Distributed SVM Learning and Support Vector Reduction

Stephen Winters-Hilt[1,2] * and Kenneth Armond Jr.[1]

[1]Department of Computer Science, University of New Orleans, New Orleans, LA 70148
[2]The Research Institute for Children, Children's Hospital, New Orleans, LA 70118

* corresponding author: winters@cs.uno.edu

Email addresses:
SWH: winters@cs.uno.edu
KA: kcarmond@uno.edu

Abstract

Background
Support Vector Machines (SVMs) are used for a growing number of applications. A fundamental constraint on SVM learning is the management of the training set. This is because the order of computations during the learning process typically goes as the square of the size of the training set. For the 150-component feature data examined here, training sets of 1000 (500 positives and 500 negatives, for example) can be managed on a PC without hard-drive I/O thrashing. Training sets of 10,000 or more, however, can't be managed with a single PC-based resource. For this reason most SVM implementations must contend with some kind of *chunking* process to learn parts of the data at a time.

Results
Two sets of binary SVM training results are examined. These results show that chunk aliasing and outlier accumulation may pose problems for distributed SVM learning. The results also present new methods and how they offer a stable learning solution to these problems at minimal cost. One of those methods extends the learning process with modified alpha-selection heuristics that enable a support-vector reduction phase.

Conclusion
What is not as commonly discussed about *distributed* SVM learning are the details of the distributed, or approximately parallel, chunk processing methods. The distributed SVM described here is implemented using Java RMI, and is developed to run on a network of multi-core processor computers.

Introduction and Background

Support Vector Machines [1] are discriminators that use structural risk minimization to find a decision hyperplane with a maximum margin between separate groupings of feature vectors. When SVMs were created in 1995, a quadratic programming algorithm was used [2]. This was slow and only small datasets could be run with them. In 1998, Platt created sequential minimal optimization (SMO), which is an algorithm that uses Lagrange multipliers to bypass having to use a quadratic algorithm [3]. The SMO SVM iterates through the dataset comparing and updating the Lagrange multipliers (alphas) two at a time. This simplification into smaller steps provides a significant increase in speed. This did much to advance the feasibility and ease-of-use of SVM classifiers. There has still been the key constraint, however, of computing the kernel matrix corresponding to the training data. In the Background sections that follow we introduce (i) the standard binary SVM; (ii) kernel variants; (iii) alpha-selection-variants (including simple chunking); and (iv) previous work with a multiclass SVM formulation. The work with the multiclass formulation is included because it demonstrates the importance of managing and tracking SV's (and other feature vector categories) during the learning process. The importance of tracking the SV's during the learning process will be revisited in the distributed learning results presented in the Results section..

Binary Support Vector Machines

In the binary SVM implementation described in what follows we follow the notation and conventions used previously [4]. Feature vectors are denoted by x_{ik}, where index i labels the feature vectors ($1 = i = M$) and index k labels the N feature vector components ($1 = k = N$). For the binary SVM, labeling of training data is done using label variable $y_i = \pm1$ (with sign according to whether the training instance was from the positive or negative class). For hyperplane separability, elements of the training set must satisfy the following conditions: $w_ß x_{iß} - b = +1$ for i such that $y_i = +1$, and $w_ß x_{iß} - b = -1$ for $y_i = -1$, for some values of the coefficients $w_1,..., w_N$, and b (using the convention of implied sum on repeated Greek indices). This can be written more concisely as: $y_i(w_ß x_{iß} - b) - 1 = 0$. Data points that satisfy the equality in the above are known as "support vectors" (or "active constraints").

Once training is complete, discrimination is based solely on position relative to the discriminating hyperplane: $w_ß x_{iß} - b = 0$. The boundary hyperplanes on the two classes of data are separated by a distance 2/w, known as the "margin," where $w^2 = w_ß w_ß$. By increasing the margin between the separated data as much as possible the optimal separating hyperplane is obtained. In the usual SVM formulation, the goal to maximize w^{-1} is restated as the goal to minimize w^2. The Lagrangian variational formulation then selects an optimum defined at a saddle point of

$$L(\mathrm{w},\mathrm{b};a) = \frac{\mathrm{w}_b \mathrm{w}_b}{2} - a_g y_g (\mathrm{w}_b \mathrm{w}_{gb} - \mathrm{b}) - a_0 \ ,$$

$$\text{where } a_0 = \sum_g a_g \ , \ a_g \geq 0 \quad (1 \leq g \leq M)$$

The saddle point is obtained by minimizing with respect to $\{w_1,...,w_N,b\}$ and maximizing with respect to $\{a_1, ..., a_M\}$. If $y_i(w_ß x_{iß} - b) - 1 = 0$, then maximization on a_i is achieved for $a_i = 0$. If $y_i(w_ß x_{iß} - b) - 1 = 0$, then there is no constraint on a_i. If $y_i(w_ß x_{iß} - b) - 1 < 0$, there is a constraint violation, and a_i ?   8 . If absolute separability is possible, the last case will eventually be eliminated for all a_i, otherwise it is natural to limit the size of a_i by some constant upper bound, i.e., $\max(a_i) = C$, for all i. This is equivalent to another set of inequality constraints with $a_i = C$. Introducing sets of Lagrange multipliers, $?_?$ and $\mu_?(1 = ? = M)$, to achieve this, the Lagrangian becomes:

$$L(\mathrm{w},\mathrm{b};a,x,m)=\frac{\mathrm{w}_b\mathrm{w}_b}{2}-a_g[y_g(\mathrm{w}_b\mathrm{x}_{gb}-\mathrm{b})+x_g]+a_0+x_0C-m_gx_g$$

$$\text{where}\, x_0=\sum_g x_g\ ,\ \ a_0=\sum_g a_g \ \text{ and }\ a_g\ge 0\ \text{ and }\ x_x\ge 0\quad (1\le g\le M)$$

At the variational minimum on the $\{w_1,...,w_N,b\}$ variables, $w_ß = a_?y_?x_{?ß}$, and the Lagrangian simplifies to:

$$L(a)=a_0-\frac{a_d\mathrm{y}_d\mathrm{x}_{db}a_g\mathrm{y}_g\mathrm{x}_{gb}}{2},$$

$$\text{with}\, 0\le a_g\le C\ \ (1\le g\le M)\ \text{and}\ a_gy_g=0\,,$$

where only the variations that maximize in terms of the $a_?$ remain (known as the Wolfe Transformation). In this form the computational task can be greatly simplified. By introducing an expression for the discriminating hyperplane: $f_i = w_ß\, x_{iß} - b = a_?y_?x_{?ß}x_{iß} - b$, the variational solution for L(a) reduces to the following set of relations (known as the Karush-Kuhn-Tucker, or KKT, relations):

(i) $a_i = 0$, $y_if_i = 1$

(ii) $0 < a_i < C$, $y_if_i = 1$

(iii) $a_i = C$, $y_if_i = 1$

When the KKT relations are satisfied for all of the $a_?$ (with $a_?y_? = 0$ maintained) the solution is achieved. The constraint $a_?y_? = 0$ is satisfied for the initial choice of multipliers by setting the a's associated with the positive training instances to 1/N(+) and the a's associated with the negatives to 1/N(-), where N(+) is the number of positives and N(-) is the number of negatives. Once the Wolfe transformation is performed it is apparent that the training data (support vectors in particular, KKT class (ii) above) enter into the Lagrangian solely via the inner product $x_{iß}x_{jß}$. Likewise, the discriminator f_i, and KKT relations, are also dependent on the data solely via the $x_{iß}x_{jß}$ inner product.

Generalization of the SVM formulation to data-dependent inner products other than $x_{iß}x_{jß}$ are possible and are usually formulated in terms of the family of symmetric positive definite functions (reproducing kernels) satisfying Mercer's conditions [1].

The SVM discriminators are trained by solving their KKT relations using the SMO procedure of [5]. The method described here follows the description of [5] and begins by selecting a pair of Lagrange multipliers, $\{a_1,a_2\}$, where at least one of the multipliers has a violation of its associated KKT relations. For simplicity it is assumed in what follows that the multipliers selected are those associated with the first and second feature vectors: $\{x_1,x_2\}$. The SMO procedure then "freezes" variations in all but the two selected Lagrange multipliers, permitting much of the computation to be circumvented by use of analytical reductions:

$$L(a_1, a_2; a_{b \geq 3}) = a_1 + a_2 - \frac{(a_1^2 \mathrm{K}_{11} + a_2^2 \mathrm{K}_{22} + 2a_1 a_2 y_1 y_2 \mathrm{K}_{12})}{2} - a_1 y_1 v_1 - a_2 y_2 v_2$$

$$+ a_{b'} U_{b'} - \frac{a_{b'} a_{y'} y_{b'} K_{b'y'}}{2} ,$$

with ß',?' = 3, and where $K_{ij} = K(x_i, x_j)$, and $v_i = a_{\text{ß}'} y_{\text{ß}'} K_{i\text{ß}'}$ with ß' = 3. Due to the constraint $a_{\text{ß}} y_{\text{ß}} = 0$, we have the relation: $a_1 + sa_2 = -?$, where $? = y_1 a_{\text{ß}'} y_{\text{ß}'}$ with ß' = 3 and $s = y_1 y_2$. Substituting the constraint to eliminate references to a_1, and performing the variation on a_2: $?L\ (a_2\,; a_{\text{ß}'} = 3)/?a_2 = (1 - s) + ?a_2 + s?(K_{11} - K_{22}) + sy_1 v_1 – y_2 v_2$, where $? = (2K_{12} - K_{11} - K_{22})$. Since v_i can be rewritten as $v_i = w_{\text{ß}} x_{i\text{ß}} - a_1 y_1 K_{i1} - a_2 y_2 K_{i2}$, the variational maximum $?L\ (a_2\,; a_{\text{ß}'} = 3)/?a2 = 0$ leads to the following update rule:

$$a_2^{new} = a_2^{old} - \frac{y_2((w_b \mathrm{x}_{1b} - \mathrm{y}_1) - (w_b \mathrm{x}_{2b} - \mathrm{y}_2))}{h}$$

