Compiling Support Vector Machines for Efficient Classification

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

Abstract

The time cost of classifying examples with support vector machines is traditionally linear in the number (L) of support vectors, since each example computes dot products with each SV. This presents a critical challenge for applying SVMs to large-scale problems, since L is often large (e.g. 5% or more of the training data). This paper introduces techniques for "compiling" a SVM classifier, such that the cost of computing an output becomes proportional to the difficulty of classifying the given example. An example far from the decision surface requires few dot products whereas a close example may require L dot products. Our compiled SVMs achieve order-of-magnitudes amortized speedups while still guaranting the same classifications as the original SVM. The related "reduced set" method (Burges, 1996) similarly lowers the effective L, but provides neither proportionality with difficulty nor guaranteed preservation of classifications.

1. Introduction

Much recent work has shown support vector machines (SVMs) consistently achieving among the lowest test error rates across a wide variety of practical applications (e.g. (DeCoste & Schölkopf, 2001; Guyon, 2000; Schoelkopf et al., 1999)). Also, recent decomposition methods for SVM training such as SMO (Platt, 1999; Keerthi et al., 1999) and SVM^{light} (Joachims, 1999), have demonstrated significant gains in scaling SVMs to handle large real-world classification tasks. The time costs of those methods appear to typically scale $O(\ell^2)$, where ℓ is the number of training examples — whereas direct application of general quadratic programming optimizers often scales $O(\ell^3)$.

Despite all this progress, wide-spread use of SVMs in practical large-scale applications has been hindered by three key obstacles:

1. test-time is typically $O(\ell)$ — classifying each test example requires L SVM kernel evaluations (e.g. dot products), one with each of the L support vectors, and L is typically a large fraction of ℓ (e.g. 5% or more),
2. training-time scales super-linear (i.e. $O(\ell^2)$),
3. efficient model selection (e.g. optimal kernel and regularization parameters) is still an open issue.

The first obstacle is often most critical for many real-world applications, including CPU and memory limited onboard spacecraft applications that have motivated our work. For example, a SVM has recently achieved the best generalization performance (DeCoste & Schölkopf, 2001) on the well-known benchmark MNIST digit recognition task (LeCun, 2000), but its test-time complexity is orders of magnitude higher than the previous best (a convolution neural network) due to the large numbers of SVs for each binary digit recognizer (around 20,000) and high dimensionality (784 pixels per image).

This paper focuses on overcoming that first obstacle. We also believe that embedding our solution into the training and model selection processes would also significantly help address those remaining two obstacles.

We exploit geometric properties of the SVM to reduce the average number of kernel evaluations required to classify a test example — by as much as a factor of L under favorable conditions. We demonstrate the effectiveness of our approach on illustrative data.

2. Support Vector Machines

This section summarizes the relevant SVM background and then gives a reformulation in terms of kernel distances that we will employ. Given an ℓ-by-D data matrix (X), an ℓ-by-1 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:

minimize:

$$\frac{1}{2}\sum_{i,j=1}^{\ell}\alpha_i\alpha_j y_i y_j K(x_i,x_j) - \sum_{i=1}^{\ell}\alpha_i$$

subject to:

$$0 \le \alpha_i \le C,\ \sum_{i=1}^{\ell}\alpha_i y_i = 0,$$

where ℓ is the number of training examples, y_i is the label (+1 for positive example, -1 for negative) for the i-th training example (x_i), and $K(x_i,x_j)$ denotes the value of the kernel function for i-th and j-th examples.

The kernel $K(x_i,x_j)$ implicitly projects the two given examples from D-dimensional input space into some (possibly infinite) feature space and returns their dot product in that feature space. That is, it computes

$$K(x_i,x_j) \equiv \phi(x_i)\cdot\phi(x_j), \tag{1}$$

for some mapping function ϕ, but without explicitly computing the coordinates of the projected vectors. In this way, kernels allow large non-linear feature spaces to be explored while avoiding curse of dimensionality.

The simplest is the *linear* kernel, implemented as a simple dot product:

$$K(u,v) = u\cdot v \equiv \sum_{i=1}^{d} u_i\cdot v_i. \tag{2}$$

The *polynomial* kernel is defined by a non-linearly squashed dot product of the following form:

$$K(u,v) = (u\cdot v + r)^d, \tag{3}$$

with polynomial degree parameter d. Varying the continuous offset parameter r changes the relative weighting of the (implicit) terms in the non-linear polynomial feature space. One of the most popular kernels is the *radial basis function* (RBF) non-linear kernel:

$$K(u,v) = e^{\frac{-||u-v||^2}{2\sigma^2}}, \tag{4}$$

with variance parameter σ, giving another non-linear squash of the dot product of the two examples. [1]

2.1 Standard Formulation of SVM Outputs

The SVM output classification, for any given example x, can be computed as:

$$F(x) = sign(G(x) - b), \tag{5}$$

$$G(x) = \sum_{i=1}^{\ell}\alpha_i y_i K(x,x_i), \tag{6}$$

where vector of alphas α (of length ℓ) are the variables determined by the above QP optimization problem.

