Manuscript Number: 2752

Decomposition Methods for Linear Support Vector Machines

Wei-Chun Kao, Kai-Min Chung, Chia-Liang Sun, and Chih-Jen Lin
Department of Computer Science and
Information Engineering
National Taiwan University
Taipei 106, Taiwan
cjlin@csie.ntu.edu.tw

Abstract In this paper, we show that decomposition methods with alpha seeding are extremely useful for solving a sequence of linear SVMs with more data than attributes. This strategy is motivated from (Keerthi and Lin 2003) which proved that for an SVM with data not linearly separable, after C is large enough, the dual solutions are at the same face. We explain why a direct use of decomposition methods for linear SVMs is sometimes very slow and then analyze why alpha seeding is much more effective for linear than nonlinear SVMs. We also conduct comparisons with other methods which are efficient for linear SVMs, and demonstrate the effectiveness of alpha seeding techniques for helping the model selection.

1 Introduction

Solving linear and non-linear support vector machines (SVM) has been considered two different tasks. For linear SVM without too many attributes in data instances, people have been able to train millions of data (e.g. (Mangasarian and Musicant 2000)); but for other types of problems, in particular, non-linear SVMs, the requirement of huge memory as well as computational time has prohibited us from solving very large problems. Currently, the decomposition method, a specially designed optimization procedure, is one of the main tools for non-linear SVMs. In this paper, we show the drawbacks of existing decomposition methods, in particular SMO-type algorithms, for linear SVMs. To remedy these drawbacks, motivating from Theorem 3 of (Keerthi and Lin 2003), we develop effective strategies so that decomposition methods become efficient for solving linear SVMs.

First, we briefly describe linear and non-linear SVMs. Given training vectors $x_i \in R^n, i = 1, \ldots, l$, in two classes, and a vector $y \in R^l$ such that $y_i \in \{1, -1\}$, the standard SVM formulation (Cortes and Vapnik 1995) is as follows:

$$\begin{aligned} \min_{w,b,\xi} \quad & \frac{1}{2}w^T w + C\sum_{i=1}^{l} \xi_i \\ \text{subject to} \quad & y_i(w^T\phi(x_i) + b) \geq 1 - \xi_i, \\ & \xi_i \geq 0, i = 1, \ldots, l. \end{aligned} \tag{1.1}$$

If $\phi(x) = x$, usually we say (1.1) is the form of a linear SVM. On the other hand, if ϕ maps x to a higher dimensional space, (1.1) a non-linear SVM.

For a non-linear SVM, the number of variables depends on the size of w and can be very large (even infinite), so people solve the following dual form:

$$\begin{aligned} \min_{\alpha} \quad & \frac{1}{2}\alpha^T Q\alpha - e^T\alpha \\ \text{subject to} \quad & y^T\alpha = 0, \\ & 0 \leq \alpha_i \leq C, i = 1, \ldots, l, \end{aligned} \tag{1.2}$$

where Q is an $l \times l$ positive semi-definite matrix with $Q_{ij} = y_i y_j \phi(x_i)^T\phi(x_j)$, e is the vector of all ones, and $K(x_i, x_j) = \phi(x_i)^T\phi(x_j)$ is the kernel function. (1.2) is solvable because its number of variables is the size of the training set, independent of the dimensionality of $\phi(x)$.

The primal and dual relation shows

$$w = \sum_{i=1}^{l} \alpha_i y_i \phi(x_i), \tag{1.3}$$

so

$$\text{sgn}(w^T\phi(x) + b) = \text{sgn}(\sum_{i=1}^{l} \alpha_i y_i K(x_i, x) + b)$$

is the decision function.

Unfortunately, for large training set, Q becomes such a huge dense matrix that traditional optimization methods cannot be directly applied. Currently, some specially designed approaches such as decomposition methods (Osuna, Freund, and Girosi 1997; Joachims 1998; Platt 1998) and finding the nearest points of two convex hulls (Keerthi, Shevade, Bhattacharyya, and Murthy 2000) are major ways of solving (1.2).

On the other hand, for linear SVMs, if $n \ll l$, w is not a huge vector variable, so (1.1) can be solved by many regular optimization methods. As at the optimal solution $\xi_i = \max(0, 1 - y_i(w^T x_i + b))$, in a sense we mainly have to find out w and b. Therefore, if the number of attributes n is small, there are not many main variables w and b in (1.1) no matter how large the training set is. Currently, on a normal computer, people have been able to train a linear SVM with millions of data (e.g. (Mangasarian and Musicant 2000)); but for a non-linear SVM with much fewer data, we already need more computational time as well as computer memory.

Therefore, it is natural to ask whether in an SVM software linear and non-linear SVMs should be treated differently and solved by two methods. It is also interesting to see how capable non-linear SVM methods (e.g. decomposition methods) are for linear SVMs. Here, by linear SVMs we mean those with $n < l$. If $n \geq l$, the dual form (1.2) has fewer variables than w of the primal, a situation similar to nonlinear SVMs. As the rank of Q is less than or (usually) equal to $\min(n, l)$, the linear SVMs we are interested in here are those with low-ranked Q.

Recently, in many situations, linear and non-linear SVMs are considered together. Some approaches (Lee and Mangasarian 2001; Fine and Scheinberg 2001) approximate non-linear SVMs by different problems which are in the form of linear SVMs (Lin and Lin 2003; Lin 2002) with $n \ll l$. In addition, for non-linear SVM model selection with Gaussian kernel, (Keerthi and Lin 2003) proposed an efficient method which has to conduct linear SVMs model selection first (i.e. linear SVMs with different C). Therefore, it is important to discuss optimization methods for linear and non-linear SVMs at the same time.

In this paper, we focus on decomposition methods. In Section 2, we show that existing decomposition methods are inefficient for training linear SVMs. Section 3 demonstrates theoretically and experimentally that the alpha seeding technique is particularly useful for linear SVMs. Some implementation issues are in Section 4. The decomposition method with alpha seeding is compared with existing linear

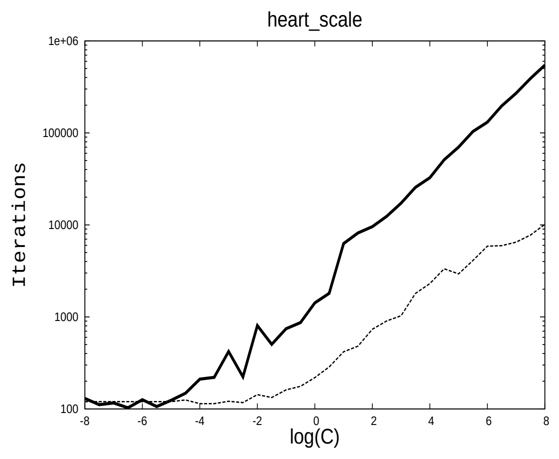

Figure 1: Number of decomposition iterations for solving SVMs with linear (the thick line) and RBF (the thin line) kernel.