Once a_2^{new} is obtained, the constraint $a_2^{new} = C$ must be re-verified in conjunction with the $a_{\text{ß}} y_{\text{ß}} = 0$ constraint. If the L $(a_2; a_{\text{ß}'} = 3)$ maximization leads to a a_2 new that grows too large, the new a_2 must be "clipped" to the maximum value satisfying the constraints. For example, if y_1 ? y_2, then increases in a_2 are matched by increases in a_1. So, depending on whether a_2 or a_1 is nearer its maximum of C, we have max (a_2) = argmin$\{a_2 + (C - a_2)\,; a_2 + (C - a_1)\}$. Similar arguments provide the following boundary conditions:
(i) if s = -1, max(a_2) = argmin$\{a_2\,; C + a_2 - a_1\}$, and min(a_2) = argmax$\{0\,; a_2 - a_1\}$, and
(ii) if s = +1, max(a_2) = argmin$\{C\,; a_2 + a_1\}$, and min(a_2) = argmax$\{0\,; a_2 + a_1 - C\}$.
In terms of the new $a_2^{new,\ clipped}$, clipped as indicated above if necessary, the new a_1 becomes:

$$a_1^{new} = a_1^{old} + s(a_2^{old} a_2^{new,clipped}) ,$$

where $s = y_1 y_2$ as before. After the new a_1 and a_2 values are obtained there still remains the task of obtaining the new b value. If the new a_1 is not "clipped" then the update must satisfy the non-boundary KKT relation: $y_1 f(x_1) = 1$, i.e., $f^{new}(x_1) - y_1 = 0$. By relating f^{new} to f^{old} the following update on b is obtained:

$$b_1^{new} = b - (f^{new}(x_1) - y_1) - y_1(a_1^{new} - a_1^{old})K_{11} - y_2(a_2^{new,clipped} - a_2^{old})K_{12}$$

If a_1 is clipped but a_2 is not, the above argument holds for the a_2 multiplier and the new b is:

$$b_2^{new} = b - (f^{new}(x_2) - y_2) - y_2(a_2^{new} - a_2^{old})K_{22} - y_1(a_1^{new,clipped} - a_1^{old})K_{12}$$

If both a_1 and a_2 values are clipped then we don’t have a unique solution for b. The Platt convention was to take:

$$b^{new} = \frac{b_1^{new} + b_2^{new}}{2}$$

and this works well much of the time. Alternatively, Keerthi [6] has devised an alternate formulation without this weakness, as have Crammer and Singer [7], with the latter described in the multiclass SVM section. Perhaps just as good as any exact solution for ‘b’ in the double-clipped scenario is to manage this special case by rejecting the update and picking a new pair of alphas to update (in this way only unique ‘b’ updates are made). Alpha-selection variants are briefly discussed in the Section after next.

Kernel Variants

The SVM Kernels of interest are “regularized” distances or divergences, where they are regularized if in the form of an exponential with argument the negative of some distance-measure squared ($d^2(x,y)$) or symmetrized divergence measure ($D(x,y)$), the former if using a geometric heuristic for comparison of feature vectors, the latter if using a distributional heuristic. The Gaussian and Absdiff kernels are regularized distances in the form of an exponential distance measure ($d^2(x,y)$). The Gaussian kernel ($d^2(x,y) = S_k(x_k - y_k)^2$) is common since it tends to produce good results when used with a wide variety of datasets. The Absdiff ($d^2(x,y) = S_k(|x_k - y_k|)^{1/2}$) and Sentropic ($D(x,y) = D(x||y) + D(y||x)$) Kernels [4] tend to work better with all of the datasets considered here and in other tests not shown. The Sentropic kernel is based on a regularized information divergence ($D(x,y)$) instead of a geometric distance.

Alpha-selection Variants and previous Chunking efforts

The SVM algorithm variants are only briefly described here. In the standard Platt SMO algorithm, η=2*K12-K11-K22, and speedup variations are described to avoid calculation of this value entirely. A middle ground is sought with the following definition “η =2*K12-2; If (η >=0) { η = -1;}” (in [4], where underflow handling and other details differ slightly).

SVM chunking provides an alternative method to running a typical SVM on a dataset by breaking up the training data and running the SVM on smaller chunks of data. In the chunking process feature vectors associated with strong data points are retained from chunk to chunk, while weak data points are discarded. See Methods for specifics on the chunking methods employed here.

Zanghirati and Zanni [8] developed the variable projection method (VPM) for training SVMs in parallel. This method is based off of Joachim's SVM light decomposition techniques [9] which delve further into the inner workings of the SMO algorithm [3,5]. First, the feature vector indices are divided into two categories, the free and fixed sets based upon their alphas (Lagrange multipliers). The free set represents the KKT violators which need to be further optimized while the fixed set is the alphas that already fulfill the KKT equations. An alpha variable from each set is used to solve each quadratic sub problem in order to optimize the free set alphas until convergence. VPM provides a parallel solution to computing the kernel matrix which is the most memory intensive part of the SVM. The kernel calculations are spread among several processing elements and the rows of the matrix are spread and usually duplicated across the memory of those processing elements. Since the rows are duplicated, they must be synchronized after each local computation. VPM is implemented using standard C and MPI communication routines.

Hans Peter Graf et al. [10] developed the Cascade SVM to parallelize SVMs. This method begins by breaking the large dataset into chunks. The SVM is run on each separate chunk in the first layer. When the SVMs have all converged, new chunks are created from the resulting support vectors from the pairs of first layer chunks which make up the second layer of chunks. This occurs until a final chunk is reached. The final set of support vectors is then fed back into each first layer chunk. If further optimization is possible and needed, the entire process is rerun until the global optimum is met. This method seems intuitive, but after testing we have found that passing 100% of support vectors down to the next set of chunks, without also passing non support vectors or using the SVR method, typically results in systematic convergence failure (with the various 150-component DNA feature datasets examined). The data run never finishes, in other words, since it cannot sufficiently reduce the support vectors to converge (see Fig. 1). The weakness of the method, not apparent at first sight, is not simply that the SVs from different chunks might be sufficiently different to pose complications. The added subtlety is to prevent the accumulation of outliers during the distributed learning/merging of the SVM with chunking.

Multiclass SVM Methods

Training speedup with large multiclass datasets has been explored in prior work [4]. In those efforts it was found that sometimes convergence itself can be weak purely because time is repeatedly wasted on alpha updates that don't reduce the penalty set (the set of alphas at the max alpha boundary). This result foreshadows developments on SV handling presented in the Results, so it is described in sufficient detail to see how penalty set members are discerned and decisions made accordingly.

SVM-external Multiclass: Binary SVM Decision Tree

The SVM binary discriminator offers high performance and is very robust in the presence of noise. This allows a variety of reductionist multiclass approaches, where each reduction is a binary classification (for classifying cards by suit, maybe classify as red or black first, then as heart or diamond for red and spade or club for black, for example). The SVM Decision Tree is one such approach used extensively with the datasets examined [11], and a collection of them (a SVM Decision Forest) can be used to avoid problems with throughput biasing. Alternatively, the variational formalism can be modified to perform a multi-hyperplane optimization situation for a direct multiclass solution [7,12,13], as is described next.

SVM-Internal Multiclass: Multi-hyperplane Optimization

In [4] we make use of a variant of a formulation by Crammer & Singer [7], where there are 'k' classes and hence 'k' linear decision functions – a description of their approach is given first briefly. For a given input 'x', the output vector corresponds to the output from each of these decision functions. The class of the largest element of the output vector gives the class of 'x'. Each decision function is given by: $f_m(x) = w_m.x + b_m$ for all m = (1,2,…,k). If y_i is the class of the input x_i, then for each input data point, the misclassification error is defined as follows: $\max_m\{f_m(x_i) + 1 – d_i^m\} - f_{yi}(x_i)$, where d_i^m is 1 if m = y_i and 0 if m ? y_i. We add the slack variable $?_i$ where $?_i = 0$ for all i that is proportional to the misclassification error: $\max_m\{f_m(x_i) + 1 – d_i^m\} - f_{yi}(x_i) = ?_i$, hence $f_{yi}(x_i) - f_m(x_i) + d_i^m = 1 - ?_i$ for all i, m. To minimize this classification error and maximize the distance between the hyper-planes (Structural Risk Minimization) we have the following formulation:

Minimize: $? _i?_i + ß(1/2)? _m w_m^T w_m + (1/2)? _m b_m^2$,
where ß > 0 is defined as a regularization constant.

Constraint: $w_{yi}.x_i + b_{yi} - w_m.x_i - b_m - 1 + ?_i + d_i^m = 0$ for all i,m

Note: the term **(1/2) ? $_m$b$_m^2$** is added for de-coupling, 1/ß = C, and $m = y_i$ in the above constraint is consistent with $?_i = 0$. The Lagrangian is:

$$L(w,b,?) = ?\,_i?_i + ß(1/2)?\,_m w_m{}^T w_m + (1/2)?\,_m b_m{}^2$$
$$- ?\,_i?\,_m a_i^m(w_{yi}.x_i + b_{yi} - w_m.x_i - b_m - 1 + ?_i + d_i^m)$$

Where all a_i^ms are positive Lagrange multipliers. Now taking partial derivatives of the Lagrangian and equating them to zero (Saddle Point solution): $?L/??_i = 1 - ?\,_m a_i^m = 0$. This implies that $?\,_m a_i^m = 1$ for all i. $?L/?b_m = b_m + ?\,_i a_i^m - ?\,_i d_i^m = 0$ for all m. Hence $b_m = ?\,_i(d_i^m - a_i^m)$. Similarly: $?L/?w_m = ßw_m + ?\,_i a_i^m x_i - ?\,_i d_i^m x_i = 0$ for all m. Hence $w_m = (1/ ß)[?\,_i(d_i^m - a_i^m)x_i]$ Substituting the above equations into the Lagrangian and after simplification reduces into the dual formalism:

Maximize: $-½?\,_{i,j}?\,_m(d_i^m - a_i^m)(d_j^m - a_j^m)(K_{ij} + ß) - ß?\,_{i,m}d_i^m a_i^m$
Constraint: $0 = a_i^m$, $?\,_m a_i^m = 1$, $i = 1…l$; $m = 1…k$

Where $K_{ij} = x_i.x_j$ is the Kernel generalization.
further details on solving the multiclass system are given in [4]. For our purposes here, however, the last formulation of the problem suffices to describe the categories of KKT violators as SV or boundary support vector (BSV).

