Trust Region Newton Method for Large-Scale Logistic Regression

Chih-Jen Lin[*], Ruby C. Weng[†], and S. Sathiya Keerthi[‡]

Abstract

Large-scale logistic regression arises in many applications such as document classification and natural language processing. In this paper, we apply a trust region Newton method to maximize the log-likelihood of the logistic regression model. The proposed method uses only approximate Newton steps in the beginning, but achieves fast convergence in the end. Experiments show that it is faster than the commonly used quasi Newton approach for logistic regression. We also compare it with existing linear SVM implementations.

1 Introduction

The logistic regression model is useful for two-class classification. Given data $\mathbf{x}$ and weights $(\mathbf{w}, b)$, it assumes the following probability model

$$P(y = \pm 1|\mathbf{x}, \mathbf{w}) = \frac{1}{1 + \exp(-y(\mathbf{w}^T\mathbf{x} + b))},$$

where y is the class label. If training instances are $\mathbf{x}_i, i = 1, \ldots, l$ and labels are $y_i \in \{1, -1\}$, one estimates $(\mathbf{w}, b)$ by minimizing the negative log-likelihood:

$$\min_{\mathbf{w},b} \quad \sum_{i=1}^{l} \log(1 + e^{-y_i(\mathbf{w}^T\mathbf{x}_i + b)}).$$

There are numerous applications of logistic regression. It can be extended to a multi-class classification model, which is a special case of conditional random

[*]Department of Computer Science, National Taiwan University, Taipei 106, Taiwan, cjlin@csie.ntu.edu.tw

[†]Department of Statistics, National Chengchi University, Taipei 116, Taiwan, chweng@nccu.edu.tw

[‡]Yahoo! Research, Santa Clara, CA 95054, USA, selvarak@yahoo-inc.com

fields, and is also called the maximum entropy model in the natural language processing community.

To have a simpler derivation without considering the scalar b, one often augments each instance with an additional dimension:

$$\mathbf{x}_i^T \leftarrow [\mathbf{x}_i^T, 1] \qquad \mathbf{w}^T \leftarrow [\mathbf{w}^T, b].$$

Moreover, to obtain good generalization abilities, one adds a regularization term $\mathbf{w}^T\mathbf{w}/2$, so in this paper we consider the following form of regularized logistic regression:

$$\min_{\mathbf{w}} f(\mathbf{w}) \equiv \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_{i=1}^{l} \log(1 + e^{-y_i\mathbf{w}^T\mathbf{x}_i}), \tag{1}$$

where $C > 0$ is a parameter decided by users so that the two terms in (1) are balanced.

There are many methods for training logistic regression models. In fact, most unconstrained optimization techniques can be considered. Those which have been used in large-scale scenarios are, for example, iterative scaling (Darroch and Ratcliff, 1972; Pietra et al., 1997; Goodman, 2002; Jin et al., 2003), nonlinear conjugate gradient, quasi Newton (in particular, limited memory BFGS) (Liu and Nocedal, 1989; Benson and Moré, 2001), and truncated Newton (Komarek and Moore, 2005). All these optimization methods are iterative procedures, which generate a sequence $\{\mathbf{w}^k\}_{k=1}^{\infty}$ converging to the optimal solution of (1). One can distinguish them according to the following two extreme situations of optimization methods:

Low cost per iteration; slow convergence.	$\longleftrightarrow$	High cost per iteration; fast convergence.

For instance, iterative scaling updates one component of $\mathbf{w}$ at a time, so the cost per iteration is low but the number of iterations is high. In contrast, Newton method, which is expensive at each iteration, has very fast convergence rates. Many have attempted to compare these methods for logistic regression. Minka (2003) experiments with small data sets, and Malouf (2002) has done an extensive comparison for large-scale sets. Currently, most argue that the limited memory BFGS method is the most efficient and effective (e.g., (Malouf, 2002), (Sutton and McCallum, 2006) and references therein). In this article, we aim at situations for which both l (number of instances) and n (number of features) are very large. In addition, the data instances $\mathbf{x}_1, \ldots, \mathbf{x}_l$ are sparse (i.e., many feature values are zero). Many recent applications from document classification and computational linguistics are of this type.

Truncated Newton methods have been an effective approach for large-scale unconstrained optimization, but their use for logistic regression has not been fully exploited. Though Komarek and Moore (2005) have considered this type of methods, their implementation does not follow rigorous optimization derivations, and hence may not be guaranteed to obtain the minimum of the negative log-likelihood. In Section 2, we discuss an efficient and robust truncated Newton method for logistic regression.

The second term in (1) can be considered as a loss function, so regularized logistic regression is related to other learning approaches such as Support Vector Machines (SVM) (Boser et al., 1992). L1-SVM solves the following optimization problem:

$$\min_{\mathbf{w}} f_1(\mathbf{w}) \equiv \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_{i=1}^{l} \max\left(0, 1 - y_i\mathbf{w}^T\mathbf{x}_i\right), \tag{2}$$

while L2-SVM solves

$$\min_{\mathbf{w}} f_2(\mathbf{w}) \equiv \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_{i=1}^{l} \left(\max(0, 1 - y_i\mathbf{w}^T\mathbf{x}_i)\right)^2. \tag{3}$$

Figure 1 shows the different shapes of these three loss functions. SVM is often used with a nonlinear kernel, where data $\mathbf{x}_i$ are mapped to a high dimensional space. However, there are certain applications for which with/without nonlinear mapping gives similar performances. For the case of no nonlinear mapping, we have the possibility of directly solving bigger optimization problems. We refer to such cases as *linear* SVM, and considerable efforts have been made on its fast training (e.g., (Kao et al., 2004; Keerthi and DeCoste, 2005; Joachims, 2006)). L1-SVM involves the optimization of a non-differentiable function of $\mathbf{w}$, so unconstrained optimization techniques cannot be directly applied. For L2-SVM, the training objection function (3) is differentiable but not twice differentiable (Mangasarian, 2002). A modified Newton method has been proposed in (Keerthi and DeCoste, 2005) for large problems. As logistic regression and SVM differ only

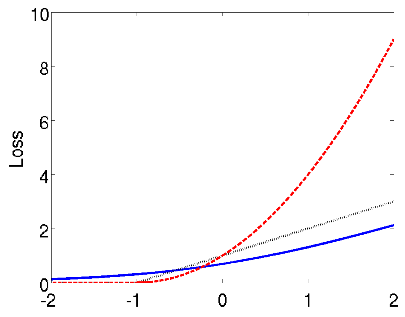

Figure 1: Different loss functions. Dotted (black): $\max(0, 1 - y\mathbf{w}^T\mathbf{x})$, dashed (red): $(\max(0, 1 - y\mathbf{w}^T\mathbf{x}))^2$, solid (blue): $\log(1 + e^{-y\mathbf{w}^T\mathbf{x}})$. The x-axis is $-y\mathbf{w}^T\mathbf{x}$.

in loss functions, very likely they may give similar performances. Some even try to approximate SVM models using logistic regression (Zhang et al., 2003). In Sections 3 and 4, we discuss some existing optimization methods for logistic regression/linear SVM and conduct comparisons. Results show that the proposed Newton method is robust and efficient. Section 5 investigates a variant of our proposed method. Finally, Section 6 gives conclusions.

All sources used in this paper are available at `http://www.csie.ntu.edu.tw/~cjlin/liblinear`. Part of this work appears in a conference paper (Lin et al., 2007).

2 Trust Region Newton Methods

In this section, we briefly discuss Newton and truncated Newton methods. For large-scale logistic regression, we then propose a trust region Newton method, which is a type of truncated Newton approach.