SVM methods in Section 5. We then, in Section 6, apply the new implementation to solve a sequence of linear SVMs required for the model selection method in (Keerthi and Lin 2003). Final discussion and concluding remarks are in Section 7.

2 Drawbacks of Decomposition Methods for Linear SVMs with $n \ll l$

The decomposition method is an iterative procedure. In each iteration, the index set of variables is separated to two sets B and N, where B is the working set. Then in that iteration variables corresponding to N are fixed while a sub-problem on variables corresponding to B is minimized. If q is the size of the working set B, in each iteration, only q columns of the Hessian matrix Q are required. They can be calculated and stored in the computer memory when needed. Thus, unlike regular optimization methods which usually require the access of the whole Q, here, the memory problem is solved. Clearly, decomposition methods are specially designed for nonlinear SVMs. Throughout this paper, we use the term DSVM to refer the solver of SVM that adopts the decomposition method, e.g. LIBSVM (Chang and Lin 2001b) and SVM^{light} (Joachims 1998). When the size of its working set is two, we say it is of SMO-type (Platt 1998).

Unlike popular optimization methods such as Newton or quasi-Newton which

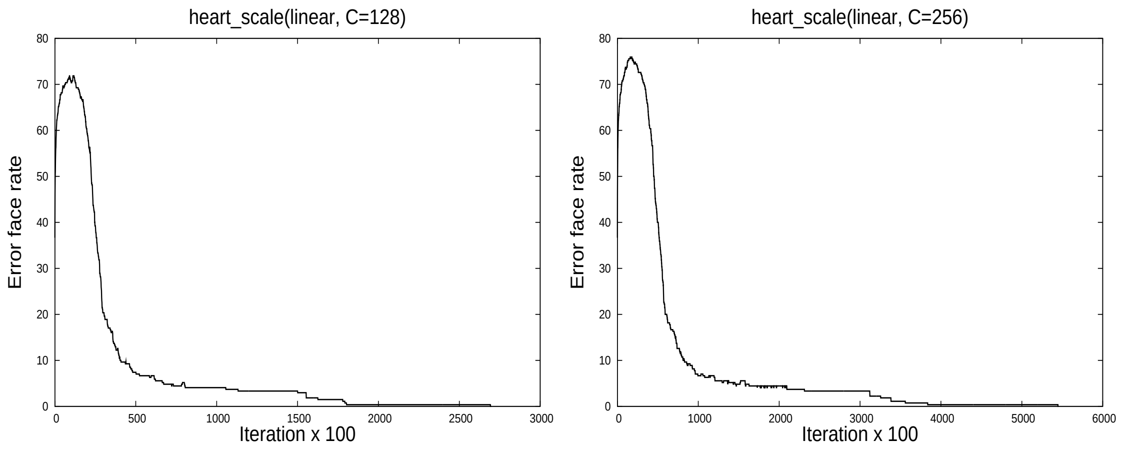

Figure 2: The error-face rate (i.e., the difference between the current face and the one at the final solution) for solving linear SVM with $C = 128$ and $C = 256$.

enjoy fast convergence, decomposition methods converges slowly as in each iteration only very few variables are updated. We will show that the situation is even worse when solving linear SVMs.

It has been demonstrated (e.g. (Hsu and Lin 2002b)) by experiments that if C is large and the Hessian matrix Q is not well-conditioned, decomposition methods converge very slowly. For linear SVMs, if $n \ll l$, then Q is a low-rank and hence ill-conditioned matrix. In Figure 1, we demonstrate a simple example by using the problem heart from the statlog database (Michie, Spiegelhalter, and Taylor 1994). Each attribute is scaled to $[-1, 1]$. We use LIBSVM (Chang and Lin 2001b) to solve linear and nonlinear (RBF kernel, $e^{-\|x_i - x_j\|^2/(2\sigma^2)}$ with $1/(2\sigma^2) = 1/n$) SVMs with $C = 2^{-8}, 2^{-7.5}, \ldots, 2^8$ and present the number of iterations. Though two different problems are solved (in particular, their Q_{ij}'s are in different ranges), Figure 1 clearly indicates the huge number of iterations for solving the linear SVMs. Note that for linear SVMs, the slope is greater than that for nonlinear SVMs and is very close to one, especially when C is large. This means that a doubled C leads to a doubled number of iterations.

The following theorems that hold only for linear SVMs help us to realize the difficulty the decomposition methods suffer from. Theorem 2 can further explain why the number of iterations is nearly doubled when C is doubled.

Theorem 1 *(Keerthi and Lin 2003) The dual linear SVM has the following prop-*

erties:

- *There is C^* such that for all $C \geq C^*$, there are optimal solutions at the same face.*
- *In addition, for all $C \geq C^*$, the primal solution w is the same.*

By the face of α we mean three types of value of each α_i: (i) lower-bounded, i.e., $\alpha_i = 0$, (ii) upper-bounded, i.e., $\alpha_i = C$, and (iii) free, i.e., $0 < \alpha_i < C$. More precisely, the face of α can be represented by a length-l vector whose components are in $\{\text{lower-bounded}, \text{upper-bounded}, \text{free}\}$.

This theorem indicates that after $C \geq C^*$, exponentially-increased numbers of iterations are wasted in order to obtain the same primal solution w. Even if we could detect C^* and stop training SVMs, for C not far below C^*, the number of iterations may be already huge. Therefore, it is important to have an efficient linear SVM solver which could handle both large and small C.

Next, we try to explain the nearly doubled iterations by the hardness of locating faces of the dual solution α.

