Journal of Machine Learning Research 6 (2005) 341-361 Submitted 11/04; Published 03/05

A Modified Finite Newton Method for Fast Solution of Large Scale Linear SVMs

S. Sathiya Keerthi SATHIYA.KEERTHI@OVERTURE.COM
Dennis DeCoste DENNIS.DECOSTE@OVERTURE.COM
Yahoo! Research Labs
210 South Delacey Avenue
Pasadena, CA 91105, USA

Editor: Thorsten Joachims

Abstract

This paper develops a fast method for solving linear SVMs with L_2 loss function that is suited for large scale data mining tasks such as text classification. This is done by modifying the finite Newton method of Mangasarian in several ways. Experiments indicate that the method is much faster than decomposition methods such as SVM$^{\text{light}}$, SMO and BSVM (e.g., 4-100 fold), especially when the number of examples is large. The paper also suggests ways of extending the method to other loss functions such as the modified Huber's loss function and the L_1 loss function, and also for solving ordinal regression.

Keywords: linear SVMs, classification, conjugate gradient

1. Introduction

Linear SVMs (SVMs whose feature space is the same as the input space of the problem) are powerful tools for solving large-scale data-mining tasks such as those arising in the textual domain. Quite often, these large-scale problems have a large number of examples as well as a large number of features and the data matrix is very sparse (e.g. $>$99.9% sparse "bag of words" in text classification). In spite of their excellent accuracy, SVMs are sometimes not preferred because of the huge training times involved (Chakrabarti et al., 2003). Thus, it is important to have fast algorithms for solving them. Traditionally, linear SVMs have been trained using decomposition techniques such as SVM$^{\text{light}}$ (Joachims, 1999), SMO (Platt, 1999), and BSVM (Hsu and Lin, 2002), which solve the dual problem by optimizing a small subset of the variables in each iteration. Each iteration costs $O(n_{nz})$ time, where n_{nz} is the number of non-zeros in the data matrix ($n_{nz} = mn$ if the data matrix is full where m is the number of examples and n is the number of features). The number of iterations, which is a function of m and the number of support vectors, tends to grow super-linearly with m and thus these algorithms can be inefficient when m is large.

With SVMs two particular loss functions for imposing penalties on slacks (violations on the wrong side of the margin, usually denoted by ξ) have been popularly used. L_1-SVMs penalize slacks linearly (penalty=ξ) while L_2-SVMs penalize slacks quadratically (penalty=$\xi^2/2$). Though both SVMs give good generalization performance, L_1-SVMs are more popularly used because they usually yield classifiers with a much less number of support vectors, thus leading to better classification speed. For linear SVMs the number of support vectors is not a matter of much concern since the final classifier is directly implemented using the weight vector in feature space. In this paper we focus on L_2-SVMs.

Since linear SVMs are directly formulated in the input space, it is possible and worthwhile to think of direct methods of solving the primal problem without using the kernel trick. Primal approaches are attractive because they assure a continuous decrease in the primal objective function.

Recently some promising primal algorithms have been given for training linear classifiers. Fung and Mangasarian (Fung and Mangasarian, 2001) have given a primal version of the least squares formulation of SVMs given by Suykens and Vandewalle (1999). Komarek (2004) has effectively applied conjugate gradient schemes to logistic regression. Zhang et al. (2003) have given an indirect algorithm for linear L_1-SVMs that works by approximating the L_1 loss function by a sequence of smooth modified logistic regression loss functions and then sequentially solving the resulting smooth primal modified logistic regression problems by nonlinear conjugate gradient methods. A particular drawback of that method is its inability to exploit the sparsity property of SVMs: that only the support vectors determine the final solution.

A direct primal algorithm for L_2-SVMs that exploits the sparsity property, called the finite Newton method, was given by Mangasarian (2002). It was mainly presented for problems in which the number of features is small. The main aim of this paper is to introduce appropriate tools that transform this method into a powerfully fast technique for solving large scale data mining problems (with a large number of examples and/or a large number of features). Our main contributions are: (1) we modify the finite Newton method by keeping the least squares nature of the problem intact in each iteration and using exact line search; (2) we bring in special, numerically robust conjugate gradient techniques to implement the Newton iterations; and (3) we introduce heuristics that speed-up the baseline implementation considerably. The result is an algorithm that is numerically robust and very fast. An attractive feature of the algorithm is that it is also very easy to implement; Appendix A gives a pseudocode that can be easily transcribed into a working code.

We show that the method that we develop for L_2-SVMs can be extended in a straight-forward way to the modified Huber's loss function (Zhang, 2004) and, in a slightly more complicated way to the L_1 loss function. We also show how the algorithm can be modifed to solve ordinal regression.

The paper is organized as follows. Section 2 formulates the problem and gives basic results. The modified finite Newton algorithm is developed in Section 3. Section 4 gives full details associated with a practical implementation of this algorithm. Section 5 gives computational experiments demonstrating the efficiency of the method in comparison with standard methods such as SVM$^{\text{light}}$ (Joachims, 1999) and BSVM (Hsu and Lin, 2002). Section 6 suggests ways for extending the method to other loss functions and Section 7 explains how ordinal regression can be solved. Section 8 contains some concluding remarks.

2. Problem Formulation and Some Basic Results

Consider a binary classification problem with training examples, $\{x_i, t_i\}_{i=1}^m$ where $x_i \in R^n$ and $t_i \in \{+1, -1\}$. To obtain a linear classifier $y = w \cdot x + b$, L_2-SVM solves the following primal problem:

$$\min_{(w,b)} \frac{1}{2}(\|w\|^2 + b^2) + \frac{C}{2}\sum_i^m \xi_i^2 \quad \text{s.t.} \;\; t_i(w \cdot x_i + b) \geq 1 - \xi_i \quad \forall\, i \tag{1}$$

where C is the regularization parameter. We have included the $b^2/2$ term so that standard regularized least squares algorithms can be used directly. Our experience shows that generalization performance is not affected by this inclusion. In a particular problem if there is reason to believe that the added term does affect performance, one can proceed as follows: take $\gamma^2 b^2/2$ as the term to be included; then define $\tilde{b} = \gamma b$ as the new bias variable and take the classifier to be $y = w \cdot x + (1/\gamma)\tilde{b}$. The parameter γ can either be chosen to be a small positive value or be tuned by cross validation.

For applying least squares ideas neatly, it is convenient to transform (1) to an equivalent formulation by eliminating the ξ_i's and dividing the objective function by the factor C. This gives[1]

$$\min_{\beta} f(\beta) = \frac{\lambda}{2}\|\beta\|^2 + \frac{1}{2}\sum_{i \in I(\beta)} d_i^2(\beta) \tag{2}$$

where $\beta = (w, b)$, $\lambda = 1/C$, $d_i(\beta) = y_i(\beta) - t_i$, $y_i(\beta) = w \cdot x_i + b$, and $I(\beta) = \{i : t_i y_i(\beta) < 1\}$.

Least Squares SVM (LS-SVM) (Suykens and Vandewalle, 1999) corresponds to (1) with the inequality constraints replaced by the equality constraints, $t_i(w \cdot x_i + b) = 1 - \xi_i$ for all i. When transformed to the form (2), this is equivalent to setting $I(\beta)=\{1,\ldots,m\}$; thus, LS-SVM is solved via a single regularized least squares solution. In contrast, the dependance of $I(\beta)$ on β complicates the L_2-SVM solution. In spite of this complexity, it can be advantageous to opt for using L_2-SVMs because they do not allow well-classified examples to disturb the classifier design. This is especially true in problems where the support vectors are a small subset of all examples.[2]

Let us now review several basic results concerning f, some of which are given in Mangasarian (2002). First note that f is a piecewise quadratic function. The presence of the $\lambda\|\beta\|^2/2$ term makes f strictly convex. Thus it has a unique minimizer. f is continuously differentiable in spite of the jumps in $I(\beta)$, the reason for this being that when an index i causes a jump in $I(\beta)$ at some β, its d_i is 0. The gradient of f is given by

$$\nabla f(\beta) = \lambda\beta + \sum_{i \in I(\beta)} d_i(\beta) \begin{pmatrix} x_i \\ 1 \end{pmatrix}. \tag{3}$$

Given an index set $I \subset \{1,\ldots,m\}$, let us define the function f_I as

$$f_I(\beta) = \frac{\lambda}{2}\|\beta\|^2 + \frac{1}{2}\sum_{i \in I} d_i^2(\beta). \tag{4}$$

Clearly f_I is a strictly convex quadratic function and so it has a unique minimizer. It follows directly from (3) that, for any $\bar{\beta}$, $\nabla f(\beta)|_{\beta=\bar{\beta}} = \nabla f_{\bar{I}}(\beta)|_{\beta=\bar{\beta}}$ where $\bar{I} = I(\bar{\beta})$. In fact, there exists an open set around $\bar{\beta}$ in which f and $f_{\bar{I}}$ are identical. It follows that $\bar{\beta}$ minimizes f iff it minimizes $f_{\bar{I}}$.