2.1 Newton and Truncated Newton Methods

To discuss Newton methods, we need the gradient and Hessian of $f(\mathbf{w})$:

$$\begin{aligned} \nabla f(\mathbf{w}) &= \mathbf{w} + C\sum_{i=1}^{l}(\sigma(y_i\mathbf{w}^T\mathbf{x}_i) - 1)y_i\mathbf{x}_i, \\ \nabla^2 f(\mathbf{w}) &= \mathcal{I} + CX^TDX, \end{aligned}$$

where $\mathcal{I}$ is the identity matrix,

$$\sigma(y_i\mathbf{w}^T\mathbf{x}_i) = (1 + e^{-y_i\mathbf{w}^T\mathbf{x}_i})^{-1}.$$

D is a diagonal matrix with

$$D_{ii} = \sigma(y_i \mathbf{w}^T \mathbf{x}_i)(1 - \sigma(y_i \mathbf{w}^T \mathbf{x}_i)), \text{ and } X = \begin{bmatrix} \mathbf{x}_1^T \\ \vdots \\ \mathbf{x}_l^T \end{bmatrix} \tag{4}$$

is an $l \times n$ matrix. The Hessian matrix $\nabla^2 f(\mathbf{w})$ is positive definite, so (1) is strictly convex. We can further prove the following theorem.

Theorem 1 (1) *attains a unique global optimal solution.*

The proof is in Appendix A.

Since $\nabla^2 f(\mathbf{w}^k)$ is invertible, the simplest Newton method updates $\mathbf{w}$ by the following way

$$\mathbf{w}^{k+1} = \mathbf{w}^k + \mathbf{s}^k, \tag{5}$$

where k is the iteration index and $\mathbf{s}^k$, the Newton direction, is the solution of the following linear system:

$$\nabla^2 f(\mathbf{w}^k)\mathbf{s}^k = -\nabla f(\mathbf{w}^k). \tag{6}$$

However, there are two issues in using this update rule:

1. The sequence $\{\mathbf{w}^k\}$ may not converge to an optimal solution. In fact, even the function value may not be guaranteed to decrease.

2. While we assume that the data matrix X is sparse, $X^T D X$ is much denser. The Hessian matrix is then too large to be stored. Thus, solving the linear system (6) is an issue that needs careful consideration.

Optimization researchers address the first issue by adjusting the length of the Newton direction. Two techniques are often used: line search and trust region.

For the second issue, there are two major types of methods for solving linear systems: direct methods (e.g., Gaussian elimination), and iterative methods (e.g., Jacobi and conjugate gradient). The main operation of certain iterative methods is the product between the Hessian matrix and a vector $\mathbf{s}$:

$$\begin{aligned} \nabla^2 f(\mathbf{w})\mathbf{s} &= (\mathcal{I} + C X^T D X)\mathbf{s} \\ &= \mathbf{s} + C \cdot X^T(D(X\mathbf{s})). \end{aligned} \tag{7}$$

As we assume sparse X, (7) can be efficiently calculated without storing the Hessian matrix $\nabla^2 f(\mathbf{w}^k)$. Therefore, for large logistic regression, iterative methods are more suitable than direct methods, which require the whole Hessian matrix. Among iterative methods, currently conjugate gradients are the most used ones in Newton methods. The optimization procedure then has two layers of iterations: at each outer iteration an inner conjugate gradient procedure finds the Newton direction. Unfortunately, conjugate gradient methods may suffer from lengthy iterations in certain situations. To save time, one may use only an "approximate" Newton direction in the early stages of the outer iterations. Such a technique is called truncated Newton method (or inexact Newton method).

Komarek and Moore (2005) are among the first to apply truncated Newton methods for logistic regression[*]. They approximately solve (6) by conjugate gradient procedures and use (5) to update $\mathbf{w}^k$. They terminate the conjugate gradient procedure if the relative difference of log likelihoods between two consecutive conjugate gradient iterations is smaller than a threshold. However, they do not provide a convergence proof. In fact, when we tried their code, we found that $\|\nabla f(\mathbf{w}^k)\|$ may not approach zero and hence $\{\mathbf{w}^k\}$ may not converge to an optimum[†].

Optimization researchers have well addressed the above two issues together. They devise the procedure of outer iterations, and specify stopping conditions for the inner iterations. The overall framework guarantees the convergence to the global minimum. The truncation rule of the inner algorithm is important as one should stop after a sufficiently good direction has been found. A survey of truncated Newton methods is in (Nash, 2000). Some comparisons between limited memory quasi Newton and truncated Newton are in (Nocedal and Nash, 1991; Zou et al., 1993).

2.2 A Trust Region Newton Method

We consider the trust region method in (Lin and Moré, 1999), which is a truncated Newton method to deal with general bound-constrained optimization problems

[*]They minimize only the negative log likelihood without the regularization term $\mathbf{w}^T\mathbf{w}/2$.

[†]We used the data set citeseer for this purpose. See Section 4 for a description of this data set.

(i.e., variables are in certain intervals). We simplify the setting to unconstrained situations, so the algorithm is close to earlier work such as (Bouaricha et al., 1997) and (Steihaug, 1983).

At each iteration of a trust region Newton method for minimizing $f(\mathbf{w})$, we have an iterate $\mathbf{w}^k$, a size Δ_k of the trust region, and a quadratic model

$$q_k(\mathbf{s}) = \nabla f(\mathbf{w}^k)^T\mathbf{s} + \frac{1}{2}\mathbf{s}^T\nabla^2 f(\mathbf{w}^k)\mathbf{s}$$

as the approximation of the value $f(\mathbf{w}^k + \mathbf{s}) - f(\mathbf{w}^k)$. Next, we find a step $\mathbf{s}^k$ to approximately minimize $q_k(\mathbf{s})$ subject to the constraint $\|\mathbf{s}\| \le \Delta_k$. We then update $\mathbf{w}^k$ and Δ_k by checking the ratio

$$\rho_k = \frac{f(\mathbf{w}^k + \mathbf{s}^k) - f(\mathbf{w}^k)}{q_k(\mathbf{s}^k)} \tag{8}$$

of the actual reduction in the function to the predicted reduction in the quadratic model. The direction is accepted if ρ_k is large enough:

$$\mathbf{w}^{k+1} = \begin{cases} \mathbf{w}^k + \mathbf{s}^k & \text{if } \rho_k > \eta_0, \\ \mathbf{w}^k & \text{if } \rho_k \le \eta_0, \end{cases} \tag{9}$$

where $\eta_0 > 0$ is a pre-specified value.

From (Lin and Moré, 1999), updating rules for Δ_k depend on positive constants η_1 and η_2 such that $\eta_1 < \eta_2 < 1$, while the rate at which Δ_k is updated relies on positive constants σ_1, σ_2, and σ_3 such that $\sigma_1 < \sigma_2 < 1 < \sigma_3$. The trust region bound Δ_k is updated by the rules

$$\begin{array}{lll} \Delta_{k+1} \in [\sigma_1 \min\{\|\mathbf{s}^k\|, \Delta_k\}, \sigma_2\Delta_k] & \text{if} & \rho_k \le \eta_1, \\ \Delta_{k+1} \in [\sigma_1\Delta_k, \sigma_3\Delta_k] & \text{if} & \rho_k \in (\eta_1, \eta_2), \\ \Delta_{k+1} \in [\Delta_k, \sigma_3\Delta_k] & \text{if} & \rho_k \ge \eta_2. \end{array} \tag{10}$$

Similar rules are used in most modern trust region methods.

A description of our trust region algorithm is given in the following table.

Algorithm 1 A trust region algorithm for logistic regression

```
1. Given $\mathbf{w}^0$.
2. For $k = 0, 1, \ldots$ (outer iterations)
    • If $\nabla f(\mathbf{w}^k) = \mathbf{0}$, stop.
    • Find an approximate solution $\mathbf{s}^k$ of the trust region sub-problem
          $\min_{\mathbf{s}} \; q_k(\mathbf{s}) \quad$ subject to $\|\mathbf{s}\| \leq \Delta_k$.      (11)
    • Compute $\rho_k$ via (8).
    • Update $\mathbf{w}^k$ to $\mathbf{w}^{k+1}$ according to (9).
    • Obtain $\Delta_{k+1}$ according to (10).
```

The conjugate gradient method to approximately solve the trust region sub-problem (11) is given in the following table.