Theorem 2 *Assume that any two parallel hyperplanes in the feature space do not contain more than $n+1$ points of $\{x_i\}$ on them. We have*[*]

1. *For any optimal solution of (1.2), it has no more than $n+1$ free components.*
2. *There is C^* such that after $C \geq C^*$, all optimal solutions of (1.2) share at least the same $l-n-1$ bounded α variables.*

The proof is available in the technical report (Chung, Kao, Sun, and Lin 2002). This result indicates that when $n \ll l$, most components of optimal solutions are at bounds. Furthermore, dual solutions at C and $2C$ share at least the same $l-2(n-1)$ upper and lower-bounded components. If upper-bounded α_i at C remains upper-bounded at $2C$, a direct use of decomposition methods means that

[*]Note that a pair of two parallel hyperplane is decided by $n+1$ numbers (n number decides one hyperplane in the feature space R^n; and another one decides the other hyperplane parallel to it.) So the assumption of Theorem 2 would be violated if m linear equations in $n+1$ variables, where $m > n+1$, have solutions. The occurrence of this scenario is of measure zero. This explains that the assumption of Theorem 2 is genetic.

Table 3.1: Comparison of iterations (linear kernel); with and without alpha seeding.

	α-seeding			without α-seeding		
Problem	#iter	C^*	w^Tw	#total iter	#iter($C = 2^{7.5}$)	#iter($C = 2^8$)
heart	27231	$2^{3.5}$	5.712	2449067	507122	737734
australian	79162	$2^{2.5}$	2.071	20353966	3981265	5469092
diabetes	33264	$2^{6.5}$	16.69	1217926	274155	279062
german	277932	2^{10}	3.783	42673649	6778373	14641135
web	24044242	unstable	unstable	$\geq 10^8$	74717242	$\geq 10^8$
adult	3212093	unstable	unstable	$\geq 10^8$	56214289	84111627
ijcnn	590645	2^6	108.6	41440735	8860930	13927522

Table 3.2: Comparison of iterations (RBF kernel); with and without alpha seeding.

Problem	l	n	α-seeding	without α-seeding
heart	270	13	43663	56792
australian	690	14	230983	323288
diabetes	768	8	101378	190047
german	1000	24	191509	260774
web	49749	300	633788	883319
adult	32561	123	2380265	4110663
ijcnn	49990	22	891563	1968396

α_i is updated from 0 to C and from 0 to $2C$, respectively. Thus, we anticipate the efforts are roughly doubled. We confirm this explanation by comparing the error-face rate (i.e., the difference between the current face and the one at the final solution) with $C = 2^7$ and $C = 2^8$. As shown in Figure 2, two curves are quite similar except that the scale of x-axis differs by twice. This indicates that α travels similar faces for $C = 2^7$ and $C = 2^8$, and the number of iterations spent on each face with $C = 2^8$ is roughly doubled.

3 Alpha Seeding for Linear SVMs

Theorem 2 implies that for linear SVMs, dual solutions may share many upper and lower-bounded variables. Therefore, we conjecture that if α^1 is an optimal solution at $C = C_1$, then $\alpha^1 C_2/C_1$ can be a very good initial point for solving (1.2) with $C = C_2$. The reason is that $\alpha^1 C_2/C_1$ is at the same face as α^1 and it is likely to be at a similar face of one optimal solution of $C = C_2$. This technique, called alpha seeding, was originally proposed for SVM model selection (DeCoste and Wagstaff 2000) where several (1.2) with different C have to be solved. Earlier work which focus on nonlinear SVMs mainly uses alpha seeding as a heuristic. Now for linear SVMs, the speed could be significantly boosted due to the above analysis.

The following theorem further supports the use of alpha seeding:

Theorem 3 *There are two vectors A, B, and a number C^* such that for any $C \geq C^*$, $AC + B$ is an optimal solution of (1.2).*

The proof is in the technical report (Chung, Kao, Sun, and Lin 2002). If $A_i > 1$, $A_iC + B > C$ after C is large enough and this violates the bounded constraints in (1.2). Similarly, A_i cannot be less than zero, so $0 \leq A_i \leq 1$. Therefore, we can consider the following three situations of vectors A and B:

1. $0 < A_i \leq 1$,
2. $A_i = 0, B_i = 0$,
3. $A_i = 0, B_i > 0$.

For the second case, $\alpha_i^1 \frac{C_2}{C_1} = A_iC_2 + B_i = 0$, and for the first case, $A_iC \gg B_i$ after C is large enough. Therefore, $\alpha_i^1 \frac{C_2}{C_1} = A_iC_2 + B_i\frac{C_2}{C_1} \approx A_iC_2 + B_i$. For both cases, alpha seeding is very useful. On the other hand, using Theorem 2, there are few ($\leq n + 1$) components satisfying the third case.

Next, we conduct some comparisons between DSVM with and without alpha seeding. Here, we consider two-class problems only. Some statistics of the data sets used are in Table 3.2. The four small problems are from the statlog collection (Michie, Spiegelhalter, and Taylor 1994). The problem adult is compiled by Platt (1998) from the UCI "adult" data set (Blake and Merz 1998). Problem web is also from Platt. Problem ijcnn is from the first problem of IJCNN challenge 2001 (Prokhorov 2001). Note that we use the winner's transformation of the raw data (Chang and Lin 2001a).

We train linear SVMs with $C \in \{2^{-8}, 2^{-7.5}, \ldots, 2^{8}\}$. That is, $[2^{-8}, 2^{8}]$ is discretized to 33 points with equal ratio. Table 3.1 presents the total number of iterations of training 33 linear SVMs using the alpha seeding approach. We also individually solve them by LIBSVM and list the number of iterations (total, $C = 2^{7.5}$, and $C = 2^8$). The alpha seeding implementation, will be described in detail in Section 4. We also list the approximate C^* for which linear SVMs with $C \geq C^*$ have the same decision function. In addition, the constant $w^T w$ after $C \geq C^*$ is also given. For some problems (e.g. web and adult), $w^T w$ has not reached a constant until C is very large so we indicate them as "unstable" in Table 3.1.

To demonstrate that alpha seeding is much more effective for linear than nonlinear SVMs, Table 3.2 presents the number of iterations using the RBF kernel $K(x_i, x_j) = e^{-\|x_i - x_j\|^2/(2\sigma^2)}$ with $1/2\sigma^2 = 1/n$. It is clear that the saving of iterations by using alpha seeding is marginal. In addition, comparing to the "total iter." column in Table 3.1, we confirm again the slow convergence for linear SVMs if without alpha seeding.

