Fast Query-Optimized Kernel Machine Classification Via Incremental Approximate Nearest Support Vectors

Dennis DeCoste DENNIS.DECOSTE@JPL.NASA.GOV
Jet Propulsion Laboratory / Caltech, 4800 Oak Grove Drive, Pasadena, CA 91109, USA

Dominic Mazzoni DOMINIC.MAZZONI@JPL.NASA.GOV
Jet Propulsion Laboratory / Caltech, 4800 Oak Grove Drive, Pasadena, CA 91109, USA

Abstract

Support vector machines (and other kernel machines) offer robust modern machine learning methods for nonlinear clasification. However, relative to other alternatives (such as linear methods, decision trees and neural networks), they can be orders of magnitude slower at query-time. Unlike existing methods that attempt to speedup query-time, such as reduced set compression (e.g. (Burges, 1996)) and anytime bounding (e.g. (DeCoste, 2002), we propose a new and efficient approach based on treating the kernel machine classifier as a special form of k nearest-neighbor. Our approach improves upon a traditional k-NN by determining at query-time a good k for each query, based on pre-query analysis guided by the original robust kernel machine. We demonstrate effectiveness on high-dimensional benchmark MNIST data, observing a greater than 100-fold reduction in the number of SVs required per query (amortized over all 45 pairwise MNIST digit classifiers), with no extra test errors (in fact, it happens to make 4 fewer).

1. Introduction

Kernel machine (KM) methods, such as support vector machines (SVMs) and kernel Fischer discrmininants (KFDs), have become promising and popular methods in modern machine learning research (Schölkopf & Smola, 2002). Using representations which scale only linearly in the number of training examples, while (implicitly) exploring large nonlinear (kernelized) feature spaces (that are exponentially larger than the original input dimensionality), KMs elegantly and practically overcome the classic “curse of dimensionality”.

However, the tradeoff for this power is that a KM's query-time complexity scales linearly with the number of training examples, making KMs often orders of magnitude more expensive at query-time than other popular machine learning alternatives (such as decision trees and neural networks). For example, an SVM recently achieved the lowest error rates on the MNIST (LeCun, 2000) benchmark digit recognition task (DeCoste & Schölkopf, 2002), but classifies much slower than the previous best (a neural network), due to digit recognizers having many (e.g. 20,000) support vectors.

1.1. The Goal: Proportionality to Difficulty

Despite the significant theoretical advantages of KMs, their heavy query-time costs are a serious obstacle to KMs becoming the practical method of choice, especially for the common case when the potential gains (in terms of reduced test error) are often relatively modest compared to that heavy cost. For example, it is not atypical in practice for a simple linear method to achieve 80-95% test accuracy, with a KM improving this by a few percentages more. Although such improvements are often significant and useful, it does raise serious “bang for buck” issues which hinder KMs from being more widely deployed, especially for tasks involving huge data sets (e.g. data mining) and/or query-time constraints (e.g. embedded onboard resource-limited spacecraft or real-time robots).

Also troubling is that KM costs are *identical* for each query, *even* for “easy” ones that alternatives (e.g. decision trees) can classify much faster than harder ones.

What would be ideal would be an approach that: 1) only uses a simple (e.g. linear) classifier for the (majority of) queries it is likely to correctly classify, 2) incurs the much heavier query-time cost of an exact KM only for those (relatively rare) queries for which such precision likely matters, and 3) otherwise, uses something in between (e.g. an approximate KM), whose complexity is proportional to the difficulty of the query.

1.2. Summary of Our Approach

The approach we propose in this paper directly attempts to find a good approximation to the above ideal. It is based on our empirical observation that one can often achieve the same classification as the exact KM by using only small fraction of the nearest support vectors (SVs) of a query. Whereas the exact KM output is a (weighted) sum over the kernel values between the query and the SVs, we will approximate it with a k nearest-neighbor (k-NN) classifier, whose output sums only over the (weighted) kernel values involving the k selected SVs.

Before query-time we gather statistics about how misleading this k-NN can be (relative to the outputs of the exact KM, for a representative set of examples), for each possible k (from 1 to the total number of SVs). From these statistics we derive upper and lower thresholds for each step k, identifying output levels for which our variant of k-NN already leans so strongly positively or negatively that a reversal in sign is unlikely, given the (weaker) SV neighbors still remaining.