Algorithm 2 Conjugate gradient procedure for approximately solving the trust region sub-problem (11)

```
1. Given $\xi_k < 1, \Delta_k > 0$. Let $\bar{\mathbf{s}}^0 = \mathbf{0}, \mathbf{r}^0 = -\nabla f(\mathbf{w}^k)$, and $\mathbf{d}^0 = \mathbf{r}^0$.
2. For $i = 0, 1, \ldots$ (inner iterations)
    • If
          $\|\mathbf{r}^i\| \leq \xi_k \|\nabla f(\mathbf{w}^k)\|$,      (12)
      then output $\mathbf{s}^k = \bar{\mathbf{s}}^i$ and stop.
    • $\alpha_i = \|\mathbf{r}^i\|^2 / ((\mathbf{d}^i)^T \nabla^2 f(\mathbf{w}^k) \mathbf{d}^i)$.
    • $\bar{\mathbf{s}}^{i+1} = \bar{\mathbf{s}}^i + \alpha_i \mathbf{d}^i$.
    • If $\|\bar{\mathbf{s}}^{i+1}\| \geq \Delta_k$, compute $\tau$ such that
          $\|\bar{\mathbf{s}}^i + \tau \mathbf{d}^i\| = \Delta_k$,      (13)
      then output $\mathbf{s}^k = \bar{\mathbf{s}}^i + \tau \mathbf{d}^i$ and stop.
    • $\mathbf{r}^{i+1} = \mathbf{r}^i - \alpha_i \nabla^2 f(\mathbf{w}^k) \mathbf{d}^i$.
    • $\beta_i = \|\mathbf{r}^{i+1}\|^2 / \|\mathbf{r}^i\|^2$.
    • $\mathbf{d}^{i+1} = \mathbf{r}^{i+1} + \beta_i \mathbf{d}^i$.
```

The main operation is the Hessian-vector product $\nabla^2 f(\mathbf{w}^k)\mathbf{d}^i$, which is implemented using the idea in Eq. (7). Note that only one Hessian-vector product is needed at each conjugate gradient iteration. Since

$$\mathbf{r}^i = -\nabla f(\mathbf{w}^k) - \nabla^2 f(\mathbf{w}^k)\bar{\mathbf{s}}^i,$$

the stopping condition (12) is the same as

$$\| - \nabla f(\mathbf{w}^k) - \nabla^2 f(\mathbf{w}^k)\bar{\mathbf{s}}^i\| \leq \xi_k \|\nabla f(\mathbf{w}^k)\|,$$

which implies that $\bar{\mathbf{s}}^i$ is an approximate solution of the linear system (6). However, Algorithm 2 is different from standard conjugate gradient methods for linear systems as the constraint $\|\mathbf{s}\| \leq \Delta$ must be taken care of. It is known that (Steihaug, 1983, Theorem 2.1) with $\bar{\mathbf{s}}^0 = \mathbf{0}$, we have

$$\|\bar{\mathbf{s}}^i\| < \|\bar{\mathbf{s}}^{i+1}\|, \forall i,$$

so in a finite number of conjugate gradient iterations, either (12) is satisfied or $\bar{\mathbf{s}}^{i+1}$ violates the trust region constraint. In the latter situation, (13) finds a point on the trust region boundary as

$$q_k(\bar{\mathbf{s}}^i + \tau \mathbf{d}^i) < q_k(\bar{\mathbf{s}}^i).$$

The whole procedure is a careful design to make sure that the approximate Newton direction is good enough and the trust region method converges.

Next, we discuss convergence properties of the trust region Newton method. Most results can be traced back to (Steihaug, 1983). However, here we follow Lin and Moré (1999) as our algorithmic settings are closer to it. For the sequence $\{\mathbf{w}^k\}$ to have at least one limit point[‡], since $f(\mathbf{w}^k)$ is decreasing, it suffices to show that the level set $\{\mathbf{w} \mid f(\mathbf{w}) \leq f(\mathbf{w}^0)\}$ is closed and bounded. This result has been explained in the proof of Theorem 1. To have this limit point to be the minimum, Theorem 2.1 of (Lin and Moré, 1999) requires that $\nabla^2 f(\mathbf{w}^k)$ is uniformly bounded. We have this property as $\nabla^2 f(\mathbf{w})$ is continuous in this bounded level set.

Eq. (12) is a relative stopping condition in solving a linear system. The parameter ξ_k effectively controls the efforts associated with the inner iterations.

[‡]That is, the sequence $\{\mathbf{w}^k\}$ has at least one convergent sub-sequence.

From Theorem 5.4 of (Lin and Moré, 1999)[§], if $\xi_k < 1$, then the trust region method Q-linearly converges to the global minimum of (1). That is,

$$\lim_{k \to \infty} \frac{\|\mathbf{w}^{k+1} - \mathbf{w}^*\|}{\|\mathbf{w}^k - \mathbf{w}^*\|} < 1, \tag{14}$$

where $\mathbf{w}^*$ is the unique optimal solution of (1). If

$$\xi_k \to 0 \text{ as } k \to \infty,$$

then the limit in (14) becomes zero, so we have Q-superlinear convergence.

Regarding the computational complexity, the cost per iteration is

$$\begin{aligned} & O(\text{nnz}) \text{ for 1 function and 0/1 gradient evaluations} \\ + \quad & O(\text{nnz}) \times \text{ number of conjugate gradient iterations,} \end{aligned} \tag{15}$$

where nnz is the number of nonzero elements in the sparse matrix X. Note that if $\mathbf{w}^k$ is not updated in (9), then the gradient is the same for the next iteration.

3 Related Methods and Implementation Issues

In this section, we discuss two related techniques, which will be compared in Section 4. The first one is from (Keerthi and DeCoste, 2005), which is a modified Newton method for L2-SVM. It is now one of the fastest implementations in training linear SVMs. The second is (Liu and Nocedal, 1989), which is a general limited memory quasi Newton implementation. Many consider it as very efficient for training logistic regression. We also discuss implementation issues of the proposed trust region Newton method.

3.1 Modified Newton Method for L2-SVM

The key idea of (Keerthi and DeCoste, 2005) to solve L2-SVM is that for any given index set $I \subset \{1, \ldots, l\}$, if the optimal solution $\mathbf{w}^*$ of the following problem

$$\min_{\mathbf{w}} \quad \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_{i \in I}(1 - y_i \mathbf{w}^T \mathbf{x}_i)^2 \tag{16}$$

[§]More precisely, see the explanation given in that paper after the proof of Theorem 5.4.

satisfies

$$1 - y_i(\mathbf{w}^*)^T\mathbf{x}_i \begin{cases} > 0 & \text{if } i \in I, \\ \leq 0 & \text{if } i \notin I, \end{cases}$$

then $\mathbf{w}^*$ is an optimal solution of the L2-SVM problem (3). Once I is fixed, (16) is a simple regularized least square problem and can be solved by the following linear system:

$$(\mathcal{I} + 2CX_{I,:}^T X_{I,:})\mathbf{w} = 2CX_{I,:}^T \mathbf{y}_I, \tag{17}$$

where $X_{I,:}$ includes X's rows corresponding to the set I. Their algorithm is described in the following table.