The alpha seeding approach performs so well to the point that its total number of iterations is much less than solving one single linear SVM with the original decomposition implementation. Therefore, if we intend to solve one linear SVM with a particular C, it may be more efficient to solve one with small initial C_0 and then use the proposed alpha seeding method by gradually increasing C.

Furthermore, since we have solved linear SVMs with different C, model selection by cross-validation is already done. From the discussion in Section 2, if without alpha seeding, solving several linear SVMs is time consuming and the model selection is not an easy task.

Note that in Table 3.1, web is the most difficult problem and requires the largest number of iterations. Theorem 2 helps to explain this: since web's large number of attributes might lead to more free variables during iterations or at the final solution, alpha seeding is less effective.

4 Implementation

Though the concept is so simple, to have an efficient and elegant implementation, there are many considerations which will be discussed in this section.

Most DSVM implementations maintain the gradient vector of the dual objective function during iterations. The gradient is used for selecting the working set or checking the stopping condition. In the non-linear SVM, calculation of the gradient $Q\alpha - e$ requires $O(l^2 n)$ operations ($O(n)$ for each kernel evaluation), which are expensive. Therefore, many DSVM software use $\alpha = 0$ as the initial solution, which makes that the initial gradient $-e$ is immediately available. However, in DSVM with alpha seeding, the initial solution is obtained from the last problem, so the initial gradient is not a constant vector any more. Fortunately, for linear SVMs, the situation is not as bad as that in the non-linear SVM. In this case, the kernel matrix is of the form $Q = X^T X$, where $X = [y_1 x_1, \ldots, y_l x_l]$ is an n by l matrix. So we can calculate the gradient by $Q\alpha - e = X^T(X\alpha) - e$, which requires only $O(ln)$ operations. The first decomposition software which uses this trick for linear SVMs is SVM^{light} (Joachims 1998).

Similarly, if α is changed by $\Delta\alpha$ between two consecutive iterations, then the change of gradient is $Q(\Delta\alpha) = X^T(X\Delta\alpha)$. Since there are only q non-zero elements in $\Delta\alpha$, the gradient can be updated with $O(nq) + O(ln) = O(ln)$ operations, where q is the size of working set. Note that because $l \gg q$, increasing q from 2 to some other small constant will not effect the time of updating gradient. In contrast, for non-linear SVMs, if Q is not in the cache, the cost for updating gradient is by computing $Q(\Delta\alpha)$, which, requiring q columns of Q, takes $O(lnq)$-time. Thus, while the implementation of non-linear SVMs may choose SMO-type implementation (i.e., $q = 2$) to have less cost per iteration, we should use a larger q for linear SVMs as the gradient update is independent of q and the number of total iterations may be reduced.

As mentioned above, in the non-linear SVM, constructing the kernel matrix Q is expensive. So the cache for storing recently used elements of Q is a must. However, in the linear SVM, either the kernel matrix Q or the cache is not needed any more.

5 Comparison with Other Approaches

It is interesting to compare the proposed alpha seeding approach with efficient methods for linear SVMs. In this section, we consider Active SVM (ASVM) (Mangasarian and Musicant 2000) and Lagrangian SVM (LSVM) (Mangasarian and Musicant 2001).

DSVM, ASVM , and LSVM solve slightly different formulations, so it is difficult to conduct a fair comparison. However, our goal here is only to demonstrate that, with alpha seeding, decomposition methods, can be several times faster and competitive with other linear-SVM methods. In the following we briefly describe the three implementations.

DSVM is the standard SVM which uses the dual formulation (1.2). ASVM and LSVM both consider a square error term in the objective function:

$$\min_{w,b,\xi} \quad \frac{1}{2}(w^T w + b^2) + C\sum_{i=1}^{l} \xi_i^2. \tag{5.1}$$

Then, the dual problem of (5.1) is

$$\begin{aligned} \min_{\alpha} \quad & \frac{1}{2}\alpha^T(Q + yy^T + \frac{I}{2C})\alpha - e^T\alpha \\ \text{subject to} \quad & 0 \le \alpha_i, \quad i = 1, \ldots, l, \end{aligned} \tag{5.2}$$

where I is the identity matrix. The solution of (5.2) has far more free components than that of (1.2) as upper-bounded variables of (1.2) are likely to be free now. With different formulations, their stopping conditions are not exactly the same. We use conditions from similar derivations and details are discussed in (Chung, Kao, Sun, and Lin 2002).

In this experiment, we consider LIBSVM for DSVM (with and without alpha seeding). For ASVM, we directly use the authors' C++ implementation available at

http://www.cs.wisc.edu/dmi/asvm. The authors of LSVM provide only MATLAB programs so we implement it by modifying LIBSVM. The experiments were done on an Intel Xeon 2.8GHz machine with 1024MB RAM using the gcc compiler.

Using the same benchmark problems as in Section 3, we perform comparisons in Table 5.1 as follows: for each problem, we randomly select two thirds data for training and leave the remaining for testing. For algorithms except DSVM without alpha seeding, five-fold cross validation with $C = 2^{-10}, 2^{-9.5}, \ldots, 2^8$ on the training set is conducted. For DSVM without alpha seeding, as the training time is huge, only C up to 2^3 is tried. Then using the C which gives the best cross-validation rate, we train a model and predict the test data. Both testing accuracy and the total computational time are reported.

Table 5.1: Comparison of different approaches for linear SVMs. Acc.: test accuracy using the parameter obtained from cross validation. Time (in seconds): total training time of five-fold cross validation by trying $C = 2^{-10}, 2^{-9.5}, \ldots, 2^{8}$ (or 2^3 if specified).

	Decomposition methods (LIBSVM)			Methods for linear SVMs		
		With α seeding	Without α seeding		ASVM	LSVM
problem	Acc.	Time: $C \leq 2^8 (C \leq 2^3)$	Time: $C \leq 2^3$	Acc.	Time	Time
australian	85.51	4.1 (3.6)	7.0	88.70	8.1	2.3
heart	85.56	1.4 (0.9)	1.0	85.56	5.8	1.8
diabetes	79.69	2.4 (2.3)	1.6	81.64	6.2	2.0
german	73.65	10.2 (6.2)	18.2	73.95	16.4	9.0
ijcnn	92.65	981.9 (746.0)	3708.6	92.51	725.2	17496.4
adult	85.02	1065.9 (724.8)	12026.8	84.90	3130.7	13445.4
web	98.67	18035.3 (1738.2)	7035.6	98.63	10315.9	43060.1

In Table 5.1, alpha seeding with C up to 2^8 is competitive with solving C up to only 2^3 without alpha seeding. For these problems, considering $C \leq 2^3$ is enough, and if alpha seeding stops at 2^3 as well, it is several times faster than without alpha seeding.