3. The Modified Finite Newton Algorithm

Mangasarian's finite newton method (Mangasarian, 2002) does iterations of the form

$$\beta_{k+1} = \beta_k + \delta_k p_k,$$

1. To help see the equivalence of (1) and (2), note that at a given (w, b), the minimization of ξ_i^2 in (1) will automatically choose $\xi_i = 0$ for all $i \notin I(\beta)$.
2. We find that L_2-SVMs usually achieve better generalization performance over LS-SVMs. Interestingly, we also find L_2-SVMs often train faster than LS-SVMs, due to sparseness arising from support vectors.

where the search direction p_k is based on a second order approximation of the objective function at β_k:

$$p_k = -H(\beta_k)^{-1}\nabla f(\beta_k).$$

Since f is not twice differentiable at β where at least one of the d_i is zero, $H(\beta)$ is taken to be the generalized Hessian defined by $H(\beta) = \lambda J + C^T DC$ where J is the $n \times n$ identity matrix, C is a matrix whose rows are $(x_i^T, 1)$ and D is a diagonal matrix whose diagonal elements are given by:

$$D_{ii} = \begin{cases} 1 & \text{if } t_i y_i(\beta) < 1 \\ \text{some specific element of } [0,1] & \text{if } t_i y_i(\beta) = 1 \\ 0 & \text{if } t_i y_i(\beta) > 1 \end{cases} \tag{5}$$

Note that examples with indices satisfying $t_i y_i(\beta) > 1$ do not affect $H(\beta)$ and p_k. (This property contributes greatly to the overall efficiency of the method.) The step size δ_k is chosen to satisfy an Armijo condition that ensures convergence, and it is found by applying a halving method of line search in the $[0,1]$ interval. (If C in (1) is sufficiently small then it is shown in Mangasarian (2002) that the fixed step size, $\delta_k = 1$ suffices for convergence.)

We modify the algorithm slightly in two ways. First, we avoid doing anything special for cases where $t_i y_i(\beta) = 1$ occurs. (Essentially, we set $D_{ii} = 0$ in (5) for such cases.) This lets us keep the least squares nature of the problem intact. More precisely, instead of computing the Newton direction p_k, we compute the Newton point, $\beta_k + p_k$, which is the solution of a regularized least squares problem. As we will see in the next section, this has useful implications on stable algorithmic implementation. Second, we do an exact line search to determine δ_k. This feature allows us to directly apply convergence results from nonlinear optimization theory. (In the next section we give a fast method for exact line search.) Thus, at one iteration, given a point β we set $I = I(\beta)$ and minimize f_I to obtain the Newton point, $\bar{\beta}$. Then an exact line search on the ray from β to $\bar{\beta}$ yields the next point of the method. These iterations are repeated till the algorithm converges. The overall algorithm is given below. Implementation details associated with each step are discussed in Section 4.

Algorithm L_2-SVM-MFN.

1. Choose a suitable starting β_0. Set $k = 0$ and go to step 2.

2. Check if β_k is the optimal solution of (2). If so, stop with β_k as the solution. Else go to step 3.

3. Let $I_k = I(\beta_k)$. Solve
$$\min_{\beta} f_{I_k}(\beta). \tag{6}$$
Let $\bar{\beta}$ denote the solution obtained.

4. Do a line search to decrease the "full" objective function, f:
$$\min_{\beta \in L} f(\beta), \tag{7}$$
where $L = \{\beta = \beta_k + \delta(\bar{\beta} - \beta_k) : \delta \geq 0\}$. Let $\delta^\star$ denote the solution of this line search. Set $\beta_{k+1} = \beta_k + \delta^\star(\bar{\beta} - \beta_k)$, $k := k+1$ and go back to step 2 for another iteration.

Theorem 1. Algorithm L_2-SVM-MFN converges to the solution of (1) in a finite number of iterations.

Proof. Let $p_k = (\bar{\beta} - \beta_k)$. Note that $p_k = -H_{I_k}^{-1} \nabla f(\beta_k)$ and $\lambda I \leq H_{I_k} \leq H_{\text{all}}$ where H_{I_k} is the Hessian of f_{I_k} and H_{all} is the Hessian of $f_{\{1,\dots,m\}}$. By Proposition 1.2.1 of Bertsekas (1999) it follows[3] that $\{\beta_k\}$ converges to the minimizer of f. Proof of finite convergence is exactly as in Mangasarian's proof of finite convergence of his algorithm, and goes as follows. Let $\beta^\star$ denote the minimizer of f and $I^\star = I(\beta^\star)$. Let $O = \{\beta : I(\beta) = I^\star\}$. Clearly O is an open set that contains $\beta^\star$. Since $\{\beta_k\}$ converges to $\beta^\star$, there exists a k such that $\beta_k \in O$. When this happens in step 2 of the algorithm, we get $\bar{\beta} = \beta^\star$ in step 3 and so $\beta_{k+1} = \beta^\star$. ■

4. Practical Implementation

In this section we discuss details associated with the implementation of the various steps of the modified finite Newton algorithm and also introduce some useful heuristics for speeding up the algorithm. The discussion leads to a fast and robust implementation of L_2-SVM-MFN. The final algorithm is also very easy to implement; Appendix A gives a pseudocode that can be easily transcribed into a working code. A number of data sets are used in this section to illustrate the effectiveness of various implementation features. These data sets are described in Appendix B. All our computations were done on a 2.4 GHz machine with Intel Xeon processor and having four Gb RAM.

4.1 Step 1: Initialization

If no guess of β is available, then the simplest starting point is $\beta_0 = 0$. For this point we have $y_i = 0$ for all i and so $I_0 = \{1, \dots, m\}$. Therefore, with such a zero initialization, the $\bar{\beta}$ obtained in step 3 is exactly the LS-SVM solution.

3. To apply Proposition 1.2.1 of Bertsekas (1999) note the following: (a) Because $\lambda > 0$ and H_{all} is positive definite, condition (1.12) of Bertsekas (1999) holds; (b) Bertsekas (1999) shows that (1.12) implies (1.13) given there; (c) in Bertsekas (1999) exact line search is referred as the minimization rule.

	Adult-9	Web-8	News20	Financial	Yahoo
No β-seeding	60.26	105.16	1321.87	1130.35	11650.56
β-seeding	36.80	24.43	944.27	692.47	4419.89

Table 1: Effectiveness of β-seeding on five data sets. The following 21 C values were used: $\sqrt{2}^k$, $k = -10, -9, \ldots, 9, 10$. All computational times are in seconds.

Suppose we have a guess $\tilde{w}$ for the weight vector w. It is possible that $\tilde{w}$ comes from an inexpensive classification method, such as the Naive-Bayes classifier. In that case it is necessary to rescale $\tilde{w}$ and also choose b_0 so as to form a β_0 that is good for starting the SVM solution. So we set $\beta_0 = (\gamma\tilde{w}, b_0)$ and choose suitable values for γ and b_0. Suppose we also assume that I, a guess of the optimal set of active indices, is available. (If no guess is available, we can simply let I be the set of all training indices.) Then choose γ and b_0 to minimize the cost

$$\frac{\lambda}{2}[\gamma^2\|\tilde{w}\|^2 + b_0^2] + \frac{1}{2}\sum_{i\in I}[\gamma\tilde{w}\cdot x_i + b_0 - t_i]^2. \tag{8}$$

It is easy to check that the resulting γ and b_0 are given by

$$\gamma = (p_{22}q_1 - p_{12}q_2)/d \quad \text{and} \quad b_0 = (p_{11}q_2 - p_{12}q_1)/d. \tag{9}$$

where $p_{11} = \lambda\|\tilde{w}\|^2 + \sum_{i\in I}(\tilde{w}\cdot x_i)^2$, $p_{22} = \lambda + |I|$, $p_{12} = \sum_{i\in I}\tilde{w}\cdot x_i$, $q_1 = \sum_{i\in I} t_i\tilde{w}\cdot x_i$, $q_2 = \sum_{i\in I} t_i$ and $d = p_{11}p_{22} - (p_{12})^2$. Once γ and b_0 are thus obtained, we can set $\beta_0 = (\gamma\tilde{w}, b_0)$ and start the algorithm.[4] Note that the set of initial indices I_0 chosen by the algorithm in step 3 is the set of active indices at β_0 and so it could be different from I.