Algorithm 3 Modified Newton Method for L2-SVM

```
1. Given $\mathbf{w}^0$.
2. For $k = 0, 1, \ldots$
    • If $\nabla f(\mathbf{w}^k) = \mathbf{0}$, stop.
    • Set up (17) using
          $I_k = \{i \mid 1 - y_i(\mathbf{w}^k)^T\mathbf{x}_i > 0\}.$
      Solve (17) by the conjugate gradient procedure and obtain $\bar{\mathbf{w}}^k$.
    • Let $\mathbf{s}^k = \bar{\mathbf{w}}^k - \mathbf{w}^k$.
      Find
          $\alpha_k = \arg\min_{\alpha \geq 0} f(\mathbf{w}^k + \alpha\mathbf{s}^k),$
      and set $\mathbf{w}^{k+1} = \mathbf{w}^k + \alpha_k\mathbf{s}^k$.
```

Keerthi and DeCoste (2005) prove that Algorithm 3 converges to the optimal solution of (3) in a finite number of iterations. Compared to our approach, this method has the following advantages and disadvantages.

Advantages:

- The final set I corresponds to the set of support vectors. Thus, the sets I_k may be small at final iterations, and solving (17) by conjugate gradient methods is fast.

Disadvantages:

- Their convergent results assume that at each iteration, (17) is exactly solved. However, they use a relative stopping condition in practical implementations, so the convergence remains an issue. In contrast, the convergence of trust region Newton method has been well studied under various early stopping conditions of the conjugate gradient procedure.

- Using $X_{I,:}$ in (17) requires an easy access of X's rows. This restricts the way how X is stored. We elaborate this point in Section 4.3.

More discussions on the stopping condition of Algorithm 3 are given in Section 4.2. To avoid exactly solving every instance of equation (17), one may also apply the trust region framework to L2-SVM. However, as L2-SVM is not twice differentiable, the convergence as well as the numerical behavior need to be carefully investigated. As this paper focuses on logistic regression, we leave this direction for future studies.

3.2 Limited Memory quasi Newton Method

We briefly introduce the approach in (Liu and Nocedal, 1989). Quasi Newton methods use certain techniques to obtain an approximate inverse Hessian H_k and can easily update it to H_{k+1}. One of the most popular updates is BFGS. The approach by (Liu and Nocedal, 1989) is almost the same as BFGS, but restricts the update to use only m vectors from the previous iterations. The matrix H_k is not formed explicitly and there is an efficient way to compute $H_k \nabla f(\mathbf{w}^k)$. This property is useful for large logistic regression as we cannot afford to store H_k. The algorithm is sketched in the following table.

Algorithm 4 Limited memory BFGS

```
1. Given $\mathbf{w}^0, H^0$ and a small integer $m$.
2. For $k = 0, 1, \ldots$
    • If $\nabla f(\mathbf{w}^k) = \mathbf{0}$, stop.
    • Using $m$ vectors from previous iterations to calculate $H_k \nabla f(\mathbf{w}^k)$, where
      $H_k$ is an approximate inverse Hessian.
    • Search $\alpha_k$ so that
          $f(\mathbf{w}^k - \alpha H_k \nabla f(\mathbf{w}^k))$
      satisfies certain sufficient decrease conditions.
    • Update $H_k$ to $H_{k+1}$.
```

Regarding convergence, problem (1) satisfies Assumption 6.1 of (Liu and Nocedal, 1989), so the algorithm is R-linearly convergent. That is, there is a constant $c < 1$ such that

$$f(\mathbf{w}^k) - f(\mathbf{w}^*) \leq c^k (f(\mathbf{w}^0) - f(\mathbf{w}^*)), \tag{18}$$

where $\mathbf{w}^*$ is the unique optimal solution of (1). Note that (14) implies (18), so Algorithm 1 has a stronger convergence property than Algorithm 4. While truncated Newton methods find an approximate direction, they still use the exact Hessian matrix. In contrast, limited memory quasi Newton methods consider only approximate Hessian matrices, so we can expect that it has slower convergence.

The cost per iteration is

$$\begin{aligned} & O(\text{nnz}) \text{ for function/gradient evaluations in line search} \\ + \quad & O(nm) \text{ for } H_k \nabla f(\mathbf{w}^k) \text{ and updating } H_k \text{ to } H_{k+1}. \end{aligned} \tag{19}$$

As generally $nm <$ nnz, function/gradient evaluations take most computational time. Moreover, compared to (15), the cost per iteration is less than that for our trust region method. However, as we will show in experiments, LBFGS' total training time is longer due to its lengthy iterations.

In this paper, we use $m = 5$, which is the default choice in the LBFGS software (Liu and Nocedal, 1989).

Problem	l	# Positive	# Negative	n	# nonzeros
a9a	32,561	7,841	24,720	123	451,592
real-sim	72,309	22,238	50,071	20,958	3,709,083
news20	19,996	9,999	9,997	1,355,191	9,097,916
citeseer	181,395	299	181,096	105,354	512,267
yahoo-japan	176,203	15,937	160,266	832,026	23,506,415
rcv1	677,399	355,460	321,939	47,236	49,556,258
yahoo-korea	460,554	145,831	314,723	3,052,939	156,436,656

Table 1: Data statistics: l is the number of instances and n is the number of features.

3.3 Implementation Issues of Trust Region Newton Method

We give details of parameters in the proposed Algorithms 1 and 2. All settings are almost the same as the TRON software (Lin and Moré, 1999).

We set the initial $\Delta_0 = \|\nabla f(\mathbf{w}^0)\|$ and take $\eta_0 = 10^{-4}$ in (9) to update $\mathbf{w}^k$. For changing Δ_k to Δ_{k+1}, we use

$$\eta_1 = 0.25, \eta_2 = 0.75,$$

$$\sigma_1 = 0.25, \sigma_2 = 0.5, \sigma_3 = 4.0.$$

As (10) specifies only the interval in which Δ_{k+1} should lie, there are many possible choices of the update rules. We use the same rules as given in (Lin and Moré, 1999). In the conjugate gradient procedure, we use $\xi_k = 0.1$ in the stopping condition (12). One may wonder how the above numbers are chosen. These choices are considered appropriate following the research on trust region methods in the past several decades. It is unclear yet if they are the best for logistic regression problems, but certainly we would like to try custom settings first.

4 Experiments

In this section, we compare our approach with other existing implementations. After describing data sets for experiments, we conduct detailed comparisons and discuss results.

4.1 Data Sets

We consider seven data sets from various sources. Table 1 lists the numbers of instances (# positive, # negative), features, and nonzero feature values. Details of data sets are described below.

a9a: This set is compiled in (Platt, 1998) from the UCI "adult" data set (Newman et al., 1998). It is available at `http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary/a9a`.

real-sim: This set is from the web site `http://people.cs.uchicago.edu/~vikass/datasets/lskm/svml/`. It, originally compiled by Andrew McCallum, includes Usenet articles from four discussion groups, for simulated auto racing, simulated aviation, real autos, and real aviation.

news20: This is a collection of news documents. We use the data processed in (Keerthi and DeCoste, 2005). They consider binary term frequencies and normalize each instance to unit length. This set is available at `http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary/news20.binary.bz2`.

citeseer: This set, obtained from `http://komarix.org/ac/ds/#spardat`, is a collection of research papers. Positive ones include those authored by "J. Lee." Note that the original data has labels 0 and 1, but for our logistic regression formulation, the label 0 must be changed to -1.

yahoo-japan: This set, obtained from Yahoo!, includes documents in hundreds of classes. We consider the class with the largest number of instances as positive and all remaining instances as negative. We use binary term frequencies and normalize each instance to unit length.

rcv1: This set (Lewis et al., 2004) is an archive of manually categorized newswire stories from Reuters Ltd. Each vector is a cosine normalization of a log transformed TF-IDF (term frequency, inverse document frequency) feature vector. The news documents are in a hierarchical structure of classes. We split the data to positive/negative by using the two branches in the first layer of the hierarchy. Data which are multi-labeled (i.e., in both branches) are not considered. The set used here can be found at `http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary/rcv1_test.binary.bz2` yahoo-korea: This set, from Yahoo!, includes documents in a hierarchy of classes. We consider the largest branch from the root node (i.e., the branch including the largest number of classes) as positive, and all others as negative.