At query-time we then incrementally update each query's partial output, stopping as soon as it exceeds the current step's predetermined statistical thresholds. For the easy queries, early stopping can occur as early as step $k = 1$. For harder queries, stopping might not occur until nearly all SVs are touched.

A key empirical observation is that this approach can tolerate very approximate nearest-neighbor orderings. Specifically, in our experiments we project the SVs and query to the top few principal component dimensions and compute neighbor orderings in that subspace. This ensures that the overhead of the nearest neighbor computations are insignificant relative to the exact KM computation.

2. Kernel Machines Summary

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 α to satisfy the Quadratic Programming (QP) dual form:

minimize: $\frac{1}{2}\sum_{i,j=1}^{n} \alpha_i\alpha_j y_i y_j K(X_i, X_j) - \sum_{i=1}^{n} \alpha_i$

subject to: $0 \leq \alpha_i \leq C, \quad \sum_{i=1}^{n} \alpha_i y_i = 0,$

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 curse of dimensionality by implicitly projecting any two d-dimensional example vectors in input space (X_i and X_j) into feature space vectors ($\Phi(X_i)$ and $\Phi(X_j)$), returning their dot product:

$$K(X_i, X_j) \equiv \Phi(X_i) \cdot \Phi(X_j). \tag{1}$$

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

linear: $K(u, v) = u \cdot v \equiv u^T v \equiv \sum_{i=1}^{d} u_i v_i,$

polynomial: $K(u, v) = (u \cdot v + a)^p,$

RBF: $K(u, v) = \exp(-\frac{1}{2\sigma^2}\ ||u - v||^2),$

normalized: $K(u, v) := K(u, v)K(u, u)^{-\frac{1}{2}}K(v, v)^{-\frac{1}{2}}.$

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

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

The exact KM output $f(x)$ is computed via: [2]

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

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

3. Related Work on Lower Query Costs

Early methods for reducing the query-time costs of KMs focussed on optimization methods to compress a KM's SVs into a "reduced set" (Burges, 1996; Burges & Schölkopf, 1997; Schölkopf et al., 1998; Schölkopf et al., 1999). This involves approximating:

$$\widehat{f}(x) = \widehat{\mathtt{W}} \cdot \mathtt{Q} - \widehat{b} \quad \approx \quad f(x) = \mathtt{W} \cdot \mathtt{Q} - b \tag{4}$$

by optimizing over all $Z_i \in \Re^d$ and $\gamma_i \in \Re$ such that:

$$\widehat{\mathtt{W}} = \sum_{i=1}^{n_Z} \gamma_i \Phi(Z_i) \tag{5}$$

[1] Where the 2-norm is defined as $||u - v||^2 \equiv u \cdot u - 2u \cdot v + v \cdot v$ and u^T indicates the matrix transpose.

[2] Assume (without loss of generality) that only the first m (for some $m \leq n$) columns of X have non-zero β_i.

minimizes the approximation cost:

$$\rho^2 = ||\mathtt{W} - \widehat{\mathtt{W}}||^2, \tag{6}$$

yielding

$$\widehat{f}(x) = \sum_{i=1}^{n_Z} \gamma_i \Phi(Z_i) \cdot \Phi(x) - b = \sum_{i=1}^{n_Z} \gamma_i K(Z_i, x) - b \tag{7}$$

When small $\rho \approx 0$ can be achieved with $n_Z \ll n$, via (often costly) global optimization, significant (e.g. 10-20 fold) speedups with little loss of classification accuracy have been reported (e.g. (Burges, 1996)).

For linear kernels, it is well known that a KM compresses with no error into a single d-dimensional $\mathtt{w}$:

$$f(x) = \mathtt{w} \cdot x - b, \quad \mathtt{w} = \sum_{i=1}^{m} \beta_i X_i. \tag{8}$$

However, for general kernels, a shortcoming of reduced sets is that every output $\widehat{f}(x)$ requires exactly $n_Z \gg 1$ kernel computations. Thus, (Romdhani et al., 2001) proposes a sequential approach, stopping early for a query as soon as its partial output drops below zero (indicating, in their application, a non-face).

A key problem with all such reduced set approaches is that they do not provide any guarantees or control concerning how much classification error might be introduced by such approximations. (DeCoste, 2002; DeCoste, 2003) develop sequential methods which quickly compute upper and lower bounds on what the KM output for a given query could potentially be, at each step k. For classification tasks, these bounds allow one to confidently stop as soon as both bounds for a given query move to the same side of zero. However, computation of these bounds involves a k^2 term (due to the use of incomplete Cholesky decomposition). Despite working well over several test data sets, this overhead makes that bounding approach often useless for the sort of very high-dimensional image classification tasks we examine in our empirical work in this paper.