There is another situation where the above initialization comes in handy. Suppose we solve (2) for one C and want to re-solve (2) for a slightly changed C. Then we can use the $\tilde{w}$ and I that are obtained from the optimal solution of the first value of C to do the above mentioned initialization process for the second value of C. For this situation we have also tried simply choosing $\gamma = 1$ and b_0 equal to the b that is optimal for the first value of C. This simple initialization also works quite well. We will refer to this initialization for the 'slightly changed C' situation as β-*seeding*; seeding is crucial to efficiency when (2) is to be solved for many C values (such as when tuning C via cross-validation). The β-seeding idea used here is very much similar to the idea of α-seeding popularly employed in the solution of SVM duals(DeCoste and Wagstaff, 2000).

Eventhough full details of the final version of our implementation of L_2-SVM-MFN is only developed further below, here let us compare the final implementation[5] with and without β-seeding. Table 1 gives computational times for several data sets. Clearly, β-seeding gives useful speed-ups.

4. To get a better β_0 one can also reset $I = I(\beta_0)$ and repeat (9) and get revised values for γ and b_0. This computation is cheap since $\tilde{w}\cdot x_i$ need not be recomputed.
5. For no β-seeding, the implementation corresponds to the use of 'both heuristics' mentioned in Table 3. For β-seeding, the two heuristics are not used since they do not contribute much when β-seeding is done.

4.2 Step 2: Checking Convergence

Checking of the optimality of β_k is done by first calculating $y_i(\beta_k)$ and $d_i(\beta_k)$ for all i, determining the active index set I_k and then checking if

$$\|\nabla f_{I_k}(\beta_k)\| = 0. \tag{10}$$

For practical reasons it is necessary to employ a tolerance parameter when checking (10). We deal with this important issue below after the discussion of the implementation of step 3.

4.3 Step 3: Regularized Least Squares Solution

The solution of (6) can be approached in one of the following ways: using factorization methods such as QR or SVD; or, using an iterative method such as the conjugate gradient (CG) method. We prefer the latter method due to the following reasons: (a) it can make effective use of knowledge of good starting vectors; and (b) it is much better suited for large-scale problems having sparse data sets. To setup the details of the CG method, let: X be the matrix whose rows are $(x_i^T, 1)$, $i \in I_k$; and t be a vector whose elements are t_i, $i \in I_k$. Then (6) is the same as the regularized least squares problem

$$\min_{\beta} f_{I_k}(\beta) = \frac{\lambda}{2}\|\beta\|^2 + \frac{1}{2}\|X\beta - t\|^2. \tag{11}$$

This corresponds to the solution of the normal system,

$$(\lambda I + X^T X)\beta = X^T t. \tag{12}$$

With CG methods there are several ways of approaching the solution of (11/12). A simple approach is to solve (12) using the CG method meant for solving positive definite systems. However, such an approach will not be numerically well-conditioned, especially when λ is small. As pointed out by Paige and Saunders (Paige and Saunders, 1982) this is mainly due to the explicit use of vectors of the form $X^T X p$. An algorithm with better numerical properties can be easily derived by an algorithmic rearrangement that is special to the regularized least squares solution, which makes use of the intermediate vector Xp. LSQR (Paige and Saunders, 1982) and CGLS (Björck, 1996) are two such special CG algorithms. Another very important reason for using one of these algorithms (as opposed to using a general purpose CG solver) is that, for the special methods it is easy to derive a good stopping criterion to approximately terminate the CG iterations using the intermediate residual vector. We discuss this issue in more detail below.

For our work we have used the version of the CGLS algorithm given in Algorithm 3 of Frommer and Maaß (Frommer and Maaß, 1999) which uses initial seed solutions neatly.[6] Following is the CGLS algorithm for solving (11).

Algorithm CGLS. Set $\beta^0 = \beta_k$ (where β_k is as in steps 2 and 3 of Algorithm L_2-SVM-MFN). Compute $z^0 = t - X\beta^0$, $r^0 = X^T z^0 - \lambda\beta^0$,[7] set $p^0 = r^0$ and do the following steps for $j = 0, 1, \ldots$

6. Frommer and Maaß have also given interesting variations of the CGLS method for efficiently solving (12) for several values of λ. But we have not tried those methods in this work.

7. It is useful to note that, at any point of the CGLS algorithm $-z^j$ is the vector containing the classifier residuals, d_i, $i \in I_k$ and r^j is the negative of the gradient of $f_{I_k}(\beta)$ at β^j. At the beginning $(j = 0)$, z^0 and r^0 are already available in view of the computations in step 2 of Algorithm L_2-SVM-MFN. This fact can be used to gain some efficiency.

```
$q^j = Xp^j$
$\gamma^j = \|r^j\|^2/(\|q^j\|^2 + \lambda\|p^j\|^2)$
$\beta^{j+1} = \beta^j + \gamma^j p^j$
$z^{j+1} = z^j - \gamma^j q^j$
$r^{j+1} = X^T z^{j+1} - \lambda\beta^{j+1}$
If $r^{j+1} = 0$ stop with $\beta^{j+1}$ as the solution.
$\omega^j = \|r^{j+1}\|^2/\|r^j\|^2$
$p^{j+1} = r^{j+1} + \omega^j p^j$
```

There are exactly two matrix-vector operations in each iteration; sparsity of the data matrix can be effectively used to do these operations in $O(n_z)$ time where n_z is the number of non-zero elements in the data matrix.

Let us now discuss the convergence properties of CGLS. It is known that the algorithm will take at most l iterations where l is the rank of X. Note that $l \leq \min\{m,n\}$ where m is the number of examples and n is the number of features. The actual number of iterations required to achieve good practical convergence is usually much smaller than $\min\{m,n\}$ and it depends on the number of singular values of X that are really significant.

Stopping the CG iterations with the right accuracy is very important because, an excessively accurate solution would lead to too much work while an inaccurate solution will not give good descent. Using a simple absolute tolerance on the size of the gradient of f_{I_k} in order to stop is a bad idea, even for a given data set since the typical size of the gradient varies a lot as λ is varied over a range of values. For a method such as CGLS it is easy to find effective practical stopping criteria. We can decide to stop when the negative gradient vector r^{j+1} has come near zero up to some relative precision. To do this we can use the bound, $\|r^{j+1}\| \leq \|X\|\|z^{j+1}\| + \|\lambda\beta^{j+1}\|$. Thus a good stopping criterion is

$$\|r^{j+1}\| \leq \varepsilon(\rho\|z^{j+1}\| + \lambda\|\beta^{j+1}\|),$$

where $\rho = \|X\|$. Since X varies at different major iterations of the L_2-SVM-MFN algorithm, we can simply take $\rho = \|\hat{X}\|$ where $\hat{X}$ is the entire $m \times n$ data matrix.

In most datamining tasks the data is normalized so that all values in the data matrix are in the unity range. For such data sets we have $\|X\| \leq \sqrt{n}$. One can take a conservative approach and simply use the stopping criterion

$$\|r^{j+1}\| \leq \varepsilon\|z^{j+1}\|. \tag{13}$$

We have found this criterion to be very effective and have used it for all the computational experiments reported in this paper. The parameter ε is a relative tolerance parameter. A value of $\varepsilon = 10^{-6}$, which roughly yields solutions accurate up to six decimal digits, is a good choice.