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

Let SV^+ represent the set of positive *support vector* examples (for which $0 < \alpha_i \le C$) and SV^- represent the set of negative SV examples (for which $0 < \alpha_i \le C$). Similarly, define their corresponding "in-bounds" subsets IN^+ and IN^-, for which $0 < \alpha_i < C$. The scalar bias (b) is typically chosen as midway between the mean of G over IN^+ and the mean of G over IN^-.

2.2 Kernel Distances

The (Euclidian) distance between examples x_i and x_j in the feature space of the kernel is, by definition:

$$d_{ij} \equiv dist(\phi(x_i),\phi(x_j)) \equiv \sqrt{(||\phi(x_i)-\phi(x_j)||^2)}. \tag{7}$$

This can be computed directly from kernel values:

$$d_{ij} \equiv \sqrt{K_{ii} - 2K_{ij} + K_{jj}}. \tag{8}$$

More generally, for any two kernel points `U` and `V` defined from sets of D-dimensional (input space) vectors:

$$\mathtt{U} = \sum_{u_i\in U}\alpha_i\phi(u_i)\ ,\quad \mathtt{V} = \sum_{v_j\in V}\beta_j\phi(v_j) \tag{9}$$

the kernel distance is defined as:

$$d_{\mathtt{UV}} \equiv dist(\mathtt{U},\mathtt{V}) \equiv \sqrt{||\mathtt{U}-\mathtt{V}||^2} \tag{10}$$

This distance can be computed from kernel values:

$$d^2_{\mathtt{UV}} = \sum_{i,j}\alpha_i\alpha_j K(u_i,u_j) - 2\sum_{i,j}\alpha_i\beta_j K(u_i,v_j) - \sum_{i,j}\beta_i\beta_j K(v_i,v_j) \tag{11}$$

2.3 Distance-Based SVM Outputs

A SVM defines a linear discriminate 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. For a SVM, these two points can be defined in terms of the two sets SV^+ and SV^-.

Let $\mathtt{Q} = \phi(x)$ be the point in kernel space to which a given query example x (implicitly) projects and let the two points defining the SVM hyperplane be:

$$\mathtt{P} \equiv \sum_{x_i\in SV^+}\alpha_i^+\phi(x_i)\ ,\quad \mathtt{N} \equiv \sum_{x_i\in SV^-}\alpha_i^-\phi(x_i), \tag{12}$$

where α^+ are the alphas of the positive SVs and α^- are the alphas of the negative SVs.

It turns out that normalizing sets α^+ and α^- such that their sums are each 1 is very useful in practice. This ensures that neither `P` nor `N` are outside the convex hull of the examples of their respective classes and helps

keep distance values smaller and within a better range for numeric stability. [2] Since the sums of the two alpha sets are constrained to be equal in the QP formulation (via $\sum_{i=1}^{\ell} \alpha_i y_i = 0$), this normalization is simply:

$$s = \sum_{i=1}^{i=\ell} \alpha_i^+ \ , \quad \alpha_i^+ := 2\alpha_i^+/s \ , \quad \alpha_i^- := 2\alpha_i^-/s, \quad (13)$$

The SVM classifications $F(x)$ can be computed via the following alternative definition of $G(x)$, based on the difference of squared distances from Q to N and to P:

$$G(x) = \frac{1}{2}[g(x)\ s - \frac{c}{s}] \quad (14)$$

where s is the above alpha normalization factor,

$$c = \sum_{x_i,x_j \in SV^-} \alpha_i \alpha_j K(x_i, x_j) \ - \sum_{x_i,x_j \in SV^+} \alpha_i \alpha_j K(x_i, x_j) \quad (15)$$

is a constant which can be precomputed from the original SVM (and before normalizing the alphas), and

$$\boxed{g(x) = d^2_{\mathtt{QN}} - d^2_{\mathtt{QP}}.} \quad (16)$$

This distance-based reformulation for computing SVM outputs allows us to employ new geometric methods to compute bounds on $g(x)$, from which bounds on $G(x)$ are directly computed using (14).

The above relation (i.e. (14)) between $G(x)$ and $g(x)$ is seen by comparing the following restatements of (6):

$$G(x) = \sum_{x_i \in SV^+} \alpha_i K(x, x_i) - \sum_{x_i \in SV^-} \alpha_i K(x, x_i) \quad (17)$$

and of (16) (using (11)):

$$g(x) = K(x,x) - 2\sum_{x_i \in SV^-} \alpha_i^- K(x, x_i) + \sum_{x_i,x_j \in SV^-} \alpha_i^- \alpha_j^- K(x_i, x_j) \quad (18)$$

$$-[K(x,x) - 2\sum_{x_i \in SV^+} \alpha_i^+ K(x, x_i) + \sum_{x_i,x_j \in SV^+} \alpha_i^+ \alpha_j^+ K(x_i, x_j)]. \quad (19)$$

3. Classification Using $\widehat{\mathtt{P}}$,$\widehat{\mathtt{N}}$

Since the high cost of SVM classification arises due to large L (i.e. large sets SV^- and SV^+), lower cost can be achieved by instead using two points $\widehat{\mathtt{P}}$ and $\widehat{\mathtt{N}}$ which involve smaller sets but are still relatively close to P and N respectively in kernel feature space.

