiv s \cdot V_{(n_Z+1),i}$ ).

Note that at any step  $k$  the dot product of any two columns of the embedding matrix of Table 1 is equal to the kernel value for those two points, by the nature of

<sup>2</sup>In practice, we add a small  $\epsilon = 10^{-8}$  to  $\mathcal{K}$ ’s diagonal, ensuring positive-definiteness even when  $W$  is a linear combination over  $Z$  (e.g. if  $Z$  contains the same vectors as  $X$ ).

the Cholesky factorization. E.g.  $\frac{1}{s}W \cdot \frac{1}{s}W = K_{WW}$  and  $V_i \cdot V_j = K(Z_i, Z_j)$ . Thus, we also pre-compute the following quantities as well (for use in Stage 3):

$$\boxed{W_y(k) = W_{k+1}, \quad W_z(k) = [s^2 \cdot K_{WW} - \sum_{i=1}^{k+1} W_i^2]^{\frac{1}{2}}.} \quad (11)$$

Note that  $W_z(k) = 0$  occurs whenever  $Z_1, \dots, Z_k$  provides a complete basis function set for  $W$  (e.g. if  $Z = X$  this always occurs for some  $k \leq m$ ).

Table 1: Embedding matrix of  $k + 2$  points at step  $k$ .

	$Z_1$	$Z_2$	$\dots$	$Z_{k-1}$	$Z_k \equiv \Phi(Z_k)$	$W$	$Q \equiv \Phi(x)$
1	$V_{1,1}$	$V_{2,1}$	$\dots$	$V_{k-1,1}$	$V_{k,1}$	$W_1$	$Q_1$
2	0	$V_{2,2}$	$\dots$	$V_{k-1,2}$	$V_{k,2}$	$W_2$	$Q_2$
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\dots$	$\vdots$	$\vdots$	$\vdots$
$k-1$	0	0	$\dots$	$V_{k-1,k-1}$	$V_{k,k-1}$	$W_{k-1}$	$Q_{k-1}$
$k$	0	0	$\dots$	0	$V_{k,k}$	$W_k$	$Q_k$
$y \equiv k+1$	0	0	$\dots$	0	0	$W_{k+1}$	$Q_y(k) = ?$
$z \equiv k+2$	0	0	$\dots$	0	0	$W_z(k)$	$Q_z(k) = ?$

### 3.2 Stage 2 (query-time): Embed $Q$

In principle,  $Q$ 's embedding *could*, similarly to  $W$  above, be computed as the last column of  $U$  in the Cholesky factorization  $K_Q = U'U$ , via extended kernel matrix:

$$K_Q = \left[ \begin{array}{c|c} \mathcal{K} & \frac{K_{ZQ}}{K_{WQ} = ?} \\ \hline \frac{K'_{ZQ}}{K'_{WQ} = ?} & K_{QQ} \end{array} \right], \quad (12)$$

computed from the  $n_Z$ -by-1 matrix  $K_{ZQ}$ , scalar  $K_{WQ}$ , and scalar  $K_{QQ}$ :

$$K_{ZQ}(i) = \Phi(Z_i) \cdot Q = K(Z_i, x) \quad (1 \leq i \leq n_Z), \quad (13)$$

$$K_{WQ} = Q \cdot \sum_{j=1}^m \beta_j \Phi(X_j) = \sum_{j=1}^m \beta_j K(X_j, x), \quad (14)$$

$$K_{QQ} = Q \cdot Q = K(x, x). \quad (15)$$

However, comparing (14) with (3) shows that  $f(x) = K_{WQ} - b$ . Thus, computing  $K_{WQ}$  would amount to the expensive traditional exact computation of output  $f(x)$  that we are trying to avoid whenever possible.

Instead, we use only the equations from the Cholesky factorization that compute as much of the embedding of  $Q$  that we can *without requiring*  $K_{WQ}$ . In particular, at each step  $k$  we compute the Cholesky equation for one new coordinate of  $Q$ :<sup>3</sup>

$$\boxed{Q_k = [K(Z_k, x) - \sum_{i=1}^{k-1} V_{k,i} \cdot Q_i] / V_{k,k}.} \quad (16)$$

As Table 1 notes, only two coordinates,  $Q_y(k)$  and  $Q_z(k)$ , then remain unknown.