Since $r = -\nabla f_{I_k}$ we can apply exactly the same criteria as in (13) for approximately checking (10) also. Almost always, termination of L_2-SVM-MFN occurs when, after the least squares solution at step 3, exact line search in step 4 gives $\beta^{k+1} = \bar{\beta}$ (i.e., $\delta^\star = 1$), and the active set remains unchanged, i.e., $I(\beta^k) = I(\bar{\beta}) = I(\beta^{k+1})$.

Let us illustrate the effectiveness of the CGLS method using the LS-SVM solution as an example. For LS-SVM we can set $I = \{1,\ldots,m\}$ and solve (12) using the CGLS method mentioned above; let us refer to such an implementation as LS-SVM-CG. In their proximal SVM implementation of LS-SVM, Fung and Mangasarian (2001) solve (12) using Matlab routines that employ factorization techniques on X^TX. For large and sparse data sets it is much more efficient to use CG methods and avoid the formation of X^TX and its factorization. Table 2 illustrates this fact using two

Table 2: Ratio of the computational cost (averaged over $C = 2^{-5}, 2^{-4}, \dots, 2^5$) of the proximal SVM algorithm to that of LS-SVM-CG. s is the sparsity factor, $s = n_{nz}/(nm)$.

	Australian ($n = 14$, $m = 690$, $s = 1.0$)	Web-7 ($n = 300$, $m = 24692$, $s = 0.04$)
Ratio	0.72	15.72

data sets. The complexity of the original proximal SVM implementation is $O(n_{nz}n + n^3)$ whereas the complexity of the CGLS implementation is $O(n_{nz}l)$, where l is the number of iterations needed by the CGLS algorithm.[8] When the data matrix is dense (e.g., *Australian*) the factorization approach is slightly faster than the CGLS approach. But, when the data matrix is sparse (e.g., *Web-7*) the CGLS approach is considerably faster, even with n being small.

Similar observations hold for Mangasarian's finite Newton method as well as our L_2-SVM-MFN. When the data matrix is sparse, factorization techniques are much more expensive compared to CG methods, even when n is not too big. Of course, factorization methods get completely ruled out when both m and n are large.

4.4 Step 4: Exact Line Search

Let $\beta(\delta) = \beta_k + \delta(\bar{\beta} - \beta_k)$. The one dimensional function, $\phi(\delta) = f(\beta(\delta))$ is a continuously differentiable, strictly convex, piecewise quadratic function. To determine the minimizer of this function analytically, we compute the points at which the second derivative jumps. For any given i, let us define: $\delta_i = (t_i - y_i^k)/(\bar{y_i} - y_i^k)$, where $\bar{y_i} = y_i(\bar{\beta})$ and $y_i^k = y_i(\beta_k)$. The jump points mentioned above are given by

$$\Delta = \Delta_1 \cup \Delta_2, \tag{14}$$

where

$$\Delta_1 = \{\delta_i : i \in I_k, t_i(\bar{y_i} - y_i^k) > 0\} \quad \text{and} \quad \Delta_2 = \{\delta_i : i \notin I_k, t_i(\bar{y_i} - y_i^k) < 0\}. \tag{15}$$

For Δ_1 we are not using i with $t_i(\bar{y_i} - y_i^k) \leq 0$ because they do not cause switching at a positive δ; similarly, for Δ_2 we are not using i with $t_i(\bar{y_i} - y_i^k) \geq 0$.

Take one $\delta_i \in \Delta_1$. When we increase δ across δ_i, the index i leaves $I(\beta(\delta))$. Thus, the term $d_i^2/2$ has to be left out of the objective function for all $\delta > \delta_i$. Similarly, for $\delta_i \in \Delta_2$, when we increase δ across δ_i, the index i enters $I(\beta(\delta))$. Thus, the term $d_i^2/2$ has to be included into the objective function for all $\delta > \delta_i$.

The slope, $\phi'(\delta)$ is a continuous piecewise linear function that changes its slope only at one of the δ_i's. The optimal point $\delta^\star$ is the point at which $\phi'(\delta)$ crosses 0.[9] To determine this point we first sort all δ_i's in Δ in non-decreasing order. To simplify the notations, let us assume that δ_i, $i = 1, 2, \dots$ denotes that ordering. Between δ_i and δ_{i+1} we know that $\phi'(\delta)$ is a linear function. Just for doing calculations extend this line both sides (left and right) to meet the $\delta = 0$ and $\delta = 1$ vertical lines.

8. It is useful to note here that the proximal SVM implementation solves (12) exactly while LS-SVM-CG uses the practical stopping condition (13) that contributes further to its efficiency.
9. Note that this point may not necessarily be at one of the δ_i's.

Let us call the ordinate values at these two meeting points as l_i and r_i respectively. It is very easy to keep track of the changes in l_i and r_i as indices get dropped and added to the active set of indices.

We move from left to right to find the zero crossing of $\phi'(\delta)$. At the beginning we are at $\delta_0 = 0$. Between δ_0 and δ_1 we have I_k as the active set. It is easy to get, from the definition of $\phi(\delta)$ that

$$l_0 = \lambda\beta_k \cdot (\bar{\beta} - \beta_k) + \sum_{i \in I_k} (y_i^k - t_i)(\bar{y}_i - y_i^k) \tag{16}$$

and

$$r_0 = \lambda\bar{\beta} \cdot (\bar{\beta} - \beta_k) + \sum_{i \in I_k} (\bar{y}_i - t_i)(\bar{y}_i - y_i^k). \tag{17}$$

(If, at step 3 of the L_2-SVM-MFN algorithm, we solve (6) exactly, then it is easy to check that $r_0 = 0$. However, in view of the use of the approximate termination mentioned in (13) it is better to compute r_0 using (17).) Find the point where the line joining $(0, l_0)$ and $(1, r_0)$ points on the (δ, ϕ') plane crosses zero. If the zero crossing point of this line is between 0 and δ_1 then that point is $\delta^\star$. If not, we move over to searching between δ_1 and δ_2. Here l_1 and r_1 need to be computed. This can be done by a simple updating over l_0 and r_0 since only the term $d_i^2/2$ enters or leaves. Thus, for a general situation where we already have l_i, r_i computed for the interval δ_i to δ_{i+1} and we need to get l_{i+1}, r_{i+1} for the interval δ_{i+1} to δ_{i+2}, we use the update formula

$$l_{i+1} = l_i + s(y_i^k - t_i)(\bar{y}_i - y_i^k) \quad \text{and} \quad r_{i+1} = r_i + s(\bar{y}_i^k - t_i)(\bar{y}_i - y_i^k), \tag{18}$$

where $s = -1$ if $\delta_i \in \Delta_1$ and $s = 1$ if $\delta_i \in \Delta_2$. Thus we keep moving to the right until we get a zero satisfying the condition that the root determined by interpolating $(0, l_i)$ and $(1, r_i)$ lies between δ_i and δ_{i+1}. The process is bound to converge since we know the existence of the minimizer (we are dealing with a strictly convex function). In a typical application of the above line search algorithm, many δ_i's are crossed before $\delta^\star$ is reached, especially in the early stages of the algorithm, causing $|I(\beta_{k+1})|$ to be much different from $|I(\beta_k)|$. This is the crucial step where the support vectors of the problem get identified.

The complexity of the above exact line search algorithm is $O(m \log m)$. Since the least squares solution (step 3) is much more expensive, the cost of exact line search is negligible.

4.5 Complexity Analysis

The bulk of the cost of the algorithm is associated with step 3, which only deals with examples that are active at the current point. (The full set of examples is involved only in step 4.) This crucial factor greatly contributes to the overall efficiency of the algorithm. The number of iterations, i.e., loops of steps 2-4 is usually small, say 5-20. Thus, the empirical complexity of the algorithm is $O(n_{nz} l_{av})$ where l_{av}, the average number of CG iterations in step 3, is bounded by the rank of the data matrix and so $l_{av} \le \min\{m, n\}$. As already mentioned, l_{av} usually turns out to be much smaller than both, m and n. For example, when applied to the *financial* data set that has 198788 examples and 252472 features, for $C = 1$ and $\beta = 0$ initialization, L_2-SVM-MFN took 11 iterations, with $l_{av} = 102$.