[2] In fact, due to geometric properties of SVMs, this alpha normalization makes P and N the closest points between those two convex hulls (Bennett & Bredensteiner, 2000).

Define approximations of P and N respectively as

$$\widehat{\mathtt{P}} \equiv \sum_{z_i \in Z^+} \beta_i^+ \phi(z_i) \ , \quad \widehat{\mathtt{N}} \equiv \sum_{z_i \in Z^-} \beta_i^- \phi(z_i). \quad (20)$$

In this section we assume vectors β^+ and β^- and sets Z^+ and Z^- (containing D-dimensional vectors) defining $\widehat{\mathtt{P}}$ and $\widehat{\mathtt{N}}$ are given. In the next section we discuss search methods to find good ones.

Reduced set methods (Burges, 1996) also employ such approximations. However, they give *estimates*

$$g(x) \ \approx \ \tilde{g}(x) = \sum_{z_i \in Z^+} \beta_i K(x, z_i) - \sum_{z_i \in Z^-} \beta_i K(x, z_i), \quad (21)$$

whereas we find *bounds*: $g_L(x) \le g(x) \le g_H(x)$

Computing bounds on g(x) directly gives bounds on the SVM classification output $F(x)$, via (14) and (5), where $F_L(x) \le F(x) \le F_H(x)$:

$$\boxed{F_L = sign(G_L(x) - b) \ , \quad F_H = sign(G_H(x) - b)} \quad (22)$$

$$\boxed{G_L(x) = \tfrac{1}{2}[g_L(x)\ s - \tfrac{c}{s}], \ G_H(x) = \tfrac{1}{2}[g_H(x)\ s - \tfrac{c}{s}]} \quad (23)$$

When F_L=F_H, $\widehat{\mathtt{P}}$ and $\widehat{\mathtt{N}}$ suffice to classify example x.

3.1 Computing Bounds on g(x) for Given $\widehat{\mathtt{P}}$,$\widehat{\mathtt{N}}$

The following summarizes a very fast two-step method for computing optimally-tight bounds on $g(x)$ for given approximations $\widehat{\mathtt{P}}$,$\widehat{\mathtt{N}}$ of P,N.

3.1.1 Step 1: Embed $\widehat{\mathtt{P}}$,$\widehat{\mathtt{N}}$,P,N in 3D

Given the four kernel points $\widehat{\mathtt{P}}$,$\widehat{\mathtt{N}}$,P,N, we first compute all six distances between them (using (11)).

We then embed these four points to respective vectors in three-dimensions:

$$p \equiv <\ 0,\ 0, 0>, \quad n \equiv <n_x,\ 0,\ 0>,$$
$$P \equiv <P_x, P_y, 0>, N \equiv <N_x, N_y, N_z>.$$

The six coordinates $n_x, P_x, P_y, N_x, N_y, Nz$ are obtained by solving the system of equations defining the six distances between these four points:

$$(P_x - N_x)^2 + (P_y - N_y)^2 + N_z^2 = dist(\mathtt{P},\mathtt{N})^2 \quad (24)$$

$$(N_x - n_x)^2 + N_y^2 + N_z^2 = dist(\mathtt{N},\widehat{\mathtt{N}})^2 \quad (25)$$

