SMOOTHED HINGE LOSS AND ℓ^1 SUPPORT VECTOR MACHINES

JEFFREY HAJEWSKI, SUELY OLIVEIRA AND DAVID E. STEWART

Abstract. A new algorithm is presented for solving the soft-margin Support Vector Machine (SVM) optimization problem with an ℓ^1 penalty. This algorithm is designed to require a modest number of passes over the data, which is an important measure of its cost for very large data sets. The algorithm uses smoothing for the hinge-loss function, and an active set approach for the ℓ^1 penalty.

Key words. support vector machine; smoothing; ℓ^1 penalty

AMS subject classifications. Primary: 65K05, Secondary: 49M15

1. Introduction. Dealing with large data sets has lead to a strong interest in methods that have low iteration costs, such as stochastic gradient descent (SGD) [12, 18]. More classical methods such as Newton’s method for optimization [13, §3.3] are generally not used as their cost per iteration involves solving linear systems which takes O(m^3) operations where m is the number of unknowns. Contrary to conventional wisdom, we argue that Newton’s method and more sophisticated line search methods are actually more appropriate for very large data problems, since there the computational issues are typically due to the large number of data items (n) rather than the dimension of the problem (m). Wide data, where the dimension of the data vectors x_i is large compared to the number of data items, is still problematic for Newton’s method as the Hessian matrix is then singular. However, in this paper we focus on ℓ^1 Support Vector Machines (ℓ^1 SVMs) and argue that Newton’s method with a suitable line search and an active-set strategy can also solve these problems very efficiently. The algorithm developed here is, in part, inspired by [15] for the basis pursuit noise-reduction problem.

This algorithm and the numerical results are reported in the conference paper [9]. The development of the line search algorithm and justification for the convergence of the overall algorithm are not reported in the conference paper.

The soft-margin SVM [7, p. 263] for given data (x_i, y_i), i = 1, 2, ..., n where each y_i = ±1 minimizes

\[ \frac{1}{2} \lambda \|w\|_2^2 + \frac{1}{n} \sum_{i=1}^{n} \max(0, 1 - y_i w^T x_i) \]  

over all \( w \in \mathbb{R}^m \). Here the value of \( \lambda > 0 \) is used to control the size of the vector \( w \). The function \( \max(0, 1 - y_i w^T x_i) \) is called the hinge-loss function as it is based on the function \( u \mapsto \max(0, u) \) whose graph looks like a hinge. The ℓ^1 SVM for the same data minimizes

\[ \frac{1}{2} \lambda \|w\|_2^2 + \frac{1}{n} \sum_{i=1}^{n} \max(0, 1 - y_i w^T x_i) + \alpha \|w\|_1 \]  

over \( w \). Here \( \alpha > 0 \) controls the level of sparsity of \( w \). Larger values tend to mean fewer components of \( w \) are non-zero; if \( \alpha \) is large enough then \( w = 0 \). Note that this formulation is similar to, but not the same as the 1-norm SVM of Zhu, Rosset, Hastie and Tibshirani [20]. Also, the algorithm obtained here is O(n) with respect to the number of data points, while the algorithm of Zhu et al. is \( \Omega(n^2) \) as it involves identifying the intersections of a descent line with the hyperplanes \( 1 - y_i w^T x_i = 0 \) for each data point. Rather we use a smoothing approach for the sum of the hinge-loss functions.
Traditionally, for optimization problems, the numbers of function, gradient, and Hessian matrix evaluations are used to measure the cost of the algorithm. For large-scale data mining types of optimization problems, perhaps a different measure of performance is more important: the number of passes over the data. The general form of most optimization problems used in data mining is

$$\min_w f(w) := R(w) + \frac{1}{n} \sum_{i=1}^n \psi(x_i, y_i; w)$$

(1.3)

where $R$ is a regularization function, and $\psi$ is a loss function. Provided $w$ has relatively low dimension (say, below $10^3$) and $n$ is large (say, $10^5$ to $10^9$), the cost of computing $R(w)$ is modest and can be computed on one processor, while the computations of $\psi(x_i, y_i; w)$ should be carried out in parallel, and then summed via a parallel reduction operation [4].

Computing the gradient of the objective function

$$\nabla f(w) = \nabla R(w) + \frac{1}{n} \sum_{i=1}^n \nabla_w \psi(x_i, y_i; w),$$

which can be computed in a similar manner to the objective function, except that the reduction (summation) is applied to the gradients $\nabla_w \psi(x_i, y_i; w)$. Similarly, the Hessian matrices can be computed in parallel but the reduction (summation) is applied to the Hessian matrices $\text{Hess}_w \psi(x_i, y_i; w)$ of the loss functions. This may become an expensive step if $m$ becomes large, as the reduction must be applied to objects of size $O(m^2)$ where $w \in \mathbb{R}^m$. In such cases, a BFGS algorithm may be appropriate instead of a direct Newton method.

If the function $R(w)$ is non-smooth in $w$ (as is the case for (1.2)), then the optimization algorithm needs to be adapted for it.

1.1. Problems with the line search. Line searches are often needed in optimization algorithms because the predicted step from Newton’s method “goes too far”, or in some other way results in an increase in the objective function value or does not decrease it significantly. Suppose the step for the Newton method is $d$. If the quadratic Taylor polynomial at $s = 0$ to $f(w + sd)$ is a poor approximation to $f(w + sd)$, then it may be necessary to perform many line search steps, which will require many function evaluations. This is costly in the context of parallel computation with high latency networks.