4. Nearest Support Vectors (NSV)

The key intuition behind the new approach proposed in this paper is that, during incremental computation of the KM output for a query (via (3), one SV per step), once the partial KM output leans "strong enough" either positively or negatively, it will not be able to completely reverse course (i.e. change sign) as remaining $\beta_i K(X_i, x)$ terms are added. To encourage this to happen in as few steps as possible (for maximum speedup at query-time), we will intelligently (and efficiently) order the SVs for each query, so that the largest terms tend to get added first. To enable us to know when a leaning is "strong enough" to "safely" stop early, we will gather statistics over representative data (before query-time), to see how strongly the partial output must lean at each step for such stopping to lead to the same classification (i.e. sign) as the exact KM output.

That such an approach could work is not so surprising if one views the KM output computation (i.e. (3)) as a form of weighted nearest-neighbor voting, in which the β's are the weights and the kernel values reflect (inverse) distances. Our inspiration is that small-k nearest-neighbor classifiers can often classify well, but that the best k will vary from query to query (especially near the discriminant border), prompting us to consider how to harness the robustness of a KM to guide determination of a good k for each query. Due to its relation to nearest-neighbors, we call our approach "nearest support vectors" (NSV).

Let NSV's distance-like scoring be defined as:

$$\mathtt{NNscore}_i(x) = |\beta_i| \; K(X_i, x). \tag{9}$$

Figure 1 shows a positive query example (a "3" digit in a "8 vs 3" binary classification task using MNIST data), followed by its top 8 nearest SVs from the training set (those with largest `NNscore`'s being first). The partial KM outputs (g_k) for the first 8 steps using this SV ordering are shown above each SV. The two factors in the `NNscore`'s are shown in the first line of text under each SV (i.e. β_i followed by the kernel value for the query and that SV). [3]

Not surprisingly, 3 of the first 4 nearest SVs are of the query's class. More importantly, the $\beta_i K(X_i, x)$ terms corresponding to the `NNscore`-ordered SVs tend to follow a steady (though somewhat noisy) "two steps forward, one step backward" progression, such that soon the remaining terms become too small to overcome any strong leanings. For example, the second four NSVs in Figure 1 already have considerable smaller $\beta_i K(X_i, x)$ than the first four.

Encouraging and exploiting this phenomena is the key behind our approach. We will classify a query as soon as our pre-query worst-case estimates of how slow this drop off might occur indicate that a strong leaning is

[3] Actually, those kernel values are *approximated* during NSV ordering, to ensure nominal time costs, as described in Section 4.3. The *exact* kernel values (shown in Figure 1 below the approximate kernel values) are computed only as needed, as partial outputs are incrementally computed. Also, the KM bias term (b=0.0322 in this example) accounts for the first partial output (g_1) starting lower than the product of the first SV's β and exact kernel value.

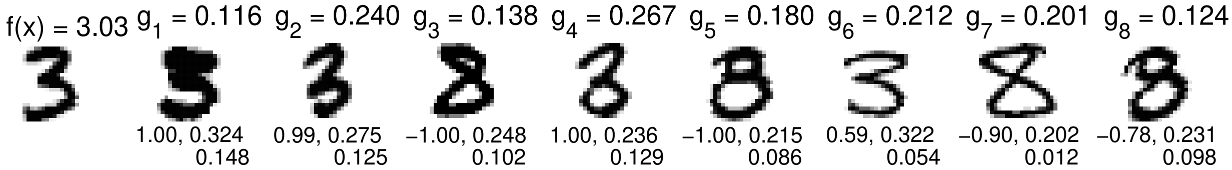

Figure 1. Example of Nearest SVs.

unlikely to be reversed in sign, as the remaining (even lower scoring) NSVs are examined.