<sup>3</sup>Note that  $\mathcal{K}$  being positive-definite ensures that  $V_{k,k} > 0$ .

### 3.3 Stage 3 (query-time): Bound $f(x)$ via Closed-Form Optimization

At step  $k$ , given  $Q_k$  from (16), we first reformulate (4) as an incremental update:

$$f_k(x) = \sum_{i=1}^k W_i \cdot Q_i - b \Rightarrow \boxed{f_k(x) = f_{k-1}(x) + W_k \cdot Q_k, \quad f_0(x) = -b,} \quad (17)$$

where the relation between exact output  $f(x)$  and approximation  $f_k(x)$  is:

$$F_k(x) \equiv f_k(x) + W_y(k)Q_y(k) + W_z(k)Q_z(k), \quad f(x) = F_k(x) \quad (1 \leq k \leq n_Z). \quad (18)$$

Second, we incrementally update the remaining ‘‘residual’’ in  $Q$ :

$$\boxed{R_k^2(x) = R_{k-1}^2(x) - Q_k^2, \quad R_0^2(x) = K(x, x),} \quad (19)$$

which constrains the two remaining (unknown) coordinates (since  $Q \cdot Q \equiv K(x, x)$ ):

$$R_k^2(x) = Q_y^2(k) + Q_z^2(k). \quad (20)$$

Third, we solve (18) with (20) for  $\frac{\delta F_k(x)}{\delta Q_y(k)} = 0$  and  $\frac{\delta F_k(x)}{\delta Q_z(k)} = 0$ , obtaining the values of  $Q_y(k)$  and  $Q_z(k)$  at the extrema of  $F_k(x)$ :<sup>4</sup>

$$\boxed{Q_{y^*}(k) = \pm \left[ \frac{R_k^2(x) W_y^2(k)}{W_y^2(k) + W_z^2(k)} \right]^{\frac{1}{2}}, \quad Q_{z^*}(k) = \pm \left[ R_k^2(x) - Q_{y^*}^2(k) \right]^{\frac{1}{2}}} \quad (21)$$

Finally, we get bounds  $L_k(x)$  and  $H_k(x)$  at step  $k$  for the output  $f(x)$  of query  $x$ :

$$\boxed{L_k(x) \equiv f_k(x) - \text{gap}_k(x) \leq f(x) \leq f_k(x) + \text{gap}_k(x) \equiv H_k(x),} \quad (22)$$

$$\boxed{\text{gap}_k(x) = |W_y(k) Q_{y^*}(k)| + |W_z(k) Q_{z^*}(k)| \geq 0,} \quad (23)$$

by plugging (21) into (18) for both extrema of  $F_k(x)$ .

### 3.4 Anytime Classification

For each query  $x$ , we repeat Stages 2 and 3 over steps  $k = 1, \dots, n_Z$ , until  $\text{sign}(L_k(x)) = \text{sign}(H_k(x))$ . Due to the nature of our incremental embedding, most embedding coordinates in  $W$  and  $Q$  persist between steps:

$$Q_i(k+1) = Q_i(k) \quad (1 \leq i \leq k), \quad W_j(k+1) = W_j(k) \quad (1 \leq j \leq k+2). \quad (24)$$

Table 1 uses gray backgrounds to highlight the three coordinates not persisting (i.e.  $W_z(k)$ ,  $Q_y(k)$ , and  $Q_z(k)$ ).  $W_z(k)$  is precomputed as in (11), while  $Q_y(k)$  and  $Q_z(k)$  are never computed (i.e. bound by computing extrema  $Q_{y^*}(k)$  and  $Q_{z^*}(k)$ ).

Computing  $Q_k$  via (16) dominates the time cost of each step  $k$ . It requires one  $\mathcal{O}(d)$  kernel ( $K(Z_k, x)$ ) and  $\mathcal{O}(k-1)$  summation over previous  $Q$  coordinates. Thus,  $k$  steps require  $\mathcal{O}(k \cdot d + k^2)$  time, whereas exact  $f(x)$  always requires  $\mathcal{O}(d \cdot m)$ . So, whenever  $k > \min(m, \sqrt{d \cdot m})$  occurs before  $\text{sign}(L_k(x)) = \text{sign}(H_k(x))$  occurs for query  $x$ , it becomes advisable to switch to computing  $f(x)$  directly (via (3)). Fortunately, empirical evidence suggests this seldom occurs in practice, particularly for the common case of  $m > d$  in large applications. Typical time overhead per query-time step  $k$  is under 25% of a kernel computation, via efficient coding tricks.