Lack of smoothness can be a cause of this, and result in many costly parallel reduction steps. Thus the shape of the $R(w)$ function must be known by the line search procedure at least to fairly good accuracy. In the case of the $\ell^1$SVM problem, this means that the nonsmoothness of the $\ell^1$ penalty must be explicitly represented and used for the line search procedure.

1.2. Non-smoothness for $\ell^1$SVM. The advantage of using $\ell^1$SVM over a standard SVM formulation is that the $\ell^1$ penalty tends to result in sparse solutions. That is, with the $\ell^1$ penalty, the number of indexes $i$ where $w_i \neq 0$ tends to be small. In fact, if the weight $\alpha > 0$ is large enough, then the solution is $w = 0$. If $\alpha$ is smaller, we usually expect $w_i \neq 0$ for a modest number of indexes $i$. Sparse solutions have a number of advantages. There is a much lower likelihood of over-fitting the data. The solution is more likely to be “explainable” in the sense that the set of $i$ where $w_i \neq 0$ is small or modest, so that the method essentially selects those parameters as being important. Finally, since fewer parameters are used to create the “fit”, there is probably less noise in each of the parameters. Models with large numbers of parameters, tend to have much less “data per parameter”, so that the numerical values obtained tend to be less reliable.
The disadvantage is that the numerical algorithm for performing the optimization has to be adapted to deal with the non-smoothness. Since the important non-smooth part of the objective function in [1.2] is $\alpha \|w\|_1$ is highly structured, we can exploit this structure to create a fast and efficient algorithm. To do this, an active set is maintained $I = \{ i \mid w_i \neq 0 \}$. This needs to be expanded when new parameters $w_i$ are made active, or available for optimization, and reduced when a line search indicates that $w_i = 0$ seems optimal for an active parameter $w_i$. If $f(w) = g(w) + \alpha \|w\|_1$ with $g$ smooth, an inactive parameter $w_i$ should be made active if $|\partial g / \partial w_i(w)| > \alpha$. With this strategy, many parameters can be made active in one step, but only one active parameter can become inactive in one step.

1.3. Smoothing the hinge-loss function and convergence of Hessian matrices. The hinge-loss function $\psi(x, y; w) = \max(0, 1 - y w^T x)$ is a piecewise linear function of $w$, and so its Hessian matrix is zero or undefined. Thus

$$\frac{1}{n} \sum_{i=1}^{n} \psi(x_i, y_i; w)$$

is also a piecewise linear function of $w$, and thus its Hessian matrix is either zero or undefined. On the other hand,

$$\frac{1}{n} \sum_{i=1}^{n} \psi(x_i, y_i; w)$$

usually appears to be very smooth. For example, if $n = 200$, and for $y = +1$, $x$ is chosen randomly and uniformly from $[0, +1]$ while for $y = -1$, $x$ is chosen uniformly and randomly from $[-1, 0]$, $n^{-1} \sum_{i=1}^{n} \psi(x_i, y_i; w)$ looks like Figure 1.1.

As $n \to \infty$ under some statistical assumptions detailed below, the function $n^{-1} \sum_{i=1}^{n} \psi(x_i, y_i; w)$ approaches a smooth function $h(w)$. Rather than compute the exact Hessian matrix of
\[ n^{-1} \sum_{i=1}^{n} \psi(x_i, y_i; w) \] with respect to \( w \), which has very little to do with the overall behavior of the function, we should aim to compute an approximation to the Hessian matrix of \( h(w) \). This can be done by means of a smoothed hinge-loss function. Using a smoothed hinge-loss function does not change the value of \( n^{-1} \sum_{i=1}^{n} \psi(x_i, y_i; w) \) significantly, but does enable us to estimate the Hessian matrix of \( h(w) \), as well as its gradient.

For large data sets which come from some statistical distribution with a \( C^2 \) probability density function, the mean of the hinge-loss functions approaches a \( C^2 \) function

\[
\frac{1}{n} \sum_{i=1}^{n} \psi(x_i, y_i; w) \rightarrow \int \psi(x, +1; w) p_1(x) \, dx + \int \psi(x, -1; w) p_2(x) \, dx =: h(w) \tag{1.4}
\]

as \( n \to \infty \). Here \( p_1(x) \) is the probability density function of \( x \) given that \( y = +1 \), while \( p_2(x) \) is the probability density function of \( x \) given that \( y = -1 \). The values of the averages \( (1/n) \sum_{i=1}^{n} \psi(x_i, y_i; w) \) can then be well-approximated by a \( C^2 \) function. The difficulty is in estimating the Hessian matrix of this unknown smooth function. We can approximate \( \psi(x, y; w) \) by a smoothed hinge-loss function \( \psi_s(x, y; w) \) given by

\[
\psi_s(x, y; w) = \frac{1}{2} (u + \sqrt{\epsilon^2 + u^2}) \text{ where } u = 1 - y w^T x. \tag{1.5}
\]

What we want is that

\[
\frac{1}{n} \sum_{i=1}^{n} \text{Hess}_w \psi_s(x_i, y_i; w) \approx \text{Hess} h(w)
\]

for \( n \) sufficiently large. Now

\[
\text{Hess} h(w) = \int \text{Hess}_w \psi(x, +1; w) p_1(x) \, dx + \int \text{Hess}_w \psi(x, -1; w) p_2(x) \, dx.
\]

and \( \text{Hess}_w \psi(x, y; w) = y^2 x x^T \delta((1 - y x^T w)/ \| y x \|) \) where \( \delta \) is the Dirac-\( \delta \) distribution. That is,

\[
\int \text{Hess}_w \psi(x, +1; w) p_1(x) \, dx = \int_{\{x | 1 - w^T x = 0\}} x x^T p_1(x) \, dS(x)
\]

where the latter is a surface integral.