4.6 Speed-up Heuristics

Suppose we are solving a problem for which the number of support vectors, i.e., $|I(\beta^\star)|$ is a small fraction of m, and we use the initialization, $\beta_0 = 0$. Since $I(\beta_0) = \{1, \ldots, m\}$, step 3 corresponds to

	Adult-9	Web-8	News20	Financial	Yahoo
No heuristics	7.28	11.79	98.06	456.85	1443.23
Heuristic 1	5.12	9.24	67.85	70.86	904.99
Heuristic 2	3.57	4.17	90.54	202.62	848.99
Both heuristics	3.00	3.90	52.73	62.18	524.16
SV fraction	0.605	0.219	0.650	0.068	0.710

Table 3: Effectiveness of the two speed-up heuristics on five data sets. The value $C = 1$ was used. All computational times are in seconds. SV fraction is the ratio of the number of support vectors to the number of training examples.

solving an unnecessarily large least squares problem; it is wasteful to solve it accurately. One (or both) of the following two heuristics can be employed to avoid this.

Heuristic 1. Whenever β_0 is a crude approximation (say, $\beta_0 = 0$), terminate the least squares solution of (6) after a fixed, small number (say, 10) of CGLS iterations at the first call to step 3. Even with the crude β thus generated, the following step 4 usually leads to a point β_1 with $|I(\beta_1)|$ much smaller than m, and a good bulk of the non-support vectors get identified correctly.

Heuristic 2. First run the L_2-SVM-MFN algorithm using a crude tolerance, say $\varepsilon = 10^{-2}$. Use the β thus generated as the starting vector and make another run with $\varepsilon = 10^{-6}$, the final desired accuracy.

Table 3 gives the effectiveness of the above heuristics on a few data sets. Clearly both heuristics are useful. It is not easy to say which one is more effective and so using both of them is the appropriate thing to do. This gives at least a 2-fold speed-up. As expected, the amount of speed-up is big if the fraction of examples that are support vectors is small. The pseudocode of Appendix B uses both heuristics.

A third heuristic may also be used when working with a very large number of examples. First choose a small random subset of the examples and run the algorithm. Then use the β thus generated to seed a second run, this time using all the examples for training.

We end this section on implementation by explaining how a solution of the SVM dual can be obtained after L_2-SVM-MFN solves the primal.

4.7 Obtaining a Dual Solution

Note that the SVM dual variables, α_i, $i = 1, \ldots, m$ are not involved anywhere in the algorithm. But it is easy to recover them once we solve (2) using L_2-SVM-MFN. From the structure of (3) it is easy to see that $\alpha_i = -t_i d_i/\lambda$, if $i \in I(\beta)$ and $\alpha_i = 0$ otherwise. In a practical solution we do not get the true solution due to the use of (13). In such a situation it is useful to ask as to how well the α defined above satisfies the KKT optimality conditions of the dual. This can be easily done. After computing α as mentioned above, set $\hat{\beta} = \sum_i \alpha_i t_i (x_i^T, 1)^T$, $\xi_i = \lambda\alpha_i \; \forall i$, $g_i = t_i y_i(\hat{\beta}) + \xi_i - 1 \; \forall i$, and obtain the maximum dual KKT violation as $\max\{\max_{i:\alpha_i>0} |g_i|, \max_{i:\alpha_i=0} \max\{0, -g_i\}\}$. If, keeping the maximum dual KKT violation within some specified tolerance (say, $\tau = 0.001$) is important for some reason, then one can proceed as follows. First solve L_2-SVM-MFN using $\varepsilon = 10^{-3}$ and then check the maximum dual KKT violation as described above. If it does not satisfy the required tolerance then continue the L_2-SVM-MFN solution with a suitably chosen smaller value of ϵ.

5. Comparison with SVM$^{\text{light}}$ and BSVM

The experiments of this section compare L_2-SVM-MFN against two dual-based methods: the popular SVM$^{\text{light}}$ (Joachims, 1999)[10] and the more modern BSVM (Hsu and Lin, 2002). In order to make a proper comparison L_2-SVM-MFN was forced to satisfy the same dual KKT tolerance of $\tau = 0.001$ as the other methods. (The procedure given at the end of the last section was used to do this.) We used default `-q` (subproblem size) for SVM$^{\text{light}}$ & BSVM; other values tried were not faster.[11] We also tried SMO (Platt, 1999), but found it slower than the others for these linear problems. An explanation for this is given by Kao et al (Kao et al., 2004) in Section 4 of their paper via the fact that, for the linear SVM implementation, the cost of updating the gradient of the dual is independent of the number of dual variables that are optimized in each basic iteration.

Tables 4-6 report training times and 10-fold cross-validation (CV) error rates for *Adult-9*, *Web-8* and *News20* data sets.

We show training times for various C's, with optimal (lowest) mean CV error rate for each method shown in bold. Due to the different loss functions used, a direct comparison of these methods is challenging and necessarily approximate on non-separable data. Therefore, in the following tables we show results for a range of C values around values of C yielding minimum cross-validation errors for each of the three methods. A reasonable conservative speedup for our method can then be determined by selecting the slowest training time for a near-optimal C value for our method versus the fastest training time for a near-optimal C value for an alternative method.

For example, for *Adult-9* one could compare the time for SVM$^{\text{light}}$'s CV-optimal (112.6 secs) versus the time for L_2-SVM-MFN's CV-optimal (1.6 secs), yielding a speedup ratio of 70.4. Alternatively, *nearly*-CV-optimal cases for *Adult-9* (e.g. 15.23%$>$15.22% for SVM$^{\text{light}}$ with C=2^{-3}) yield other speedups (e.g. 13.2 if both SVM$^{\text{light}}$ and L_2-SVM-MFN use C=2^{-3}). Over all such nearly-optimal cases for all three data sets, speedups are consistently significant (e.g. 4-100 over SVM$^{\text{light}}$ and 4-40 over BSVM). Even for *News20* which has more than a million features[12] L_2-SVM-MFN is more than four times faster than SVM$^{\text{light}}$ and BSVM.

For the *Yahoo* data set having a million examples, SVM$^{\text{light}}$ and BSVM could not complete the solution even after one full day, while L_2-SVM-MFN took only about 10 minutes to obtain a solution.

We also did an experiment to study how the algorithms scale with m, the number of examples. Figure 1 gives log-log plots showing the variation of training times as a function of m, for four of the conventional *Adult* and *Web* subsets. The times plotted for each data subset/method pair are for the corresponding CV-optimal C's. These plots show that not only was L_2-SVM-MFN always faster, but it also scaled much better with m.

10. We report results using version 5.0 of SVM$^{\text{light}}$. We also tried the newer version 6.0, but found for our particular experiments with linear kernels that it was no faster, and sometimes even slower.
11. Specifically, the values used for q were 10 for SVM$^{\text{light}}$ and 30 for BSVM.
12. Dual algorithms such as SVM$^{\text{light}}$ and BSVM are efficient when the number of features is large. Their cost scales linearly with the number of features.

	SVM$^{\text{light}}$		BSVM		L_2-SVM-MFN	
$\log_2 C$	secs	CV%	secs	CV%	secs	CV%
-4.5	16.8	15.29	7.3	15.29	1.2	15.30
-4.0	15.3	15.28	8.1	15.27	1.3	15.26
-3.5	18.6	15.25	8.9	15.27	1.4	15.22
-3.0	21.1	15.23	10.0	15.23	**1.6**	**15.21**
-2.5	25.8	15.24	11.7	15.23	1.8	15.21
-2.0	44.4	15.25	13.3	15.24	1.9	15.23
-1.5	47.0	15.23	15.9	15.24	2.2	15.23
-1.0	58.9	15.26	20.3	15.25	3.1	15.22
-0.5	80.9	15.23	25.8	15.24	2.9	15.22
0.0	**112.6**	**15.22**	**32.4**	**15.22**	2.9	15.23
0.5	189.2	15.23	41.7	15.23	3.1	15.23
1.0	235.9	15.24	54.4	15.22	3.5	15.23

