Trust Region Newton Method for Large-Scale Logistic Regression

Chih-Jen Lin   CJLIN@CSIE.NTU.EDU.TW
Department of Computer Science
National Taiwan University
Taipei 106, Taiwan

Ruby C. Weng   CHWENG@NCCU.EDU.TW
Department of Statistics
National Chengchi University
Taipei 116, Taiwan

S. Sathiya Keerthi   SELVARAK@YAHOO-INC.COM
Yahoo! Research
Santa Clara, CA 95054, USA

Editor: Alexander Smola

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 extend the proposed method to large-scale L2-loss linear support vector machines (SVM).

Keywords: logistic regression, newton method, trust region, conjugate gradient, support vector machines

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

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

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

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:

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

where $C > 0$ is a parameter decided by users so that the two terms in (2) are balanced. One can easily check that (2) is twice continuously differentiable.

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 (2). One can distinguish them according to the following two extreme situations of optimization methods:

$$\begin{array}{ccc} \text{Low cost per iteration;} & & \text{High cost per iteration;} \\ \text{slow convergence.} & \longleftrightarrow & \text{fast convergence.} \end{array}$$

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. This approach, called trust region Newton method, uses only approximate Newton steps in the beginning, but takes full Newton directions in the end for fast convergence.

In Sections 3 and 4, we discuss some existing optimization methods for logistic regression and conduct comparisons. As Newton method uses the exact Hessian (second derivative), it has quadratic convergence near the optimum. Results indicate that our proposed method converges much faster than quasi-Newton methods, which use only an approximate Hessian. Section 5 investigates a variant of our proposed method by using preconditioned conjugate gradients in the trust region framework. In Section 6, we extend the proposed trust region method to solve L2-loss support vector machines. Finally, Section 7 gives conclusions.

All sources used in this paper are available at

<div align="center"><code>http://www.csie.ntu.edu.tw/~cjlin/liblinear</code>.</div>

A preliminary version 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 $f(\mathbf{w})$:

$$
\begin{aligned}
\nabla f(\mathbf{w}) &= \mathbf{w} + C \sum_{i=1}^{l} (\sigma(y_i \mathbf{w}^T \mathbf{x}_i) - 1) y_i \mathbf{x}_i, & (3) \\
\nabla^2 f(\mathbf{w}) &= I + C X^T D X, & (4)
\end{aligned}
$$

where 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

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

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

Theorem 1 (2) *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 $\mathbf{s}$:

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

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.[1] 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 by Nash (2000). Some comparisons between limited memory quasi Newton and truncated Newton are by Nocedal and Nash (1991) and Zou et al. (1993).

2.2 A Trust Region Newton Method

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

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

---

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

Algorithm 1 A trust region algorithm for logistic regression

1. Given $\mathbf{w}^0$.

2. For $k = 0, 1, \dots$ (outer iterations)

   - If $\nabla f(\mathbf{w}^k) = \mathbf{0}$, stop.
   - Find an approximate solution $\mathbf{s}^k$ of the trust region sub-problem

     $$\min_{\mathbf{s}} \ q_k(\mathbf{s}) \quad \text{subject to } \|\mathbf{s}\| \le \Delta_k. \tag{11}$$

   - Compute ρ_k via (8).
   - Update $\mathbf{w}^k$ to $\mathbf{w}^{k+1}$ according to (9).
   - Obtain Δ_{k+1} according to (10).

---

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

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

Similar rules are used in most modern trust region methods. A description of our trust region algorithm is given in Algorithm 1. The main difference between our algorithm and those by Steihaug (1983) and Bouaricha et al. (1997) is on the rule (10) for updating Δ_k.