Figure 2 summarizes our query-time algorithm. It trades off speedup (m/k) versus fidelity (likelihood of $\text{sign}(g(x)) = \text{sign}(f(x))$) by the choice of upper ($\mathtt{H_k}$) and lower ($\mathtt{L_k}$) thresholds, as the next section describes.

```
Inputs: query x, SVs X_i, weights β_i, bias b,
        and statistical thresholds (L_k,H_k).
Output: g(x), an approximation of f(x).

sort X_i's and β_i's by NNscore_i(x); (large 1st)
g  =  −b;
for k = 1 to m
   g  =  g  +  β_k  K(X_k,x);
   if (g < L_k) or (g > H_k) then stop;
end
```

Figure 2. Pseudo-code for query-time NSV classification.

4.1. Statistical Thresholds for NSV

We derive thresholds $\mathtt{L_k}$ and $\mathtt{H_k}$ by running the algorithm of Figure 2 over a large representative sample of pre-query data and gathering statistics concerning the partial outputs (g). A reasonable starting point is to include the entire training set X (not just the SVs) in this sample. Section 5.2 will discuss refinements.

Let $g_k(x)$ denote $g(x)$ after k steps. A natural initial approach to thresholding (denoted `Simple`) is to compute $\mathtt{L_k}$ as the minimum value of $g_k(x)$ over all x such that $g_k(x) < 0$ and $f(x) > 0$. This identifies $\mathtt{L_k}$ as the worst-case wrong-way leaning of any sample that the exact KM classifies as positive. Similarly, $\mathtt{H_k}$ is assigned the maximum $g_k(x)$ such that $g_k(x) > 0$ and $f(x) < 0$.

When the query data is very similar to the training data, these `Simple` thresholds will often suffice to classify queries the same as the exact KM (but faster). However, in practice, the test and training data distributions will not be identical. Therefore, we introduce a second method (denoted `MaxSmoothed`) which is based on outwardly adjusting the thresholds to conservatively account for some local variance in the extrema of the $g_k(x)$ around each step k.

Specifically, we replace each $\mathtt{H_k}$ ($\mathtt{L_k}$) with the maximum (minimum) of all threshold values over adjacent steps $k-w$ through $k+w$. In our experiments in Section 6, we used a smoothing window of w=10. Our analysis suggests this was more conservative than required to avoid introducing test errors. However, experiments with narrower windows (e,g, w=2, giving slightly tighter thresholds) did not yield much additional speedups, suggesting that a moderate window size such as w=10 (relative to 1000's of SVs) is probably usually prudent. In any case, we assume that in practice an appropriate method of smoothing (e.g. choice of w) would be selected via some sort of pre-query cross-validation process, to see what works best for a given task and data set.

4.2. "Tug of War": Balancing SV^+ vs SV^-

Although the above approach suffices to provide some query speedups without introducing significant test errors, we have noticed that its speedups can be significantly suboptimal. This is because sorting NSVs solely by $\mathtt{NNscore_i}$(x) in the algorithm of Figure 2 leads to relatively wide and skewed thresholds whenever there is even a slight imbalance in the number of positive SVs (i.e. set SV^+, with $\beta_i > 0$) versus negative SVs (i.e. SV^-, with $\beta_i < 0$). For example, since an SVM constrains the sum of β over SV^+ to be the same as that over SV^-, the β values for the smaller set will be proportionally larger, making early (small k) leanings of $g_k(x)$ tend towards the class with fewer SVs. This results in thresholds that are skewed and much larger than desired. In particular, we desire all but the earliest (strongest scoring) NSVs to effectively cancel each other, and thus make it unnecessary to explicit touch them during queries. However, such skewed leanings make that especially difficult to achieve.

To overcome this problem, we replace the simple NSV sort with an ordering we call "tug of war". This involves adjusting the `NNscore`-based ordering so that the cumulative sums of the positive β and the negative β at each step k are as equal as possible. This often results in orderings which alternate between the top scoring positive and negative SVs, though not always – especially at later steps (large k), when β values are smaller and more widely varying.

4.3. Fast Approximate NSV Ordering

Metric (e.g. vp-trees, (Yianilos, 1998)) and spatial (e.g. kd-trees) indexing methods are often employed to avoid the expensive of full linear scans in nearest-neighbor search. However, for the high-dimensional data targeted by this work, we find such indexing methods to often be practically useless. For example, the distribution of kernel distances using a degree 9 polynomial kernel on the MNIST data is so narrow that, for instance, triangular inequalities used by metric trees prune almost no neighbor candidates. That is, the minimum (non-zero) distance is greater than half the maximum distance, forcing full linear scans.