$$N_x^2 + N_y^2 + N_z^2 = dist(\mathtt{N},\widehat{\mathtt{P}})^2 \quad (26)$$

$$(P_x - n_x)^2 + P_y^2 = dist(\mathtt{P},\widehat{\mathtt{N}})^2 \quad (27)$$

$$P_x^2 + P_y^2 = dist(\mathtt{P},\widehat{\mathtt{P}})^2 \quad (28)$$

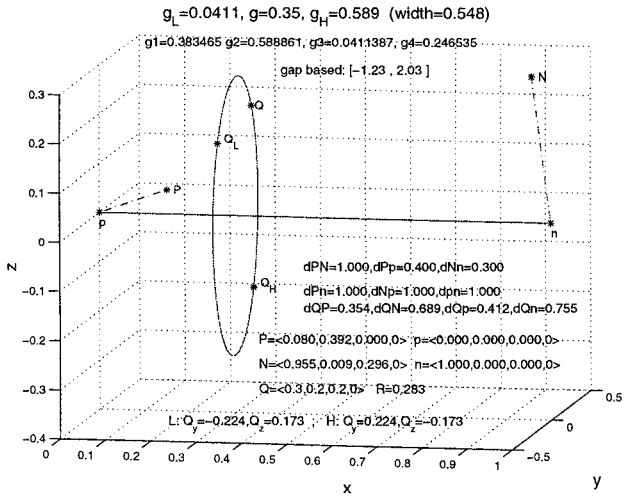

Figure 1. Example embedding of P,N,$\widehat{\text{P}}$, $\widehat{\text{N}}$, Q. Note that naive bounds using $g_l = (d_{Qn} - gap)^2 - (d_{Qp} + gap)^2$ and $g_h = (d_{Qn} + gap)^2 - (d_{Qp} - gap)^2$, where $gap = d_{Pp} + d_{Nn}$, result in an interval $[g_l, g_h] = [-1.23, 2.03]$, for which the sign is unclear. Whereas our interval bound of $[0.0411, 0.589]$ gives a clear sign.

$$n_x^2 = dist(\widehat{\text{P}}, \widehat{\text{N}})^2 \tag{29}$$

Using notational shorthand of $d_{Pp} \equiv dist(\text{P}, \widehat{\text{P}})$, $d_{Np} \equiv dist(\text{N}, \widehat{\text{P}})$, etc., solutions are:

$$\boxed{n_x = d_{pn},\ P_x = \tfrac{1}{2d_{pn}}[d_{Pp}^2 - d_{Pn}^2 + d_{pn}],\ P_y = \sqrt{d_{Pp}^2 - P_x^2}} \tag{30}$$

$$\boxed{N_x = \tfrac{1}{2d_{pn}}[d_{pn}^2 + d_{Np}^2 - d_{Nn}^2],\ N_z = \sqrt{d_{Np}^2 - N_x^2 - N_y^2}} \tag{31}$$

$$\boxed{N_y = \tfrac{1}{-2P_y}[d_{PN}^2 - d_{Np}^2 - {P_x}^2 - {P_y}^2 + 2P_x N_x]} \tag{32}$$

3.1.2 Step 2: Bound $g(x)$ via Optimization

The first step above need only be done once (e.g. before test-time). To compute bounds on $g(x)$ for each example x, we now embed all five points ($\text{Q} = \phi(x)$, along with P,N,$\widehat{\text{P}}$, and $\widehat{\text{N}}$) into a 4D space (with coordinates labeled x,y,z,w).

The vectors P, N, p, n retain their first 3 coordinates as defined in the previous section and have $P_w = N_w = p_w = n_w = 0$. The query point Q embeds into vector $Q \equiv < Q_x, Q_y, Q_z, Q_w >$. A system of four equations arises from the definitions of distances between Q and the other four points:

$$(Q_x - N_x)^2 + (Q_y - N_y)^2 + (Q_z - N_z)^2 + Q_w^2 = d_{QN}^2 \tag{33}$$

$$(Q_x - P_x)^2 + (Q_y - P_y)^2 + Q_z^2 + Q_w^2 = d_{QP}^2 \tag{34}$$

$$(Q_x - n_x)^2 + Q_y^2 + Q_z^2 + Q_w^2 = d_{Qn}^2 \tag{35}$$

$$Q_x^2 + Q_y^2 + Q_z^2 + Q_w^2 = d_{Qp}^2 \tag{36}$$

Solving (35), (36), and (30) for Q_x yields:

$$\boxed{Q_x = \tfrac{1}{2d_{pn}}[d_{Qp}^2 - d_{Qn}^2 + d_{pn}^2].} \tag{37}$$

Unfortunately, unlike the previous 3D case, we cannot directly solve for the remaining three coordinates, because d_{QP} and d_{QN} are also unknown. Indeed, they are the expensive distances we are trying to avoid computing. The radius of the sphere on which Q must lie, given the triangle imposed by the known distances d_{Qp} and d_{Qn} and the anchor points p and n, is given by:

$$R^2 = Q_y^2 + Q_z^2 + Q_w^2 \tag{38}$$

Solving (38) and (36) for R yields

$$\boxed{R^2 = d_{Qp}^2 - Q_x^2.} \tag{39}$$

With five remaining unknowns (Q_y,Q_z,Q_w,d_{QP},d_{QN}), the problem is under-constrained. Q could be anywhere inside or on the circle (of radius R and center at $< Q_x, 0, 0, Q_w >$) shown in Figure 1, depending on Q_w. Due to our choice of embedding, Q_x is fixed, and so the plane of that circle orthogonal to the x-axis.

So, instead, we settle for finding the minimum ($g_L(x)$) and maximum ($g_H(x)$) of function $g(x)$, using function optimization over Q_y, Q_z, Q_w to find the extrema of:

$$g = d_{QN}^2 - d_{QP}^2 \tag{40}$$

It turns out that both extrema occur when Q is exactly on that circle (i.e. $Q_w = 0$), since Q_w factors cancel in g. This also means (via (38)) that Q_y determines Q_z:

$$Q_z^2 = R^2 - Q_y^2 \tag{41}$$

Thus, we only need to optimize over Q_y.