The conjugate gradient method to approximately solve the trust region sub-problem (11) is given in Algorithm 2. 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

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

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
     $$\|\mathbf{r}^i\| \le \xi_k \|\nabla f(\mathbf{w}^k)\|, \tag{12}$$
     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 $\|\bar{\mathbf{s}}^{i+1}\| \ge \Delta_k$, compute τ such that
     $$\|\bar{\mathbf{s}}^i + \tau \mathbf{d}^i\| = \Delta_k, \tag{13}$$
     then output $\mathbf{s}^k = \bar{\mathbf{s}}^i + \tau \mathbf{d}^i$ and stop.
   - $\mathbf{r}^{i+1} = \mathbf{r}^i - \alpha_i \nabla^2 f(\mathbf{w}^k) \mathbf{d}^i$.
   - $\beta_i = \|\mathbf{r}^{i+1}\|^2 / \|\mathbf{r}^i\|^2$.
   - $\mathbf{d}^{i+1} = \mathbf{r}^{i+1} + \beta_i \mathbf{d}^i$.

---

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 $\|\mathbf{s}\| \le \Delta$ 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,[2] since $f(\mathbf{w}^k)$ is decreasing, it suffices to show that the level set $\{\mathbf{w} \mid f(\mathbf{w}) \le f(\mathbf{w}^0)\}$ 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. The following theorem summarizes the convergence of Algorithm 1.

---

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

Theorem 2 *The sequence $\{\mathbf{w}^k\}$ generated by Algorithm 1 globally converges to the unique minimum of* (2). *If $\xi_k < 1$, then the trust region method Q-linearly converges:*

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

where $\mathbf{w}^*$ is the unique optimal solution of (2). *If*

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

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

We do not provide a proof here as it follows from Theorem 5.4 of Lin and Moré (1999). Since the Hessian matrix $\nabla^2 f(\mathbf{w})$ is continuously differentiable, $\nabla^2 f(\mathbf{w})$ is Lipschitz continuous around the optimal solution. Hence, as explained by Lin and Moré (1999),[3] if $\xi_k \leq \kappa_0 \|\nabla f(\mathbf{w}^k)\|$ for a positive constant κ_0, then at final iterations, our algorithm has quadratic convergence:

$$\lim_{k \to \infty} \frac{\|\mathbf{w}^{k+1} - \mathbf{w}^*\|}{\|\mathbf{w}^k - \mathbf{w}^*\|^2} < 1.$$

Regarding the computational complexity, the cost per iteration is

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

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 a general limited memory quasi Newton implementation (Liu and Nocedal, 1989). Many consider it to be very efficient for training logistic regression. We also discuss implementation issues of the proposed trust region Newton method.

3.1 Limited Memory Quasi Newton Method

We briefly introduce the approach by 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 procedure is sketched in Algorithm 3.

Regarding the convergence rate, Assumption 7.1 of Liu and Nocedal (1989) requires:

1. $f(\mathbf{w})$ is twice continuously differentiable.

2. The level set $\{\mathbf{w} \mid f(\mathbf{w}) \leq f(\mathbf{w}^0)\}$ is bounded.

---

3. See the explanation given in that paper after the proof of Theorem 5.4.

Algorithm 3 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 α_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}.

---

3. There are positive constants M_1 and M_2 such that

$$M_1 \|\mathbf{s}\|^2 \leq \mathbf{s}^T \nabla^2 f(\mathbf{w}) \mathbf{s} \leq M_2 \|\mathbf{s}\|^2, \quad \forall \mathbf{s}.$$

The function we are minimizing satisfies the first condition. The second condition follows from our proof of Theorem 1 (see Eq. 29). The third condition follows from choosing

$$M_1 = 1 \text{ and } M_2 = 1 + C\|X^T\|\|X\|.$$

Then Algorithm 3 is R-linearly convergent. That is, there is a constant $c < 1$ such that

$$f(\mathbf{w}^k) - f(\mathbf{w}^*) \leq c^k(f(\mathbf{w}^0) - f(\mathbf{w}^*)), \tag{16}$$

where $\mathbf{w}^*$ is the unique optimal solution of (2). Note that (14) implies (16), so Algorithm 1 has a stronger convergence property than Algorithm 3. 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

$$O(\text{nnz}) \text{ for function/gradient evaluations in line search}$$
$$+ \quad O(nm) \text{ for } H_k \nabla f(\mathbf{w}^k) \text{ and updating } H_k \text{ to } H_{k+1}. \tag{17}$$

As generally $nm < \text{nnz}$, 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).