SVM-Internal Speedup via differentiating BSVs and SVs

Since the algorithm by Crammer & Singer [7] does not differentiate between SV and BSV, a lot of time is spent in trying to adjust the weights of the BSV i.e. weak data. We briefly examine this in [4], where the modification to the algorithm is shown below. Once the BSV's are identified (as specified by Case III conditions), their weights are no longer adjusted. This results in faster convergence without sacrificing accuracy. For the BSV/SV-tracking speedup, the KKT violators are redefined as:

For all m ? y_i we have:
$a_i^m\{f_{yi} - f_m - 1 + ?_i\} = 0$
Subject to: $1 = a_i^m = 0$; $?\,_m a_i^m = 1$; $?_i = 0$ for all i,m
Where $f_m = (1/ß)[w_m.x_i + b_m]$ for all m

Case I:
If $a_i^m = 0$ for m S.T $f_m = f_m^{max}$
Implies $a_i^{yi} > 0$ and hence $?_i = 0$
Hence $f_{yi} - f_m^{max} - 1 = 0$

Case II:
If $1 > a_i^m > 0$ for m S.T $f_m = f_m^{max}$ and $a_i^{yi} > a_i^m$
Implies $?_i = 0$
Hence $f_{yi} - f_m^{max} - 1 = 0$

Case III:
If $1 = a_i^m > 0$ for m S.T $f_m = f_m^{max}$ and $a_i^{yi} = a_i^m$
Implies $?_i > 0$

Hence $f_{yi} – f_m^{max} – 1 + ?_i = 0$
Or $f_{yi} – f_m^{max} – 1 < 0$

Results

Four sets of experimental Results are presented. The first two sets concern the optimal performance/learning-rate configurations for feature vector passing (“pass-tuning”), where the SVM training is performed using chunking with different learning topologies (sequential, partially-sequential distributed). The experiments are: (1) SV/non-SV pass-tuning on binary subsets of {9AT,9TA,9CG,9GC,8GC}; and, (2) SV/non-SV pass-tuning for binary classification of (9AT,9TA) vs (9CG,9GC).

There appear to be minor instabilities when learning on distributed topologies, and there are a couple of new approaches that appear to be robust in addressing these instabilities. In turn, this allows the hopes of a distributed speedup to be directly realized. One of those new approaches is a post-processing phase, after all KKT violators have been eliminated, during which SV alpha’s near the boundaries are coerced to their boundary sets (i.e., to the polarization set at alpha=0 or the penalty set at alpha=C). This is the subject of the third section: (3) Support Vector Reduction (SVR). The fourth and last section of the Results shows the combined operation of various methods for comparative purposes, and indicates that robust distributed SVM learning may be possible: (4) Distributed SVM with pass-tuning and SVR.

SV/non-SV pass-tuning on binary subsets of {9AT,9TA,9CG,9GC,8GC}: an outlier-management chunking heuristic

For DNA hairpin feature vector datasets, our observations have shown the best kernels to be the Gaussian, Absdiff, and Sentropic kernels (see Background). Of the three kernels indicated, Absdiff and Sentropic produce similar results when measuring accuracy as the average of the Sensitivity (SN) and Specificity (SP), typically significantly outperforming the third best kernel, Gaussian. The Gaussian kernel, on the other hand, is the best performing of the three at keeping the growth in chunk-size as small as possible.

Sequential chunking is a simple form of chunking which is not multi-threaded. This method runs the SVM on the first chunk, and then sends the support feature vectors (SVs) and sometimes non-SVs to be added onto the training data for the next chunk. This continues until the final chunk has been run. When using sequential chunking, feature vector passing can be difficult since passing too many features on to the next chunk can result in training datasets that are too large in the later chunks in the process. For the sequential chunking method, the accuracy of Absdiff (0.898) is shown in Table 1. Sentropic (0.891) produces similar results (see Table A1 in Additional file 1). Gaussian has accuracy 0.864 (see Table A2 in Additional file 2). In these data runs, 100% of the support vector set was passed to the next set of chunks (see Methods). The chunking parameters indicated for the table represent the best accuracy for the given chunking method, and all of the parameter selections are verified for stability (in that minor changes of parameter do not strongly alter classifier accuracy).

For the multi-threaded chunking method, the average accuracy of Sentropic is best (0.855) (see Table 2). Absdiff (0.854) is very similar in performance, and is shown in Table A3 in Additional file 3. Gaussian has average accuracy 0.833 (and is shown in Table A4 in Additional file 4). In these data runs, 30% of the support vector set was passed to the next set of chunks. If 100% SV-passing is attempted there is typically failure to converge. As with the sequential Results, these chunking parameters chosen represent the best accuracy for the given chunking method, and all of the parameter selections are verified for stability.

The SVs in the final distributed chunk with Gaussian kernel have an average 78% reduction from the original data-set to final chunk SV decision-set, while the Sentropic kernel has a 72% reduction. The SV number in the final sequential chunk had a 22.5% reduction for the Absdiff kernel in the sequential setting, compared with a 44.3% reduction for the Gaussian kernel. So the improved accuracy of the Absdiff and Sentropic kernels, over the standard Gaussian kernels, comes at a minor cost in computational time in the distributed-chunking setting, while it can involve significantly more time in the sequential-chunking setting.

From tuning over the number of SVs to pass, we find that sequential learning topologies strongly benefit from 100% SV passing, whereas distributed learning topologies have a non-optimality at 100% SV passing (and is prone to non-convergence to a solution – see Fig. 1), while 30% SV-passing performs as well and with greater stability. There are a variety of ways to deal with the distributed learning instabilities found with passing 'base' SV's, including the solution of pipelining the learning process to always have SV's merge into an untrained chunk to avoid outlier accumulation (and gridlock) in the learning process. In the Discussion we suggest that the low SV-passing percentage that is found to work in *distributed* chunking might fundamentally be an issue of outlier control during distributed learning.

SV/non-SV pass-tuning on (9AT,9TA) vs (9CG,9GC): an outlier-management chunking heuristic

For the DNA hairpin datasets considered in the previous section, and considered here on a larger dataset, we find that the ideal chunking parameter for sequential chunking is 100% of the support vector set. This produced the best accuracy (0.855) with stable conditions (see Fig. 2). Table 3 displays a sample run and the size of each chunk as the algorithm progresses through the chunks. Table 3 also shows the feature vector set composition of each chunk.

For the DNA hairpin datasets considered in the previous section, and considered here on a larger dataset, results have shown that the ideal chunking parameter for *distributed* chunking can be as low as 30% of the support vector set. This produced the best accuracy (0.83) with stable conditions (see Fig. 3).

For the data runs shown in Table 3 & 4, where 100% of the support vector set and 50% of the polarization set were passed for the sequential chunking method Table 3) and 80% of the support vector set and 60% of the polarization set were passed for the multi-threaded chunking method (Table 4). These chunking parameters were chosen since they produce a similar accuracy when compared to the parameters discussed above. Most notably, only 30% SV-passing was needed for multi-threaded – while here we proceed with using 80% SV-passing. This is done since as no weakening of performance or convergence instabilities are observed, and so as to have a more challenging chunk-growth problem to manage in our analysis that follows.

Support Vector Reduction

Support Vector Reduction (SVR) is a process that is run right after the SVM learning step is complete. Instead of going on to merge subsets of feature vectors or to test data against known results, the idea is to further reduce the support vector set. One way to do this is to coerce some alphas to zero which means they would now fall into the polarization set. This process is described further in the Methods.

Figure 4 shows the results of the SVR method on the non-chunking SMO SVM. For this dataset, 0.19 was found to be the best cut off value since it retains accuracy while reducing the support vectors. For the 9GC9CG_9AT9TA dataset, 140 support vectors (10.5% of total) were dropped without affecting the accuracy.

SVR-enabled data runs using sequential chunking methods (Figure A1 in Additional file 9) and multi-threaded chunking methods (Figure A2 in Additional file 10) show similar results. The chunking results tend to be noisier since the SVM algorithm makes some approximations, thus the hyperplane will not be exactly the same for every data run and this behavior is amplified in the chunking methods. Nonetheless, the SVR method cuts down on support vectors and decreases testing time. For sequential chunking (Figure A1 in Additional file 9), an alpha cut-off value of 0.25 caused 87 support vectors (7.2%) to be dropped without affecting accuracy. For multi-threaded chunking (Figure A2 in Additional file 10), an alpha cut-off value of 0.22 dropped 26 support vectors (6.2%) while retaining accuracy.

Distributed SVM with pass-tuning and SVR

Multi-threaded Chunking

The multi-threaded chunking method simultaneously runs the chunks using multiple threads. Once all of the threaded chunks are finished training, the chunk results are collected into an array. The same user defined percentages of feature vector sets are used here except this time those percentages of feature vectors are extracted from each chunk. All of the chosen feature vectors to be passed are stored together then re-chunked if the current data set is large enough to be chunked again. Re-chunking occurs when the data set is greater than or equal to twice the specified chunk size. If this is not the case, the final chunk is run alone to get the final result. The main use of the multi-threaded chunking method is with a single computer with multiple processors/cores. Results are shown in Table 5.

Multi-threaded Distributed Chunking

The multi-threaded distributed chunking implementation is a multi-server/multi-CPU (core) version of the previous multi-threaded chunking method. Java RMI is used to handle the remote calls between the client and servers. The client program runs multi-threaded remote calls to a user specified set of servers (round robin). Each server and the client machine have an SVM Server listening. When the client program runs, a chunk is passed to each available processor/core in the network until all or as many as possible are training simultaneously. As the chunks finish, the results are passed back to the client. Each “chunk level” may take multiple batches depending on the chunk size and amount of processors/cores available. The final chunk is largest so the client program should be processed on the machine with the most computing power. This not only speeds up the final chunk but allowing larger chunks should produce better final results. The main benefit of this method is a significant decrease in run time for large datasets. As shown below in Table 5, multi-threaded distributed chunking performs almost as well as non-chunked learning. Network overhead causes it to be slightly slower than the single-server multi-threaded chunking method. With extremely large datasets (i.e. 60,000 feature vectors and larger), the multi-threaded and distributed method is shown to fill a critical need.

