SimpleSVM

S.V.N. Vishwanathan VISHY@AXIOM.ANU.EDU.AU
Machine Learning Program, National ICT for Australia, Canberra, ACT 0200, Australia

Alexander J. Smola ALEX.SMOLA@ANU.EDU.AU
Machine Learning Group, RSISE , Australian National University, Canberra, ACT 0200, Australia

M. Narasimha Murty MNM@CSA.IISC.ERNET.IN
Dept. Of Comp. Sci. and Automation, Indian Institute of Science, Bangalore 560012, India

Abstract

We present a fast iterative support vector training algorithm for a large variety of different formulations. It works by incrementally changing a candidate support vector set using a greedy approach, until the supporting hyperplane is found within a finite number of iterations.

It is derived from a simple active set method which sweeps through the set of Lagrange multipliers and keeps optimality in the unconstrained variables, while discarding large amounts of bound-constrained variables. The hard-margin version can be viewed as a simple (yet computationally crucial) modification of the incremental SVM training algorithms of Cauwenberghs and Poggio.

Experimental results for various settings are reported. In all cases our algorithm is considerably faster than competing methods such as Sequential Minimal Optimization or the Nearest Point Algorithm.

1. Introduction

Support Vector Machines (SVM) have gained prominence in the field of machine learning and pattern classification. In this paper we propose a fast iterative active set method for training a SVM.

1.1. Notation and Background

In the following we denote by $\{(x_1, y_1), \ldots, (x_n, y_n)\} \subset \mathcal{X} \times \{\pm 1\}$ the set of labeled training samples, where x_i are drawn from some domain $\mathcal{X}$ and $y_i \in \{\pm 1\}$, denotes the class labels $+1$ and -1 respectively. Furthermore, let n be the total number of points and let n_+ and n_- denote the number of points in class $+1$ and -1 respectively. Denote by $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ a Mercer kernel and by $\Phi : \mathcal{X} \to \mathcal{F}$ the corresponding feature map, that is $\langle \Phi(x), \Phi(x') \rangle = k(x, x')$.

It is well known that the optimal separating hyperplane between the sets with $y_i = 1$ and $y_i = -1$ is spanned by a linear combination of points in feature space (Schölkopf & Smola, 2002). Consequently the classification rule can be expressed in terms of dot products in feature space and we have

$$f(x) = \langle \mathbf{w}, \Phi(x) \rangle + b = \sum_{j \in A} \alpha_j y_j k(x_j, x) + b, \quad (1)$$

where $\alpha_j \geq 0$ is the coefficient associated with a support vector x_j and b is an offset.

In the case of a *hard-margin* SVM, all SVs satisfy $y_i f(x_i) = 1$ and for all other points we have $y_i f(x_i) > 1$. Furthermore (to account for the constant offset b) we have the condition $\sum_i y_i \alpha_i = 0$. This means that if we knew all SVs beforehand, we could simply find the solution of the associated quadratic program by a simple matrix inversion.

To cope with errors, a soft margin loss function was introduced by Bennett and Mangasarian (1993), leading to various quadratic optimization problems depending on the type of error penalty used. For quadratic penalty it is well known (Cortes & Vapnik, 1995) that such loss functions give rise to a modified hard-margin SV problem,[1] where the kernel $k(x, x')$ is replaced by $k(x, x') + \chi \delta_{x,x'}$ for some $\chi > 0$. Again, if we knew

[1] For convenience of notation we assume that there are no duplicates in the observations.

Proceedings of the Twentieth International Conference on Machine Learning (ICML-2003), Washington DC, 2003.

which vectors become SVs beforehand, we could find the solution by a simple matrix inversion.

Finally, for the linear soft margin formulation, matters are somewhat more complicated since SVs could be margin SVs or margin errors. Here, if we knew the sets of margin errors, margin SVs and their complement, the points classified correctly with a minimum margin, beforehand, we could solve the optimization problem by solving a linear system.

Such a strategy would be particularly useful if the number of unconstrained SVs was comparatively small with respect to the overall size of the dataset, since in such cases the search for the unconstrained SVs is rather swift.

1.2. The Basic Idea

We propose an active set method inspired by the chunking strategies used in the early days of SVs at AT&T: given an optimal solution on a subset, add only *one point* to the set of SVs at a time and compute the exact solution.