Clearly, except a9a, all other sets are from document classification. Past experiences have shown that for such data, linear classifiers are often as good as kernalized ones. We find that normalizations are usually needed so that the length of each instance is not too large. Otherwise, when the number of features is large, $\mathbf{w}^T\mathbf{x}_i$ may be huge and cause difficulties in solving optimization problems (For good performance also, such normalizations are usually needed).

Except citeseer, all other problems are quite balanced. It is known that unbalanced sets usually lead to shorter training time. Therefore, problems used in this article are more challenging in terms of training time.

4.2 Comparisons

We compare four methods. Two solve logistic regression, one solves L2-SVM, and one solves L1-SVM.

- TRON: the trust region Newton method discussed in Section 2.2.
- LBFGS: the limited memory quasi Newton implementation (Liu and Nocedal, 1989). See the discussion in Section 3.2. The source code is available online at `http://www.ece.northwestern.edu/~nocedal/lbfgs.html`.
- SVMlin: the modified Newton method for L2-SVM (Keerthi and DeCoste, 2005). See the discussion in Section 3.1. The source code (implemented by Vikas Sindhwani) is available online at `http://people.cs.uchicago.edu/~vikass/svmlin.html`.
- SVM^{perf} (Joachims, 2006): it is an approximation algorithm to train L1-SVM. A threshold controls how well the approximation is. The code is online available at `http://svmlight.joachims.org/svm_perf.html`. Since this approach is very different from the other three, we conduct a separate comparison between it and TRON at the end of this sub-section.

	Logistic regression			L2-SVM	
	TRON		LBFGS	SVMlin	
C	CV	Time	Time	CV	Time
0.25	84.69%	4	14	84.79%	15
1	84.71%	6	28	84.78%	23
4	84.72%	11	54	84.78%	30
16	84.71%	18	107	84.78%	52

(a) a9a

	Logistic regression			L2-SVM	
	TRON		LBFGS	SVMlin	
C	CV	Time	Time	CV	Time
0.25	95.85%	10	32	97.43%	20
1	96.97%	16	63	97.53%	29
4	97.41%	23	96	97.24%	47
16	97.51%	35	175	96.86%	92

(b) real-sim

	Logistic regression			L2-SVM	
	TRON		LBFGS	SVMlin	
C	CV	Time	Time	CV	Time
0.25	89.74%	62	187	95.78%	79
1	93.36%	96	434	96.72%	147
4	95.49%	144	806	96.90%	270
16	96.30%	198	1424	96.86%	495

(c) news20

	Logistic regression			L2-SVM	
	TRON		LBFGS	SVMlin	
C	CV	Time	Time	CV	Time
0.25	99.84%	9	15	99.85%	19
1	99.84%	14	28	99.85%	21
4	99.84%	20	50	99.80%	100
16	99.85%	32	83	99.72%	492

(d) citeseer

	Logistic regression			L2-SVM	
	TRON		LBFGS	SVMlin	
C	CV	Time	Time	CV	Time
0.25	91.91%	60	176	92.87%	144
1	92.50%	96	337	93.00%	290
4	92.81%	163	702	92.71%	651
16	92.86%	283	1184	92.14%	1396

(e) yahoo-japan

	Logistic regression			L2-SVM	
	TRON		LBFGS	SVMlin	
C	CV	Time	Time	CV	Time
0.25	97.18%	78	241	97.76%	131
1	97.56%	122	988	97.74%	275
4	97.72%	181	946	97.53%	595
16	97.69%	261	1422	97.07%	1370

(f) rcv1

	Logistic regression			L2-SVM	
	TRON		LBFGS	SVMlin	
C	CV	Time	Time	CV	Time
0.25	81.34%	528	2512	85.43%	1734
1	84.03%	912	4368	86.48%	3813
4	85.75%	1771	8937	86.38%	8896
16	86.40%	4475	14074	85.66%	19751

(g) yahoo-korea

Table 2: The comparison of three methods. Here time (in seconds) is the total training time in the CV procedure. As TRON and LBFGS minimize the same formulation and their CV accuracy values are almost the same, we present only the result of TRON. The number of CV folds is five for small problems, and is two for larger ones (yahoo-japan, rcv1, yahoo-korea). Note for both models, the CV values do not increase using $C > 16$.

We do not consider the code by (Komarek and Moore, 2005) because of two reasons. First, we have mentioned its convergence problems in Section 2.1. Second, for sparse data, it handles only problems with 0/1 feature values, but most our data have real-numbered features.

All the above methods are implemented in high-level languages such as C/C++ or FORTRAN. For easier experiments, we use Matlab interfaces of the first three codes. Experiments are conducted on an AMD64 Opteron (2.4GHz) computer with 32 GB RAM. All sources used for this comparison can be found at `http://www.csie.ntu.edu.tw/~cjlin/liblinear`.

Regarding the stopping conditions, for TRON and LBFGS, we use

$$\|\nabla f(\mathbf{w}^k)\|_\infty \leq 10^{-3}. \tag{20}$$

Though SVMlin minimizes a different function $f_2(\mathbf{w})$, a similar condition should be $\|\nabla f_2(\mathbf{w}^k)\|_\infty \leq 10^{-3}$. Note that

$$\nabla f_2(\mathbf{w}) = (\mathcal{I} + 2CX_{I,:}^T X_{I,:})\mathbf{w} - 2CX_{I,:}^T \mathbf{y}_I,$$

where $I = \{i \mid 1 - y_i \mathbf{w}^T \mathbf{x}_i > 0\}$ is an index set depending on $\mathbf{w}$. Recall in Algorithm 3 that we sequentially obtain the following items:

$$\mathbf{w}^k \rightarrow I_k \rightarrow \bar{\mathbf{w}}^k.$$

In practice, we use

$$\|(\mathcal{I} + 2CX_{I_k,:}^T X_{I_k,:})\bar{\mathbf{w}}^k - 2CX_{I_k,:}^T \mathbf{y}_{I_k}\|_\infty \leq 10^{-3}, \tag{21}$$

and

$$1 - y_i(\bar{\mathbf{w}}^k)^T \mathbf{x}_i \begin{cases} \geq -10^{-3} & \text{if } i \in I_k, \\ \leq 10^{-3} & \text{if } i \notin I_k, \end{cases} \tag{22}$$

as the stopping condition. We also use (21) as the stopping rule when solving (17) by conjugate gradients[¶].

Regarding the initial $\mathbf{w}^0$, all methods use $\mathbf{0}$.

It is important to check the prediction ability of logistic regression and L2-SVM, so we first report cross-validation (CV) accuracy. For the larger sets (yahoo-japan, rcv1, and yahoo-korea), we use two-fold CV. For others, five-fold CV is

[¶]This default setting of SVMlin is similar, but a relative stopping condition is used for (17).

conducted. We do not consider other measurements such as AUC (Area Under Curve) or F-measure as all problems except citeseer are rather balanced, and CV accuracy is suitable. Moreover, different values of the regularization parameter C may affect the performance as well as training time. So we try four different C values: 0.25, 1, 4, and 16. Table 2 presents the result of comparisons. We show CV accuracy and the total training time in the CV procedure.

Regarding CV accuracy, when C is small, logistic regression is slightly worse than L2-SVM. However, as C increases, the two give similar CV values. Overall, the two learning models give comparable generalization performance. On training time, TRON is better than LBFGS, so truncated Newton methods are effective for training logistic regression. One may wonder if any implementation-specific details cause unfair timing comparisons. Indeed we have made the experimental environments as close as possible. For all the three methods compared here, we store the sparse matrix X by the same compressed row format. Section 4.3 discusses that different sparse formats may lead to dissimilar computational time. For LBFGS, one main cost is on function/gradient evaluations, which are provided by users. We implement the same code in TRON and LBFGS for function/gradient evaluations. Thus, our timing comparison is quite fair.