Discussion and Conclusion

Support Vector Machines are extremely useful for classifying data and therefore dominate over other methods in a variety of fields and applications. Since the main weakness of SVMs is the long training time when running large datasets it is only natural to develop multi-threaded distributed SVM training methods, especially since multiple cores/processors are becoming commonplace, each with a steadily growing capacity for RAM.

An overall comparison of the SVM methods explained here can be found in Table 5 (above). Sequential chunking has the benefit of holding onto accuracy when compared to running the straight SVM (SMO) but the run times can be higher since the method does not run in parallel. Multi-threaded chunking has a significant run time performance improvement, which is further improved when employing the SVR method. The multi-threaded aspect allows training of extremely large datasets which may not be possible using sequential chunking. Additionally, using the multi-threaded distributed method allows users to add machines to make the algorithm train even faster. This aspect makes the size of the dataset no longer as significant a limitation in SVM training, which opens up many possibilities for the practical use of SVM methods in SVM-intensive applications, such as SVM-based clustering [4].

Methods

Chunking Protocols

Chunking becomes a necessity when classifying large datasets. The number and size of the chunks depends on the size of the dataset to be trained. In the Java implementation used here, the user specifies the size of each chunk and the chunks are broken up accordingly. If the chunks don't divide evenly, which is the case most of the time, the few remaining feature vectors are added to the last chunk. When training on the chunk is complete, the resulting trained feature vectors split into distinct sets (support vectors, polarization set, penalty set, and KKT violator). If the SVM learning is done well, the largest set consists of the support and polarization feature vectors. The polarization set consists of the feature vectors that have been properly classified. These feature vectors pass the KKT relations and have an alpha coefficient equal to zero. The penalty set consists of the feature vectors which pass the KKT relations and have alpha coefficients equal to C (the max value). The KKT violators make up another set consisting of feature vectors that violate one of the KKT relations. (The KKT violator set is usually zero at the end of the training process, unless some minimal number of violators is allowed upon learning completion.). These sets give the user different categories of feature vectors that they can pass to the next chunk(s). To keep the SVM converging to a better solution on the next chunk run, however, support vectors (and sometimes some of the polarization set) are passed to the next chunk(s). The optimal pass-percentages of each feature vector set depend on which kernel is used and the dataset.

There are different methods of extracting the feature vectors from the different sets. The specified percentages of feature vectors are pseudo-randomly chosen from each of the sets except for one. *The support feature vectors extraction method differs since it extracts the support vectors that are nearest to the decision hyperplane.* We choose feature vectors whose scores are closer to the hyperplane in order to pass a tighter hyperplane on to the next chunk(s), and manage accumulation of outliers.

The chunk learning topology used in our distributed approach is slightly different from the Binary Tree splitting described in the Cascade SVM presented in [10]. As discussed above, the large dataset is broken into smaller chunks and the SVM is run on each separate chunk. Instead of bringing the results of paired chunks together, all chunk results are brought together and re-chunked as occurred in the first layer. This process occurs until the final chunk is calculated which gives the trained result. At each training stage, the user has the option to tune the percentage of support vectors and non support vectors to pass to the next set of chunks. Additionally, passed support vectors can be chosen to satisfy some max value (approx. C/10 in cases examined) to produce a tighter hyperplane to better distinguish the polarization sets and eliminate outliers. We also incorporate SVR post-processing in some of the dataruns (method below), where SVR runs as part of the core SVM learning task on each chunk. It uses a user-defined alpha cutoff value for further tuning and can significantly reduce the number of support vectors passed to the next set of chunks (with bias towards elimination of outliers and the large non-boundary alphas). These additional steps reduce the size of the chunks, thus making the algorithm run faster without loss of accuracy. The SVR post-processing also appears to offer similar immunity to the convergence pathology (noted previously for 100% SV passing on distributed learning topologies).

SVR Method

Support Vector Reduction (SVR) is a process that is run right after the SVM learning step is complete. Instead of going on to testing data against the training results to get accuracy, we further reduce the support vector set. One way to do this is to coerce some alphas to zero which means they would now fall into the polarization set. Converting the smaller alphas to zeros makes the most sense since a larger alpha indicates that the data point is stronger towards its grouping (polarized sign). This is done using a user-defined alpha cut off value. All alpha values that are under the cut off are pushed to zero. It is not entirely trivial since certain mathematical constraints must be met. The constraint that must be met for this method is the linear equality constraint [1-4]:

$$\sum_{i=1}^{N} y_i a_i = 0$$

Therefore, the alpha values not meeting the cutoff cannot just be forced to zero unless the value is retained somewhere else in the set. This is done by first sorting the alpha values of the support vectors. Then for each alpha that does not meet the cut off value, the small left over value is added to the largest alpha of the same polarity (further biases towards SVR). Since the list is sorted it can loop through and evenly distribute the left over values through the larger alphas starting with the largest. The reduction process can cut the number of support vectors significantly, while not significantly diminishing the accuracy. Other observations have shown that the easier the dataset to classify, the larger the reduction via this process.

Competing interests

The author declares that there are no competing interests.

Author's Contribution

KA did the datarun experiments and helped with writing the paper. SWH proposed the problem and the SVR methods and helped write the paper. Data analyzed was from nanopore detector experiments that were performed at the Children's Hospital nanopore detector facilities directed by SWH.

Acknowledgments

Funding was derived from grants from NIH and the LA Board of Regents. Federal funding was provided by NIH K-22 (SWH PI, 5K22LM008794). State funding was provided from a LaBOR Enhancement (SWH PI).

References

1. Vapnik VN. The Nature of Statistical Learning Theory, Springer-Verlag, New York, 1995.
2. Cortes C, and Vapnik VN, “Support Vector Networks”, Machine Learning, 20:273-297. 1995.
3. Platt JC, “Sequential Minimal Optimization: A Fast Algorithm for Training Support Vector Machines”, Microsoft Research, Technical Report MSR-TR-98-14. 1998.
4. Winters-Hilt S, Yelundur A, McChesney C, Landry M: Support Vector Machine Implementations for Classification & Clustering. BMC Bioinf. 7 Suppl 2: S4, 2006.
5. Platt JC. Fast Training of Support Vector Machines using Sequential Minimal Optimization. In *Advances in Kernel Methods -- Support Vector Learning*. Edited by Scholkopf B, Burges CJC, and Smola AJ. MIT Press, Cambridge, USA; Ch. 12. 1998.
6. Keerthi SS, Shevade SK, Bhattacharyya C and Murthy KRK. Improvements to Platt's SMO algorithm for SVM classifier design. *Neural Computation,* Vol. 13, 637-649. 2001.
7. Crammer K and Singer Y. On the Algorithmic Implementation of Multiclass Kernel-based Vector Machines. Journal of Machine Learning Research 2 pp. 265-292. 2001.
8. Zanghirati G and Zanni L, “A parallel solver for large quadratic programs in training support vector machines”, Parallel Computing, Vol. 29, pp.535-551, 2003.
9. Joachims T, Making large-scale SVM learning practical, in: B. Sch€lkopf, C.J.C. Burges, A. Smola (Eds.), Advances in Kernel Methods—Support Vector Learning, MIT Press, Cambridge, MA, 1998.
10. Graf HP, Cosatto E, Bottou L, Durdanovic I, and Vapnik VN. Parallel Support Vector Machines: The Cascade SVM, in proceedings NIPS, 2004.
11. Winters-Hilt S, Vercoutere W, DeGuzman VS, Deamer DW, Akeson M, and Haussler D. Highly Accurate Classification of Watson-Crick Basepairs on Termini of Single DNA Molecules. Biophys. J. 84:967-976. 2003.
12. Hsu CW, Lin CJ: A Comparison of Methods for Multi-class Support Vector Machines. *IEEE Transactions on Neural Networks* 13;:415-425. 2002.
13. Lee Y, Lin Y, Wahba G: Multicategory Support Vector Machines. *Technical Report 1043, Dept of Statistics* http://citeseer.ist.psu.edu/lee01multicategory.html *University of Wisconsin, Madison, WI.* 2001.

Figure Legends

Figure 1. SVM convergence failure seen with 100% SV passing on distributed learning topologies. SVM training dataset reduction with 100% SVs passed on a distributed learning topology.

Figure 2 Sequential Learning Topology SV pass-tuning. Dataset = 9GC9CG_9AT9TA (1600 feature vectors). SVM Parameters: Absdiff kernel with sigma=.5, C = 10, Epsilon = .001, Tolerance = .001. This result shows the trend for sequential chunking when using different support vector and polarization set percentage parameters. (During the tuning operation, every variation of multiples of ten up to 100 was used for each of the two sets. For example, when the SV % parameter was 10, the polarization set % parameter would vary from 0 to 100 in multiples of ten.) For most of the data run, especially the more stable part at 100 % SVs, the variation of the small polarization set did not seem to have much effect on the outcome.

Figure 3. Distributed Learning Topology SV pass-tuning. Dataset = 9GC9CG_9AT9TA (1600 feature vectors). SVM Parameters: Absdiff kernel with sigma=.5, C = 10, Epsilon = .001, Tolerance = .001. This shows the trend for multi-threaded chunking when using different support vector and polarization set percentage parameters. (Every variation of multiples of ten up to 100 was used for each of the two sets. For example, when the SV % parameter was 10, the polarization set % parameter would vary from 0 to 100 in multiples of ten.) For most of the data run, especially the more stable part around 30 % SV-passing, the variation of the polarization set did not have much effect on the outcome.

Figure 4. SMO (non-chunking) Support Vector Reduction. Dataset: 9GC9CG_9AT9TA (1600 feature vectors). SVM Parameters: Absdiff kernel with sigma=.5, C = 10, Epsilon = .001, Tolerance = .001. This graph shows the rate of support vectors reduced as the alpha cutoff value is increased. The alpha cutoff value 0.19 is chosen as the best since it is the last value before accuracy begins to degrade. This chosen value reduces 140 support vectors.