If we had to re-compute the solution from scratch every time a new point is added, this would be an extremely wasteful procedure. Instead, as we will see below, it is possible to perform such computations at $O(m^2)$ cost, where m is the number of current SVs *and* obtain the exact solution on the new subset of points. As one would expect, this modification works well whenever the number of SVs is small relatively to the size of the dataset, that is, for "clean" datasets. In many cases the kernel matrix may be rank degenerate or may be approximated by a low rank matrix (Schölkopf & Smola, 2002). In case the kernel matrix is rank degenerate of rank l, updates can be performed at $O(ml)$ cost using a novel factorization method of Smola and Vishwanathan (2003), thereby further reducing the computational burden (see Vishwanathan (2002) for technical details).

To cope with "dirty" datasets or the linear soft margin SVM another modification is needed: whenever a point is classified as a margin error (i.e., whenever it hits the upper boundary), it is removed from the current working set and the value of its Lagrange multiplier α_i is frozen until the point is re-visited.

While there is no guarantee that one sweep through the data set will lead to a full solution of the optimization problem (and it almost never will, since some points may be left out which will become SVs at a later stage and a similar cautionary note applies to margin errors), we empirically observe that a small number of passes through the entire dataset (less than 10) is sufficient for the algorithm to converge. Algorithm 1 gives a high-level description of SimpleSVM in the hard-margin case.

Algorithm 1 Hard-Margin SimpleSVM

input Dataset Z
  Initialize: Find a close pair from opposing classes (x_{i_+}, x_{i_-})
  $A \leftarrow \{i_+, i_-\}$
  Compute f and α for A
  while there are x_v with $y_v f(x_v) < 1$ **do**
    $A \leftarrow A \cup \{v\}$
    Recompute f and α and remove non-SVs from A.
  end while
  Output: A, $\{\alpha_i \text{ for } i \in A\}$

1.3. Outline of the Paper

In the following section we describe the active set method underlying SimpleSVM. This section also includes a proof that all algorithms of the SimpleSVM family have linear convergence rate and converge in finite time (this, of course, does not preclude each step from taking up to quadratic time in size of the current active set). Subsequently, Section 3 discusses implementation details and initialization strategies. Experimental evidence of the performance of SimpleSVM is given in Section 4 and we compare it to other state-of-the art optimization methods. We conclude with a discussion in Section 5.

Due to space limitations, most technical details concerning the factorization of matrices and rank-one modifications is relegated to Vishwanathan (2002). The latter plus initial experimental code will be made freely available at `http://www.axiom.anu.edu.au/~vishy`.

1.4. Related Work

Cauwenberghs and Poggio (2001) proposed an incremental SVM algorithm for the hard-margin case where at each step only one point is added to the training set and one re-computes the exact SV solution of the *whole dataset seen so far*. Although similar in spirit to SimpleSVM this algorithm suffers from a serious drawback:

While adding one point at a time converges in a finite number of steps (clearly after n additions we have seen the whole training set), the condition to remain optimal at every step means that at every step the algorithm has to *test and potentially train* on all the observations seen so far. Such a requirement is clearly

expensive. The authors suggest various online variants to alleviate the problem (e.g., by introducing tolerance terms into the margin definition), however the main problem remains.

The way to overcome the above limitations is to drop the requirement of optimality with respect to all the data seen so far and only require that the new solution strictly decrease the value of the dual objective function (in SV terms the margin of separation) and be optimal with respect to a subset of variables. The advantage of doing this is twofold: firstly, we can deal with a larger family of optimization problems and SVM formulations. Secondly, we can maintain and update a numerically stable $LDL^\top$ factorization of a subset of the kernel matrix.

Similar steps are proposed in DirectSVM (Roobaert, 2000), which starts off with a pair of points in the candidate SV set. It works on the conjecture that the point which incurs the maximum error (i.e., minimal $y_i f(x_i)$) during each iteration is a SV. This is a heuristic and as such it may fail. In DirectSVM's case this invokes a random restart of the algorithm, as it has no provisions to backtrack and remove points from the kernel expansion.

2. Active Set Algorithm

Instead of describing the technical details of SimpleSVM in terms of SVs, it is advantageous to consider general constrained optimization problems and describe the SimpleSVM family in terms of an active set method. This alleviates the need of an immediate "geometric" interpretation of every optimization step.