3.2 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).

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
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. # nonzeros indicates the number of nonzeros among $l \times n$ values.

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 by 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 a quasi Newton implementation for logistic regression. After describing data sets for experiments, we conduct detailed comparisons and discuss results.

4.1 Data Sets

We consider six 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 by Platt (1998) from the UCI "adult" data set (Asuncion and Newman, 2007). 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 by Keerthi and DeCoste (2005). They consider binary term frequencies and normalize each inst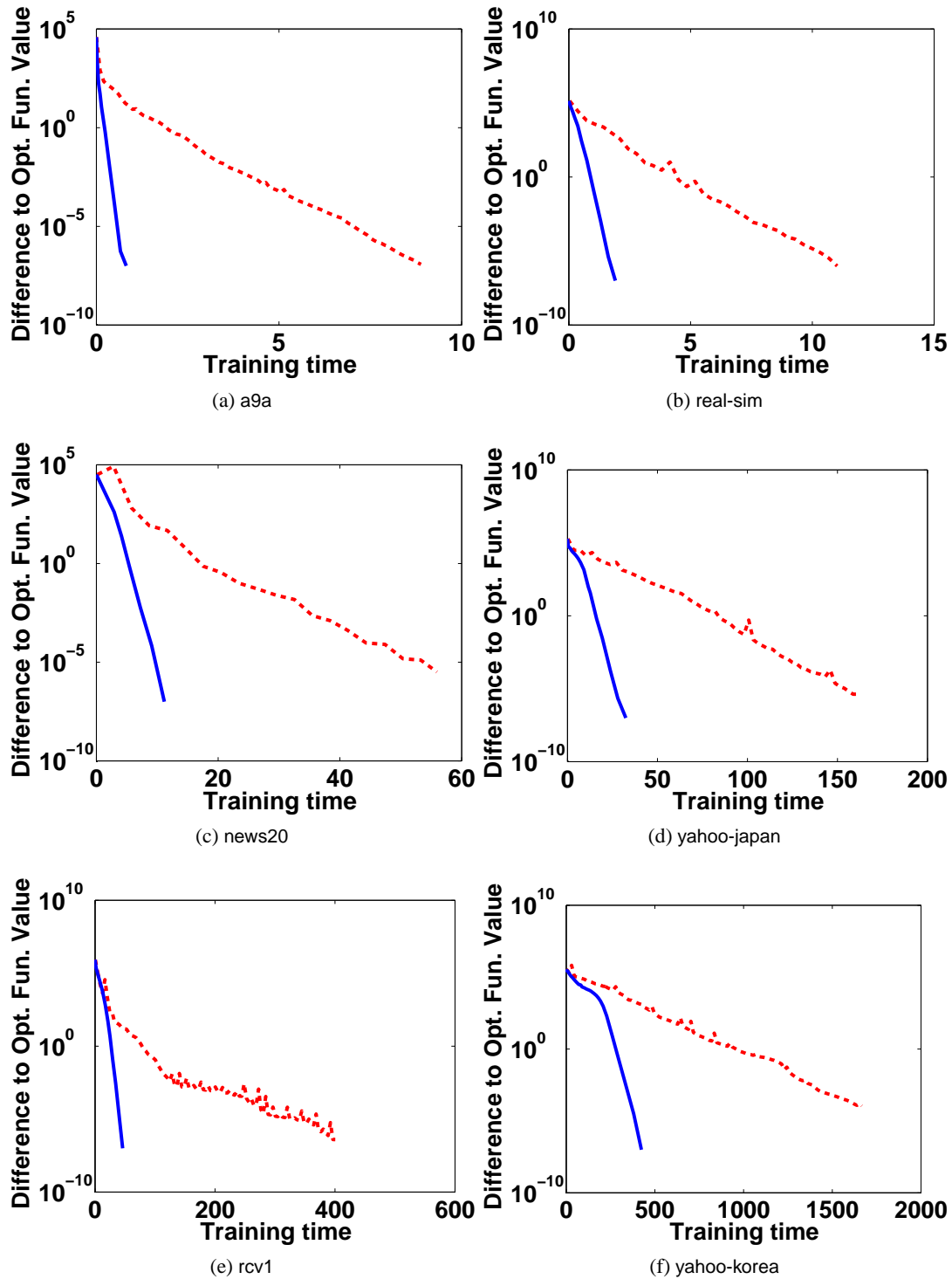ance to unit length. This set is available at `http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary/news20.binary.bz2`.