Tables and captions

Sequential Chunked SMO			Chunk Size 200 of 800 total feature vectors			
Data	**Iterations**	**# of SVs**	**SN**	**SP**	**(SN+SP)/2**	**Elapsed Time (ms)**
8GC9AT	100	554	0.96	0.95	0.955	12610
8GC9CG	114	557	0.92	0.92	0.92	16901
8GC9GC	58	524	0.94	0.97	0.955	8914
8GC9TA	68	542	0.97	0.95	0.96	10000
9AT9CG	37	727	0.83	0.8	0.815	10936
9AT9GC	23	727	0.83	0.83	0.83	9757
9AT9TA	9	661	0.93	0.93	0.93	7563
9CG9GC	15	751	0.78	0.77	0.775	9218
9CG9TA	41	597	0.92	0.89	0.905	10267
9GC9TA	51	567	0.95	0.92	0.935	9695
Mean	**51.6**	**620.7**	**0.903**	**0.893**	**0.898**	**10586.1**

Table 1. Sequential chunking using different DNA hairpin datasets. This table shows the different sequential chunking data runs performed on assortments of DNA hairpin pairs. The last line of the table presents the mean of the data runs. SVM Parameters: Absdiff kernel with sigma=.5, C = 10, Epsilon = .001, Tolerance = .001. Passing 100% of support vectors.

Distributed Chunked SMO			Chunk Size 200 of 800 total feature vectors			
Data	**Iterations**	**# of SVs**	**SN**	**SP**	**(SN+SP)/2**	**Elapsed Time (ms)**
8GC9AT	14	221	0.97	0.89	0.93	2667
8GC9CG	30	202	0.91	0.9	0.905	1993
8GC9GC	27	208	0.91	0.93	0.92	2003
8GC9TA	38	208	0.95	0.88	0.915	2017
9AT9CG	8	232	0.79	0.72	0.755	2531
9AT9GC	21	237	0.71	0.8	0.755	2121
9AT9TA	8	234	0.85	0.87	0.86	2318
9CG9GC	9	237	0.74	0.69	0.715	2132
9CG9TA	8	230	0.84	0.94	0.89	2003
9GC9TA	10	224	0.94	0.87	0.905	1945
Mean	**17.3**	**223.3**	**0.86**	**0.849**	**0.855**	**2173**

Table 2. Multi-threaded chunking using different DNA hairpin datasets. This table shows the different multi-threaded chunking data runs performed on assortments of DNA hairpin pairs. The last line of the table presents the mean of the data runs. SVM Parameters: **Sentropic kernel** with sigma=.5, C = 10, Epsilon = .001, Tolerance = .001. Passing 30% of support vectors.

	Chunk 1	Chunk 2	Chunk 3	Chunk 4
Total Chunk Size	400	787	1143	1472
Support Vectors	373	700	1002	1320
Polarization Set	27	86	140	152
Penalty Set	0	0	0	0
Violator Set	0	1	1	0
Support Vectors Passed	373	700	1002	
Polarization Set Passed	14	43	70	
Total Passed Set	387	743	1072	

Table 4. Sequential chunking with the Absdiff kernel. Dataset = 9GC9CG_9AT9TA (1600 feature vectors). SVM Parameters: Absdiff kernel with sigma=.5, C = 10, Epsilon = .001, Tolerance = .001. Pass 100% of support vectors and 50% of polarization set. Final Chunk Performance: {SN, SP} = {.87, .84}. A breakdown of each feature vector set is displayed to show how the percentage parameters are used to pass portions of each set to the next chunk.

	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10
Total Chunk Size	400	400	400	400	423	423	425	504	504	791
Support Vectors	373	377	378	388	402	402	403	466	460	699
Polarization Set	27	23	22	12	21	21	22	38	43	92
Penalty Set	0	0	0	0	0	0	0	0	0	0
Violator Set	0	0	0	0	0	0	0	0	1	0
Support Vectors Passed	1218	-	-	-	968	-	-	742	-	-
Polarization Set Passed	53	-	-	-	40	-	-	49	-	-
Total Passed Set	1271	-	-	-	1008	-	-	791	-	-

Table 4. Multi-threaded chunking with the Absdiff kernel. Dataset = 9GC9CG_9AT9TA (1600 feature vectors). SVM Parameters: Absdiff kernel with sigma=.5, C = 10, Epsilon = .001, Tolerance = .001. Pass 80% of support vectors and 60% of polarization set. Final Chunk Performance: {SN, SP} = {.855, .795}. A breakdown of each feature vector set is displayed to show how the percentage parameters are used to pass portions of each set to the next set of chunks.

SVM Method	**Sensitivity**	**Specificity**	**(SN + SP) / 2**	**Total Time (ms)**
SMO (non-chunked)	0.87	0.84	0.86	47708
Sequential Chunking	0.84	0.86	0.85	27515
Multi-threaded Chunking	0.88	0.78	0.83	7855
SMO (non-chunked) with SV Reduction	0.91	0.81	0.86	43662
Sequential Chunking with SV Reduction	0.90	0.82	0.86	18479
Multi-threaded Chunking with SV Reduction	0.85	0.83	0.84	5232
Multi-threaded Distributed Chunking with SV Reduction	0.85	0.83	0.84	5973

Table 5. Performance comparison of the different SVM methods. The distributed chunking used three identical networked machines. Dataset = 9GC9CG_9AT9TA (1600 feature vectors). SVM Parameters: Absdiff kernel with sigma=.5, C = 10, Epsilon = .001, Tolerance = .001. For chunking methods: Pass 90% of support vectors, Starting chunk size = 400, maxChunks = 2. For SV Reduction methods: Alpha cut off value = .15.

Additional files

Additional file 1
File format: DOC
Title: Table A1. Sequential chunking using different DNA hairpin datasets.
Description: This table shows the different sequential chunking data runs performed on assortments of DNA hairpin pairs. The last line of the table presents the mean of the data runs. SVM Parameters: **Sentropic kernel** with sigma=.5, C = 10, Epsilon = .001, Tolerance = .001. Passing 100% of support vectors.

Additional file 2
File format: DOC
Title: Table A2. Sequential chunking using different DNA hairpin datasets.
Description: This table shows the different sequential chunking data runs performed on assortments of DNA hairpin pairs. The last line of the table presents the mean of the data runs. SVM Parameters: **Gaussian kernel** with sigma=.05, C = 10, Epsilon = .001, Tolerance = .001. Passing 100% of support vectors.

Additional file 3
File format: DOC
Title: Table A3. Multi-threaded chunking using different DNA hairpin datasets.
Description: This table shows the different multi-threaded chunking data runs performed on assortments of DNA hairpin pairs. The last line of the table presents the mean of the data runs. SVM Parameters: **Absdiff kernel** with sigma=.5, C = 10, Epsilon = .001, Tolerance = .001. Passing 30% of support vectors.

Additional file 4
File format: DOC
Title: Table A4. Multi-threaded chunking using different DNA hairpin datasets.
Description: This table shows the different multi-threaded chunking data runs performed on assortments of DNA hairpin pairs. The last line of the table presents the mean of the data runs. SVM Parameters: **Gaussian kernel** with sigma=.05, C = 10, Epsilon = .001, Tolerance = .001. Passing 30% of support vectors.

Additional file 5
File format: DOC
Title: Table A5. Sequential chunking method focusing on the chunk sizes during the data run, using the sentropic kernel.
Description: The breakdown of each feature vector set is displayed to show how the percentage parameters are used to pass portions of each set to the next chunk.Dataset = 9GC9CG_9AT9TA (1600 feature vectors). SVM Parameters: Sentropic kernel with sigma=.5, C = 10, Epsilon = .001, Tolerance = .001. Pass 100% of support vectors and 50% of polarization set. Final Chunk Performance: {SN, SP} = {.875, .82}.

Additional file 6
File format: DOC

Title: Table A6. Sequential chunking method focusing on the chunk sizes during the data run, using the Gaussian kernel.
Description: The breakdown of each feature vector set is displayed to show how the percentage parameters are used to pass portions of each set to the next chunk. Dataset = 9GC9CG_9AT9TA (1600 feature vectors). SVM Parameters: Gaussian kernel with sigma=.05, C = 10, Epsilon = .001, Tolerance = .001. Pass 100% of support vectors and 50% of polarization set. Final Chunk Performance: {SN, SP} = {.715, .85}.

Additional file 7
File format: DOC
Title: Table A7. The multi-threaded chunking method focusing on the chunk sizes during the data run and using the sentropic kernel.
Description: The breakdown of each feature vector set is displayed to show how the percentage parameters are used to pass portions of each set to the next set of chunks. Dataset = 9GC9CG_9AT9TA (1600 feature vectors). SVM Parameters: Sentropic kernel with sigma=.5, C = 10, Epsilon = .001, Tolerance = .001. Pass 80% of support vectors and 60% of polarization set. Final Chunk Performance: {SN, SP} = {.845, .755}.

Additional file 8
File format: DOC
Title: Table A8. This table shows the multi-threaded chunking method focusing on the chunk sizes during the data run using the Gaussian kernel.
Description: The breakdown of each feature vector set is displayed to show how the percentage parameters are used to pass portions of each set to the next set of chunks. Dataset = 9GC9CG_9AT9TA (1600 feature vectors). SVM Parameters: Gaussian kernel with sigma=.05, C = 10, Epsilon = .001, Tolerance = .001. Pass 80% of support vectors and 60% of polarization set. Final Chunk Performance: {SN, SP} = {.85, .83}.

Additional file 9
File format: DOC
Title: Figure A1. Sequential Chunking Support Vector Reduction.
Description: Dataset: 9GC9CG_9AT9TA (1600 feature vectors), Starting chunk size=400. SVM Parameters: Absdiff kernel with sigma=.5, C = 10, Epsilon = .001, Tolerance = .001. Passing 100% of Support Vectors. This graph shows the rate of support vectors reduced as the alpha cutoff value is increased. The alpha cutoff value 0.25 is chosen as the best since it is the last value before accuracy begins to degrade. This chosen value reduces 87 support vectors.