Assume a constrained optimization problem with a small-to-moderate number of equality constraints and a large number of (easy-to-check) box constraints. This situation occurs in all currently known SV settings, be it classification, regression or novelty detection, independent of the parameterization (ν vs. C), independent of the number of parametric basis functions (bias, semiparametric estimation, etc.), and independent of the loss function used (as long as it is piecewise constant, linear, quadratic, or infinite). See Schölkopf and Smola (2002) for explicit formulations. Consider

$$\begin{array}{ll} \underset{\alpha}{\text{minimize}} & \frac{1}{2}\alpha^\top H\alpha + c^\top\alpha \\ \text{subject to} & 0 \le \alpha_i \le C \text{ and } A\alpha = b \end{array} \tag{2}$$

where $H \in \mathbb{R}^{m\times m}$ is a positive semidefinite matrix, $\alpha, c \in \mathbb{R}^m$, $A \in \mathbb{R}^{d\times m}$, and $b \in \mathbb{R}^d$. In general, interior point codes are among the most efficient ones to solve this type of problems.

However, if the problem or its solution has special structure, faster solutions can be obtained. This is quite often the case in SVMs, since only a small, and, in the large sample size limit, negligible, fraction of Support Vectors actually ends up lying *on* the margin for linear soft-margin loss (Schölkopf & Smola, 2002). As a consequence, out of the large number of Lagrange multipliers, almost all end up hitting either the lower constraint $\alpha_i = 0$ or upper constraint $\alpha_i = C$, thereby rendering the optimization problem much simpler than a direct solution suggests.

2.1. The Algorithm

Denote by $S_0 := \{i|\alpha_i = 0\}$, $S_C := \{i|\alpha_i = C\}$, and $S_{\text{work}} := \{i|\alpha_i \in (0, C)\}$ such that $S_0 \cup S_{\text{work}} \cup S_C = [1:m]$. In analogy to that denote by $\alpha_0, \alpha_C, \alpha_{\text{work}}$ the corresponding sets of variables. Furthermore denote by O_{work} the optimization problem (2), constrained to the α_{work}, while keeping α_0, α_C fixed:

$$\begin{array}{ll} \underset{\alpha_{\text{work}}}{\text{minimize}} & \frac{1}{2}\alpha_{\text{work}}^\top H_{ww}\alpha_{\text{work}} + (c_w + H_{wC}\alpha_C)^\top\alpha_{\text{work}} \\ \text{subject to} & 0 \le \alpha_i \le C \text{ and } A_w\alpha_{\text{work}} = b - A_C\alpha_C \end{array} \tag{3}$$

Here H_{ww} is the (quadratic) submatrix of H given by the entries determined by S_{work} only, H_{wC} is the (rectangular) submatrix of H formed by selecting rows from S_{work} and columns from S_C, and finally c_w, A_w and A_C are the submatrices of c and A arising by selecting columns via S_{work} and S_C respectively.

Finally assume that we have an initial α for which α_{work} solves O_{work} and satisfies the constraints imposed by (2). The algorithm works as follows:

1. At each step add one variable from S_C or S_0 which violates the Kuhn-Tucker optimality condition into S_{work}
   (a) Solve the new optimization problem O_{work} ignoring the box constraints $0 \le \alpha_i \le C$.
   (b) If the resulting new vector $\alpha_{\text{work}}^{\text{new}}$ satisfies the box constraints automatically we are done and proceed to the next variable.
   (c) If the box constraint is not satisfied pick the point where the line $[\alpha_{\text{work}}^{\text{old}}, \alpha_{\text{work}}^{\text{new}}]$ intersects with the box constraints and remove the coordinate for which the box constraint becomes active from S_{work} and add it to S_C. This decreases the number of free variables by one and we proceed with Step a).
2. Check whether the resulting solution is optimal and unless this is the case we repeat step 1) using the new set of variables given by α_{work}.

This simple algorithm has several desirable properties: it is relatively simple to implement (see next section) and it enjoys attractive convergence properties:

Proposition 1 *The active set algorithm described above converges in a finite number of steps to exact solution of (2). The convergence is linear for positive definite H.*

Proof Observe that the dual objective function must strictly improve at every step. This is so since at every step we proceed from a sub-optimal solution (we add in a variable at a time which does not satisfy the Kuhn-Tucker conditions yet) to an optimal solution in these variables.