Note that \( \psi(x, y; w) = j(1 - y w^T x) \) with \( j(u) = \max(0, u) \), while \( \psi_s(x, y; w) = j_s(1 - y w^T x) \) with \( j_s(u) = \frac{1}{2} (u + \sqrt{\epsilon^2 + u^2}) \). Now \( j''(u) = \delta(u) \) while

\[
j_s''(u) = \frac{1}{2} ((\epsilon^2 + u^2)^{-1/2} - u^2 (\epsilon^2 + u^2)^{-3/2}) = \frac{1}{2} (\epsilon^2 + u^2)^{3/2},
\]

which converges to \( \delta(u) \) in the sense of distributions (and the sense of measures, although weakly) as \( \epsilon \to 0 \). Thus for continuous \( p_1 \),

\[
\int \text{Hess}_w \psi_s(x, +1; w) p_1(x) \, dx \to \int_{\{x | 1 - w^T x = 0\}} x x^T p_1(x) \, dS(x) \quad \text{as } \epsilon \to 0.
\]

The integral on the right is an \((m - 1)\)-dimensional integral over the hyperplane. Also, if we choose \( x_i \) independently, and distributed according to the probability distribution \( p_1 \), and the
Since the same arguments apply for \( p_1 \) is finite, then by the Strong Law of Large Numbers \[11\] p. 239],

\[
\frac{1}{n} \sum_{i=1}^{n} \text{Hess}_w \psi(x_i, +1; w) \to \int \text{Hess}_w \psi(x, +1; w) p_1(x) \, dx \quad \text{almost surely.} \tag{1.6}
\]

Since the same arguments apply for \( p_2 \) and the samples where \( y_i = -1 \),

\[
\lim_{\epsilon \to 0} \lim_{n \to \infty} \frac{1}{n} \sum_{i=1}^{n} \text{Hess}_w \psi(x_i; y_i; w) = \text{Hess}_w h(w) \quad \text{almost surely.} \tag{1.7}
\]

To make this work in a practical sense, we need the number of samples \( n \) to be “sufficiently large” for a given \( \epsilon > 0 \) in order to have

\[
\frac{1}{n} \sum_{i=1}^{n} \text{Hess}_w \psi(x_i; y_i; w) \approx \text{Hess}_w h(w),
\]

at least with high probability. A natural question is how large \( n \) has to be for a given \( \epsilon \) in order to have a good approximation. Since \( \psi_r(x, +1; w) \) only depends on \( w^T x \) (and similarly for \( \psi_r(x, -1; w) \)), we only need \( n \sim \text{const} \|w\|^{-1} \) as \( \epsilon \to 0 \) in order to achieve a given level of accuracy in approximating \( \text{Hess} h(w) \). Thus the number of data points needed to obtain a good approximation of the curvature of the objective function is not exorbitant.

2. Development of the Algorithm.

2.1. Choice of \( \epsilon \). We wish to use a modified Newton method for minimizing \( f(w) \) from \[1.3\] As noted in Section \[1.3\], Hess \( f(w) \) is either undefined or Hess \( R(w) \), if it has a Hessian matrix. This is misleading, and will not lead to fast convergence. Instead we use the smoothed hinge-loss function \( \psi_\epsilon \) for suitable \( \epsilon > 0 \). The problem then is to choose \( \epsilon \). From the analysis in Section \[1.3\], we can choose \( \epsilon \) to be inversely proportional to \( n \), the number of data points. With this approach, it might also be necessary to adapt \( \epsilon \) according to the distribution of the data points \( (x_i, y_i) \).

However, there is another approach which is agnostic regarding \( n \) and the distribution of the data points. This is to simply begin with a large value of \( \epsilon \), minimize \( f_\epsilon(w) \) over \( w \), then repeatedly reduce \( \epsilon \) by (for example) halving \( \epsilon \), and then minimizing \( f_\epsilon(w) \) over \( w \) with this new value of \( \epsilon \).

2.2. Line-search algorithm. In the context of parallel computing, it is important to keep the number of function evaluations small. So it is important to use a “good” first guess. With Newton methods applied to smooth functions \( \psi \), it is traditional to use the step length \( s = 1 \) with the Newton step \( d = -(\text{Hess} \psi(w))^{-1} \nabla \psi(w) \) followed by the Armijo line search (see \[2\], \[14\] p. 33)). However, with non-smooth functions, such as the \( \ell^1 \) penalty, this choice can result in many function evaluations for a single line search.