If we take TRON and SVMlin, though it may not be right to compare the training speed using the same C, TRON has the advantage of fast convergence. Note that as C increases, the stopping condition (20) is more strict. But TRON's training time does not increase as much as that of SVMlin. If we compare the training time of the model with the highest CV accuracy, TRON may not be faster than SVMlin. However, as one usually tries several settings such as various C or stopping tolerances, SVMlin more easily gets into a situation of long training time than TRON. Both approaches spend most of their time on the operation (7) in the conjugate gradient procedure. SVMlin is slower because it accurately solves Newton directions in early iterations. Hence, many conjugate gradient iterations are wasted.

In the above comparisons, we do not include SVM^{perf} because it significantly differs from the other three codes, which follow more traditional optimization derivations. SVM^{perf} uses a threshold to control how well the obtained solution

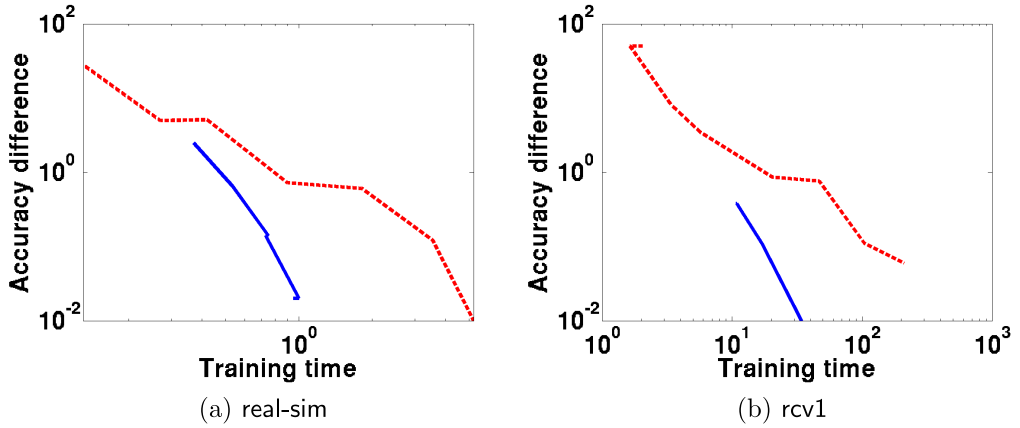

Figure 2: A comparison between TRON (blue solid line) and SVM^{perf} (red dotted line). The y-axis shows the accuracy differences between current models and that of using the stopping tolerance 0.1. The x-axis (training time) is in seconds. We use $C = 1$. One must be careful that SVM^{perf}'s C parameter is different. We take our C value and set SVM^{perf}'s C parameter as $Cl/100$.

approximates the optimal one. This threshold is very different from the tolerance used in the stopping condition (20). A suitable comparison is to check the relationship between training time and testing accuracy. In other words, we can check how closely the algorithm can achieve an accuracy similar to that of using the minimum of the optimization problem.

We conduct the following experiment. A data set is equally split into training and testing sets. We then run SVM^{perf} with its default threshold 0.1, and TRON with the stopping condition $\|\nabla f(\mathbf{w}^k)\|_\infty \leq 0.1$ to obtain their respective models and testing accuracy. Since accuracy stabilizes for thresholds/tolerances smaller than 0.1, the accuracies obtained from these solutions can be taken to be very close to those of the true minima. So we check against these accuracy values as training time increases from zero. Figure 2 shows the results for real-sim and rcv1. We do not present results for other data sets as for unknown reasons SVM^{perf} needs lengthy time for predicting data with a large number of features (e.g., news20 and yahoo-japan).

From Figure 2, we see that SVM^{perf} quickly obtains models with some accuracy values, but the convergence is slow. For TRON, it needs at least a certain amount of time to have the first model, but the accuracy of this model is already close to that of the optimal one. The final convergence is also faster. The experiment indicates that unless the problem is extremely huge so that limits on computational time force one to use very early sub-optimal stopping[‖], TRON is a more effective method than SVM^{perf}.

4.3 Row and Column Formats in Storing X

A sparse matrix can be represented by many ways. Two commonly used ones are "compressed column" and "compressed row" formats (Duff et al., 1989). For example, if

$$X = \begin{bmatrix} -10 & 0 & -20 & 0 \\ 30 & 0 & 0 & 10 \end{bmatrix},$$

then its compressed column format is by three arrays:

$$\texttt{X\_val} = [-10, 30, -20, 10], \quad \texttt{X\_rowind} = [1, 2, 1, 2], \quad \texttt{X\_colptr} = [1, 3, 3, 4, 5],$$

where `rowind` means row indices and `colptr` means column pointers[**]. Alternatively, compress row format has

$$\texttt{X\_val} = [-10, -20, 30, 10], \quad \texttt{X\_colind} = [1, 3, 1, 4], \quad \texttt{X\_rowptr} = [1, 3, 5].$$

There are two important properties: First, X's column (row) format is X^T's row (column) format. Second, using the column (row) format for X leads to easy accesses of all values of one column (row). For data classification, the column (row) format thus lets us easily access any particular feature (any particular instance).

The main conjugate gradient operation (7) involves two matrix-vector products – one is with X^T, and the other is with X. In using the column format, there are ways so that for both operations, sequentially X's columns are accessed. Similarly, if using the row format, we only need to access X's rows. Thus, one may think that using the two (row and column) sparse formats does not cause many differences. Table 3 presents a comparison. Surprisingly, for some problems the difference is huge. One possible reason is the different number of nonzero entries per column and per row in X. During the matrix-vector product, as a column (or a row) is

[‖]For example, the accuracy is still 10% away from that of the final model.

[**]This way of using three arrays is common in FORTRAN programs, which did not support pointers. One can implement this format using pointers, where each pointer associates with values and indices of a row.

Problem	Row	Column
a9a	18	19
real-sim	35	42
news20	198	111
citeseer	32	40
yahoo-japan	283	267
rcv1	261	734
yahoo-korea	4475	7391

Table 3: Total training time (in seconds) in the CV procedure by storing X in compress row and column formats. We use $C = 16$ and $\epsilon = 0.001$.

used at a time, its elements should be put in the higher level of the computer memory hierarchy. If the number of nonzeros in a column is significantly larger than those in a row, very likely a column cannot be fit into the same memory level as that for a row. We think that this situation occurs for rcv1, for which the number of instances is significantly larger than the number of features.

Of course the practical behavior depends on the computer architectures as well as how nonzero elements are distributed across rows and columns. We do not intend to fully address this issue here, but would like to point out the importance of implementation details in comparing learning algorithms. In Table 2, all three methods are implemented using the row format. Without being careful on such details, very easily we may get misleading conclusions.

4.4 Stopping Tolerance

The stopping tolerance used in Section 4.2 is 10^{-3}, but one may wonder whether it is too strict. We thus compare the three methods (TRON, LBFGS, and SVMlin) using two larger stopping tolerances: 0.1 and 10. Instead of listing all results in tables, here we summarize some observations.

1. Stopping tolerance 0.1:

   All CV accuracy values are almost the same, so for these data sets, a tolerance value of 0.1 is usually small enough. The training time of all three approaches is less. Overall, TRON is still the fastest among the three methods.

2. Stopping tolerance 10:

   For some situations, SVMlin gives an erroneous CV accuracy. The reason is apparently that the index set I is wrongly identified under the loose stopping condition (21)-(22). Thus, it is possible that in (22), one should use a smaller tolerance than in (21). In contrast, TRON and LBFGS yield CV values close to those by using tolerances 0.1 or 0.001, since their gradient function is calculated in a more straightforward way. While TRON is still the fastest, the difference between it and LBFGS is smaller. We expect this result as for such a large tolerance, LBFGS does not suffer from slow convergence.