Since alpha seeding is not applied to ASVM and LSVM, we admit that their computational time can be further improved. Results here also serve as the first comparison between ASVM and LSVM. Clearly ASVM is faster. Moreover, due to the huge computational time, we set the maximal iterations of LSVM to be 1,000. For problems adult and web, after C is large, iteration limit is reached before stopping conditions are satisfied.

In addition to comparing DSVM with ASVM and LSVM, we compare the performance of SMO-type ($q = 2$) and that with a larger working set ($q = 30$) for DSVM with alpha seeding in Table 5.2 by modifying the software SVM^{light}, which allows adjustable q. All default settings of SVM^{light} are used. In this experiment, we solve linear SVMs with $C = 2^{-8}, 2^{-7.5}, \ldots, 2^{8}$ and report their computational time and total number of iterations. Note that when q is two, SVM^{light} and LIBSVM use the same algorithm and differ only in some implementation details.

The results in Table 5.2 show that the implementation with a larger working set takes less time than that with a smaller one. This is consistent with our earlier statement that for linear SVMs, SMO-type decomposition methods are

Table 5.2: Comparison of different subproblem size in decomposition methods for linear SVMs (time in second; q: size of the working set). Note that time here is shorter than that in Table 5.1 because we do not perform cross validation.

	Decomposition methods with alpha seeding			
	$q = 2$ (SVM^{light})		$q = 30$ (SVM^{light})	
problem	total iter.	time	total iter.	time
australian	50145	0.71	6533	1.19
heart	25163	0.21	1317	0.33
diabetes	30265	0.4	5378	0.31
german	182051	2.9	6006	3.68
ijcnn	345630	185.85	79847	115.03
adult	1666607	1455.2	414798	516.71
web	N/A*	N/A*	2673578	1885.1

*: SVM^{light} faced numerical difficulties.

less favorable.

Regarding the computational time reported in this section, we must exercise the caution that quite a few implementation details may affect it. For example, each iteration of ASVM and LSVM involves several matrix-vector multiplications. Hence, it is possible to use finely-tuned dense linear algebra subroutines. For the LSVM implementation here, by using ATLAS (Whaley, Petitet, and Dongarra 2000), for large problems, the time is reduced by two third. Thus, it is possible to further reduce the time of ASVM in Table 5.1 though we find it too complicated to modify the authors' program. Using such tools also means X is considered as a dense matrix. In contrast, X is currently treated as a sparse matrix in both LIBSVM and SVM^{light}, where each iteration requires two matrix-vector multiplications $X(X^T(\alpha^{k+1} - \alpha^k))$. This sparse format creates some overheads when data are dense.

6 Experiments on Model Selection

If the RBF kernel

$$K(x_i, x_j) = e^{-\|x_i - x_j\|^2/(2\sigma^2)}$$

is used, (Keerthi and Lin 2003) proposes the following model selection procedure for finding good C and σ^2:

Algorithm 1 *Two-line model selection*

1. *Search for the best C of linear SVMs and call it $\tilde{C}$.*

2. *Fix $\tilde{C}$ from step 1 and search for the best (C, σ^2) satisfying* $\log \sigma^2 = \log C - \log \tilde{C}$ *using the RBF kernel.*

That is, we solve a sequence of linear SVMs first and then a sequence of nonlinear SVMs with the RBF kernel. The advantage of Algorithm 1 over an exhausted search of the parameter space is that only parameters on two lines are considered. If decomposition methods are directly used for both linear and nonlinear SVMs here, due to the huge number of iterations, solving the linear SVMs becomes the bottleneck. Our goal is to show that by applying the alpha seeding technique to linear SVMs, the computational time spent on the linear part becomes similar to that on the nonlinear SVMs.

Earlier in (Keerthi and Lin 2003), due to the difficulty on solving linear SVMs, Algorithm 1 is only tested on small two-class problems. Here, we would like to evaluate this algorithm on large multi-class data sets. We consider problems dna, satimage, letter, and shuttle, which were originally from the statlog collection (Michie, Spiegelhalter, and Taylor 1994) and were used in (Hsu and Lin 2002a). Except dna, which takes two possible values 0 and 1, each attribute of all training data is scaled to [-1,1]. Then, test data are adjusted using the same linear transformation.

Since LIBSVM contains a well-developed cross-validation procedure, we use it as the DSVM solver in this experiment. We search for $\tilde{C}$ by five-fold cross-validation on linear SVMs using uniformly spaced $\log_2 \tilde{C}$ value in $[-10, 10]$ (with grid space 1). As LIBSVM considers $\gamma = 1/2\sigma^2$ as the kernel parameter, the second step is to search for good (C, γ) satisfying

$$-1 - \log_2 \gamma = \log_2 C - \log_2 \tilde{C}. \tag{6.1}$$

We discretize $[-10, 4]$ as values of $\log_2 \gamma$ and calculate $\log_2 C$ from (6.1). To avoid that $\log_2 C$ locates in an abnormal region, we consider only points with $-2 \le \log_2 C \le 12$ so the second step may solve less SVMs than the first step. The same computational environment as that for Section 3 is used.

Since this model selection method is based on the analysis of binary SVMs, a multi-class problem has to be decomposed to several binary SVMs. We employ the "one-against-one" approach: if there are k classes of data, all $k(k-1)/2$ two-class combinations are considered. For any two classes of data, the model selection is conducted to have the best (C, σ^2). With the $k(k-1)/2$ best (C, σ^2) and corresponding decision functions, a voting strategy is used for the final prediction. In Table 6.1, we compare this approach with two versions complete grid searches. First, for any two classes of data, five-fold cross-validation is conducted on 225 points, a discretization of the $(\log_2 C, \log_2 \gamma) = [-2, 12] \times [-10, 4]$ space. The second way is from the cross-validation procedure adopted by LIBSVM for multi-class data, where a list of (C, σ^2) is selected first and then for each (C, σ^2), one-against-one method is used for estimating the cross-validation accuracy of the whole multi-class data. Therefore, for the final optimal model, $k(k-1)/2$ decision functions share the same C and σ^2. Since the same number of nonlinear SVMs are trained, the time for the two complete grid searches is exactly the same but the performance (test accuracy) may be different. There is no comparison so far, so we present a preliminary investigation here.