Instead, we find that simple (though approximate) low-dimensional embeddings work much better on such data. Specifically, we do pre-query principal component analysis (PCA) on the matrix of SVs:

$$U\ S\ V^T \quad = \quad X. \tag{10}$$

The k-dimensional embeddings of the SVs (pre-query) and the query x (during query) are given by projection:

$$X^{(k)} = V(:,1:k)^T\ X, \quad x^{(k)} = V(:,1:k)^T\ x. \tag{11}$$

We use these small k-dimensional vectors, instead of the much larger original d-dimensional ones, to quickly compute (approximate) kernels and to approximately order NSVs for each Q as needed. When $k \ll d$, the cost of NSV ordering becomes an insignificant part of the overall cost. For example, our empirical results indicate that even k=20 (denoted as PCA(20)), for MNIST images with d=784, gives sufficiently accurate NSV orderings, incuring a query-time projection overhead equivalent to computing only about 20 dot-product kernels (i.e. touching 20 SVs) per query.

It is also useful to note that using excessively approximate kernels (e.g. small PCA embedding dimension) will not make our threshold approach "unsafe" – it will just require more query time steps because the `MaxSmoothed` thresholds will be proportionally wider. So, as with the threshold smoothing method, we assume in practice that pre-query cross-validation would select appropriate levels and methods of approximation. Indeed, a wide assortment of other existing approximate nearest-neighbors methods would be compatible with our approach and worth consideration. In particular, we suspect promising future work would be to use FastMAP (Faloutsos & Lin, 1995) to approximate kernel PCA (Schölkopf et al., 1999), exploiting FastMAP's ability to quickly find good low-dimensional nonlinear embeddings. This would improve the kernel approximations we currently get using linear PCA (which really just approximates the dot products themselves, and then squashes those approximate dot products using the kernel function).

5. Enhancements

Several enhancements to the above basic approach can be useful, as described below.

5.1. Linear Methods as Initial Filters

To directly attempt to achieve the speed of linear methods, we find it useful to train (pre-query) a linear SVM and use it as an initial filter (with an overhead equivalent to touching just one SV), using (8). This can be done by computing upper and lower `Simple` thresholds as before (in Sectionsect:statsthresholds), except using the linear SVM's output as the "partial output" $g_1(x)$, for the one and only step, and with $f(x)$ still being the main (nonlinear) KM's output.

However, to avoid outlier training examples from dominating the thresholds for the linear SVM, we find it better to compute the high threshold as the mean linear SVM output (for positive-leaning negative examples) plus 3 standard deviations, and, similarly, the low threshold being the mean output for negative-leaning positive examples minus 3 standard derivations, whenever that gives tighter bounds. When the training set is larger or more varied than the expected query set (as seems true for MNIST), this seems reasonable (and we note it does work well for MNIST). In any case, as mentioned earlier, we believe all such choices must ultimately be based on some pre-query search process.

As reported in Section 6, this linear filtering typically boosts our amortized speedups on the MNIST tasks by an additional factor of 3 or more, without introducing any new test errors.

5.2. Better Thresholds via Data Generation

Perhaps the main concern with our approach is that it could potentially introduce large numbers of test errors (with respect to the exact KM's classifications) if the queries does not fall within the statistical extremas identified for the "representative" sample (e.g. training) set used to compute the thresholds.

To address this concern, we have begun considering methods that generate additional data for which there are reasons to believe both: 1) the current thresholds would likely be insufficient and 2) they could plausibly occur in actual query sets. One promising method involves generating data which falls within the convex hull of the training data (i.e. plausibility) and occurs close to a large cluster of SVs from one class and yet is on the other side of the discriminant border (i.e. challenging the sufficiency of the current thresholds).

However, none of the tasks we have explored so far have ultimately required this level of care, so we omit details here, except to note that this would seem to be an important area for future work, ensuring wider and safer applicability of our NSV approach.

Nevertheless, we suspect that query-time checks – such as running the exact KM machine on, say, a random 1% of a large query set and comparing them to those based on early stopping using our NSV method – might provide reasonable diagnostics for detecting in practice when the current set of thresholds is inappropriate to the given query set. Even better would be checks with probability proportional to dissimilarity of a query to the training data as a whole. In short, although the issue of training and test data distributions being "significantly different" is a classic and well-known issue in machine learning, approaches such as NSV provide additional motivations (and contexts) to seriously explore this issue in future work.