Next note that the value after each step of the optimization only depends on the choice of sets S_0, S_{work}, and S_C. Since there exists only a finite number of them and the algorithm cannot cycle (since we make steady progress at each step), we must reach the optimal partition in finite time.

Finally, to show linear convergence, note that we are performing updates which are strictly better than coordinate descent at every step (in coordinate descent we only optimize over one variable at a time, whereas in our case we optimize over S_{work} which includes a new variable at every step). Coordinate descent, however, has linear convergence for strictly convex functions (Fletcher, 1989). ■

2.2. Applying the Optimality Conditions

It follows from Farkas' Lemma (Fletcher, 1989) that for the optimization problem

$$\underset{\alpha}{\text{minimize}}\ \frac{1}{2}\alpha^\top H\alpha + c^\top\alpha \text{ subject to } A\alpha = b \qquad (4)$$

the optimal solution in α can be found by requiring that the gradient outside the constraints vanishes. This means that we are solving the linear system

$$\begin{bmatrix} H & A^\top \\ A & 0 \end{bmatrix}\begin{bmatrix} \alpha \\ \beta \end{bmatrix} = \begin{bmatrix} -c \\ b \end{bmatrix} \qquad (5)$$

If we add one variable to (4), the overall matrix only changes by one row/column, making it the perfect candidate for a rank-one update (the cost is quadratic rather than cubic for a full matrix inversion), which can be done very efficiently .

Note, however, that we have to solve the box-constrained optimization problem (2) instead of (4).

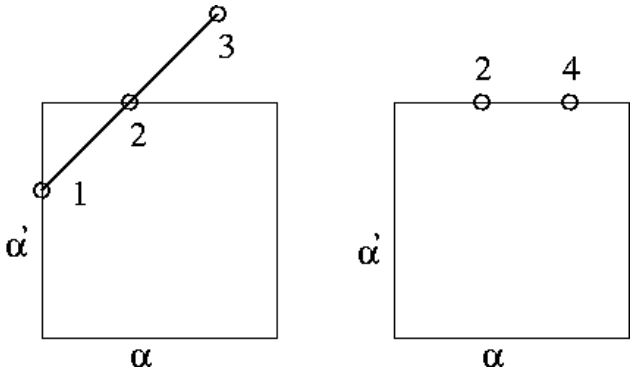

Figure 1. Progress of the active set algorithm (without equality constraints): from a feasible solution (1) we optimize over the currently free variables plus a new variable, i.e., (α, α') to obtain an unconstrained optimal solution (3). Next we progress from (1) to (3) until we hit the constraints (2). Here α' becomes bound and we optimize over the remaining free variables (α) to obtain an (unconstrained) optimal solution (4) which automatically satisfies the box constraints. If the latter is not the case, we repeat the previous steps, i.e., 1-3-2, however now starting with (2) instead of (1).

So unless the unconstrained solution ends up lying inside the box we can use the unconstrained solution only as an indication on how to proceed in the optimization. Moving towards the unconstrained part along the boundary guarantees that we progress in the objective function. Once we hit the boundary this will remove one variable from the set of free variables[2]

Finally, observe that by moving on the line between the old set of variables satisfying the equality constraints and the new solution (possibly violating the box constraints) we will always strictly satisfy the equality constraint $A\alpha = b$. Hence we will remain strictly feasible at every step.

2.3. Selecting new Variables

The issue of selecting a new variable, say i, to be added to the working set S_{work} is relatively straightforward: for the convergence of our algorithm we need to ensure that each variable is visited in a sweep through the dataset. However, it only pays to choose it if we can make progress in this variable. For this purpose we check the gradient after correction by the constraints imposed by (5).

Note that the set of variables we started with, was optimal in the free variables, hence it solves the un-

[2]The pathological case of more than one variable to become bound at the same time can be treated as if we were successively removing one variable at a time.

modified system (5). Furthermore α is always strictly feasible in the constraints. Given the particular form of the RHS of (5), which is determined by (3), only the i-th row of the system (5) may not be satisfied after enlarging it by one row and column and updating its RHS. This means that we need to check

$$\sum_{j \in S_{\text{work}}} \alpha_j H_{ij} + \sum_l \beta_l A_{il} = -c_i - \sum_{j \notin S_{\text{work}}} H_{ij}\alpha_j \quad (6)$$

or, in short $g_i(\alpha, \beta) := [H\alpha]_i + [A^\top \beta]_i + c_i = 0$.