C	TRON CV	Time	LBFGS Time
0.25	84.69%	1	12
1	84.71%	2	24
4	84.72%	4	47
16	84.71%	7	92

(a) a9a

C	TRON CV	Time	LBFGS Time
0.25	95.85%	4	17
1	96.97%	6	34
4	97.41%	10	52
16	97.51%	14	126

(b) real-sim

C	TRON CV	Time	LBFGS Time
0.25	89.74%	24	78
1	93.36%	38	181
4	95.49%	59	331
16	96.30%	82	614

(c) news20

C	TRON CV	Time	LBFGS Time
0.25	91.91%	28	94
1	92.50%	42	185
4	92.81%	64	326
16	92.86%	113	534

(d) yahoo-japan

C	TRON CV	Time	LBFGS Time
0.25	97.18%	39	106
1	97.56%	62	427
4	97.72%	94	615
16	97.69%	118	821

(e) rcv1

C	TRON CV	Time	LBFGS Time
0.25	81.34%	221	1066
1	84.03%	385	2165
4	85.75%	773	3480
16	86.40%	1888	6329

(f) yahoo-korea

Table 2: The comparison between TRON and LBFGS. 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 that the CV values do not increase using $C > 16$.

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. 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, past experiences show that such normalizations are usually needed). After normalization, we include the bias term using (1).

All data sets 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.