5.3. Nearest Reduced Set Neighbors

Using reduced sets (summarized in Section 3) in place of the SVs also seems to be a promising direction for future work to improve NSV. In principal, the compression-based speedups from reduced sets are largely disjoint from NSV's speedups from ordering SVs by weighted similarity to the query. However, our initial attempts to exploit reduced sets did not seem to speedup NSV much. Reduced sets involve a costly global optimzation and our current simple greedy methods yield vectors which are not very orthogonal (i.e. ideally kernel values between reduced set vectors would be near 0, but ours rarely are). We suspect nearly-orthogonal reduced set vectors could avoid a key limitation of our current approach, which is that clusters of nearly-identical SVs each must be touched during NSV if they are near the query.

Fortunately, for our MNIST experiments, this does not seem to be common, as is reflected in the narrow distribution of kernel distances that makes nearest-neighbor indexing methods degrade on that domain (for highly nonlinear kernels), as mentioned earlier.

6. Experiments

For reproducibility and comparison to other work, we report on pairwise digit classification tasks using the well-known benchmark MNIST data set (LeCun, 2000). This data has very high input dimensionality (d) and large numbers of SVs (m), typical of the sort of challenging tasks our approach is intended to help.

Table 1 details 3 of the 45 pairwise MNIST cases. Rows labeled 1-2 summarize input dimension and number of positive and negative training examples. [4] Rows 3-6 summarize the trained SVMs for each case.

Row 10 notes the number of test examples (out of MNIST's 10,000 total), over both digit classes. Row 11 indicates the test errors using exact SVM classification, while row 12 similarly reports tests errors using an exact linear SVM (which is generally worse). [5]

Row 21 indicates that our NSV method never disagreed with the classifications of the exact SVM for these 3 cases. Row 22 illustrates that linear filtering alone typically pruned 50-90% of the test queries.

Rows 23-24 shows statistics on how many steps k were required by NSV per query. Row 25 computes the speedup, relative to exact classification.

Rows 31-35 similarly report without linear filtering, indicating that filtering improves by factors of 3-4.

Rows 40-45 similarly reports when using *all* 10,000 test examples (with linear filtering). Speedup is generally much less (though still significant) in this case, apparently because many test digits which are not of either pairwise class will tend to get small partial and exact KM outputs and thus will seldom lean strongly enough and early enough for our current NSV approach to exploit. This suggests that future research is required if NSV is to hope to achieve the same sort of large speedups on multi-class classification (e.g. using (Platt, 1999)) that it seems to enjoy on binary ones.

Figure 3 plots all 1984 queries for the "8 vs 3" case (without using linear filtering), with exact SVM outputs ($f(x)$) on the y-axis and steps k, at which each query's partial output ($g_k(x)$) exceeds the thresholds, on the x-axis. It illustrates "proportionality to difficulty" – queries requiring the largest k tend to have $f(x) \sim 0$. Also, the solid lines are `MaxSmoothed`'s thresholds. Note that some queries plot within those

[4] To enable many experiments, we trained SVMs with only the first 4,000 digits of about 12,000 per case (given 10 similar-sized digit class sets and 60,000 training digits).

[5] It is well-known (and we verified) that fixed-k NN does not work nearly as well as SVMs for MNIST. So, NSV's good performance here is not matched by simpler k-NN.

Table 1. Details of some MNIST experiments.

		8 vs 3	0 vs 1	2 vs 5
1	d	784	784	784
2	n^+,n^-	1994, 2006	2181, 1819	1944, 2056
3	kernel (normed)	$(u \cdot v)^9$	$(u \cdot v)^9$	$(u \cdot v)^9$
4	C	1	1	1
5	m SVs $(\alpha = C)$	1753 (36)	789 (25)	1840 (11)
6	$\lvert SV^+\rvert$, $\lvert SV^-\rvert$	849, 904	191, 598	951, 889
10	test examples	1984	2115	1924
	test errors:			
11	kernel SVM	12	10	5
12	linear SVM	37	2	24
	w/ linear filter:			
21	NSV disagrees	0	0	0
22	filter skips	1402	1766	1687
23	k min,max	1, 1478	1, 435	1, 999
24	k mean,median	132.2, 1.0	14.6, 1.0	12.2, 1.0
25	m/k speedup	**13.3**	**54.2**	**150.2**
	w/o linear filter:			
31	NSV disagrees	0	0	0
33	k min,max	4, 1477	5, 434	1, 998
34	k mean,median	453.9, 400.5	62.6, 53.0	37.0, 20.0
35	m/k speedup	**3.9**	**12.6**	**49.7**
40	all test data	10000	10000	10000
42	filter skips	3795	7108	2912
43	k min,max	1, 1703	1, 566	1, 1134
44	k mean,median	407.4, 330.0	35.2, 1.0	211.0, 42.0
45	m/k speedup	**4.3**	**22.4**	**8.7**

solid lines, indicating that their $g_k(x)$ leaned outside $[\mathtt{L_k},\mathtt{H_k}]$, even though $f(x)$ does not.