Optimization theory tells us that if $g_i(\alpha, \beta) > 0$, we can make progress by shrinking α_i, and likewise, we can decrease the objective function by increasing α_i if $g_i(\alpha, \beta) < 0$. This means that only if $\alpha_i = 0$ and $g_i < 0$ or alternatively $\alpha_i = C$ and $g_i > 0$ we need to consider this point for optimization.

2.4. SVM Interpretation

We now return to the SV formulation for some more geometric intuition.

In our case H is the kernel matrix, possibly adorned by the labels, that is $H_{ij} = y_i y_j k(x_i, x_j)$. The vector c has entries all -1 (for C-classification) and $c = 0$ for ν-classification. Moreover, $A = (y_1, \ldots, y_n)^\top$ for classification and larger constraint matrices are found for ν-SVM and semiparametric estimation. In the hard margin case, $C = \infty$, that is, we ignore upper constraints.

Having a set of Lagrange multipliers satisfying the equality constraint $A\alpha = b$ means, in SV language, that the free variables (constant offset, margin, or semiparametric terms) of the optimization problem are optimally determined. In particular, their values are given by β from (5). A detailed description on free variables and dual constraints can be found in Schölkopf and Smola (2002).

Adding a variable to the set of free variables means that we want to find a (partial) solution for which the contributions of the other points to the weight vector w are kept fixed. For hard margin or quadratic soft-margin SVM this simply means that we ignore points which currently are not SVs. In the case of a linear soft-margin formulation it means that we also ignore points which have turned into margin errors, while keeping their contribution to the weight vector $\mathbf{w}$ in (1) fixed.

In SV terms the selection criterion (6) $g_i(\alpha, \beta) \neq 0$ means that we add only points which are erroneously flagged as margin errors ($\alpha_i = C$ but $y_i f(x_i) > 1$) or those which turn out to be margin errors but with vanishing Lagrange multiplier ($\alpha_i = 0$). The connection to (6) stems from the fact that

$$[H\alpha]_i + [A^\top \beta]_i = y_i f(x_i)$$

In this sense it is quite obvious why Algorithm 1 is a special instance of the optimization steps discussed above. However, it would be much more difficult to describe the updates in SV language than it is in a purely algebraic description.

3. Implementation Issues

This section contains details on initialization strategies plus cases where we have rank degenerate kernel matrices.

3.1. Initialization

Since we want to find the optimal separating hyperplane of the overall dataset Z, a good starting point is the pair of observations (x_+, x_-) from opposing sets X_+, X_- closest to each other (Roobaert, 2000). This means that we already start with a relatively small upper bound on the margin.

Brute force search for this pair costs $O(n^2)$ kernel evaluations, which is clearly not acceptable for the search of a good starting point. Instead one may use one of the following two operations: an iterative scheme which will find the closest pair in log-linear time, or a randomized method, which will find a pair almost as good as the best pair in constant time.

Algorithm 2 Closest Pair

Input: Sets X_+, X_-
Initialize: Draw x_+, x_- at random from X_+, X_-.
repeat
  $D_{\text{old}} \leftarrow d(x_+, x_-)$.
  $x_- \leftarrow \text{argmin}_{x \in X_-} d(x_+, x)$,
  $x_+ \leftarrow \text{argmin}_{x \in X_+} d(x_-, x)$
  $D_{\text{new}} \leftarrow d(x_+, x_-)$.
until $D_{\text{old}} = D_{\text{new}}$
Output: x_+, x_-

The Best Pair: Algorithm 2 finds the best pair by iteratively finding the closest point from one set with respect to another point and vice versa. This algorithm is known to converge in log-linear time.

A Randomized Method: denote by $\xi := d(x_+, x_-)$ the random variable obtained by randomly choosing $x_+ \in X_+$ and $x_- \in X_-$. Then the shortest distance between a pair x_+, x_- is given by the minimum of the random variables ξ. Therefore, if

we are only interested in finding a pair whose distance is, with high probability, much better than the distance of any other pair, we need only draw random pairs and pick the closest one.