Table 4: Results for *Adult-9*

	SVM$^{\text{light}}$		BSVM		L_2-SVM-MFN	
$\log_2 C$	secs	CV%	secs	CV%	secs	CV%
-0.5	14.3	1.36	11.4	1.35	3.2	1.34
0.0	14.9	1.35	15.8	1.35	4.2	1.34
0.5	19.6	1.34	20.7	1.34	5.0	1.33
1.0	29.1	1.34	28.4	1.34	**5.0**	**1.33**
1.5	46.8	1.33	-	1.33	4.7	1.33
2.0	**60.3**	**1.33**	61.4	1.33	5.7	1.33
2.5	110.5	1.33	**82.4**	**1.33**	7.6	1.33
3.0	131.6	1.33	139.0	1.33	8.9	1.34
3.5	232.6	1.33	191.4	1.33	10.8	1.34
4.0	279.8	1.34	282.9	1.34	10.8	1.34
4.5	347.5	1.35	441.1	1.35	11.6	1.34
5.0	615.1	1.35	647.1	1.34	11.2	1.34

Table 5: Results for *Web-8*

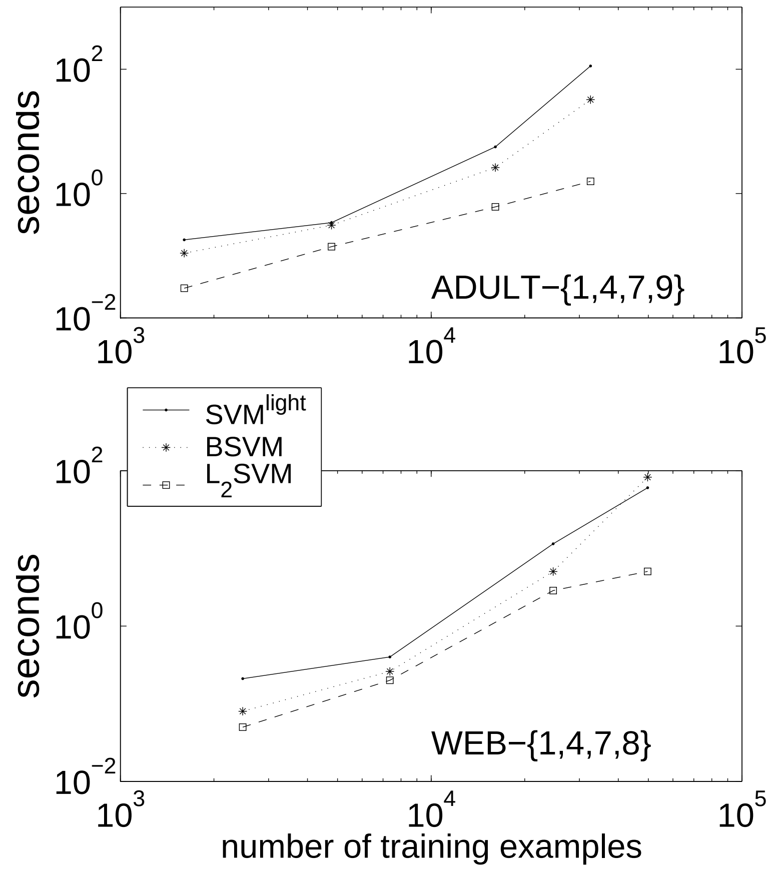

Figure 1: Training time versus *m* for the Adult and Web data sets, on a log-log plot. Note that the vertical axes are only marked at 10^{-2}, 10^{0} and 10^{2}.

	SVM$^{\text{light}}$		BSVM		L_2-SVM-MFN	
$\log_2 C$	secs	CV%	secs	CV%	secs	CV%
0.0	335.9	2.93	330.4	2.93	69.3	3.35
0.5	407.6	2.82	392.6	2.81	55.4	3.16
1.0	437.1	2.78	436.1	2.78	84.5	3.07
1.5	442.2	2.78	434.4	2.78	62.7	2.98
2.0	**466.8**	**2.74**	**437.3**	**2.73**	76.0	2.89
2.5	466.9	2.77	432.3	2.78	75.0	2.88
3.0	455.9	2.85	450.4	2.85	91.6	2.88
3.5	471.3	3.02	439.0	3.02	**98.0**	**2.86**
4.0	513.7	3.07	432.2	3.07	114.0	2.86
4.5	554.7	3.07	437.0	3.07	120.4	2.89
5.0	525.7	7.76	421.6	3.07	159.2	2.90
5.5	535.7	3.07	426.5	3.08	193.9	2.97

Table 6: Results for *News20*

6. Extension to Other Loss Functions

The previous sections addressed the solution of L_2-SVM, i.e., the SVM primal problem that uses the L_2 loss function:

$$\min_\beta f(\beta) = \frac{\lambda}{2}\|\beta\|^2 + \sum_i L(\xi_i) \tag{19}$$

where $\xi_i = 1 - t_i y_i(\beta)$ and $L = L_2$ where

$$L_2(\xi_i) = \begin{cases} 0 & \text{if } \xi_i \le 0 \\ \xi_i^2/2 & \text{if } \xi_i > 0 \end{cases} \tag{20}$$

The modified Newton algorithm can be adapted for other loss functions too. We briefly explain how to do this for the following loss functions: the modifed Huber's loss function (Zhang, 2004) and the L_1 loss function.

Consider, first, the modified Huber's loss function. The solution for this loss function forms the basis of the solution for the L_1 loss function. Recently, Zhang (Zhang, 2004) pointed out that the modified Huber's loss function has some attractive theoretical properties. The loss function is given by

$$L_h(\xi_i) = \begin{cases} 0 & \text{if } \xi_i \le 0 \\ \xi_i^2/2 & \text{if } 0 < \xi_i < 2 \\ 2(\xi_i - 1) & \text{if } \xi_i \ge 2 \end{cases} \tag{21}$$

With $L = L_h$, the primal objective function f in (19) is strictly convex, continuously differentiable and piecewise quadratic, very much as when (20) is used. So the extension of the modified Newton algorithm to L_h is rather easy. A basic iteration proceeds as follows. Given β_k let $I_0 = \{i : \xi_i(\beta_k) \le 0\}$, $I_1 = \{i : 0 < \xi_i(\beta_k) < 2\}$ and $I_2 = \{i : \xi_i(\beta_k) \ge 2\}$. The natural quadratic approximation of f to minimize is the one which keeps these index sets unchanged, i.e.,

$$\min_\beta \tilde{f}(\beta) = \frac{\lambda}{2}\|\beta\|^2 + \frac{1}{2}\sum_{i\in I_1}(y_i(\beta) - t_i)^2 - 2\sum_{i\in I_2} t_i y_i(\beta). \tag{22}$$

Let

$$q = \frac{2}{\lambda} \sum_{i \in I_2} t_i \begin{pmatrix} x_i \\ 1 \end{pmatrix}$$

and $\tilde{\beta} = \beta - q$ so that (22) can be equivalently rewritten as the solution of

$$\min_{\beta} \bar{f}(\beta) = \frac{\lambda}{2} \|\beta - q\|^2 + \frac{1}{2} \sum_{i \in I_1} (y_i(\beta) - t_i)^2. \tag{23}$$

This is nothing but a regularized least squares solution that is shifted in β space; the CG techniques described in Section 4 can be used to solve for $\tilde{\beta} = \beta - q$ and then β can be obtained. The exact line search for minimizing f on a ray is only slightly more complicated than the one in Section 4: with (21) we need to watch for jumps of examples from/to three sets of the type I_0, I_1 and I_2 defined above. The proof of finite convergence of the overall algorithm is very much as for the L_2 loss function.

Let us now consider the L_1 loss function given by

$$L_1(\xi_i) = \begin{cases} 0 & \text{if } \xi_i \le 0 \\ \xi_i & \text{if } \xi_i > 0. \end{cases} \tag{24}$$

Choose τ, a positive tolerance parameter and define $\tilde{\xi} = (\xi_i/\tau) + 1$. The L_1 loss function can be approximated by $L_h(\tilde{\xi})$. Thus, we can solve the primal problem corresponding to the L_1 loss function as follows. Take a sequence of τ values, say $\tau_j = 2^{-j},\ j = 0, 1, \ldots$. Start by solving the problem for $j = 0$. Use the β thus obtained to seed the solution of the problem for $j = 1$ and so on until a solution that approximates the true solution of the L_1 loss function satisfactorily is obtained. This is only a rough outline of the main scheme. Several details need to be worked out in order to arrive at an overall method that is actually very efficient. Currently we are working on these details; we will report the results in a future paper.