<sup>4</sup>Plug (20) into (18):  $0 = \frac{\delta F_k(x)}{\delta Q_y(k)} = W_y(k) + \frac{1}{2}W_z(k)[R_k^2(x) - Q_y^2(k)]^{-\frac{1}{2}}[-2Q_y(k)] \rightarrow W_y(k) = W_z(k)[R_k^2(x) - Q_y^2(k)]^{-\frac{1}{2}}Q_y(k) \rightarrow W_y^2(k) = W_z^2(k)Q_y^2(k)[R_k^2(x) - Q_y^2(k)]^{-1} \rightarrow Q_y^2(k) = \frac{R_k^2(x) W_y^2(k)}{W_y^2(k) + W_z^2(k)}$ .

## 4 Greedy Selection of $Z$ From $X$ (and Query Samples)

Simple, yet effective, sequences of  $Z_i$ 's can be determined by greedily ordering the columns of  $X$ . In (DeCoste, 2002) we simply ordered the columns using Sparse Greedy Matrix Approximation (SGMA) (Smola & Schölkopf, 2000). Our new Cholesky-based approach suggests the following better alternatives.

First, consider greedy search over  $k = 1, \dots, m$ , with iteration  $k$  selecting the next best column  $Z_k \in X$ . We select the candidate  $X_i$  ( $1 \leq i \leq m$ ) which minimizes:

$$\text{cost}_k(Z_k = X_i) = W_z(k), \quad (25)$$

where  $W_z(k)$  is computed as in (11), using a simple rank-1 Cholesky factorization update of our embedding  $V$  (since the new candidate ( $X_i$ ) for  $Z_k$  extends  $\mathcal{K}$  (of (10)) by rank-1). This cost is proportional to the approximation error between  $W$  and its embedding  $W(k)$ , yielding  $Z_i$  sequences essentially equivalent to reduced set search (e.g. (Burges, 1996)) results over the same candidates.

This approach (call it `minWz`) improves upon our earlier use of SGMA in (DeCoste, 2002). The existence of well-tuned Cholesky factorization code (e.g. MATLAB) makes our computation of cost  $W_z(k)$  particularly efficient, enabling search over more candidates at each iteration  $k$  than our previous SGMA approach. Furthermore, our use of SGMA approximated matrix  $K_{ZZ}$ , whereas our bounds tighten quickest with better approximations of  $W$  (which occurs when  $W_z(k)$  is minimized).

Second, we find that using all  $n$  training examples ( $X$ ), instead of just the  $m$  SVs, as candidates for  $Z_k$  often helps. Often a non-SV provides a better (e.g. more orthogonal) basis function than multiple SVs. We refer to this cost as  $\text{cost}_k(Z_k = X_i^{(n)}) = W_z(k)$  and ordering based on it as `minWzn`. Since  $n$  is often much larger than  $m$  for SVMs, we limit search cost by employing the “59-trick” of (Smola & Schölkopf, 2000), considering at most 59 randomly-selected non-SVs at every iteration  $k$ , in addition to all unchosen SVs.

Third, given a sample set of representative queries ( $\mathcal{Q}$ ), we can do even better. Consider the same greedy ordering search over  $X$  as above, but with a new cost:

$$\text{cost}_k(Z_k = X_i^{(n)}, \mathcal{Q}) = \sum_{x \in \mathcal{Q}^-} \max(H_k(x), 0) - \sum_{x \in \mathcal{Q}^+} \min(L_k(x), 0) \quad (26)$$

where  $\mathcal{Q}^- \equiv \{x \in \mathcal{Q} : f(x) < 0\}$  and  $\mathcal{Q}^+ \equiv \{x \in \mathcal{Q} : f(x) > 0\}$ . That is, we optimize the sequence of  $Z_i$ 's to encourage the lower bounds to rapidly raise (across  $k$ ), until above 0, for query samples classified as positive by the kernel machine, while similarly encouraging the upper bound to drop (for negative queries). This method (call it `meanLHn`) is motivated by the fact that correct classification only requires the correct sign.