Figure 1: A comparison between TRON (blue solid line) and LBFGS (red dotted line). The *y*-axis shows the difference to the optimal function value. The *x*-axis (training time) is in seconds. We use the training set from the first training/validation split of the CV procedure and set $C = 4$.
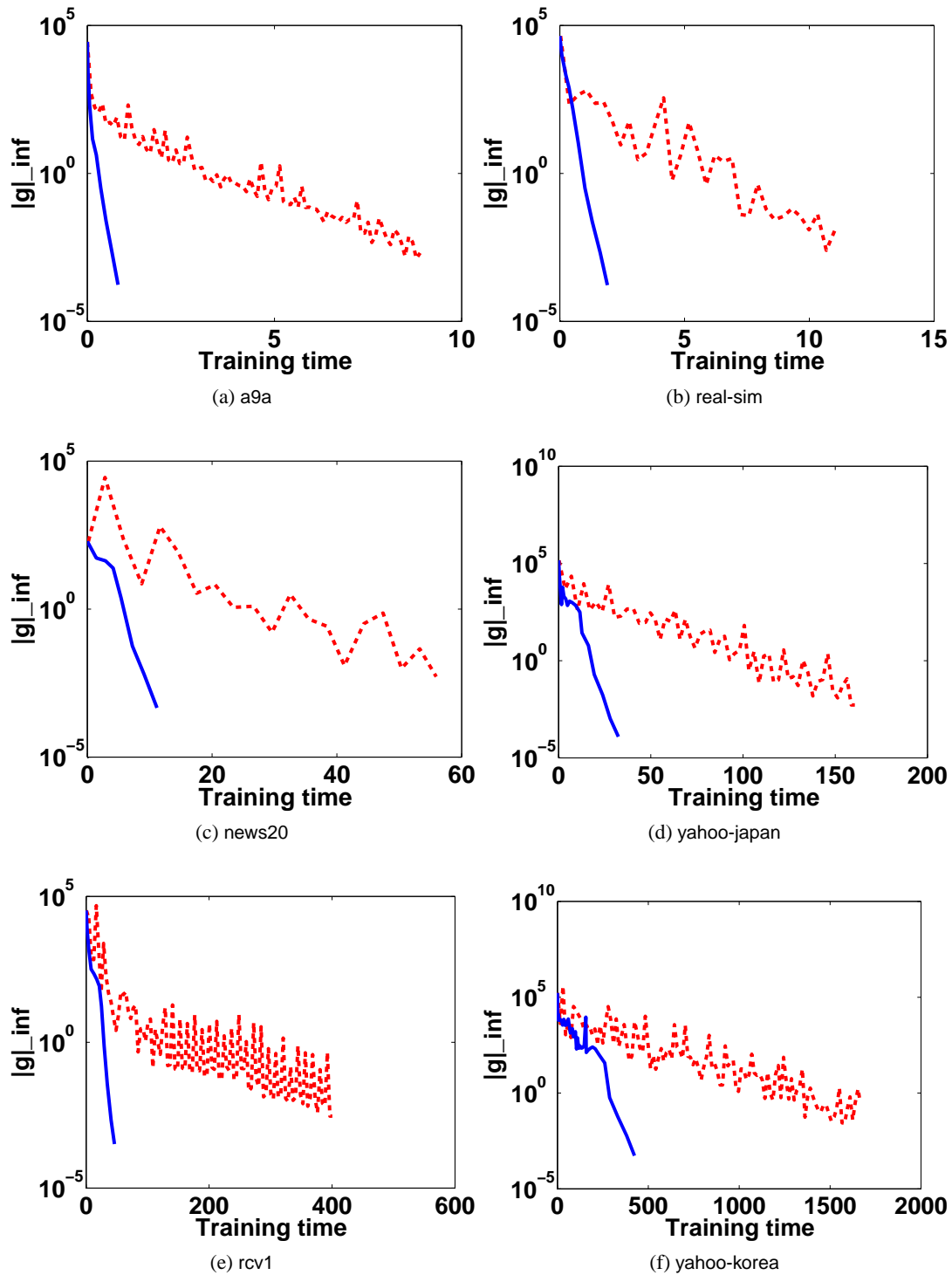
Figure 2: A comparison between TRON (blue solid line) and LBFGS (red dotted line). The y-axis shows $\|\nabla f(\mathbf{w})\|_\infty = \max_j |\nabla f(\mathbf{w})_j|$. The x-axis (training time) is in seconds. We use the training set from the first training/validation split of the CV procedure and set $C = 4$.

4.2 Comparisons

We compare two logistic regression implementations:

- 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 3.1. The source code is available online at

    http://www.ece.northwestern.edu/~nocedal/lbfgs.html.

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.

These methods are implemented in high-level languages such as C/C++ or FORTRAN. For easier experiments, we use their Matlab interfaces. Experiments are conducted on an Intel Core2 Quad (2.66GHz) computer with 8 GB RAM. All sources used for this comparison can be found at

    http://www.csie.ntu.edu.tw/~cjlin/liblinear.

We set the initial $\mathbf{w}^0 = \mathbf{0}$.

We conduct two types of experiments. For the first one, we simulate the practical use of logistic regression by setting a stopping condition and checking the prediction ability. Most unconstrained optimization software use gradient information to terminate the iterative procedure, so we use

$$\|\nabla f(\mathbf{w}^k)\|_\infty \leq 10^{-3} \tag{18}$$

as the stopping condition. We then 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 conducted. We do not consider other measurements such as AUC (Area Under Curve) or F-measure as all problems 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.

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 the 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.