In particular, one can check that roughly 59 pairs are sufficient for a pair better than 95% of all pairs with 0.95 probability (Schölkopf & Smola, 2002).

Once a good pair (x_+, x_-) has been found, we use the latter as S_{work} (for C-SVM the vector $\alpha = 0$ is feasible) and begin with the optimization.

Whenever we have upper box constraints $\alpha_i \leq C$ and somewhat more complex equality constraints (e.g., in ν-SVM), that is, cases where a single pair of points cannot influence the outcome by too much, a simple random initialization has proven to yield good results. After all, the algorithm finds the sets S_0, S_C and S_{work} relatively quickly.

3.2. Rank-Degenerate Kernels

Regardless of the type of matrix factorization we use to compute the SV solutions, we still encounter the problem that the memory requirements scale as $O(m^2)$ and the overall computation is of the order of $O(m^3 + mn)$ for the whole algorithm (recall that m is the total number of SVs). This may be much better than other methods (see Section 4 for details), yet we would like to take further advantage of kernels which are rank degenerate, that is, if $k(x, x')$ can be approximated on the training set X by $z(x)z(x')^\top$ where $z(x) \in \mathbb{R}^l$ with $l \ll m$ (in the following we assume that this approximation is exact). See (Schölkopf & Smola, 2002; Fine & Scheinberg, 2000) for details how such an approximation can be obtained efficiently. This means that the kernel matrix to be used in the quadratic soft margin algorithm can be written as

$$K = Z^\top Z + \lambda \mathbf{1} \tag{7}$$

where $Z_{ij} := z_j(x_i)$ and $Z \in \mathbb{R}^{n \times l}$. Extending the work of Fine and Scheinberg (2000) recently an algorithm was proposed by Smola and Vishwanathan (2003) which allows one to find an $LDL^\top$ factorization of K in $O(nl^2)$ time and which can be updated efficiently in $O(nl)$ time. This technique of using a low rank matrix decomposition is faster than using the full matrix inversion. Technical details about the factorization can be found in Vishwanathan (2002).

Alternatively, a Sherman-Morrison-Woodbury formulation could be used, albeit at the expense of much reduced numerical stability.

4. Experiments

Since the main goal of the paper is to give an *algorithmic* improvement over existing SVM training algorithms, we will not report generalization performance figures here.[3] Instead, we will compare our method with the performance of the NPA algorithm for the quadratic soft-margin formulation (Keerthi et al., 1999) and the popular SMO algorithm for the linear soft-margin formulation (Platt, 1999), as both are comparable in speed to other methods such as SVMLight (Joachims, 1999).

Following (Keerthi et al., 1999), we compare the number of kernel evaluations performed by a support vector algorithm as an effective measure of its speed. The latter is relevant in particular if the kernel evaluations are expensive (this happens to be the case with most custom-tailored kernels). Other measures are fraught with difficulty, since comparing different implementations, compilers, platforms, operating systems, etc., causes a large amount of variation even between identical algorithms.

4.1. Experimental Setup and Datasets

We uniformly used a value of $1e{-}5$ for the error bound i.e. we stop the algorithm when $\alpha_i \max(y_i f(x_i) - 1, 0) + (C - \alpha_i) \max(1 - y_i f(x_i), 0) < 10^{-5}$. This means that we stop if the contribution of every point to the KKT gap (Schölkopf & Smola, 2002) is less than 10^{-5}. This is a much more stringent requirement than what can typically be satisfied with other optimization codes in practice, the exception being interior point methods.

The NPA results are those reported by Keerthi et al. (1999). For the sake of comparability we used the same kernel, namely a Gaussian RBF kernel, for all the experiments (including the SMO and the NPA), i.e.

$$k(x, x') = \exp\left(-\frac{1}{2\sigma^2}\|x - x'\|^2\right). \tag{8}$$

The datasets chosen for our experiments are described in Table 1. The *Spiral* dataset was proposed by Alexis Wieland of MITRE Corporation and it is available from the CMU Artificial Intelligence repository. Both *WSPBC* and the *Adult* datasets are available from the UCI Machine Learning repository (Blake & Merz, 1998). We used the same values of σ^2 as in Keerthi

[3] It is a common misperception that different SV optimization algorithms lead to *different* estimation results. This is not true, however, as long as all optimizers actually minimize the same objective function (things differ if approximations are made). Hence it is irrelevant to report *generalization performance* results in a paper concerned with the *speed* of an estimator.