Table 6.1: Comparison of different model selection methods (time in second).

	Complete grid search			Algorithm 1			
	1 (C, σ^2)	$k(k-1)/2$ (C, σ^2)		Time	Time	Time	Accuracy
Problem	Accuracy	Time	Accuracy		(linear)	(non-linear)	
dna	95.62	4945	95.11	202	123	79	94.86 (94.77)
satimage	91.9	7860	92.2	1014	743	271	91.55 (90.55)
letter	97.9	56753	97.72	5365	3423	1942	96.54 (95.9)
shuttle	99.92	104904	99.94	4196	2802	1394	99.81 (99.7)

Accuracies of Algorithm 1 enclosed in parentheses are the accuracies if we search $\log_2 \tilde{C} \in [-10, 3]$ in step 1 of Algorithm 1.

Table 6.2: Mean and Standard deviation of two model selection methods (each method applied 10 times).

	Complete grid search						Algorithm 1			
	$\log_2 C$		$\log_2 \gamma$		Accuracy		$\log_2 \tilde{C}$		Accuracy	
Problem	Mean	Std.	Mean	Std.	Mean	Std.	Mean	Std.	Mean	Std.
banana	7	4.45	-0.4	1.51	87.91	0.47	-1.9	2.18	76.36	12.21
adult_small	5.4	2.37	-7.6	1.71	83.82	0.27	0.3	4.08	83.20	1.28
dna	5.4	3.34	-5	0	95.56	0.19	-	-	94.85	0.20
satimage	2.5	0.71	0.1	0.57	91.74	0.24	-	-	91.19	0.28

Table 6.1 presents experimental results. For each problem, we compare test accuracy by two complete grid searches and by Algorithm 1. The two grid searches

are represented as "1 (C, σ^2)" and "$k(k-1)/2$ (C, σ^2)," respectively, depending on how many (C, σ^2) used by the decision functions. The performance of the three approaches are very similar. However, the total model selection time of Algorithm 1 is much shorter. In addition, we also list the accuracy of Algorithm 1 in parentheses if we only search $\log_2 \tilde{C}$ value in $[-10, 3]$ in step 1. We can discover that the accuracy is consistently lower if we only search $\tilde{C}$ in this smaller region. In fact, if we search $\log_2 \tilde{C}$ in $[-10, 3]$, there are many $\log_2 \tilde{C}$ equals to 3 in this experiment. This means that $[-10, 3]$ is too small to cover good parameter regions. We make Algorithm 1 practical because of the use of alpha seeding. Otherwise, time for solving linear SVMs is a lot more so the proposed model selection does not possess any advantage.

We then investigate the stability of the new model selection approach. Due to timing restriction, we consider two smaller problems banana and adult_small tested in (Keerthi and Lin 2003). Note that the adult_small is a subset of the adult used in Section 3. It is a binary problem with 1,605 examples. Table 6.2 shows the means and standard deviations of parameters and accuracy using 10-time the "$k(k-1)/2$ (C, σ^2)" grid search and Algorithm 1. For Algorithm 1, we list only $\tilde{C}$'s variances because the variances of parameters C and σ^2, which are computed from (6.1), are less meaningful. Note that different parameters as well as accuracy by applying the same method 10 times are due to the randomness of cross-validation.

From Table 6.2, we can see that although the performance (testing accuracy and consumed time) of the model selection Algorithm 1 is good, it might be less stable. That is, the variance of accuracy is significantly larger than that of the complete grid search method while the variances of parameters are both large. We think that in the complete grid search method, the cross-validation estimation bounds the overall error. Thus, the variances of gained parameters do not affect the testing performance. However, in the two-line search method (Algorithm 1), two-stage cross-validations are utilized. Thus, the variance in the first stage may affect the best performance of the second stage.

7 Discussion and Conclusion

It is arguable that we may have used a too strict stopping condition in DSVM when C is large. One possibility is to use the stopping tolerance that is proportional to C. This will reduce the number of iterations so that directly solving linear SVMs with large C may be possible. However, in Appendix of the technical report (Chung, Kao, Sun, and Lin 2002), we show that even in these settings, DSVM with alpha seeding still makes the computational time several times faster than the original DSVM, especially for large datasets. Moreover, too large stopping tolerance will cause DSVM stops with wrong solutions.

In conclusion, we hope that based on this work, SVM software using decomposition methods can be suitable for all types of problems, no matter $n \ll l$ or $n \gg l$.

Acknowledgments

This work was supported in part by the National Science Council of Taiwan via the grant NSC 90-2213-E-002-111. The authors thank Thorsten Joachims for the help on modifying SVM^{light} for experiments in Section 5.

References

Blake, C. L. and C. J. Merz (1998). UCI repository of machine learning databases. Technical report, University of California, Department of Information and Computer Science, Irvine, CA. Available at `http://www.ics.uci.edu/~mlearn/MLRepository.html`.

Chang, C.-C. and C.-J. Lin (2001a). IJCNN 2001 challenge: Generalization ability and text decoding. In *Proceedings of IJCNN*. IEEE.

Chang, C.-C. and C.-J. Lin (2001b). *LIBSVM: a library for support vector machines.* Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

Chung, K.-M., W.-C. Kao, C.-L. Sun, and C.-J. Lin (2002). Decomposition methods for linear support vector machines. Technical report, Department of Computer Science and Information Engineering, National Taiwan University.

Cortes, C. and V. Vapnik (1995). Support-vector network. *Machine Learning 20*, 273–297.

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

Fine, S. and K. Scheinberg (2001). Efficient svm training using low-rank kernel representations. *Journal of Machine Learning Research 2*, 243–264.

Hsu, C.-W. and C.-J. Lin (2002a). A comparison of methods for multi-class support vector machines. *IEEE Transactions on Neural Networks 13*(2), 415–425.

Hsu, C.-W. and C.-J. Lin (2002b). A simple decomposition method for support vector machines. *Machine Learning 46*, 291–314.