For $C \geq 16$, the CV accuracy does not improve. One can clearly see that the training time is higher as C increases. One reason is that the second term of (4) plays a more important role and hence the Hessian matrix is more ill-conditioned. Another reason is that (18) becomes a stricter condition. In (3), the second term of $f(\mathbf{w})$ is proportional to C. Hence, practically one may use a stopping condition relative to C.
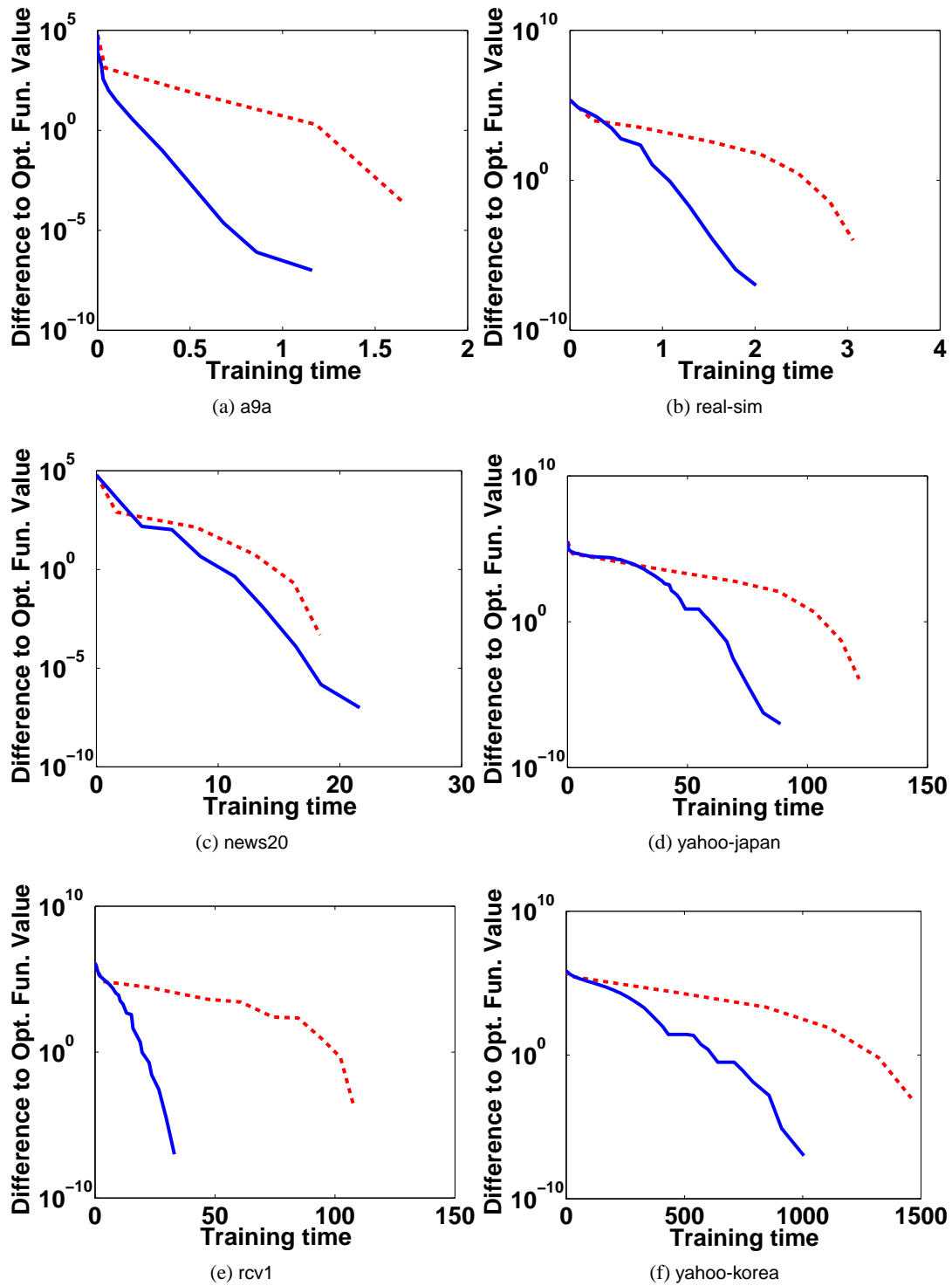
Figure 3: A comparison between TRON (blue solid line) and SVMlin (red dotted line) for L2-SVM. The *y*-axis shows the difference to the optimal function value. The *x*-axis (training time) is in seconds. We use the same training set as in Figure 1 and set $C = 4$.