Replacing Q_z^2 with $R^2 - Q_y^2$ for (34) and (33) yields:

$$A \equiv d_{QP}^2 = (Q_x - P_x)^2 + (Q_y - P_y)^2 + (R^2 - Q_y^2) + Q_w^2 \tag{42}$$

$$B \equiv d_{QN}^2 = (Q_x - N_x)^2 + (Q_y - N_y)^2 + B_k + Q_w^2 \tag{43}$$

$$B_k = [sign(Q_z)(R^2 - Q_y^2)^{\frac{1}{2}} - N_z]^2 \tag{44}$$

The gradient of g with respect to Q_y is:

$$\frac{\delta g}{\delta Q_y} = \frac{\delta B}{\delta Q_y} - \frac{\delta A}{\delta Q_y} \tag{45}$$

$$\frac{\delta A}{\delta Q_y} = 2(Q_y - P_y) - 2Q_y = -2P_y \quad (46)$$

$$\frac{\delta B}{\delta Q_y} = 2(Q_y - N_y) + \frac{\delta B_k}{\delta Q_y} \quad (47)$$

Reorganizing B_k to make $\frac{\delta B_k}{\delta Q_y}$ more obvious gives:

$$B_k = (R^2 - Q_y^2) - 2N_z sign(Qz)(R^2 - Q_y^2)^{\frac{1}{2}} + N_z^2 \quad (48)$$

$$\frac{\delta B_k}{\delta Q_y} = -2Q_y + 2N_z Q_y sign(Qz)(R^2 - Q_y^2)^{-\frac{1}{2}} \quad (49)$$

Combining results for $\frac{\delta B}{\delta Q_y}$ and $\frac{\delta A}{\delta Q_y}$ yields:

$$\frac{\delta g}{\delta Q_y} = [2N_z Q_y sign(Q_z)(R^2 - Q_y^2)^{-\frac{1}{2}} - 2N_y] + 2P_y \quad (50)$$

Setting this gradient for g to zero and squaring both sides indicates that extremas occur when:

$$N_z^2 Q_y^2 (R^2 - Q_y^2)^{-1} = (N_y - P_y)^2 \quad (51)$$

Solving (51) for Q_y gives:

$$\boxed{Q_y = \pm\sqrt{\frac{(N_y - P_y)^2 R^2}{N_z^2 + (N_y - P_y)^2}}} \quad (52)$$

and (via (41)):

$$\boxed{Q_z = \pm\sqrt{R^2 - Q_y^2}} \quad (53)$$

Finally, evaluating g (by plugging Q_y and Q_z into the expression for $B - A$), for all four combinations of signs for Q_y and Q_z, includes the minimum $g_L(x)$ and the maximum $g_H(x)$.

The degenerate case, where both $N_y = P_y$ and $N_z = 0$ hold, means P,N,$\widehat{\text{P}}$,$\widehat{\text{N}}$ all embed in a 2D plane and the line connecting P and N is parallel to the (x-axis) line connecting $\widehat{\text{P}}$ and $\widehat{\text{N}}$. For that (rare) fortunate case, $g_L(\text{x}) = g(x) = g_H(x)$, because there are no extrema (i.e. g is identical for all $0 \leq Q_y \leq R$).

This optimization process is very fast, involving no iterations and only basic scalar computations and some square roots. In practice, we have found the time to compute $g_L(x)$ and $g_H(x)$ to be roughly that of one 10-dimensional dot product. [3] Thus, the overhead in computing bounds easily pays for any savings in fewer dot products obtained due to the smaller sets of the $\widehat{\text{P}}$ and $\widehat{\text{N}}$ approximations.

[3] Tests performed on 450 Mhz Sun System Ultra 60.

3.2 Direct Application: Safe Weight-Folding

In the special case of the linear kernel, outputs can be computed efficiently via the "weight folding" trick. First, precompute a D-dimensional weight vector w, such that $w_j = \sum_{i=1}^{n} y_i a_i X_{ij}$. Then compute each output using $G(x) = xw$. This requires only a single dot product, instead of the L required for the general kernel case.

One motivation behind our new techniques was to allow general kernels to achieve speedups approaching that of weight folding whenever possible. As an extreme example, a degree-1 polynomial kernel $K(u, v) = (u \cdot v + 0.01)^1$ behaves very similar to a linear kernel, but the standard SVM output formulation still requires L more work per test example in this case than one would expect is necessary.

Consider approximations $\widehat{\text{P}} = \phi(w^+)$ and $\widehat{\text{N}} = \phi(w^-)$, for D-dimensional vectors w^+ and w^- given by:

$$w_j^+ = \sum_{x_i \in SV+} \alpha_i^+ X_{ij} \ , \quad w_j^- = \sum_{x_i \in SV-} \alpha_i^- X_{ij}.$$

Across a variety of simple randomly-generated problems we have found that these cheap $\widehat{\text{P}}$,$\widehat{\text{N}}$ approximations indeed usually suffice for classifying (i.e. $F_L(x) = F_H(x)$) almost all data using that nearly-linear polynomial degree-1 kernel — including the SVs (which are close to the discriminate hyperplane).