With the \( \ell^1 \) penalty, we have to consider the problem of minimizing \( \psi(w + s d) + \alpha \|w + s d\|_1 \) over \( s \geq 0 \) efficiently where \( \psi \) is a smooth function. Since we can estimate the Hessian matrices accurately, we can use a quadratic approximation for \( \psi(w + s d) \approx a s^2 + b s + c \). Then our line search seeks to minimize

\[
j(s) := a s^2 + b s + c + \alpha \|w + s d\|_1 \quad \text{over } s \geq 0. \tag{2.1}
\]

Provided \( a, \alpha \geq 0 \), this is a convex function, and so the derivative \( j'(s) \) is a non-decreasing function of \( s \). Provided \( \|d\|_1 > b \) or \( a > 0 \) or \( \alpha > 0 \), there is a global minimizer of \( j \); if
\( a > 0 \) then it is unique. The task is to compute this minimizer efficiently. This minimizer is characterized by either \( j'(s) = 0 \), or \( j'(s^-) \leq 0 \) and \( j'(s^+) \geq 0 \).

This can be done using a binary search algorithm that just uses the data mentioned: \( a, b, \alpha, w \) and \( d \), and no additional function evaluations. All the information needed from \( \psi \) is \( a \) and \( b \), which can be computed from the gradient and the Hessian matrix of \( \psi \) at \( w \).

Note that

\[
 j'(s) = 2as + b + \alpha \sum_{i=1}^{m} \text{sign}(w_i + sd_i) d_i. \tag{2.2}
\]

If \( \alpha = 0 \) and \( a > 0 \) then clearly the minimizing \( s = -b/(2a) \). Assuming \( a > 0 \) and \( \alpha \geq 0 \), the minimizing value of \( s \) must lie in the interval \([0, s_{\text{max}}]\) where \( s_{\text{max}} = (|b| + \alpha \|d\|_1)/(2a) \).

The points of discontinuity of \( j'(s) \) are \( \sigma_i = -w_i/d_i, i = 1, 2, \ldots, m \). If any \( d_i = 0 \), we can simply ignore \( \sigma_i \). Let \( \{\tilde{\sigma}_1, \tilde{\sigma}_2, \ldots, \tilde{\sigma}_r\} = \{\sigma_i | \sigma_i > 0\} \) with \( \tilde{\sigma}_1 < \tilde{\sigma}_2 < \cdots < \tilde{\sigma}_r \).

Set \( \tilde{\sigma}_0 = 0 \). We check \( j'(-\tilde{\sigma}_0^+) \) and \( j'(\tilde{\sigma}_0^+) \). If \( j'(\tilde{\sigma}_0^+) \geq 0 \) then the optimal \( s \) is \( s^* = 0 = \tilde{\sigma}_0 \).

If \( a = 0 \) and \( j'(\tilde{\sigma}_r^+) < 0 \) then \( j(s) \rightarrow -\infty \) as \( s \rightarrow \infty \) and there is no minimum. If \( a > 0 \) and \( j'(\tilde{\sigma}_r^+) < 0 \) then the optimal \( s \) is

\[
 s^* = -\frac{1}{2a} \left( b + \sum_{i=1}^{m} \text{sign}(d_i)d_i \right) = -\frac{b + \alpha \|d\|_1}{2a} = \tilde{\sigma}_r - \frac{j'(\tilde{\sigma}_r^+)}{2a} > \tilde{\sigma}_r,
\]

since \( j'(\tilde{\sigma}_r^+) = 2a\tilde{\sigma}_r + b + \alpha \|d\|_1 \).

Consider the sequence

\[
 j'(\tilde{\sigma}_0^+), j'(\tilde{\sigma}_1^+), j'(\tilde{\sigma}_2^+), \ldots, j'(\tilde{\sigma}_r^+). \tag{2.3}
\]

Since this sequence is a non-decreasing sequence, if \( j'(\tilde{\sigma}_i^+)<0 \) and \( j'(\tilde{\sigma}_{i+1}^+)>0 \) then it crosses from being \( \leq 0 \) to \( > 0 \) at some point. If \( j'(\tilde{\sigma}_i^+)=0 \) for some \( i \) and choice of sign, then \( s^* = \tilde{\sigma}_i \). So we assume without loss of generality that \( j'(\tilde{\sigma}_i^+) \neq 0 \) for any \( i \) and choice of sign. In this case, either there is an \( i \) where \( j'(\tilde{\sigma}_i^+)<0 \) and \( j'(\tilde{\sigma}_{i+1}^+)>0 \), or there is an \( i \) where \( j'(\tilde{\sigma}_i^-)<0 \) and \( j'(\tilde{\sigma}_{i+1}^+)>0 \). If \( j'(\tilde{\sigma}_i^-)<0 \) and \( j'(\tilde{\sigma}_{i+1}^-)>0 \), then \( s^* = \tilde{\sigma}_i \). If \( j'(\tilde{\sigma}_i^-)<0 \) and \( j'(\tilde{\sigma}_{i+1}^-)>0 \), then \( s^* \in (\tilde{\sigma}_i, \tilde{\sigma}_{i+1}) \).

Finding the point where the sequence \( \{2.3\} \) crosses zero can be carried out by binary search or a discrete version of the bisection algorithm. Thus it can be computed in \( O(\log m) \) time as \( r \leq m \).

If the optimal value for \( s^* \) is zero, then \( d \) is not a descent direction [Ref] and so some other direction should be used. This can only occur if \( \sigma_i = 0 \) for some \( i \), indicating that \( w_i = 0 \). Then in this case, we need to remove \( w_i \) from the set of active variables.