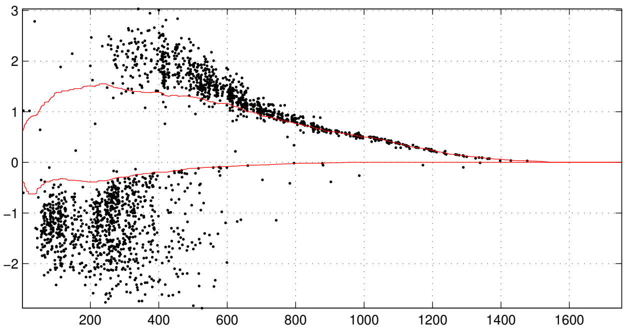

Figure 3. Example proportionality to difficulty ($f(x)$ vs k).

We initially experimented on just two cases – "3 vs 8" (hard) and "0 vs 1" (easy) – to develop our method and fix our design choices (such as `MaxSmoothed`'s window size (w=10) and the sufficiency of PCA(20) for fast NSV ordering). This minimizes concerns of overfitting design choices to this data, enabling us to test overall robustness of our method by then running experiments over all 45 pairwise classifiers (using linear filtering and test queries only from the pairwise classes). The speedup results are summarized in Table 2, with a mean m/k speedup of 111 per query over all 45 cases. Table 3 shows that this large speedup was achieved with only 6 disagreements. Investigation showed that the 2 disagreements on "4 vs 9" are a wash (NSV is correct on one and wrong on the other).

For all 4 other disagreements, NSV was correct, resulting in NSV achieving 4 *less* test errors than the exact SVMs. Although we cannot claim, nor expect, that this will usually happen, it is encouraging and interesting – especially given that some of the very best speedups achieved in related reduced set work have typically came at some small, but significant, price (e.g. 22-fold with 10% more test errors, in (Burges & Schölkopf, 1997)). [6] This result seems likely to be a consequence of the fact that, unlike reduced set methods, our NSV approach tunes under the (more aggressive) assumption that the available training data is very representative of the test data. Thus, we would expect reduced set approaches to likely better ours when this assumption does not hold well (and cannot be easily detected at query-time).

	0	1	2	3	4	5	6	7	8	9
0		**54**	**18**	**66**	**34**	**41**	**16**	**333**	**35**	**160**
1	*789*		**70**	**27**	**26**	**31**	**49**	**23**	**28**	**19**
2	*1560*	*1070*		**22**	**47**	**150**	**47**	**25**	**13**	**57**
3	*1389*	*905*	*1752*		**355**	**20**	**988**	**86**	**13**	**49**
4	*1552*	*1034*	*1794*	*1615*		**250**	**73**	**53**	**31**	**9**
5	*1686*	*1096*	*1840*	*1929*	*1887*		**25**	**145**	**14**	**50**
6	*1474*	*872*	*1585*	*1407*	*1631*	*1739*		**877**	**93**	**402**
7	*1273*	*815*	*1596*	*1404*	*1655*	*1601*	*1302*		**83**	**9**
8	*1420*	*910*	*1783*	*1754*	*1703*	*1869*	*1477*	*1414*		**24**
9	*1269*	*754*	*1498*	*1395*	*2055*	*1659*	*1277*	*1546*	*1492*	

Table 2. 45 pairwise MNIST classifiers. Lower: *SV counts* (m). Upper: Amortized m/k **speedups** over all queries.