Additional file 10
File format: DOC
Title: Figure A2. Multi-threaded Chunking Support Vector Reduction.
Description: Dataset: 9GC9CG_9AT9TA (1600 feature vectors), Starting chunk size=400. SVM Parameters: Absdiff kernel with sigma=.5, C = 10, Epsilon = .001, Tolerance = .001. Passing 30% of Support Vectors. This graph shows the rate of support vectors reduced as the alpha cutoff value is increased. The alpha cutoff value 0.22 is chosen as the best since it is the last value before accuracy begins to degrade. This chosen value reduces 26 support vectors.

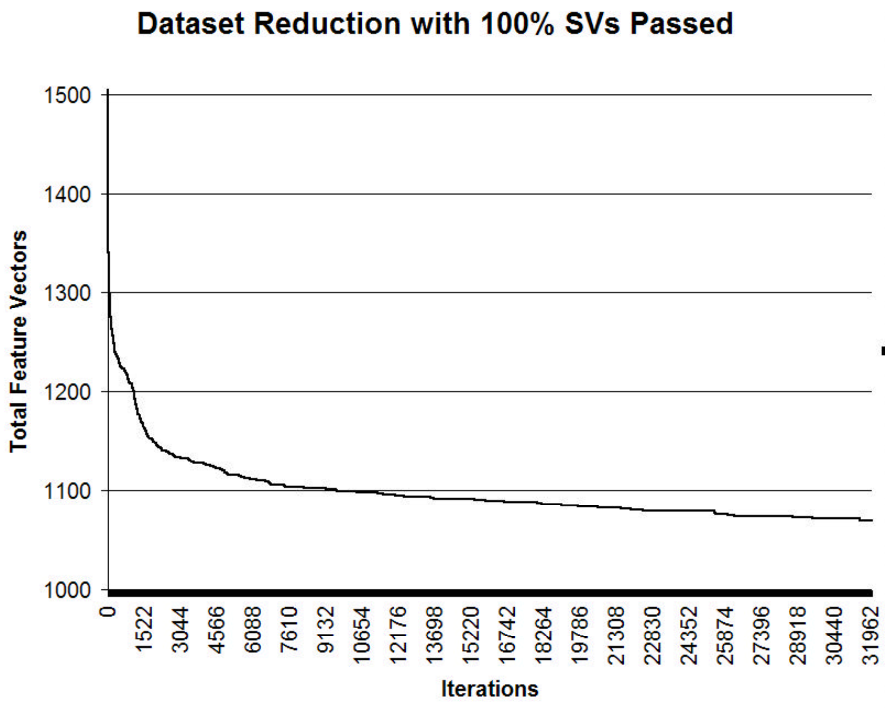
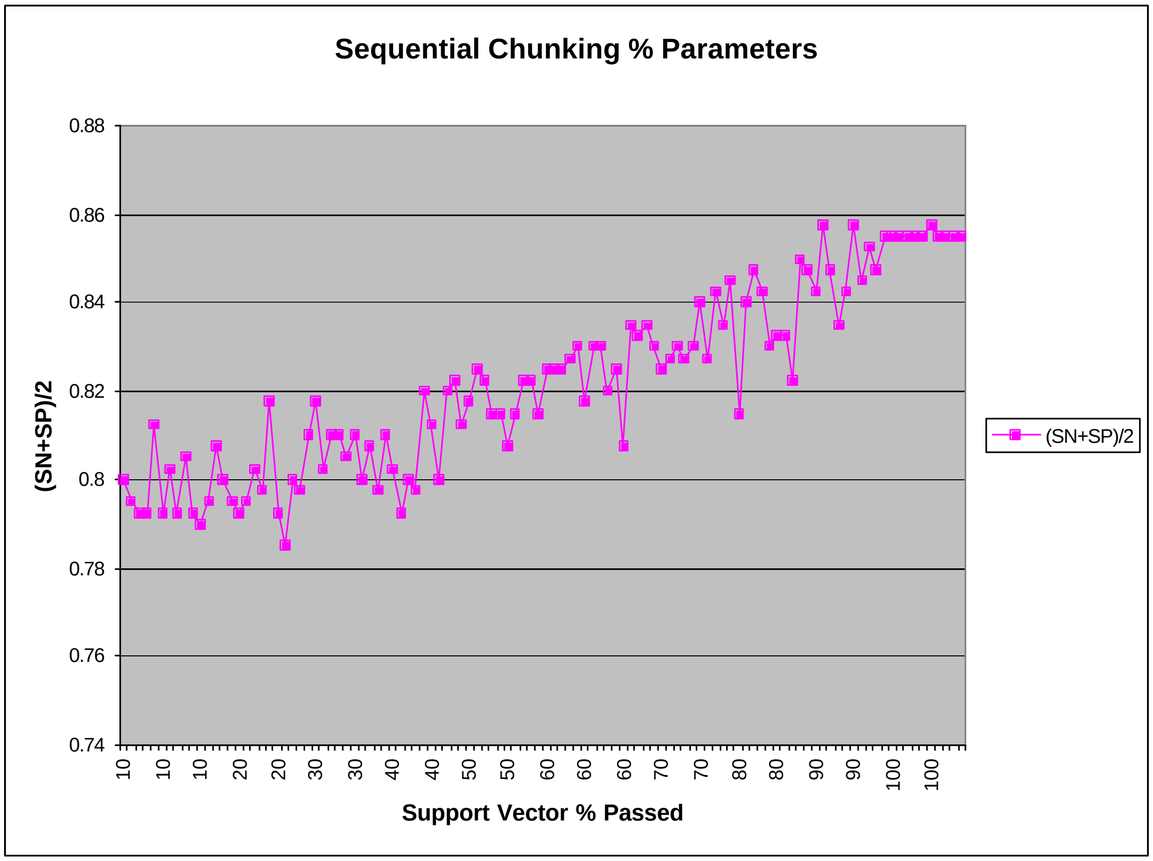

Dataset Reduction with 100% SVs Passed
Total Feature Vectors
1500
1400
1300
1200
1100
1000
Iterations
0
1522
3044
4566
6088
7610
9132
10654
12176
13698
15220
16742
18264
19786
21308
22830
24352
25874
27396
28918
30440
31962

Figure 1.



Sequential Chunking % Parameters
0.88
0.86
0.84
0.82
0.8
0.78
0.76
0.74
(SN+SP)/2
10
10
10
20
20
30
30
40
40
50
50
60
60
60
70
70
80
80
90
90
100
100
Support Vector % Passed
(SN+SP)/2

Figure 2.

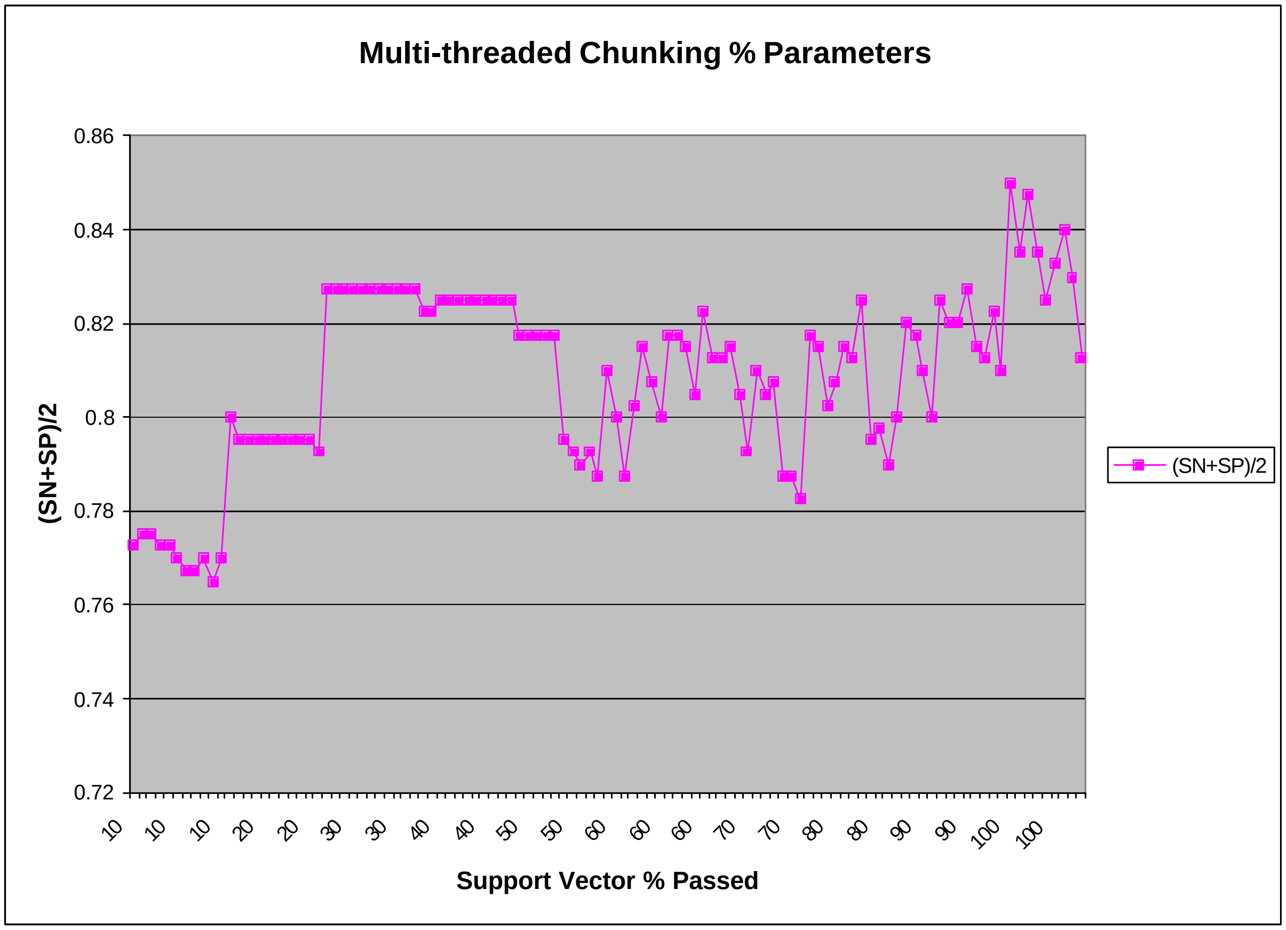

Multi-threaded Chunking % Parameters
(SN+SP)/2
0.86
0.84
0.82
0.8
0.78
0.76
0.74
0.72
10
10
10
20
20
30
30
40
40
50
50
60
60
60
70
70
80
80
90
90
100
100
Support Vector % Passed
(SN+SP)/2

Figure 3.