Unfortunately, performance quickly degrades as nonlinearity increases. For polynomial degree 2 kernels, only a small percentage of (far from the hyperplane) examples can typically be classified using this simple $\widehat{\text{P}}$ and $\widehat{\text{N}}$. As expected, in general we must find better approximations, which is the focus of the next section.

4. Finding Effective $\widehat{\text{P}}$,$\widehat{\text{N}}$

The method of the previous section works for any approximation of P and N, including the results from any existing reduced sets methods. Its key contribution is that any classification it confidently outputs (i.e. when $F_L(x) = F_H(x)$) will agree in sign with the original SVM. However, to fully harness that method, one must first find good-yet-cheap approximations $\widehat{\text{P}}$ and $\widehat{\text{N}}$, as explained below.

4.1 Optimizing Beta Weights

One way to improve approximations is simply to tune just the β^+ and β^- values of $\widehat{\text{P}}$ and $\widehat{\text{N}}$. Unlike the α^+ and α^- of the standard SVM QP formulation, there are no constraints on these betas. We consider two ways to define the cost function when optimizing.

4.1.1 Cost Based on Distance

One natural cost function to optimize β^+ is $dist(\mathtt{P}, \widehat{\mathtt{P}})$, computed via (11). As shown in (Schölkopf et al., 1999), optimal betas for this cost function can be directly computed using a matrix inversion:

$$\beta^+ \;=\; (K^z)^{-1}\;\; K^{zx}\;\; \alpha^+, \qquad (54)$$

where K^z is a matrix with elements $K^z_{ij} = \phi(z_i) \cdot \phi(z_i)$ and K^{zx} has $K^{zx}_{ij} = \phi(z_i) \cdot \phi(x_j)$, for all $z_i \in Z^+$ and all $x_j \in SV^+$. Given $\mathtt{N}$ and $\widehat{\mathtt{N}}$, β^- can be similarly optimized.

4.1.2 Cost Based on Bounds-Width

Whereas the above cost treats $\widehat{\mathtt{P}}$ and $\widehat{\mathtt{N}}$ separately, we have developed new methods that optimize them together. This uses cost based on the average width of the bounds on $g(x)$, over some subsample of data. In particular, we use a subsample of the SVs which are closest to the discriminate hyperplane (i.e. select x_i with lowest $|G(x_i) - b|$ computed by SVM training).

The intuition is that what we really want are $\widehat{\mathtt{P}}$ and $\widehat{\mathtt{N}}$ that work together to maximize the chance that any given example x will get $F_L(x) = F_H(x)$. For very close approximations, the distance-based approach will do this well also. However, for Z^+ and Z^- sets small enough to allow massive speedups, the approximation distances will typically still be quite large.

In fact, for classification signs per se, we really only care about the minimizing the width of the bounding intervals that is on the wrong side of zero. So, our cost averages the costs of each selected training sample x_i:

$$\begin{aligned} cost_w(x_i) &= |G_L - b| \text{ if } y_i = +1 \text{ and } G_L - b < 0, \\ cost_w(x_i) &= G_H - b \text{ if } y_i = -1 \text{ and } G_H - b > 0, \\ cost_w(x_i) &= 0 \text{ otherwise.} \end{aligned}$$

Evaluating this cost over samples is efficient because our g(x) bounding method is so fast. For ease of experimentation with cost function variates, we currently use a quasi-Newton method, using finite difference to compute gradients. However, our analytic solution for extrema $g_L(x)$ and $g_H(x)$ should enable future analytic solutions for gradients and Hessians.

Preliminary experiments indicate that optimizing with bounds-width costs may involve more local minima than for distanced-based costs. So, as a heuristic, the first seed values we use for betas during multiple (global) optimizations is the best betas according to distanced-based cost (and possibly small perturbations as well). This provides a useful baseline against which to evaluate the results from other seeds, and ensures that we do no worse than the distance-based approach for any approximation.

4.2 Optimizing Z Vectors

Sets Z^+ and Z^- of $\widehat{\mathtt{P}}$ and $\widehat{\mathtt{N}}$ need not necessarily contain only training examples (as SV^+ and SV^- must). As the reduced set work showed, exploiting this often allows significant additional speedup without additional approximation errors.

For distanced-based cost, the eigenvectors with highest eigenvalues resulting from kernel Principal Component Analysis (PCA) for SV^+ give good Z^+_i for $\widehat{\mathtt{P}}$ (similarly for $\widehat{\mathtt{N}}$), as discussed in (Schölkopf et al., 1999). We then fix Z^+ and Z^- and optimize betas via (54). Further improvement is possible by then optimizing over both Z and β sets at the same time (as discussed in (Burges, 1996)), but we have not explored that yet.