	0	1	2	3	4	5	6	7	8	9
0		**0**	**0**	**0**	**0**	**1**	**0**	**0**	**0**	**0**
1	*10*		**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
2	*13*	*14*		**0**	**0**	**0**	**0**	**1**	**0**	**0**
3	*3*	*11*	*15*		**0**	**1**	**0**	**0**	**0**	**0**
4	*2*	*10*	*8*	*4*		**0**	**0**	**0**	**0**	**2**
5	*15*	*11*	*5*	*24*	*1*		**0**	**0**	**0**	**0**
6	*15*	*13*	*8*	*2*	*7*	*20*		**0**	**0**	**0**
7	*4*	*11*	*18*	*11*	*10*	*10*	*1*		**0**	**0**
8	*8*	*11*	*15*	*12*	*7*	*13*	*3*	*7*		**1**
9	*8*	*12*	*17*	*17*	*20*	*20*	*3*	*11*	*17*	

Table 3. Number of *test errors* for each SVM. Number of NSV classification **disagreements** with each SVM.

6.1. Implementation Issues

For common dot-product kernels and high d, the cost of dotting each query point against all of the SVs dominates computation of exact KM outputs. When classifying large numbers of querys at once, these dot products can be computed using optimized matrix-multiply code (e.g. cache-efficient BLAS `dgemm` from ATLAS

[6] *However*, we must note that such reduced set results cannot be directly and fully compared to our new results, for various reasons, including that they performed full 10-way classification, and used 10 "one vs rest" – instead of the 45 pairwise binary classifications we explored here.

(Whaley & Dongarra, 1997)), often being 3-8x faster (depending on platform) than a naive matrix-multiply.

Our NSV approach greatly reduces the number of SVs that each query point must dot against on average, *but*, because the order of the SVs is unique to most every query point, it is difficult to be as efficient as ATLAS. Projecting each query point using PCA and sorting its NSV orderings also adds some overhead.

As a result, the best speedups, measured in actual clock time, that we have achieved so far are not yet as impressive as NSV's potential, though they are $\mathcal{O}((m/\mathrm{k}))$. Using modern cache-prefetch instructions and more carefully blocked computations, we suspect much of our existing inefficiencies are avoidable. For now, we must note in fairness that, in the worst-case, NSV speedups may necessarily be (for current memory cache hardware) 3-8x less than the ideal factor of m/k over exact KM computations.

7. Discussion

We have proposed a method which essentially treats a kernel machine at query-time as an improved k nearest-neighbor method. Empirical work indicates that it can tolerate quite approximate (i.e. very fast) nearest-neighbor computations and effectively automatically pick an appropriate k at query-time (for each query), guided by pre-query analysis of the KM's performance on representative data (e.g. large train set).

It is important to note that this approach is applicable to *any* form of KM classifier, regardless of the way it is trained (e.g. KFD (Mika et al., 2001), MPM (Lanckriet et al., 2002), PSVM (Fung & Mangasarian, 2001), etc.) – including both linear programming and quadratic programming approaches.

We report some promising and exciting speedups observed to date (including 111-fold average speedup across 45 pairwise MNIST digit classification tasks), offering a compelling existence proof that this idea has merit for at least some challenging tasks. Nevertheless, further and deeper understanding of the practical and theoretical limitations and tradeoffs between tighting thresholds and introducing additional test errors is required. We believe that a particularly promising future direction to address this central issue involves finding (recursive) partitionings of input space, within which different thresholds might be appropriate. This might involve applying our variant of the nearest-neighbors method at the leaves of some sort of decision tree structure. This work thus motivates and positions us towards further study to combine three distinct and important threads of machine learning methods: kernel machines, nearest-neighbors, and decision trees.

References

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. *Proceedings of ICML-02.*

DeCoste, D. (2003). Anytime query-tuned kernel machines via cholesky factorization. *Proceedings of SIAM International Conference on Data Mining (SIAMDM-03).*

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

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.*

Fung, G., & Mangasarian, O. (2001). Proximal support vector machine classifiers. *Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining.*

Lanckriet, G., Ghaoui, L. E., Bhattacharyya, C., & Jordan, M. I. (2002). Minmax probability machine. *Advances in Neural Information Processing Systems (NIPS) 14.*

LeCun, Y. (2000). MNIST handwritten digits dataset. http://www.research.att.com/~yann/ocr/mnist/.

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

Platt, J. (1999). Large margin dags for multiclass classification. *NIPS 11.*

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.

Whaley, R. C., & Dongarra, J. J. (1997). *Automatically tuned linear algebra software* (Technical Report UT-CS-97-366).

Yianilos, P. N. (1998). *Excluded middle vantage point forests for nearest neighbor search* (Technical Report). NEC Research Institute, Princeton, NJ.