**2.3. Combining the parts.** A complete algorithm is outlined in Algorithm \([\text{I}]\). In this algorithm it should be noted that we use the following definitions:

\[
 a \circ b = c \quad \text{where} \quad c_i = a_i b_i \quad \text{(Hadamard product)}
\]

\[
 \hat{f}_\alpha(w) = (1/m) \sum_{i=1}^{m} \psi_\alpha(x_i, y_i; 1 - y_i x_i^T w) + \frac{1}{2} \lambda(w)^T w
\]

\[
 f_\alpha(w) = f_\alpha(w) + \mu \|w\|_1.
\]

The inputs to \( \psi_\alpha(1 - y_i x_i^T w) \) form the vector \( e - y \circ (X w) \). Note that \( e \) is the vector of 1’s of the appropriate size. This vector formulation is helpful in languages such as Matlab\(^T\).
Also, the matrix \( X = [x_1, x_2, \ldots, x_n]^T \) so that \( Xw = [x_1^T w, \ldots, x_n^T w]^T \). Note that \( \nabla \hat{f}_\alpha(w) \) is well-defined for all \( w \) provided \( \alpha > 0 \), but that \( f_\alpha \) is not smooth.

The algorithm used can be broken down into a number of pieces. At the top level, the method can be considered as applying Newton’s method to a smoothed problem (smoothing parameter \( \alpha \)) keeping an inactive set \( I = \{ i \mid w_i = 0 \} \). This inactive set will need to change, either by gaining elements where \( w_j \neq 0 \) but \( w_j + sd_j = 0 \) resulting from the line search procedure, or by losing elements where \( w_i = 0 \) but the gradient component \( g_i = \partial \hat{f}_\alpha / \partial w_i(w) \) satisfies \( |g_i| > \mu \) indicating that allowing \( w_i \neq 0 \) will result in a lower objective function value. Note that \( \hat{f}_\alpha \) does not include the \( \ell^1 \) penalty term \( \mu \| w \|_1 \). The top-level computations are shown in Algorithm 1.

An essential choice in this algorithm is not to smooth the \( \ell^1 \) penalty term, and instead use an active/inactive set approach. If we had chosen to smooth the \( \ell^1 \) penalty term, then the computational benefits of the smaller linear system in the Newton step \( d_\alpha \leftarrow -H_{\hat{\alpha}}^{-1}g_\alpha \) would be lost. Instead, smoothing the \( \ell^1 \) term would mean that the linear system to be solved would have size \( m \times m \) where \( m \) is the dimension of \( w \). This would be particularly important for problems with wide data sets where \( m \) can be very large. Instead, we expect that there would be bounds on the size of \( I \), the number of active weights \( w_i \neq 0 \).

Algorithm 1 Algorithm for SVM with \( \ell^1 \) penalty

Require: \( \alpha, \alpha_{\text{min}}, \mu, \lambda > 0 \)

1: \textbf{function} SVMsmooth\((X, y, w, \lambda, \mu, \alpha, \alpha_{\text{min}})\)
2: \( I \leftarrow \{ i \mid w_i = 0 \} \)
3: \( J \leftarrow \{ i \in I \mid |g_i| > \mu \} \)
4: \textbf{while} \( \alpha > \alpha_{\text{min}}/\beta \) \textbf{do} \hfill \triangleright \text{While smoothing parameter not at threshold}
5: \hspace{1em} Carry out Newton step on smoothed problem
6: \textbf{end while}
7: \textbf{return} \( w \)
8: \textbf{end function}

The Newton step computations are shown in Algorithm 2. We first compute the gradient and the Hessian matrix. Care must be taken at this point to ensure that we compute the correct gradient for the components \( j \) where \( w_j = 0 \) but \( |g_j| > \mu \). The full Hessian matrix is not actually needed, just the “active” part of the Hessian matrix: \( H_{I \setminus \tau} \). The Newton step \( d \) is computed. If the predicted reduction of the function value is sufficiently small, then we can assume the problem for the current inactive set \( I \) and smoothing parameter \( \alpha > 0 \) has been solved to sufficient accuracy. Then we can either reduce the current inactive set \( I \) or reduce the smoothing parameter \( \alpha \) as shown in Algorithm 3. The Newton steps then continue until either the inactive set or the smoothing parameter is reduced. If the smoothing parameter goes below \( \alpha_{\text{min}} \), then the algorithm terminates.

The linesearch algorithm is shown as Algorithm 4.

The actual implementation differs slightly from the pseudo-code in that the recomputation of \( I \) on line 2 of Algorithm 2 uses some additional information returned from LINESearch1: in floating point arithmetic there is no guarantee that \( I \leftarrow \{ i \mid w_i = 0 \} \) will identify components \( w_i \) that would be set to zero in exact arithmetic. Specifically, setting \( s \leftarrow -w_j/d_j \) in does not ensure that \( w_j + s d_j \) evaluates to zero in floating point arithmetic. So the linesearch function LINESearch1 actually returns both \( s \) and \( j_1 \) and \( j_2 \): if \( j_1 = j_2 \), then \( s = -w_j/d_j \) for \( j = j_1 = j_2 \) and we would set \( w_j + s d_j = 0 \) and the new set \( I \) is the old \( I \) plus \( j \).