Dataset	Size	Dimensions	σ^2
Spiral	195	2	0.5
WPBC	683	9	4
Adult-1	1,605	123	10
Adult-4	4,781	123	10
Adult-7	16,100	123	10

Table 1. Datasets used for our experiments

et al. (1999) and Platt (1999) to allow for a fair comparison. Note that in general lower values of the regularization parameter imply larger number of Support Vectors and vice versa. We compare the scaling behaviour of various algorithms with respect to the number of Support Vectors by varying the regularization parameter. Experimental results can be found in Figures 2, 3 and 4.

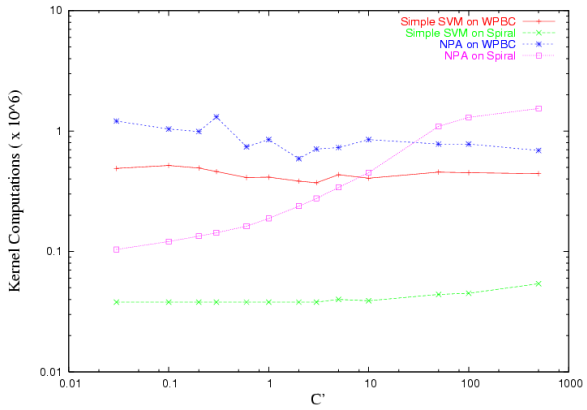

Figure 2. Performance comparison between SimpleSVM and NPA on the *Spiral* and *WPBC* datasets.

4.2. Discussion of the Results

As can be seen SimpleSVM outperforms the NPA considerably on all five datasets. For instance, on the *Spiral* dataset the SimpleSVM is an order of magnitude faster than the NPA (for $C' > 5$). On the *Adult-4* dataset for some values of the regularization constant the SimpleSVM algorithm is nearly 50 times faster than the NPA. SimpleSVM also outperforms the SMO when the number of margin SV's is reasonably small. For the extreme case when the number of margin SV's is a significant fraction of the dataset SMO tends to require fewer number of kernel computations. This is exactly what one expects from an algorithm which is geared towards the situation where there are only small numbers of margin SVs.

Furthermore, unlike NPA and SMO, SimpleSVM's runtime behaviour, given by the number of kernel evaluations, does not critically depend on the value of the regularization constant. It strictly outperforms NPA, in most cases by more than one order of magnitude. Given a Support Vector set, the solution obtained by SimpleSVM is *exact* within machine precision, whereas algorithms such as NPA and SMO will only yield *approximate* expansions for the same Support Vector set.

The good performance is due to the fact that often, e.g., in the quadratic soft-margin case, we observed that, we do not require to cycle through the dataset many times (at most 2 - 3), indicating that a "wrong" support vector is rarely picked up or removed from the active set.

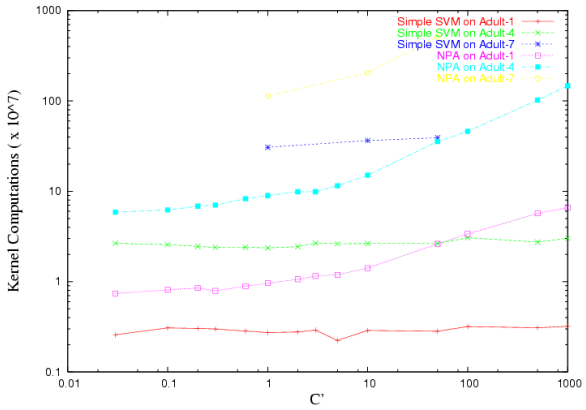

Figure 3. Performance comparison between SimpleSVM and NPA on the *Adult* datasets.

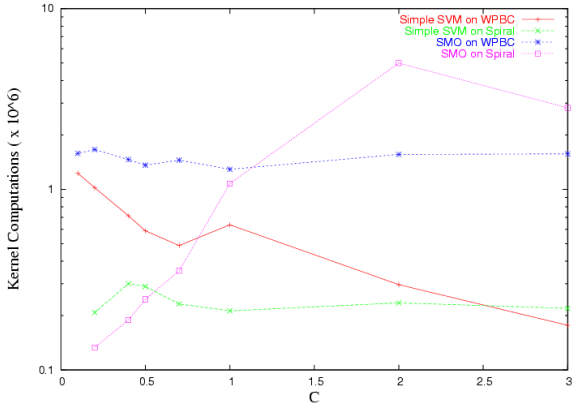

Figure 4. Performance comparison between SimpleSVM and SMO on the *Spiral* and *WPBC* datasets. Note that here we use the linear soft margin formulation as opposed to the quadratic soft margin formulation of Figure 2. Consequently the runtime of SimpleSVM in the current graph is different.