Joachims, T. (1998). Making large-scale SVM learning practical. In B. Schölkopf, C. J. C. Burges, and A. J. Smola (Eds.), *Advances in Kernel Methods - Support Vector Learning*, Cambridge, MA. MIT Press.

Keerthi, S. S. and C.-J. Lin (2003). Asymptotic behaviors of support vector machines with Gaussian kernel. *Neural Computation 15*(7), 1667–1689.

Keerthi, S. S., S. K. Shevade, C. Bhattacharyya, and K. R. K. Murthy (2000). A fast iterative nearest point algorithm for support vector machine classifier design. *IEEE Transactions on Neural Networks 11*(1), 124–136.

Lee, Y.-J. and O. L. Mangasarian (2001). RSVM: Reduced support vector machines. In *Proceedings of the First SIAM International Conference on Data Mining.*

Lin, K.-M. (2002). Reduction techniques for training support vector machines. Master's thesis, Department of Computer Science and Information Engineering, National Taiwan University.

Lin, K.-M. and C.-J. Lin (2003). A study on reduced support vector machines. *IEEE Transactions on Neural Networks*. To appear.

Mangasarian, O. L. and D. R. Musicant (2000). Active set support vector machine classification. In *Advances in Neural Information Processing Systems*, pp. 577–583.

Mangasarian, O. L. and D. R. Musicant (2001). Lagrangian support vector machines. *Journal of Machine Learning Research 1*, 161–177.

Michie, D., D. J. Spiegelhalter, and C. C. Taylor (1994). *Machine Learning, Neural and Statistical Classification.* Englewood Cliffs, N.J.: Prentice Hall. Data available at `http://www.ncc.up.pt/liacc/ML/statlog/datasets.html`.

Osuna, E., R. Freund, and F. Girosi (1997). Training support vector machines: An application to face detection. In *Proceedings of CVPR'97*, New York, NY, pp. 130–136. IEEE.

Platt, J. C. (1998). Fast training of support vector machines using sequential minimal optimization. In B. Schölkopf, C. J. C. Burges, and A. J. Smola (Eds.), *Advances in Kernel Methods - Support Vector Learning*, Cambridge, MA. MIT Press.

Prokhorov, D. (2001). IJCNN 2001 neural network competition. Slide presentation in IJCNN'01, Ford Research Laboratory. `http://www.geocities.com/ijcnn/nnc_ijcnn01.pdf` .

Whaley, R. C., A. Petitet, and J. J. Dongarra (2000). Automatically tuned linear algebra software and the ATLAS project. Technical report, Department of Computer Sciences, University of Tennessee.

A Proof of Theorem 2

For the first result, if it were wrong, there was an optimal α with more than $n+1$ free components. From the optimality condition (C.2) and the primal-dual relation (1.3), for those $0 < \alpha_i < C$,

$$(Q\alpha + by)_i = y_i(w^T x_i + b) = 1.$$

Thus, more than $n+1$ x_i are at two parallel hyperplanes. This contradicts the assumption.

Next, we consider from Theorem 1 and define

$$A \equiv \{i \mid \text{for all optimal solutions with } C > C^*, \alpha_i \text{ is at the upper (lower) bound}\}.$$

The second result of this theorem will hold if we can prove $|A| \geq l - n - 1$. If its result is wrong, we can assume all those $i \notin A$ are from $\alpha^1, \ldots, \alpha^s$ which are optimal at $C = C_1, \ldots, C_s$, $C^* < C_1 < \cdots < C_s$ and $s > n + 1$. Thus, for any $i \notin A$, it is impossible that

$$\alpha_i^1 = \cdots = \alpha_i^s = 0 \text{ or } \alpha_i^1 = C_1, \ldots, \alpha_i^s = C_s. \tag{A.1}$$

For any convex combination with weights $0 < \lambda_j < 1$ and $\sum_{j=1}^s \lambda_j = 1$,

$$0 < \sum_{j=1}^s \lambda_j \alpha_i^j < \sum_{j=1}^s \lambda_j C_j, \quad \forall i \notin A. \tag{A.2}$$

Remember from Theorem 1, for all $C \geq C^*$, the optimal w is the same. If we define

$$\bar{\alpha} \equiv \sum_{j=1}^s \lambda_j \alpha^j,$$

then

$$\sum_{i=1}^l y_i \bar{\alpha}_i x_i = \sum_{j=1}^s \lambda_j \sum_{i=1}^l y_i \alpha_i^j x_i = \sum_{j=1}^s \lambda_j w = w.$$

Therefore, we have constructed an $\bar{\alpha}$ which is optimal at $C = \sum_{j=1}^s \lambda_j C_j > C^*$ with more than $n + 1$ free components (from (A.1) and (A.2)). This contradicts the first part of this theorem so the proof is complete.

B Proof of Theorem 3

First we note that for any given C, if $A_i C + B_i, i = 1, 2, \ldots$ are all optimal solutions of (1.2), and there are vectors A and B such that

$$\lim_{i \to \infty} A_i C + B_i = AC + B, \tag{B.1}$$

then $AC + B$ is an optimal solution as well. This is from the continuity of (1.2)'s objective function and the compactness of its feasible region.

From Theorem 3 of (Keerthi and Lin 2003), there is a C^* such that for any $\bar{C} > C^*$, there exists a linear function $AC + B$ which is optimal for (1.2) when

$C \in [C^*, \bar{C}]$. Thus, we can consider $C^1 < C^2 < \cdots$ with $\lim_{i\to\infty} C^i = \infty$ and functions $A_iC + B_i$ which are optimal solutions at $C \in [C^*, C^i]$.