Recently Zhang et al (Zhang et al., 2003) gave a primal algorithm for SVMs with L_1 loss function in which a modified logistic regression function is used to approximate the L_1 loss function and a sequential approximation scheme similar to what we described above is employed. Our method is expected to be more efficient since the approximating loss function (modified Huber) helps keep the sparsity propery, i.e., examples with $|\xi_i| \ge \tau$ are inactive during the solution of the linear least squares problem at each iteration. However, this claim needs to be corraborated by proper implementation of both methods and detailed numerical experiments.

7. Extension to Ordinal Regression

In this section we explain how the L_2-SVM-MFN algorithm can be adapted to solve ordinal regression problems. In ordinal regression the target variable, t_i takes a value from a finite set, say, $\{1, 2, \ldots, p\}$. Thus, p is the total number of possible ordinal values. Let $J_s = \{i : t_i = s\}$. Let w denote the weight vector and $y_i(w) = w \cdot x_i$ denote the 'score' of the SVM for the i-th example. To set up the SVM formulation we follow the approach given in Chu and Keerthi (2005) and use $p - 1$ thresholds, b_s, $s = 1, \ldots, p-1$ to divide the scores into p bins so that the interval, (b_{s-1}, b_s) is assigned for examples which have $t_i = s$.[13] Let β denote the vector which contains w together with

13. To make this statement properly, we take $b_0 = -\infty$ and $b_p = \infty$.

b_s, $s = 1, \ldots, p-1$. For a given β and an $s \in \{1, \ldots, p-1\}$ define the following 'margin-violating' index sets:

$$\mathcal{L}_s(\beta) = \{i : i \in J_l \text{ for some } l \le s \text{ and } y_i(w) - b_s > -1\}$$

$$\mathcal{U}_s(\beta) = \{i : i \in J_l \text{ for some } l > s \text{ and } y_i(w) - b_s < 1\}.$$

Then the primal SVM problem can be written as

$$\min_\beta f(\beta) = \frac{\lambda}{2}\|\beta\|^2 + \sum_{s=1}^{p-1}\Big(\frac{1}{2}\sum_{i \in \mathcal{L}_s(w)} (y_i(w) - b_s + 1)^2 + \frac{1}{2}\sum_{i \in \mathcal{U}_s(w)} (y_i(w) - b_s - 1)^2\Big). \qquad (25)$$

A nice property of the above formulation is that, as shown in Chu and Keerthi (2005), the solution of (25) automatically satisfies the condition, $b_1 \le b_2 \le \cdots \le b_{p-1}$.

Clearly, f is a differentiable, strictly convex, piecewise quadratic function of β, very much like the f in (2). So, the extension of L_2-SVM-MFN to solve (25) is easy. A basic iteration goes as follows. Given β_k, let $\bar{\mathcal{L}}_k = \mathcal{L}_s(\beta_k)$, $\bar{\mathcal{U}}_k = \mathcal{U}_s(\beta_k)$ for $s = 1, \ldots, p-1$ and solve the following quadratic approximation of f corresponding to keeping those index sets unchanged:

$$\min_\beta \tilde{f}(\beta) = \frac{\lambda}{2}\|\beta\|^2 + \sum_{s=1}^{p-1}\Big(\frac{1}{2}\sum_{i \in \bar{\mathcal{L}}_k} (y_i(w) - b_s + 1)^2 + \frac{1}{2}\sum_{i \in \bar{\mathcal{U}}_k} (y_i(w) - b_s - 1)^2\Big). \qquad (26)$$

Let $\bar\beta$ denote the solution of (26). Exact line search to minimize f on the ray from β_k to $\bar\beta$ is more complicated than the line search we described in Section 4, but it is quite easy to program in code; also, if p is small, the algorithm is not expensive. As in Section 4, we need to identify the points along the ray at which jumps in the second derivative of f take place. Take one example, say the i-th. Let $\bar\beta = (\bar w, \bar b_1, \ldots, \bar b_{p-1})$, $\bar y_i = y_i(\bar w)$ and $l = t_i$. For each $s = 1, \ldots, l-1$, calculate δ_{si} such that $y_i(\beta_k) + \delta_{si}(\bar y_i - y_i(\beta_k)) = b_s + 1$. Similarly, for each $s = l, \ldots, p-1$, calculate δ_{si} such that $y_i(\beta_k) + \delta_{si}(\bar y_i - y_i(\beta_k)) = b_s - 1$. Each positive δ_{si} is a point where the second derivative jumps. By calculating all such points (there are at most pm of them, where m is the number of training examples), sorting them and using the ideas of Section 4 to locate the minimizer where the slope crosses the zero value, exact line search can be performed. The proof of convergence of the overall algorithm is very much similar to the proof of Theorem 1.

The ideas outlined above for the L_2 loss function can be extended to other loss functions such as the modified Huber's loss function and the L_1 loss function using the ideas of Section 6.

8. Conclusion

In this paper we have modified the finite Newton method of Mangasarian in several ways to obtain a very fast method for solving linear SVMs that is easy to implement and trains much faster than existing alternative SVM methods, making it attractive for solving large-scale classification problems.

We have also tried another method for linear L_2-SVMs. This corresponds to the direct application of a nonlinear CG method (such as Polak-Ribierre) to (2); note that f is a differentiable function. This method also works well, but it is not as efficient and numerically robust as L_2-SVM-MFN. One of the main reasons for this is that the bulk of the computations of L_2-SVM-MFN takes place in the CGLS iterations which operate only with potential support vectors. On the other hand, the nonlinear CG method has to necessarily deal with all examples in each iteration, unless clever shrinking strategies are designed.

It is interesting to ask if the modified finite Newton algorithm can be extended to nonlinear kernels. If (6) is solved via its dual (say, by using an algorithm for LS-SVMs) the new algorithm can indeed be extended to nonlinear kernels. That would be an interesting primal algorithm that is implemented using dual variables. But it is not yet clear whether such an algorithm will be more efficient than existing good dual methods (e.g. SMO or SVM$^{\text{light}}$).

Appendix A. A Pseudocode for L_2-SVM-MFN

Below, β represents the current point; y and I denote the output vector and active index set at β. $F = f(\beta)$ and $I_{all} = \{1,\ldots,m\}$.

1. *Initialization.*

   - If no initial guess of β is available, set $ini = 0$, $\beta = 0$, $y_i = 0\ \forall i \in I_{all}$ and $I = I_{all}$.
   - If a guess of w is obtained from another method (say, the Naive Bayes method), set $ini = 0$, use (9) to form β and then compute $y_i\ \forall i \in I_{all}$ and the active index set I at β.
   - If continuing the solution from one C value to another nearby C value, set $ini = 1$ and simply start with the β, y_i, $i \in I_{all}$ and I available from the previous C solution.

   If $ini = 0$ set $\varepsilon = 10^{-2}$ and *Need_Second_Round*=1. If $ini = 1$ set $\varepsilon = 10^{-6}$ and *Need_Second_Round*=0. Compute $F = f(\beta)$.[14] Set $F_{previous} = F$.

2. Set $iter = 0$ and $itermax = 50$.

3. Define X to be a restricted data matrix whose rows are $(x_i^T, 1)$, $i \in I_k$ and t to be the corresponding target vector whose elements are t_i, $i \in I_k$. (X and t are defined just for stating the steps easily here. In the actual implementation there is no need to actually form them.[15]) Set: $iter = iter + 1$, $\bar\beta = \beta$, $z = t - X\beta$, $r = X^T z - \lambda\beta$, $\phi_1 = \|r\|^2$, $p = r$, $\phi_2 = \phi_1$. If ($ini = 0$ and $iter = 1$) set $cgitermax = 10$; else set $cgitermax = 5000$. Set $cgiter = 0$, $optimality = 0$ and $exit = 0$.