5 Preconditioned Conjugate Gradient Methods

To reduce the number of conjugate gradient iterations, in the truncated Newton method one often uses preconditioned conjugate gradient procedures. Instead of solving the Newton linear system (6), we consider a preconditioner which approximately factorizes the Hessian matrix

$$\nabla^2 f(\mathbf{w}^k) \approx PP^T \tag{23}$$

and then solve a new linear system

$$(P^{-1}\nabla^2 f(\mathbf{w}^k)P^{-T})\hat{\mathbf{s}} = -P^{-1}\nabla f(\mathbf{w}^k),$$

where $\hat{\mathbf{s}} = P^T\mathbf{s}$. If the approximate factorization (23) is good enough, the matrix $P^{-1}\nabla^2 f(\mathbf{w}^k)P^{-T}$ is close to the identity and less conjugate gradient iterations are needed. However, as we need extra efforts to find P and the cost per conjugate iteration is higher, a smaller number of conjugate gradient iterations may not lead to shorter training time. Thus, finding suitable preconditioners is usually difficult. Popular preconditioners include, for example, diagonal matrix of the Hessian and incomplete Cholesky factorization.

Our situation differs from other unconstrained optimization applications in two aspects. First, lengthy conjugate gradient iterations often occur at final outer steps, but for machine learning applications the algorithm may stop before reaching such a stage (see the discussion in Section 4.4). Thus we may not benefit from using preconditioners. Second, preconditioners are more easily obtained by assuming that the whole Hessian matrix $\nabla^2 f(\mathbf{w}^k)$ is available. As we never multiply $X^T DX$ out, $\nabla^2 f(\mathbf{w}^k)$ is not stored and the selection of preconditioners may be more restricted. In this section, we conduct experiments by using the simple diagonal preconditioner

$$P = P^T = \sqrt{\text{Diag}(\nabla^2 f(\mathbf{w}^k))}.$$

Since

$$\nabla^2 f(\mathbf{w}^k)_{ii} = 1 + C \sum_{j=1}^{l} X_{ji}^2 D_{jj},$$

one goes through all X's nonzero elements once for finding diagonal elements. The cost of obtaining the preconditioner is thus no more than that of one conjugate gradient iteration.

The trust region sub-problem needs to be adjusted. Here we follow the derivation in (Lin and Moré, 1999) by considering a scaled version

$$\min_{\mathbf{s}} \; q_k(\mathbf{s}) \quad \text{subject to } \|P^T\mathbf{s}\| \leq \Delta_k. \tag{24}$$

With $\hat{\mathbf{s}} = P^T\mathbf{s}$, we transform (24) to

$$\min_{\hat{\mathbf{s}}} \; \hat{q}_k(\hat{\mathbf{s}}) \quad \text{subject to } \|\hat{\mathbf{s}}\| \leq \Delta_k, \tag{25}$$

where

$$\hat{q}_k(\hat{\mathbf{s}}) = \hat{\mathbf{g}}^T\hat{\mathbf{s}} + \frac{1}{2}\hat{\mathbf{s}}^T\hat{H}\hat{\mathbf{s}},$$

and

$$\hat{\mathbf{g}} = P^{-1}\nabla f(\mathbf{w}^k), \qquad \hat{H} = P^{-1}\nabla^2 f(\mathbf{w}^k)P^{-T}.$$

Eq. (25) is in the same form as (11), the sub-problem without using preconditioners, so the procedure to approximately solve (25) is almost the same as Algorithm 2. We give details in the following table.

Algorithm 5 Preconditioned conjugate gradient procedure for approximately solving the trust region sub-problem (25)

```
1. Given $\xi_k < 1, \Delta_k > 0$. Let $\hat{\mathbf{s}}^0 = \mathbf{0}, \mathbf{r}^0 = -\hat{\mathbf{g}}$, and $\mathbf{d}^0 = \mathbf{r}^0$.
2. For $i = 0, 1, \dots$ (inner iterations)
    • If
          $\|\mathbf{r}^i\| \le \xi_k \|\hat{\mathbf{g}}\|,$
      then output $\mathbf{s}^k = P^{-T}\hat{\mathbf{s}}^i$ and stop.
    • $\alpha_i = \|\mathbf{r}^i\|^2/((\mathbf{d}^i)^T \hat{H}\mathbf{d}^i)$.
    • $\hat{\mathbf{s}}^{i+1} = \hat{\mathbf{s}}^i + \alpha_i \mathbf{d}^i$.
    • If $\|\hat{\mathbf{s}}^{i+1}\| \ge \Delta_k$, compute $\tau$ such that
          $\|\hat{\mathbf{s}}^i + \tau\mathbf{d}^i\| = \Delta_k,$
      then output $\mathbf{s}^k = P^{-T}(\hat{\mathbf{s}}^i + \tau\mathbf{d}^i)$ and stop.
    • $\mathbf{r}^{i+1} = \mathbf{r}^i - \alpha_i \hat{H}\mathbf{d}^i$.
    • $\beta_i = \|\mathbf{r}^{i+1}\|^2/\|\mathbf{r}^i\|^2$.
    • $\mathbf{d}^{i+1} = \mathbf{r}^{i+1} + \beta_i \mathbf{d}^i$.
```

Note that in practical implementations we calculate $\hat{H}\mathbf{d}^i$ by a way similar to (7)

$$P^{-1}(P^{-T}\mathbf{d}^i + C(X^T(D(X(P^{-T}\mathbf{d}^i))))).$$

In Table 4, we present the average number of conjugate gradient iterations per fold in the CV procedure. The approach of using diagonal preconditioning reduces the number of iterations for only two problems. The number is increased for all other data sets. This experiment indicates the difficulty of doing preconditioning. Identifying effective preconditioners is thus a challenging future research issue.

6 Discussion and Conclusions

The experiments indicate that because our method (TRON) only approximately finds the Newton direction, it is faster than the Newton method for L2-SVM by Keerthi and DeCoste (2005). We may apply the same trust region strategy to their formulation. However, as L2-SVM is not twice differentiable, theoretical

Problem	CG	PCG
a9a	567	263
real-sim	104	160
news20	71	155
citeseer	113	115
yahoo-japan	278	326
rcv1	225	280
yahoo-korea	779	736

Table 4: Average number of conjugate gradient iterations per fold in the CV procedure. CG: without preconditioning. PCG: using diagonal preconditioning. We use $C = 64$ and $\epsilon = 0.001$.

convergence and implementation issues must be investigated. This is a useful future research topic.

As logistic regression is a special case of maximum entropy models and conditional random fields, it is possible to extend the proposed approach for them. The main challenge is to derive the Hessian matrix and efficiently calculate the Hessian-vector product. This topic deserves a thorough investigation in the future.

One may use a different regularized term for logistic regression. For example, the two-norm $\|\mathbf{w}\|^2/2$ could be replaced by a one-norm term $\|\mathbf{w}\|_1$. Then (1) becomes

$$\min_{\mathbf{w}} \quad \|\mathbf{w}\|_1 + C\sum_{i=1}^{l}\log(1+e^{-y_i\mathbf{w}^T\mathbf{x}_i}). \tag{26}$$

This formula has been used for some applications. See (Balakrishnan and Madigan, 2005; Koh et al., 2007) and references therein. Unfortunately, (26) is not differentiable on $\mathbf{w}$. We can transform it to a twice-differentiable bound-constrained problem by using $\mathbf{w} \equiv \mathbf{w}^+ - \mathbf{w}^-$:

$$\begin{aligned} \min_{\mathbf{w}^+,\mathbf{w}^-} \quad & \sum_{j=1}^{n} w_j^+ + \sum_{j=1}^{n} w_j^- + C\sum_{i=1}^{l}\log(1+e^{-y_i(\mathbf{w}^+-\mathbf{w}^-)^T\mathbf{x}_i}) \\ \text{subject to} \quad & w_j^+ \geq 0, w_j^- \geq 0, \quad j=1,\ldots,n. \end{aligned} \tag{27}$$

As the truncated Newton method by Lin and Moré (1999) exactly targets at such bound-constrained problems, we can thus extend the proposed approach for (27). A comparison to investigate if our method is better than existing ones is an interesting direction for future work.