Thus, elements can be added to \( I \) (line 14 of Algorithm 2) as well as removed from
Algorithm 2 Newton step

1: \( g \leftarrow \nabla \hat{f}_\alpha(w) \)
2: \( \bar{g} \leftarrow g + \mu \text{sign}(w) \)
3: \( g_j \leftarrow g_j + \mu \text{sign}(g_j) \) for all \( j \in \mathcal{J} \)
4: \( H = \lambda I + \left( \frac{1}{m} \right) X^T \text{diag}(\psi''_\alpha(z)) X \) \quad \triangleright H \text{ is Hessian matrix} \)
5: \( d_x \leftarrow -H^{-1}x; d_x \leftarrow 0 \) \quad \triangleright \text{Newton step} \)
6: \( \text{if } |d^T \bar{g}| < \alpha/10 \text{ then } \quad \triangleright \text{If smoothed problem nearly solved for } \alpha \text{ and } I \ldots \)
7: \( \text{Adjust active set & reduce smoothing parameter} \)
8: \( \text{continue} \)
9: \( \text{end if} \)
10: \( s \leftarrow \text{LinesearchL1}(w, d, g^T d, \frac{1}{2} d^T H d, \mu); w^+ \leftarrow w + s d \)
11: \( \text{while } f_\alpha(w^+) > f_\alpha(w) + c_1 s d^T \bar{g} \text{ do} \quad \triangleright \text{Armijo line search} \)
12: \( \quad s \leftarrow s/2; w^+ \leftarrow w + s d \)
13: \( \text{end while} \)
14: \( w \leftarrow w^+; I \leftarrow \{ i \mid w_i = 0 \} \) \quad \triangleright \text{Add to } I \text{ if line search indicates} \)

Algorithm 3 Adjust active set & reduce smoothing parameter

1: \( \mathcal{J}' \leftarrow \{ i \in I \mid |g_i| > \mu \} \)
2: \( \text{if } \mathcal{J}' \neq \mathcal{J} \text{ then} \)
3: \( \quad \mathcal{J} \leftarrow \mathcal{J}'; I \leftarrow I \setminus \mathcal{J}' \); \( \text{continue} \)
4: \( \text{end if} \)
5: \( \alpha \leftarrow \alpha/\beta \) \quad \triangleright \text{Reduce } \alpha \text{ and optimize for this new } \alpha \)

\( I \) (line 3 of Algorithm [3]). Note, however, that while this approach can remove multiple elements of \( I \) in a single iteration, only a single element can be added per iteration. This means that the dimension of \( w \) can strongly affect the number of iterations if \( I \) at the optimum has many elements. As removal of elements of \( I \) is easier than addition of elements, it is probably better to begin with \( I = \{ 1, 2, \ldots, m \} \) and \( w = 0 \).

3. Results. Our experiments explore SmSVM’s performance using both real and synthetic data (see Table 3.1 for a detailed description of the data). We look at the ability of our models to accurately classify test data while maintaining, and in many cases improving, state of the art training time. Additionally, we study the robustness of the model as the training data becomes increasingly sparse by increasing the number of components equal to zero in the two centroids used to generate the synthetic data. This is discussed in greater detail in Section 3.1. The results of this Section were previously published in [2].

**Table 3.1**

Description of datasets used in performance comparison experiments. Sparsity refers to the percentage of the data with a value of 0. Note that for the synthetic datasets, the sparsity levels vary based on experiment.

| Name            | Count | Dimension | Sparsity |
|-----------------|-------|-----------|----------|
| Australian      | 690   | 14        | 13%      |
| Colon Cancer    | 62    | 2,000     | 0%       |
| CoverType       | 581,012 | 54        | 78%      |
| Synthetic (tall)| 10,000 | 50        | N/A      |
| Synthetic (wide)| 50    | 2,500     | N/A      |
We compare our algorithms against conjugate gradient (Polak-Ribi`ere Plus [13, 17]), subgradient descent, stochastic subgradient descent, and coordinate descent (via LIBLINEAR [8]). In the case of conjugate gradient, since our loss function is non-smooth, we use a subgradient descent, stochastic subgradient descent, and coordinate descent (via LIBLINEAR, µ, s_max ≥ 0).

Table 3.2 describes the naming convention used in the following sections along with a brief description of the algorithms.

We consider four different optimization problems in the following experiments. SmSVM–ℓ2 minimizes the loss function defined in equation (1.1), while SmSVM–ℓ1–ℓ2 minimizes the loss function defined in equation (1.2). The standard conjugate gradient optimizer minimizes (3.1).

\[ \frac{1}{n} \sum_{i=1}^{n} \max \{ 0, 1 - y_i w^T x_i \} \]
### Table 3.2
Summary of objectives and algorithms.

| Name                  | Description                                      |
|-----------------------|--------------------------------------------------|
| SmSVM–$\ell^2$       | $\ell^2$ regularization                         |
| SmSVM–$\ell^1$–$\ell^2$ | $\ell^2$ and $\ell^1$ regularization           |
| LinearSVC             | LIBLINEAR [8]                                   |
| SGD $\ell^2$         | SGD $\ell^2$ regularization                     |
| SSGD $\ell^2$ mb     | SGD $\ell^2$ regularization mini-batch size of 32 |
| CG                    | Polak-Ribi`ere Plus [17] conjugate gradient solves equation (3.1) |
| CG – $\ell^2$        | Polak-Ribi`ere Plus [17] conjugate gradient with $\ell^2$ regularization |

solves a scaled version of equation (1.1), shown in equation (3.2).

$$C \sum_{i=1}^{n} \max\{0, 1 - y_i \mathbf{w}^T \mathbf{x}_i\} + \frac{1}{2} \| \mathbf{w} \|_2^2, \quad C > 0$$

(3.2)

In our case, this is optimized via coordinate descent (see [8] for details).