5. Summary and Outlook

We presented a new SV training algorithm that is efficient, intuitive, fast and numerically stable. It significantly outperforms other iterative algorithms like the NPA and SMO in terms of the number of kernel computations. Moreover, it does away with the problem of overall optimality on all previously seen data that was one of the major drawbacks of Incremental SVM, as proposed by Cauwenberghs and Poggio (2001).

It should be noted that SimpleSVM performs particularly well whenever the datasets are relatively "clean", that is, whenever the number of SVs is rather small. On noisy data, on the other hand, methods such as SMO may be preferable to our algorithm. This is mainly due to the fact that we need to store the $LDL^\top$ factorization of H_{ww} in main memory (256 MB of main memory suffice to store a matrix corresponding to as many as $10,000$ SVs). Storage therefore becomes a serious limitation of SimpleSVM when applied to generic dense matrices on large noisy datasets. One possibility to address this problem is to use low-rank approximation methods which make the problem amenable to the low-rank factorizations described in Section 3.2.

Due to the $LDL^\top$ factorization used in finding the SV solution our algorithm is numerically more stable than using a direct matrix inverse. This helps us deal with round off errors that can plague other algorithms. We suspect that similar modifications could be successfully applied to other algorithms as well.

At present, our algorithm does not use any kind of kernel cache to reuse kernel computations (which would further reduce the number of kernel evaluations required). The design of an efficient caching scheme to scale up the behaviour of our algorithm is currently being investigated.

It can be observed that the addition of a vector to the support vector set is entirely reversible. Using this property and following the derivation in Cauwenberghs and Poggio (2001) we can calculate leave one out errors.

Acknowledgements

S.V.N. Vishwanathan was supported by a Infosys Fellowship. Alexander J. Smola was supported by a grant of the Australian Research Council. We thank Prof. Sathiya Keerthi for useful comments and discussion.

References

Bennett, K. P., & Mangasarian, O. L. (1993). Multicategory separation via linear programming. *Optimization Methods and Software*, *3*, 27–39.

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

Cauwenberghs, G., & Poggio, T. (2001). Incremental and decremental support vector machine learning. *Advances in Neural Information Processing Systems 13* (pp. 409–415). MIT Press.

Cortes, C., & Vapnik, V. (1995). Support vector networks. *Machine Learning*, *20*, 273–297.

Fine, S., & Scheinberg, K. (2000). *Efficient SVM training using low-rank kernel representation* (Technical Report). IBM Watson Research Center, New York.

Fletcher, R. (1989). *Practical methods of optimization.* New York: John Wiley and Sons.

Joachims, T. (1999). Making large-scale SVM learning practical. *Advances in Kernel Methods—Support Vector Learning* (pp. 169–184). Cambridge, MA: MIT Press.

Keerthi, S. S., Shevade, S. K., Bhattacharyya, C., & Murthy, K. R. K. (1999). *A fast iterative nearest point algorithm for support vector machine classifier design* (Technical Report Technical Report TR-ISL-99-03). Indian Institute of Science, Bangalore.

Platt, J. (1999). Fast training of support vector machines using sequential minimal optimization. *Advances in Kernel Methods—Support Vector Learning* (pp. 185–208). Cambridge, MA: MIT Press.

Roobaert, D. (2000). DirectSVM: A simple support vector machine perceptron. *Neural Networks for Signal Processing X—Proceedings of the 2000 IEEE Workshop* (pp. 356–365). New York: IEEE.

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

Smola, A. J., & Vishwanathan, S. V. N. (2003). Cholesky factorization for rank-k modifications of diagonal matrices. *SIAM Journal of Matrix Analysis.* in preparation.

Vishwanathan, S. V. N. (2002). *Kernel methods: Fast algorithms and real life applications.* Doctoral dissertation, Indian Institute of Science, Bangalore, India.