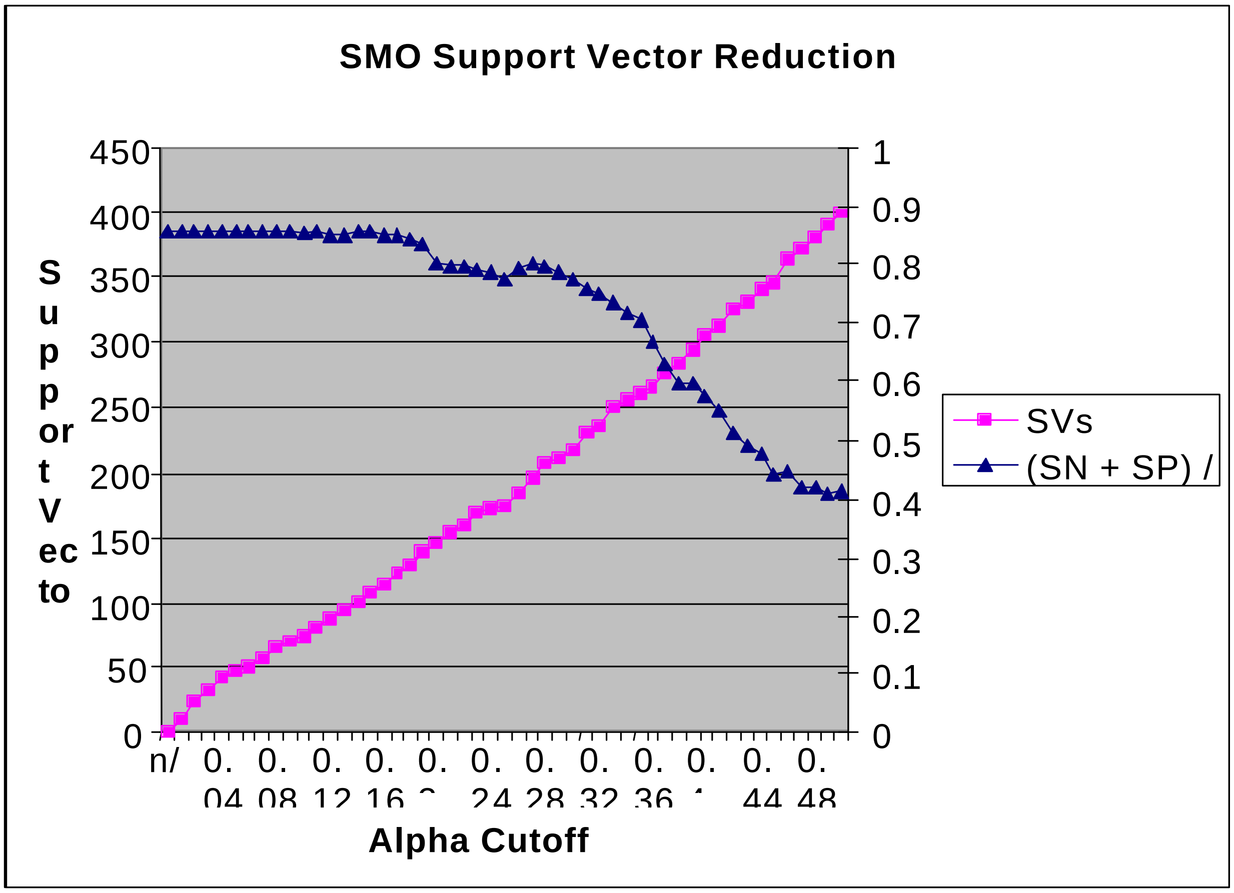

SMO Support Vector Reduction
S u p p or t V ec to
450
400
350
300
250
200
150
100
50
0
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
n/
0. 04
0. 08
0. 12
0. 16
0.
0. 24
0. 28
0. 32
0. 36
0.
0. 44
0. 48
Alpha Cutoff
SVs
(SN + SP) /

Figure 4.

Additional file 1

Sequential Chunked SMO			**Chunk Size 200 of 800 total feature vectors**			
Data	**Iterations**	**# of SVs**	**SN**	**SP**	**(SN+SP)/2**	**Elapsed Time (ms)**
8GC9AT	52	570	0.96	0.95	0.955	14479
8GC9CG	78	545	0.93	0.93	0.93	16844
8GC9GC	22	525	0.92	0.95	0.935	10065
8GC9TA	130	550	0.97	0.95	0.96	18304
9AT9CG	32	722	0.83	0.82	0.825	15273
9AT9GC	44	734	0.81	0.82	0.815	15452
9AT9TA	39	693	0.9	0.9	0.9	14419
9CG9GC	33	747	0.75	0.79	0.77	14855
9CG9TA	73	616	0.89	0.91	0.9	16978
9GC9TA	54	597	0.91	0.93	0.92	14159
Mean	**55.7**	**629.9**	**0.89**	**0.895**	**0.891**	**15082.8**

Table A1. Sequential chunking using different DNA hairpin datasets. This table shows the different sequential chunking data runs performed on assortments of DNA hairpin pairs. The last line of the table presents the mean of the data runs. SVM Parameters: **Sentropic kernel** with sigma=.5, C = 10, Epsilon = .001, Tolerance = .001. Passing 100% of support vectors.

Additional file 2

Sequential Chunked SMO			**Chunk Size 200 of 800 total feature vectors**			
Data	**Iterations**	**# of SVs**	**SN**	**SP**	**(SN+SP)/2**	**Elapsed Time (ms)**
8GC9AT	97	434	0.93	0.88	0.905	14978
8GC9CG	83	395	0.88	0.85	0.865	15653
8GC9GC	134	396	0.94	0.88	0.91	17783
8GC9TA	114	396	0.96	0.82	0.89	15988
9AT9CG	62	503	0.79	0.82	0.805	20789
9AT9GC	89	488	0.81	0.85	0.83	20833
9AT9TA	111	477	0.91	0.89	0.9	18057
9CG9GC	60	523	0.91	0.54	0.725	20233
9CG9TA	78	436	0.88	0.9	0.89	17910
9GC9TA	89	409	0.9	0.94	0.92	12422
Mean	**91.7**	**445.7**	**0.891**	**0.84**	**0.864**	**17464.6**

Table A2. Sequential chunking using different DNA hairpin datasets. This table shows the different sequential chunking data runs performed on assortments of DNA hairpin pairs. The last line of the table presents the mean of the data runs. SVM Parameters: **Gaussian kernel** with sigma=.05, C = 10, Epsilon = .001, Tolerance = .001. Passing 100% of support vectors.

Additional file 3

Distributed Chunked SMO			**Chunk Size 200 of 800 total feature vectors**			
Data	**Iterations**	**# of SVs**	**SN**	**SP**	**(SN+SP)/2**	**Elapsed Time (ms)**
8GC9AT	8	222	0.97	0.83	0.9	1947
8GC9CG	8	226	0.91	0.89	0.9	1471
8GC9GC	65	205	0.93	0.96	0.945	1412
8GC9TA	28	209	0.84	0.93	0.885	1489
9AT9CG	8	238	0.77	0.65	0.71	1308
9AT9GC	10	228	0.74	0.71	0.725	1342
9AT9TA	10	232	0.9	0.91	0.905	1265
9CG9GC	8	238	0.66	0.85	0.755	1236
9CG9TA	10	222	0.92	0.91	0.915	1232
9GC9TA	12	224	0.92	0.88	0.9	1233
Mean	**16.7**	**224.4**	**0.856**	**0.852**	**0.854**	**1393.5**

Table A3. Multi-threaded chunking using different DNA hairpin datasets. This table shows the different multi-threaded chunking data runs performed on assortments of DNA hairpin pairs. The last line of the table presents the mean of the data runs. SVM Parameters: **Absdiff kernel** with sigma=.5, C = 10, Epsilon = .001, Tolerance = .001. Passing 30% of support vectors.

Additional file 4

Distributed Chunked SMO			**Chunk Size 200 of 800 total feature vectors**			
Data	**Iterations**	**# of SVs**	**SN**	**SP**	**(SN+SP)/2**	**Elapsed Time (ms)**
8GC9AT	24	174	0.93	0.83	0.88	2629
8GC9CG	13	170	0.87	0.85	0.86	1732
8GC9GC	59	167	0.95	0.76	0.855	1970
8GC9TA	66	158	0.96	0.8	0.88	1713
9AT9CG	35	192	0.88	0.72	0.8	1490
9AT9GC	34	191	0.79	0.81	0.8	1750
9AT9TA	36	181	0.79	0.87	0.83	1456
9CG9GC	75	192	0.78	0.63	0.705	1912
9CG9TA	38	178	0.82	0.89	0.855	1533
9GC9TA	37	166	0.84	0.89	0.865	1922
Mean	**41.7**	**176.9**	**0.861**	**0.81**	**0.833**	**1810.7**

Table A4. Multi-threaded chunking using different DNA hairpin datasets. This table shows the different multi-threaded chunking data runs performed on assortments of DNA hairpin pairs. The last line of the table presents the mean of the data runs. SVM Parameters: **Gaussian kernel** with sigma=.05, C = 10, Epsilon = .001, Tolerance = .001. Passing 30% of support vectors.

Additional file 5

-	Chunk 1	Chunk 2	Chunk 3	Chunk 4
Total Chunk Size	400	792	1150	1481
Support Vectors	383	707	1011	1320
Polarization Set	17	85	139	160
Penalty Set	0	0	0	0
Violator Set	0	0	0	1
Support Vectors Passed	383	707	1011	
Polarization Set Passed	9	43	70	
Total Passed Set	392	750	1081	

Table A5. Sequential chunking method focusing on the chunk sizes during the data run, using the sentropic kernel. The breakdown of each feature vector set is displayed to show how the percentage parameters are used to pass portions of each set to the next chunk.Dataset = 9GC9CG_9AT9TA (1600 feature vectors). SVM Parameters: Sentropic kernel with sigma=.5, C = 10, Epsilon = .001, Tolerance = .001. Pass 100% of support vectors and 50% of polarization set. Final Chunk Performance: {SN, SP} = {.875, .82}.