In the extreme singleton case of  $\mathcal{Q} = \{X_i\}$ , this `meanLHn` yields  $Z_1 = X_i$ . In general, optimizing with  $\text{cost}_k(Z_k = X_i, \mathcal{Q})$  over a large representative  $\mathcal{Q}$  explicitly moves us towards our ultimate goal: classifying future queries with the lowest expected  $k$ .

Unfortunately, we observe that query-ignorant `minWzn` often produces  $Z_i$  orderings yielding much higher classification speedups than query-tuned `meanLHn` does, when used for the entire  $Z$  ordering. However, we find that `meanLHn` provides an excellent *tie-breaker*, for selecting among candidates for  $Z_k$  which have similar `minWz` cost.<sup>5</sup> This hybrid (call it `minWzn&meanLHn`) balances `meanLHn`'s greed to exploit the sufficiency of classification signs against `minWzn`'s typically less aggressive but steady improvements. Other hybrids are likely even better and worthy of future research.

<sup>5</sup>In our experiments, “ties” are candidates with `minWz` cost within 1% of the best.

## 5 Examples

We checked our approach on two UCI datasets (Blake & Merz, 1998), Sonar and Haberman, and the MNIST digit-recognition dataset (LeCun, 2000). We confirmed that  $L_k(x) \leq f(x) \leq H_k(x)$  always held. Table 2 summarizes some of our results.

Rows labelled 1-2 summarize input dimension and number of positive and negative examples for each dataset. Rows 3-6 summarize the trained SVMs for each.

Row 11 shows statistics on how many steps  $k$  were required to classify (i.e. determine  $\text{sign}(f(x))$ ) for all  $n$  examples, where the  $Z_i$  are simply the SVs in the original order given in each dataset. Row 12 computes speedup relative to exact classification, as the ratio of the number of SVs ( $m$ ) vs the mean  $k$  required (e.g.  $165/47.7 = 3.5$ ). Successive pairs of rows give similar statistics for other ordering methods.

(DeCoste, 2002) also examined Sonar and Haberman, using SGMA ordering, giving mean  $k$  of 26.2 and 4.5, respectively. In contrast, minWz ordering gives 21.3 (better) and 7.0 (worse), respectively. Poor behavior on Haberman (versus both SGMA ordering and even unordered) is corrected using minWz<sup>n</sup>&meanLH<sup>n</sup>. In Haberman many candidates give similar minWz costs, making the greedy search very noisy. Comparing across rows 32 versus 42 shows that hybrid minWz<sup>n</sup>&meanLH<sup>n</sup> overcomes such problematic cases, while not suffering significant degradation in other cases.

We tried both an easy (0 vs 1) and a hard (3 vs 8) MNIST pairwise classification task, using the small 10,000 digit MNIST dataset. minWz<sup>n</sup>&meanLH<sup>n</sup> achieved significant average speedup for the easy task, which is impressive given MNIST's high dimensionality ( $d = 784$ ). For the hard task, the speedup was relatively small (1.7). We suspect that speedups for hard cases such as MNIST(3vs8) will require more clever partitioning of query space, such as using different  $Z_i$  sequences for different clusters, instead of using one global sequence as in our current methods. <sup>6</sup>

Table 2: SVM classification speedups summary.