Since

$$0 \leq A_iC^1 + B_i \leq C^1 \text{ and } 0 \leq A_iC^2 + B_i \leq C^2,$$

are bounded for all i, there is an infinity set I such that

$$\lim_{i\to\infty, i\in I} A_iC^1 + B_i = \alpha^1 \text{ and } \lim_{i\to\infty, i\in I} A_iC^2 + B_i = \alpha^2. \tag{B.2}$$

If $\alpha^1 \neq \alpha^2$, then two different points uniquely determine vectors A and B such that

$$\alpha^1 = AC^1 + B \text{ and } \alpha^2 = AC^2 + B. \tag{B.3}$$

For any $C > C^2$, there is $0 < \lambda < 1$ such that $C^2 = \lambda C^1 + (1-\lambda)C$ so

$$A_iC^2 + B_i = \lambda(A_iC^1 + B_i) + (1-\lambda)(A_iC + B_i). \tag{B.4}$$

Taking the limit, (B.2), (B.3), and (B.4) imply

$$AC^2 + B = \lambda(AC^1 + B) + (1-\lambda) \lim_{i\in I, i\to\infty} (A_iC + B_i).$$

Thus,

$$\lim_{i\in I, i\to\infty} (A_iC + B_i) = AC + B, \tag{B.5}$$

so (B.1) is valid. The situation for $C^* \leq C \leq C^2$ is similar. Thus $AC + B$ is optimal for (1.2), for all $C \geq C^*$. On the other hand, if $\alpha^1 = \alpha^2$, since $C^2 > C^1$,

$$\lim_{i\in I, i\to\infty} A_i(C^2 - C^1) = 0$$

from (B.2) implies

$$\lim_{i\in I, i\to\infty} A_i = 0 \text{ and } \lim_{i\in I, i\to\infty} B_i = \alpha^1 = \alpha^2.$$

By defining $A \equiv 0$ and $B \equiv \alpha^1$, for any $C \geq C^*$, (B.1) also holds so $AC + B$ is optimal for (1.2). Thus, the proof is complete. $\square$

C Stopping Criteria for Experiments in Section 5

We try to use similar stopping criteria for the four approaches. If $f(\alpha)$ is the objective function, the stopping condition of LIBSVM is

$$\begin{aligned}&\max\{-y_i \nabla f(\alpha)_i \mid y_i = 1, \alpha_i < C \text{ or } y_i = -1, \alpha_i > 0\} - \\ &\min\{-y_i \nabla f(\alpha)_i \mid y_i = -1, \alpha_i < C \text{ or } y_i = 1, \alpha_i > 0\} \leq \epsilon, \qquad \text{(C.1)}\end{aligned}$$

where $\epsilon = 0.001$. This is from the Karush-Kuhn-Tucker (KKT) condition (i.e. the optimality condition) of (1.2): α is optimal if and only if α is feasible and there is a number b and two nonnegative vectors λ and μ such that

$$\begin{gathered}\nabla f(\alpha) + by = \lambda - \mu, \\ \lambda_i \alpha_i = 0, \mu_i (C - \alpha)_i = 0, \lambda_i \geq 0, \mu_i \geq 0, i = 1, \ldots, l,\end{gathered}$$

where $\nabla f(\alpha) = Q\alpha - e$ is the gradient of $f(\alpha)$, the objective function of (1.2). This can be rewritten as

$$\begin{aligned}\nabla f(\alpha)_i + by_i \leq 0 \quad &\text{if } \alpha_i > 0, \\ \nabla f(\alpha)_i + by_i \geq 0 \quad &\text{if } \alpha_i < C.\end{aligned}$$

Using $y_i = \pm 1$ and reformulating (C.2) as upper and lower bounds of b and introducing a stopping tolerance 0.001, we have (C.1). For BSVM, without the linear constraint $y^T \alpha = 0$, (C.1) can be simplified to

$$\max_{\alpha_i > 0} \nabla f(\alpha)_i - \min_{\alpha_i < C} \nabla f(\alpha)_i \leq \epsilon. \qquad \text{(C.2)}$$

For ASVM, the $\alpha_i \leq C$ constraints are removed so (C.2) is further reduced to

$$\max_{\alpha_i > 0} \nabla f(\alpha)_i - \min_{i} \nabla f(\alpha)_i \leq \epsilon. \qquad \text{(C.3)}$$

However, (C.3) is not suitable for LSVM as it does not keep $\alpha_i \geq 0$ throughout all iterations. Thus, we modify (C.3) to be

$$\max_{\alpha_i > \epsilon/100} \nabla f(\alpha)_i - \min_{i} \nabla(\alpha)_i \leq \epsilon$$

and

$$\alpha_i \geq -\epsilon/100, \forall i,$$

where $\epsilon = 0.001$, as the stopping condition of LSVM. This has been used in (Lin and Lin 2003) which implements LSVM for solving Reduced SVM (Lee and Mangasarian 2001).

D Conclusion Considering Stopping Criteria

Table D.1: Comparison of the running time for linear SVMs with stopping tolerance $C\epsilon$ (time in second).

Problem	with alpha seeding	without alpha seeding
australian	2.30	6.90
heart	0.36	1.16
diabetes	0.75	7.97
german	17.58	46.24
ijcnn	1052.07	25779.50
adult	923.76	40059.15
web	2311.79	7547.94

It is arguable that we may have used a too strict stopping condition in the DSVM when C is large. One possibility is to use the stopping tolerance that is proportional to C, instead of ϵ is used in (C.1), the number of iterations (without alpha seeding) would be much smaller. Thus, directly solving linear SVMs with large C becomes possible. However, in Table D.1, we show that even in these settings, DSVM with alpha seeding still makes the computational time several times faster than the original DSVM, especially for large datasets.

The stopping condition has long been a controversial issue for SVM software design. So far we have not found out a satisfactory way which is not too loose or too strict for most problems. This is why in most software, ϵ is left to be adjusted by users. For example, although using $C\epsilon$ takes a significant advantage on running time, especially for solving linear SVMs, it may lead us to a wrong optimal solution when C is too large. An extreme scenario is as follows: Let the initial solution $\alpha = 0$ and

$$\nabla f(\alpha) = Q\alpha - e = e.$$

The stopping condition (C.1) becomes

$$\begin{aligned}
&\max\{-y_i e_i \mid y_i = 1, \alpha_i < C \text{ or } y_i = -1, \alpha_i > 0\} - \\
&\min\{-y_i e_i \mid y_i = -1, \alpha_i < C \text{ or } y_i = 1, \alpha_i > 0\} \leq 2.
\end{aligned}$$

If $C\epsilon > 2$, the initial α already satisfies the stopping condition. Then, the optimization procedure stops with $w = 0$, an obviously wrong solution. Since a large stopping tolerance may cause a fake convergence, decomposition methods should also be efficient enough under the same strict setting.

In conclusion, we hope that based on this work, SVM software using decomposition methods can be suitable for all types of problems, no matter $n \ll l$ or $n \gg l$.