For experiments involving synthetic data, new data is generated each repetition of the experiment. Unless otherwise noted, all experiments are performed 50 times.

#### 3.1. Data.

We use both synthetic and real data to compare the SmSVM algorithms against the conjugate gradient and gradient descent algorithms mentioned in Table 3.2. Table 3.1 describes the data used in the experiments. The synthetic data is generated by creating two centroids with components randomly sampled from $N(0, 1)$, scaling the centroids, and then sampling $\mathbf{x} \sim N(\mathbf{c}_i, \mathbb{I}_m)$ where $\mathbf{c}_i \in \mathbb{R}^m$ is the respective centroid. Sparse data is created by setting randomly selected components of the centroids to zero, and then randomly sampling about the updated centroids. The real datasets used in the experiments were sourced from the UCI Machine Learning Repository [6]. The Australian and Colon Cancer [1] datasets were chosen for their shapes, with the Australian dataset being tall and narrow while the Colon Cancer dataset is short and wide. The CoverType [3] dataset was chosen due to its size and is the largest dataset we ran in our experiments. As noted in Table 3.3, the CoverType dataset was only run 20 times, due to compute time constraints.

Table 3.3 summarizes the overall results of test accuracy and training time on the four datasets.

#### 3.2. Performance and Implementation.

As seen in Table 3.3, SmSVM–$\ell^1$–$\ell^2$ performs well across a variety of dataset types, and is beaten only by other SmSVM algorithms and LIBLINEAR [8]. Most notable is the incredibly fast training time, which is due to the optimizations made available via the feature selection property of the $\ell^1$ norm. We optimize the matrix-vector and vector-vector operations by reducing the problem size to that of the active set dimension. The reduction in problem size yields substantial computational savings in problems where the active set is small. The savings are apparent in the real-world datasets where SmSVM–$\ell^1$–$\ell^2$ finished training, in some cases, by an order of magnitude shorter time.

One interesting aspect of SmSVM–$\ell^1$–$\ell^2$’s performance is its apparent struggle in terms of training time on the Colon Cancer dataset, which is a dense dataset. Although SmSVM–$\ell^1$–$\ell^2$ tied with SGD $\ell^2$ for the top test accuracy, the SmSVM family of algorithms were among the slowest to finish training.
Table 3.3
Numerical results for the real world datasets. These results are the average of 50 independent runs.

| Algorithm          | Australian Acc. | Australian Time (s) | Colon Cancer Acc. | Colon Cancer Time (s) | CoverType Acc. | CoverType Time (s) |
|--------------------|-----------------|---------------------|-------------------|-----------------------|---------------|-------------------|
| SmSVM              | 44.5            | 0.051               | 38.0              | 5.143                 | 51.2          | 44.2              |
| SmSVM–ℓ₂           | 85.9            | 0.058               | 66.3              | 32.851                | 69.8          | 148.7             |
| SmSVM–ℓ₁–ℓ₂        | 86.1            | 0.002               | 84.0              | 0.918                 | 69.5          | 1.1               |
| LinearSVC-Hinge    | 85.2            | 0.007               | 66.9              | 0.008                 | 76.3          | 182.9             |
| SGD ℓ²             | 85.9            | 0.008               | 84.0              | 0.023                 | 68.3          | 30.7              |
| SSGD ℓ² mb         | 85.9            | 0.058               | 80.9              | 0.016                 | 63.9          | 53.0              |
| CG                 | 86.0            | 1.375               | 75.1              | 0.940                 | 68.4          | 754.6             |
| CG – ℓ²            | 85.9            | 1.355               | 77.1              | 2.350                 | 68.4          | 746.3             |

Table 3.3
Numerical results for the real world datasets. These results are the average of 50 independent runs.

| Algorithm          | Synthetic Tall Acc. | Synthetic Tall Time (s) | Synthetic Wide Acc. | Synthetic Wide Time (s) |
|--------------------|---------------------|-------------------------|----------------------|-------------------------|
| SmSVM              | 49.9                | 0.15                    | 100                  | 11.89                   |
| SmSVM–ℓ₂           | 84.3                | 0.16                    | 100                  | 23.41                   |
| SmSVM–ℓ₁–ℓ₂        | 76.0                | 0.01                    | 93.6                 | 0.21                    |
| LinearSVC-Hinge    | 100                 | 0.03                    | 100                  | 0.01                    |
| SGD ℓ²             | 61.1                | 0.16                    | 51.2                 | 0.02                    |
| SSGD ℓ² mb         | 77.5                | 0.62                    | 54.0                 | 0.02                    |
| CG                 | 94.9                | 4.84                    | 82.0                 | 0.19                    |
| CG – ℓ²            | 78.6                | 5.64                    | 80.8                 | 1.29                    |

*Results based on 20 runs due to computational requirements.

Perhaps the most surprising result is the performance on the CoverType [3] dataset. Consisting of nearly 600,000 data points and roughly 70MB in uncompressed libSVM sparse format (only non-zero values and their indices are given, everything else is assumed 0). LIBLINEAR took nearly 3 minutes to train on this dataset, achieving a best-in-class test accuracy, while SmSVM–ℓ₁–ℓ₂ trained in just over one second and achieving nearly a second-place test accuracy. The closest algorithm to SmSVM–ℓ₁–ℓ₂ in terms of training time is SGD ℓ², which was nearly 30 seconds slower and had a lower test accuracy.