Additional file 6

	Chunk 1	Chunk 2	Chunk 3	Chunk 4
Total Chunk Size	400	754	1036	1264
Support Vectors	309	521	697	881
Polarization Set	90	229	334	372
Penalty Set	1	4	4	11
Violator Set	0	0	1	0
Support Vectors Passed	309	521	697	
Polarization Set Passed	45	115	167	
Total Passed Set	354	636	864	

Table A6. Sequential chunking method focusing on the chunk sizes during the data run, using the Gaussian kernel. The breakdown of each feature vector set is displayed to show how the percentage parameters are used to pass portions of each set to the next chunk. Dataset = 9GC9CG_9AT9TA (1600 feature vectors). SVM Parameters: Gaussian kernel with sigma=.05, C = 10, Epsilon = .001, Tolerance = .001. Pass 100% of support vectors and 50% of polarization set. Final Chunk Performance: {SN, SP} = {.715, .85}

Additional file 7

.	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10
Total Chunk Size	400	400	400	400	423	423	425	503	504	787
Support Vectors	383	380	383	379	409	396	442	465	446	666
Polarization Set	17	17	17	21	21	14	27	38	56	121
Penalty Set	0	0	0	0	0	0	0	0	0	0
Violator Set	0	1	0	0	0	0	1	0	2	0
Support Vectors Passed	1224	-	-	-	966	-	-	730	-	-
Polarization Set Passed	47	-	-	-	41	-	-	57	-	-
Total Passed Set	1271	-	-	-	1007	-	-	787	-	-

Table A7. The multi-threaded chunking method focusing on the chunk sizes during the data run and using the sentropic kernel. The breakdown of each feature vector set is displayed to show how the percentage parameters are used to pass portions of each set to the next set of chunks. Dataset = 9GC9CG_9AT9TA (1600 feature vectors). SVM Parameters: Sentropic kernel with sigma=.5, C = 10, Epsilon = .001, Tolerance = .001. Pass 80% of support vectors and 60% of polarization set. Final Chunk Performance: {SN, SP} = {.845, .755}.

Additional file 8

	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10
Total Chunk Size	400	400	400	400	401	401	403	458	458	690
Support Vectors	291	309	316	305	318	320	313	341	354	495
Polarization Set	108	90	83	93	83	81	88	116	103	194
Penalty Set	1	1	1	0	0	0	2	1	0	1
Violator Set	0	0	0	2	0	0	0	0	1	0
Support Vectors Passed	980	-	-	-	764	-	-	558	-	-
Polarization Set Passed	225	-	-	-	152	-	-	132	-	-
Total Passed Set	1205	-	-	-	913	-	-	690	-	-

Table A8. This table shows the multi-threaded chunking method focusing on the chunk sizes during the data run using the Gaussian kernel. The breakdown of each feature vector set is displayed to show how the percentage parameters are used to pass portions of each set to the next set of chunks. Dataset = 9GC9CG_9AT9TA (1600 feature vectors). SVM Parameters: Gaussian kernel with sigma=.05, C = 10, Epsilon = .001, Tolerance = .001. Pass 80% of support vectors and 60% of polarization set. Final Chunk Performance: {SN, SP} = {.85, .83}.

Additional file 9

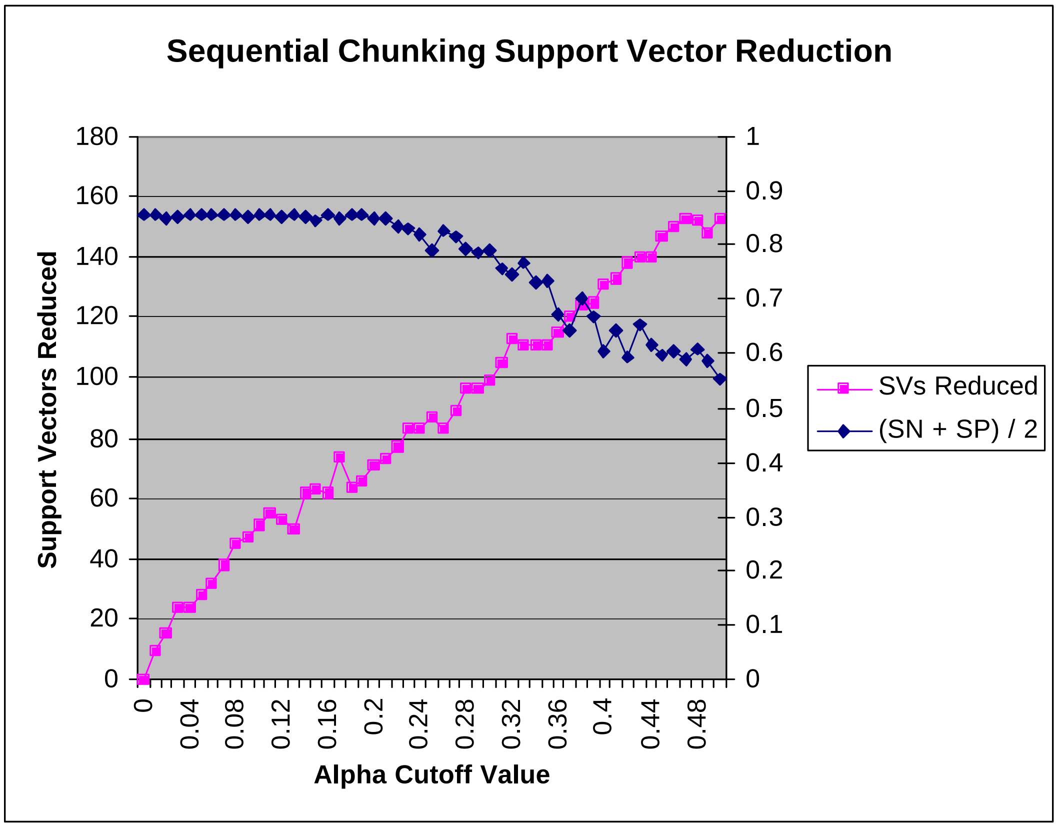


Figure A1. Sequential Chunking Support Vector Reduction. Dataset: 9GC9CG_9AT9TA (1600 feature vectors), Starting chunk size=400. SVM Parameters: Absdiff kernel with sigma=.5, C = 10, Epsilon = .001, Tolerance = .001. Passing 100% of Support Vectors. This graph shows the rate of support vectors reduced as the alpha cutoff value is increased. The alpha cutoff value 0.25 is chosen as the best since it is the last value before accuracy begins to degrade. This chosen value reduces 87 support vectors.

Additional file 10

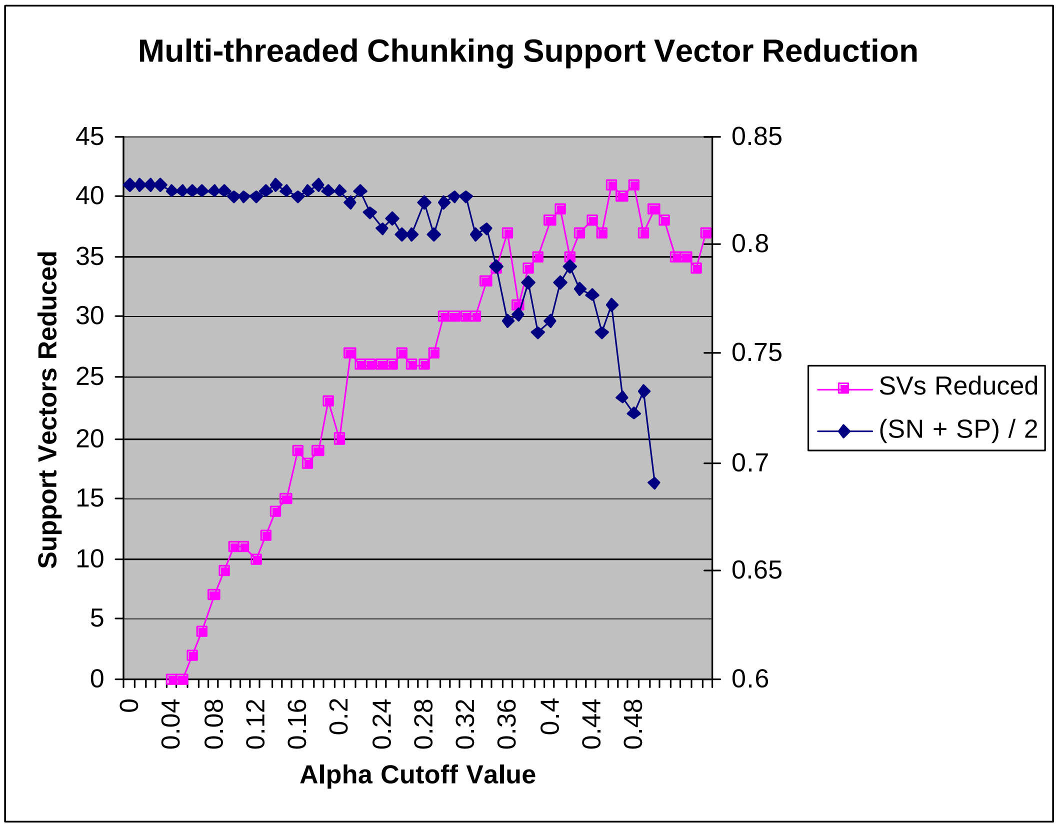


Figure A2. Multi-threaded Chunking Support Vector Reduction. Dataset: 9GC9CG_9AT9TA (1600 feature vectors), Starting chunk size=400. SVM Parameters: Absdiff kernel with sigma=.5, C = 10, Epsilon = .001, Tolerance = .001. Passing 30% of Support Vectors. This graph shows the rate of support vectors reduced as the alpha cutoff value is increased. The alpha cutoff value 0.22 is chosen as the best since it is the last value before accuracy begins to degrade. This chosen value reduces 26 support vectors.