In summary, we have shown that a trust region Newton method is effective for training large-scale logistic regression problems. The method has nice optimization properties following past developments for large-scale unconstrained optimization. It is interesting that we do not need many special settings for logistic regression; a rather direct use of modern trust region techniques already yields excellent performances. From this situation, we feel that many useful optimization techniques have not been fully exploited for machine learning applications.

Acknowledgments

The authors thank Jianxiong Dong of Yahoo! for helping to provide the data sets yahoo-japan and yahoo-korea. Part of this work was done when the first and the second authors visited Yahoo! Research. The first and the second authors were partially supported by grants from the National Science Council of Taiwan.

References

Suhrid Balakrishnan and David Madigan. Algorithms for sparse linear classifiers in the massive data setting. 2005. URL `http://www.stat.rutgers.edu/~madigan/PAPERS/sm.pdf`.

Steven Benson and Jorge J. Moré. A limited memory variable metric method for bound constrained minimization. Preprint MCS-P909-0901, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, Illinois, 2001.

Bernhard E. Boser, Isabelle Guyon, and Vladimir Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, pages 144–152. ACM Press, 1992.

Ali Bouaricha, Jorge J. Moré, and Zhijun Wu. Newton's method for large-scale optimization. Preprint MCS-P635-0197, Argonne National Laboratory, Argonne, Illinois, 1997.

John N. Darroch and Douglas Ratcliff. Generalized iterative scaling for log-linear models. *The Annals of Mathematical Statistics*, 43(5):1470–1480, 1972.

Iain S. Duff, Roger G. Grimes, and John G. Lewis. Sparse matrix test problems. *ACM Transactions on Mathematical Software*, 15:1–14, 1989.

Joshua Goodman. Sequential conditional generalized iterative scaling. In *ACL*, pages 9–16, 2002.

Rong Jin, Rong Yan, Jian Zhang, and Alex G. Hauptmann. A faster iterative scaling algorithm for conditional exponential model. In *Proceedings of the Twentieth International Conference on Machine Learning (ICML)*, 2003.

Thorsten Joachims. Training linear SVMs in linear time. In *Proceedings of the ACM Conference on Knowledge Discovery and Data Mining (KDD)*. ACM, 2006.

Wei-Chun Kao, Kai-Min Chung, Chia-Liang Sun, and Chih-Jen Lin. Decomposition methods for linear support vector machines. *Neural Computation*, 16(8):1689–1704, 2004. URL `http://www.csie.ntu.edu.tw/~cjlin/papers/linear.pdf`.

S. Sathiya Keerthi and Dennis DeCoste. A modified finite Newton method for fast solution of large scale linear SVMs. *Journal of Machine Learning Research*, 6:341–361, 2005.

Kwangmoo Koh, Seung-Jean Kim, and Stephen Boyd. An interior-point method for large-scale l1-regularized logistic regression. *Journal of Machine Learning Research*, 2007. URL `http://www.stanford.edu/~boyd/l1_logistic_reg.html`. To appear.

Paul Komarek and Andrew W. Moore. Making logistic regression a core data mining tool: A practical investigation of accuracy, speed, and simplicity. Technical Report TR-05-27, Robotics Institute, Carnegie Mellon University, 2005.

David D. Lewis, Yiming Yang, Tony G. Rose, and Fan Li. RCV1: A new benchmark collection for text categorization research. *Journal of Machine Learning Research*, 5:361–397, 2004.

Chih-Jen Lin and Jorge J. Moré. Newton's method for large-scale bound constrained problems. *SIAM Journal on Optimization*, 9:1100–1127, 1999.

Chih-Jen Lin, Ruby C. Weng, and S. Sathiya Keerthi. Trust region Newton method for large-scale logistic regression. In *Proceedings of the 24th International Conference on Machine Learning (ICML)*, 2007. Software available at `http://www.csie.ntu.edu.tw/~cjlin/liblinear`.

Dong C. Liu and Jorge Nocedal. On the limited memory BFGS method for large scale optimization. *Mathematical Programming*, 45(1):503–528, 1989.

Robert Malouf. A comparison of algorithms for maximum entropy parameter estimation. In *Proceedings of the 6th conference on Natural language learning*, pages 1–7. Association for Computational Linguistics, 2002.

Olvi L. Mangasarian. A finite Newton method for classification. *Optimization Methods and Software*, 17(5):913–929, 2002.

Thomas P. Minka. A comparison of numerical optimizers for logistic regression, 2003. URL `http://research.microsoft.com/~minka/papers/logreg/`.

Stephen G. Nash. A survey of truncated-Newton methods. *Journal of Computational and Applied Mathematics*, 124(1-2):45–59, 2000.

D. J. Newman, S. Hettich, C. L. Blake, and C. J. Merz. UCI repository of machine learning databases. Technical report, University of California, Irvine, Dept. of Information and Computer Sciences, 1998. URL `http://www.ics.uci.edu/~mlearn/MLRepository.html`.

Jorge Nocedal and Stephen G. Nash. A numerical study of the limited memory BFGS method and the truncated-newton method for large scale optimization. *SIAM Journal on Optimization*, 1(3):358–372, 1991.

Stephen Della Pietra, Vincent Della Pietra, and John Lafferty. Inducing features of random fields. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(4):380–393, 1997.

John C. Platt. Fast training of support vector machines using sequential minimal optimization. In Bernhard Schölkopf, Christopher J. C. Burges, and Alexander J. Smola, editors, *Advances in Kernel Methods - Support Vector Learning*, Cambridge, MA, 1998. MIT Press.

Trond Steihaug. The conjugate gradient method and trust regions in large scale optimization. *SIAM Journal on Numerical Analysis*, 20:626–637, 1983.

Charles Sutton and Andrew McCallum. An introduction to conditional random fields for relational learning. In Lise Getoor and Ben Taskar, editors, *Introduction to Statistical Relational Learning*. MIT Press, 2006.

Jian Zhang, Rong Jin, Yiming Yang, and Alex G. Hauptmann. Modified logistic regression: an approximation to SVM and its applications in large-scale text categorization. In *Proceedings of the Twentieth International Conference on Machine Learning (ICML)*, 2003.

Xiaolei Zou, I. Michael Navon, M. Berger, P. K. H. Phua, Tamar Schlick, and F.X. Le Dimet. Numerical experience with limited-memory quasi-Newton and truncated Newton methods. *SIAM Journal on Optimization*, 3(3):582–608, 1993.

A Proof of Theorem 1

Since $f(\mathbf{w})$ is strictly convex, a minimum attained is unique and global. The remaining issue is to check if a minimum exists (as strictly convex functions like e^x do not attain a minimum). It suffices to prove that the level set is bounded:

$$\{\mathbf{w} \mid f(\mathbf{w}) \leq f(\mathbf{w}^0)\}, \tag{28}$$

where $\mathbf{w}^0$ is any vector. If this property is wrong, there is a sequence $\{\mathbf{w}^k\}$ in the set (28) satisfying $\|\mathbf{w}^k\| \to \infty$. However,

$$f(\mathbf{w}^k) \geq \frac{1}{2}\|\mathbf{w}^k\|^2 \to \infty$$

contradicts the fact that $f(\mathbf{w}^k) \leq f(\mathbf{w}^0), \forall k$.