The SmSVM, CG, and SGD optimizers were all implemented in pure python and make extensive use of Numpy [19]. We implemented these algorithms as efficiently as possible, and in particular, focused on reducing data-copying as much as possible. The LIBLINEAR implementation was accessed via Scikit-learn [16], which provides a Python wrapper on the C++ implementation.

4. Discussion. We have introduced SmSVM, a new approach to solving soft-margin SVM, which is capable of strong test accuracy without sacrificing training speed. This is achieved by smoothing the hinge-loss function and using an active set approach to the ℓ₁ penalty. SmSVM provides improved test accuracy over LIBLINEAR with comparable, and in some cases reduced, training time. SmSVM uses orders of magnitude fewer gradient calculations and a modest number of passes over the data to achieve its results, meaning it will scales well for increasing problem sizes. SmSVM–ℓ₁–ℓ₂ optimizes its matrix-vector and vector-vector calculations by reducing the problem size to that of the active set. For even modestly sized problems this results in significant savings with respect to computational complexity.

Overall, the results are quite promising. On the real and synthetic datasets, our algorithms
outperform or tie the competition in test accuracy 80% of the time and have the fastest training time 60% of the time. The time savings are increasingly significant as the number of data points grows. The results of the wide synthetic dataset are somewhat surprising in that the SmSVM–$\ell^3$–$\ell^2$ algorithm performed worse than the SmSVM–$\ell^2$ algorithm with respect to test accuracy. This is likely due to the SmSVM–$\ell^1$–$\ell^2$ algorithm pushing features out of the active set too aggressively. On the other hand, training time was nearly two orders of magnitude faster, due to the active set being considerably smaller, which allows us to optimize some of the linear algebra operations.

SmSVM is implemented in Python, making it easy to modify and understand. The use of Numpy keeps linear algebra operations optimized—this is important when competing against frameworks such as LIBLINEAR, which is implemented in C++. Testing SmSVM on larger datasets, incorporating GPU acceleration to the linear algebra, and exploring distributed implementations are promising future directions.

REFERENCES

[1] U. Alon, N. Barkai, D. Notterman, K. Gish, S. Ybarra, D. Mack, and A. Levine, Broad patterns of gene expression revealed by clustering analysis of tumor and normal colon tissues probed by oligonucleotide arrays. Cell Biology, 96 (1999), pp. 6745–6750.
[2] L. Armijo, Minimization of functions having Lipschitz continuous first partial derivatives, Pacific J. Math., 16 (1966), pp. 1–3.
[3] J. A. Blackard and D. J. Dean, Comparative accuracies of neural networks and discriminant analysis in predicting forest cover types from cartographic variables, in Second Southern Forestry GIS Conference, 1998. Taken from UCI Machine Learning Repository.
[4] G. E. Blelloch, Programming parallel algorithms, Commun. ACM, 39 (1996), pp. 85–97.
[5] L. Bottou, F. E. Curtis, and J. Nocedal, Optimization methods for large-scale machine learning, 2016.
[6] D. Dheeru and E. Karra Taniskidou, UCI machine learning repository, http://archive.ics.uci.edu/ml, 2017.
[7] R. O. Duda, P. E. Hart, and D. G. Stork, Pattern Classification (2nd Edition), Wiley-Interscience, 2000.
[8] R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin, LIBLINEAR: A library for large linear classification, Journal of Machine Learning Research, 9 (2008), pp. 1871–1874.
[9] J. Hajewski, S. Oliveira, and D. E. Stewart, Smoothed hinge loss and $\ell^1$ support vector machines, in International Conference on Data Mining, 2018. Submitted.
[10] J. Hiriart-Urruty and C. Lemaréchal, Fundamentals of Convex Analysis, Springer-Verlag, New York, NY, 2nd ed., 2004.
[11] M. Loève, Probability Theory I, Springer-Verlag, New York, NY, 4th ed., 1977.
[12] F. Niu, B. Recht, C. Re, and S. J. Wright, Hogwild!: A lock-free approach to parallelizing stochastic gradient descent, in Proceedings of the 24th International Conference on Neural Information Processing Systems, NIPS’11, USA, 2011, Curran Associates Inc., pp. 693–701.
[13] J. Nocedal and S. J. Wright, Numerical Optimization, Springer, New York, 2nd ed., 2006.
[14] J. Nocedal and S. J. Wright, Numerical Optimization, Springer Series in Operations Research and Financial Engineering, Springer, New York, second ed., 2006.
[15] M. R. Osborne, B. Presnell, and B. A. Turlach, A new approach to variable selection in least squares problems,IMA J. Numer. Anal., 20 (2000), pp. 389–403.
[16] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, Scikit-learn: Machine learning in Python, Journal of Machine Learning Research, 12 (2011), pp. 2825–2830.
[17] E. Polak and G. Ribiere, Note sur la convergence de méthodes de directions conjuguées, Revue française d’informatique et de recherche opérationnelle. Série rouge, 3 (1969), pp. 35–43.
[18] H. Robbins and S. Monro, A stochastic approximation method, Ann. Math. Statistics, 22 (1951), pp. 400–407.
[19] S. Van der Walt, S. C. Colbert, and G. Varoquaux, The numpy array: A structure for efficient numerical computation, 2011.
[20] J. Zhu, S. Rosset, T. Hastie, and R. Tibshirani, $1$-norm support vector machines, in Proceedings of the 16th International Conference on Neural Information Processing Systems, NIPS’03, Cambridge,