	Sonar	Haberman	MNIST(3vs8)	MNIST(0vs1)
1 $d$	60	3	784	784
2 $n^+, n^-$	97, 111	81, 225	1010, 974	1135, 980
3 kernel (normalized)	$(u \cdot v + 1)^2$	$(u \cdot v + 1)^3$	$(u \cdot v + 1)^2$	$(u \cdot v + 1)^2$
4 $C$	1	1000	2	2
5 $m$ SVs (count of $\alpha_i = C$ )	165 (153)	180 (150)	262 (71)	76 (1)
6 $m^+, m^-$	81, 84	79, 101	132, 130	24, 52
$Z_i = \text{unordered SVs:}$				
11 $k$ min,mean,median,max	1, 47.7, 45, 140	1, 4.2, 3, 22	1, 220.8, 233, 258	1, 61.1, 60, 76
12 $m/k$ mean speedup	3.5	42.2	1.2	1.2
$Z_i = \text{ordered SVs via minWz:}$				
21 $k$ min,mean,median,max	1, 21.3, 17, 123	1, 7.0, 8, 9	1, 150.3, 154, 205	1, 21.3, 21, 32
22 $m/k$ mean speedup	7.7	25.5	1.7	3.6
$Z_i = \text{ordered } X, \text{ via minWz}^n \text{ cost:}$				
31 $k$ min,mean,median,max	1, 22.5, 19, 109	1, 6.1, 6, 9	1, 148.8, 155, 215	1, 17.9, 18, 35
32 $m/k$ mean speedup	7.3	29.5	1.8	4.2
$Z_i = \text{ordered } X, \text{ via minWz}^n \& \text{meanLH}^n \text{ cost } (Q=X):$				
41 $k$ min,mean,median,max	1, 21.3, 18, 129	1, 4.0, 3, 12	1, 151.4, 157, 217	1, 17.1, 17, 33
42 $m/k$ mean speedup	7.7	44.5	1.7	4.4

<sup>6</sup>We also observe that non-SVM kernel machines such as Kernel Fisher Discriminants, where usually all  $\beta_i \neq 0$ , often enjoy even greater speedups (not shown due to space limits).

## 6 Discussion

(DeCoste, 2002) explored various motivations and applications of anytime output bounding. Such bounding applies to any kernel machine of the form  $f(x) = \sum_i \beta_i K(X_i, x) - b$ . Below, we discuss insights suggested by our new improvements.

This work shows that reducing the error in approximating a kernel machine (as is the focus of previous reduced set (e.g. (Burges, 1996)) work) is not the only viable way to speedup classification. Our approach makes explicit that  $F_k(x) \rightarrow f(x)$  convergence (as step  $k$  increases) actually depends on *either* the machine's  $W$  residual  $W_z(k) \rightarrow 0$  *or* the query's  $Q$  residual  $R_k(x) \rightarrow 0$ , whichever is faster. The approximated output  $F_k(x)$  for a given query  $x$  is increased (decreased) by the product ( $W_k \cdot Q_k$ ) between the embedding coordinates of  $W$  and  $Q$ , if they are the same (different) sign. Yet, classification tasks do not necessarily care if the approximate output is too high (low) for a positive (negative) query, so long as we know the uncertainty (i.e. our  $\text{gap}_k(x)$ ) sufficient to tell us the sign of the output.

These insights suggest new costs (e.g. (26)) and methods (e.g.  $\text{minWz}^n \& \text{meanLH}^n$ ) to exploit them. This paper only begins to explore the many possibilities opened up by this work. In particular, we suspect that recursive partitioning of the query space, via tree structures, might allow our approach to provide significant further speedups (such as exploiting the existence of clusters in the expected query space).

Other promising directions that we are investigating include: optimizing over vectors in  $\mathcal{R}^d$  to obtain more optimal  $Z_i$  vectors (i.e. reduced set search, but using our new cost functions) and using anytime bounding *during* training itself (to speedup the KKT condition checks that dominate training times for many huge datasets).

### Acknowledgments

Dominic Mazzoni and Michael Turmon provided useful 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

- 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. (2002). Anytime interval-valued outputs for kernel machines: Fast support vector machine classification via distance geometry. *ICML*. <http://citeseer.nj.nec.com/decoste02anytime.html>.
- 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.
- LeCun, Y. (2000). MNIST handwritten digits dataset. Available at <http://www.research.att.com/~yann/ocr/mnist/>.
- Romdhani, S., Torr, P., Schölkopf, B., & Blake, A. (2001). Computationally efficient face detection. *Intl. Conf. on Computer Vision (ICCV-2001)*.

- Schölkopf, B., Knirsch, P., Smola, A., & Burges, C. (1998). Fast approximation of support vector kernel expansions, and an interpretation of clustering as approximation in feature spaces. *Mustererkennung 1998 — 20. DAGM-Symposium*. Springer.
- 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*.