4. Repeat the following steps until $exit = 1$ occurs:

```
$cgiter = cgiter + 1$
$q = Xp$, $\phi_3 = \|q\|^2$
$\gamma = \phi_1/(\phi_3 + \lambda\phi_2)$, $\bar\beta = \bar\beta + \gamma p$
$z = z - \gamma q$, $\phi_4 = \|z\|^2$
$r = -\lambda\bar\beta + X^T z$, $\phi_1^{old} = \phi_1$, $\phi_1 = \|r\|^2$
If $\phi_1 \le \varepsilon^2\phi_4$ set $optimality = 1$
If ($optimality = 1$ or $cgiter \ge cgitermax$) set $exit = 1$
$\omega = \phi_1/\phi_1^{old}$, $p = r + \omega p$, $\phi_2 = \|p\|^2$
```

14. If $\beta = 0$ then note that $F = m/2$.
15. It is ideal to store the input data, $\{x_i, t_i\}$ in the SVM$^{\text{light}}$ format where each example is specified by the target value together with a bunch of (feature-index,value) pairs corresponding to the non-zero components.

5. Compute[16] $\bar{y}_i = y_i(\bar{\beta})$, $\forall i \in I_{all}$. Check if the following conditions hold: (a) *optimality = 1*; (b) $t_i\bar{y}_i \leq 1 + tol$ $\forall i \in I$; and (c) $t_i\bar{y}_i \geq 1 - tol$ $\forall i \notin I$.[17] If all three conditions hold, go to step 8.

6. Compute[18] Δ_1, Δ_2 and Δ using (14) and (15). Sort the δ_i in Δ in non-decreasing order. Let $\{i_1, i_2, \ldots, i_q\}$ denote the list of ordered indices obtained. Compute *ls* and *rs* using (16) and (17). Set $exit = 0$, $j = 0$. Repeat the following steps until $exit = 1$ occurs:

```
$j = j + 1$, $\delta = \delta_{i_j}$
$delslope = ls + \delta(rs - ls)$
If $delslope \geq 0$ set $\delta^\star = -\delta\, ls/(delslope - ls)$ and $exit = 1$
Use (18) to update $ls$ and $rs$ using $i = i_j$.
```

7. Set $\beta := \beta + \delta^\star(\bar{\beta} - \beta)$, $y := y + \delta^\star(\bar{y} - y)$, and compute the new active index set: $I = \{i \in I_{all} : t_i y_i < 1\}$. Compute $F = f(\beta)$. If ($iter \geq itermax$ or $F > F_{previous}$) stop with an error message. Else, set $F_{previous} = F$ and go back to step 3 for another iteration.

8. If *Need_Second_Round*=0 stop with $\beta = \bar{\beta}$, $y = \bar{y}$ and I_k as the optimal active index set. Else, set $\varepsilon = 10^{-6}$, *Need_Second_Round*=0 and go back to step 3.

Appendix B. A Description of Data Sets Used

As in the main paper, let m, n and n_{nz} denote, respectively, the number of examples, the number of features and the number of non-zero elements in the data matrix. Let $s = n_{nz}/(mn)$ denote the sparsity in the data matrix.

Australian is a small dense data set taken from the UCI repository(Blake and Merz, 1998) and it has $m = 690$, $n = 14$ and $s = 1$.

Adult and *Web* are data sets exactly as those used by Platt(Platt, 1999). For *Adult*, n is 120 and s is 0.21, while, for *Web*, n is 300 and $s = 0.04$. With each of these two data sets, Platt created a sequence of data sets with increasing number of examples in order to study the scaling properties of his SMO algorithm with respect to m. *Adult-1*, *Adult-4*, *Adult-7* and *Adult-9* have the m values 1605, 4781, 16100 and 32561. *Web-1*, *Web-4*, *Web-7* and *Web-8* have the m values 2477, 7366, 24692 and 49749.

We generated *News20* for easily reproducible results on a text classification task having both n and m large. It is a size-balanced two-class variant of the UCI “20 Newsgroups” data set (Blake and Merz, 1998). The positive class consists of the 10 groups with names of form `sci.*`, `comp.*`, or `misc.forsale`, and the negative class consists of the other 10 groups. We tokenized via McCallum’s Rainbow(McCallum, 1996), using: `rainbow -g 3 -h -s -O 2 -i` (i.e. trigrams, skip message headers, no stoplist, drop terms occurring less than two times), giving $m = 19996$, $n = 1355191$

16. The arrays, p, r, q and z are local to step 4 and are not used elsewhere. Hence it is alright to use the same arrays elsewhere in the implementation. This can help save some memory. For example, p can be used to store the $\bar{y}_i$ computed in step 5 and r can be used to store the δ_i computed in step 6.
17. In view of numerical errors it is a good idea to employ the parameter tol in these checks. A value of $tol = 10^{-8}$ is a good choice.
18. This step refers to several equations from the main paper. To match the notations given there, take: y_i in this algorithm to be y_i^k of the main paper; *ls* in this pseudocode to stand for l_0, l_i, l_{i+1} etc; and *rs* in this pseudocode to stand for r_0, r_i, r_{i+1} etc.

and $s = 0.000336$. We used binary term frequencies and normalized each example vector to unit length.

Financial is a text classification data set that we created and corresponds to classifying news stories as financial or non-financial. Unigrams occuring in the news texts were taken as the features and a tf-idf representation was used to form the data. The data set has $m = 198788$, $n = 252472$ and $s = 0.00094$.

Yahoo is a large data set obtained from Yahoo! and is a classification problem concerning the prediction of behavior of customers. It has $m =$1 million, $n = 80$ and $s = 0.098$.

References

D. P. Bertsekas. *Nonlinear Programming*. Athena Scientific, Belmont, Massachussetts, 1999.

A. Björck. *Numerical Methods for Least Squares Problems*. SIAM, Philadelphia, 1996.

C. L. Blake and C. J. Merz. UCI repository of machine learning databases. Technical report, University of California, Irvine, 1998. www.ics.uci.edu/∼mlearn/MLRepository.html.

S. Chakrabarti, S. Roy, and M. V. Soundalgekar. Fast and accurate text classification via multiple linear discriminant projections. *The VLDB Journal*, 12:170–185, 2003.

W. Chu and S. S. Keerthi. New approaches to support vector ordinal regression. Technical report, Yahoo! Research Labs, Pasadena, California, USA, 2005.

D. DeCoste and K. Wagstaff. Alpha seeding for support vector machines. In *Proceedings of the International Conference on Knowledge Discovery and Data Mining*, pages 345–359, 2000.

A. Frommer and P. Maaß. Fast CG-based methods for Tikhonov-Phillips regularization. *SIAM Journal of Scientific Computing*, 20(5):1831–1850, 1999.

G. Fung and O. L. Mangasarian. Proximal support vector machine classifiers. In *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 77–86, 2001.

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

T. Joachims. Making large-scale SVM learning practical. In *Advances in Kernel Methods - Support Vector Learning*. MIT Press, Cambridge, Massachussetts, 1999.

W. C. Kao, K.M. Chung, T. Sun, and C. J. Lin. Decomposition methods for linear support vector machines. *Neural Computation*, 16:1689–1704, 2004.

P. Komarek. Logistic regression for data mining and high-dimensional classification. Ph.d. thesis, Carnegie Mellon University, Pittsburgh, Pennsylvania, USA, 2004.

O. L. Mangasarian. A finite Newton method for classification. *Optimization Methods and Software*, 17:913–929, 2002.

A. McCallum. Bow: A toolkit for statistical language modeling, text retrieval, classification and clustering. Technical report, University of Massachssetts, Amherst, Massachussetts, USA, 1996. www.cs.cmu.edu/∼mccallum/bow.

C. C. Paige and M. A. Saunders. LSQR: An algorithm for sparse linear equations and sparse least squares,. *ACM Transactions on Mathematical Software*, 8:43–71, 1982.

J. Platt. Sequential minimal optimization: A fast algorithm for training support vector machines. In *Advances in Kernel Methods - Support Vector Learning*. MIT Press, Cambridge, Massachussetts, 1999.

J. Suykens and J. Vandewalle. Least squares support vector machine classifiers. *Neural Processing Letters*, 9(3):293–300, 1999.

J. Zhang, R. Jin, Y. Yang, and A. Hauptmann. Modified logistic regression: An approximation to SVM and its applications in large-scale text categorization. In *Twentieth International Conference on Machine Learning*, pages 472–479, 2003.

T. Zhang. Statistical behavior and consistency of classification methods based on convex risk minimization. *The Annals of Statistics*, 32:56–85, 2004.