We have focussed instead on exploring our cost based on bounds-width to optimize Z^+ and Z^- together. Due to local minima concerns, using a variety of seed vectors is useful, including all zeros, small random values, support vectors (of the corresponding class), and the weight-folded vectors w^+,w^-.

4.2.1 Greedy Z Construction

We have only explored greedy optimization, where one pair of new vectors (one for Z^+ and one for Z^-) is optimized at a time (and all previous ones are fixed). However, all betas are optimized as well during each iteration of the optimization process.

4.2.2 Greedy SV Selection

A cheaper approach is to only do greedy optimization over sets of SVs (i.e. $Z^+ \subset SV^+$ and $Z^- \subset SV^-$). This discrete optimization will usually provide a less impressive compression ratio (from $\mathtt{P}$ and $\mathtt{N}$ to $\widehat{\mathtt{P}}$ and $\mathtt{N}$), but has the clear advantage that all kernel evaluations are between SVs and have thus probably been cached during SVM training. This can speedup the compilation process by as much as a factor of D. Another advantage is that any kernel values computed during test-time can be reused if the full original SVM is later required, whereas kernel values using general Z vectors will be wasted if no approximations succeed and the full SVM is required.

5. Compiling Sequences of $\widehat{\mathtt{P}}$,$\widehat{\mathtt{N}}$

Any approximation of $\mathtt{P}$ and $\mathtt{N}$ will yield $F_L(x) \neq F_H(x)$ for some x, particularly x close to the SVM decision hyperplane. So, we produce a succession of

approximations, with decreasing approximation error but increasing cost to compute distances. During classification of a test example x, progression along this sequence is stopped as soon as $F_L(x) = F_H(x)$ occurs. This results in proportionality to difficulty: examples farthest from the decision boundary tend to be classified earliest in the sequence.

Furthermore, we have found that maintaining the intersection of the sequence of intervals $[g_L(x), g_H(x)]$ over all $\widehat{\mathtt{P}}$,$\widehat{\mathtt{N}}$ sometimes results in even tighter bounds. This is because sometimes one approximation constrains the upper bound better but another (later) one constrains the lower bound better. Maintaining these interval intersections is a cheap way to partially account for the global geometric constraints among $\phi(x)$, P, N, and the many approximation points.

Taken together, this sequence of approximations provides a "compiled SVM", with the final level being the full original SVM, which is only invoked for x when earlier approximations fail to achieve $F_L(x) = F_H(x)$.

Currently, our sequences consist of 20 approximations (Z^+,Z^- pairs of sizes 1 through 20, or less if too few SVs) followed by the full SVM. Later Z sets are supersets of the previous ones, due to our greedy construction. In this way, we focus on massive speedups for easy examples, and resort to the full SVM for the others. Obviously there are many other types of sequences to consider as well.

It is also useful to note that our bounds provide a cheap anytime approach to classification. Even when $F_L(x) = F_H(x)$ is not yet true, our computed bounds $[G_L - b, G_H - b]$ gives us a solid basis for guessing the most likely class cheaply (i.e. which ever side of zero is wider). In fact, we have found that when the width of one side of zero is larger than four times the other side, the classification very seldom flips. This is probably related to why reduced set estimates have been reported to usually introduce few additional test errors. Our bounding methods provide a principled basis for doing even more aggressive compression than practical using the reduced set methods, since our approach prevents approximation errors from leading to any disagreements with the original SVM.

6. Example

This paper has focussed on developing the bounding technique of Section 3, but fairly evaluating it requires more optimal (in terms of both speed and finding global optimals) versions of the approximation methods of Section 4. Therefore our empirical work so far is suggestive but preliminary.

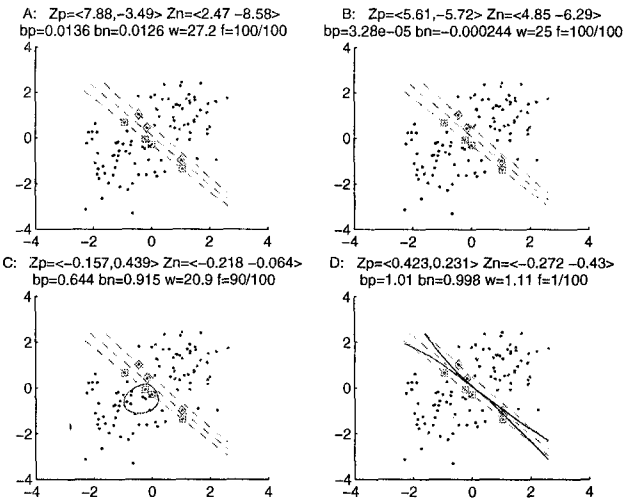

Figure 2. Random 2-dimensional data example. For each subplot, the middle dashed contour line is the original SVM's discriminate (i.e. connecting x for which $G(x) - b = 0$) and the surrounding ones are its margins (where $G(x) - b = \pm 1$, including all $x \in IN^+$ and IN^-). Boxed points are the SVs. Solid contour lines (if visible) indicate where $G_L(x) - b = 0$ or where $G_H(x) - b = 0$.

For ease of visualization, we illustrate our approach in Figure 2 using random 2-dimensional data of 50 positive and 50 negative training examples, generated from two noisy Gaussians centered at $< 1, 1 >$ and $< -1, -1 >$ respectively. For $C = 10$ and polynomial degree 2 kernel, the trained SVM has 3 positive SVs (2 at C) and 4 negative SVs (1 at C).

All 4 cases (A—D) show the performance of the first approximation ($\widehat{\mathtt{P}}$,$\widehat{\mathtt{N}}$) in a compiled SVM sequence. Cases A and B tuned just betas, using distance-based and width-based costs respectively. Both are too weak to classify any examples (e.g. failure (f) counts indicate 100) although B's average bounds width (w) is somewhat smaller. C and D both also tuned the pair of 2-dimensional Z vectors, for distance-based and width-based cost respectively. C fails to classify for all but 10 while D succeeds on all but 1. Thus, case D will achieve about a three-fold speedup (7 SVs compressed to a single pair) for classifying almost any example.

The area between the solid two curves is where the approximations are insufficient (i.e. $F_L(x) \neq F_H(x)$ for x in that region). In the last plot (D), that area is small, indicating that sufficient bounds can usually be computed. In A and B, no solid curves are shown because they are off the plot. For C, only some negative examples can be classified correctly — those 10 inside the region in which the lower bound's solid curve enclosed on itself. Width (w) drops across A through D, reflecting the order of bad to best approximations.

7. Conclusions

We have presented initial methods for compiling SVMs to achieve large amortized classification speedups while guaranting preservation of classifications and having cost proportional to the difficulty of each test example. We exploit the geometry of the SVM kernel space to compute tight bounds on the SVM output using a sequence of approximations, until the sign (class) becomes clear. We are currently preparing tests on large NASA data sets.

It is important to realize that our results apply to other SVMs, regardless of training methods and cost functions used (e.g. linear programming approaches). It further applies to other structures of same form ($f(x) = sign(\sum \alpha_i K(x, x_i) - b)$), such as some classifiers based on large mixture models or neural networks.

This work opens up several new directions for future work. One is to embed these techniques into SVM training itself, to speedup up the cost of checking KKT conditions (which dominates for very large problems). Another is finding cheaper methods for better approximations $\widehat{\mathtt{P}}$ and $\widehat{\mathtt{N}}$ — we have barely scratched the surface so far, especially for our new costs based on bounds-width. Especially useful would be better understandings of how the local minima for bounds-width and distance-based costs differ. Complementary work for reducing the effective D (whereas this work reduces the effective L) would seem particularly useful.

Also, our framework might provide a bridge between SVMs and other methods. For example, one could find approximations $\widehat{\mathtt{P}}$ and $\widehat{\mathtt{N}}$ based on neural networks (e.g. hidden unit weight vectors become Z_i vectors), using them to speed up the test-time performance of SVMs based on tanh kernels.

Finally, compiling SVMs into sequences is only the beginning. We suspect that other structures, such as decision trees, would more effectively adapt bounding computations to different types of test examples.

8. Acknowledgements

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. *Proceedings, 17th Intl. Conf. on Machine Learning.*

Burges, C. J. C. (1996). Simplified support vector decision rules. *Proceedings, 13th Intl. Conf. on Machine Learning* (pp. 71–77). San Mateo, CA: Morgan Kaufmann.

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

DeCoste, D., & Burl, M. (2000). Distortion-invariant recognition via jittered queries. *Computer Vision and Pattern Recognition (CVPR-2000).*

DeCoste, D., & Schölkopf, B. (2001). Training invariance support vector machines. *Machine Learning.* To appear.

DeCoste, D., & Wagstaff, K. (2000). Alpha seeding for support vector machines. *International Conference on Knowledge Discovery and Data Mining (KDD-2000).*

Guyon, I. (2000). Online SVM application list. (See http://www.clopinet.com/isabelle/Projects/SVM/applist.html.).

Joachims, T. (1999). Making large-scale support vector machine learning practical. In *Advances in Kernel Methods: Support Vector Machines* (Schoelkopf et al., 1999).

Keerthi, S., Shevade, S., Bhattacharyya, C., & Murthy, K. (1999). *Improvements to Platt's SMO algorithm for SVM classifier design* (Technical Report CD-99-14). Dept. of Mechanical and Production Engineering, National University of Singapore.

LeCun, Y. (2000). MNIST dataset. Available at www.research.att.com/ ~yann/ ocr/ mnist/.

Platt, J. (1999). Fast training of support vector machines using sequential minimal optimization. In *Advances in Kernel Methods: Support Vector Machines* (Schoelkopf et al., 1999).

Schoelkopf, B., Burges, C., & Smola, A. (1999). *Advances in kernel methods: Support vector machines.* Cambridge, MA: MIT Press.

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* (pp. 124 - 132). Berlin: 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.*