For the second experiment, we check the convergence speed of both methods. Figure 1 presents the results of time versus the difference to the optimal function value. We use the training set from the first training/validation split of the CV procedure and set $C = 4$. In Figure 2, we check time against $\|\nabla f(\mathbf{w})\|_\infty$. Both figures indicate that TRON more quickly decreases the function as well as the gradient values than LBFGS. This result is consistent with the faster theoretical convergence rate of TRON.

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.[4] 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 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, both methods are implemented using the row format. Without being careful on such details, very easily we may get misleading conclusions.

---

4. 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	7	7
real-sim	14	22
news20	82	55
yahoo-japan	113	127
rcv1	118	226
yahoo-korea	1888	2060

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 $\varepsilon = 0.001$.

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

$$\nabla^2 f(\mathbf{w}^k) \approx PP^T \tag{19}$$

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 (19) is good enough, $P^{-1}\nabla^2 f(\mathbf{w}^k)P^{-T}$ 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. 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 of Lin and Moré (1999) by considering a scaled version

$$\min_{\mathbf{s}} \; q_k(\mathbf{s}) \quad \text{subject to } \|P^T\mathbf{s}\| \leq \Delta_k. \tag{20}$$

Algorithm 4 Preconditioned conjugate gradient procedure for approximately solving the trust region sub-problem (21)

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 $i = 0, 1, \ldots$ (inner iterations)

   - If
     $$\|\mathbf{r}^i\| \leq \xi_k \|\hat{\mathbf{g}}\|,$$
     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 $\|\hat{\mathbf{s}}^{i+1}\| \geq \Delta_k$, compute τ 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.
   - $\mathbf{r}^{i+1} = \mathbf{r}^i - \alpha_i \hat{H} \mathbf{d}^i$.
   - $\beta_i = \|\mathbf{r}^{i+1}\|^2 / \|\mathbf{r}^i\|^2$.
   - $\mathbf{d}^{i+1} = \mathbf{r}^{i+1} + \beta_i \mathbf{d}^i$.

With $\hat{\mathbf{s}} = P^T \mathbf{s}$, we transform (20) to

$$\min_{\hat{\mathbf{s}}} \ \hat{q}_k(\hat{\mathbf{s}}) \quad \text{subject to } \|\hat{\mathbf{s}}\| \leq \Delta_k, \tag{21}$$

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. (21) is in the same form as (11), the sub-problem without using preconditioners, so the procedure to approximately solve (21) is almost the same as Algorithm 2. We give details in Algorithm 4. 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.

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 $C = 16$ and the stopping condition $\|\nabla f(\mathbf{w}^k)\|_\infty \leq 0.001$.

6. Trust Region Method for L2-SVM

The second term in (2) 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:

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

while L2-SVM solves

$$\min_{\mathbf{w}} f_2(\mathbf{w}) \equiv \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_{i=1}^{l} \left(\max(0, 1 - y_i\mathbf{w}^T\mathbf{x}_i)\right)^2. \tag{22}$$

SVM is often used with a nonlinear kernel, where data $\mathbf{x}_i$ are mapped to a high dimensional space. However, for document classification, past experiments show that 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; Shalev-Shwartz et al., 2007; Smola et al., 2008). L1-SVM is not differentiable, so our method cannot be applied. For L2-SVM, the training objection function (22) is differentiable but not twice differentiable (Mangasarian, 2002). In this section, we extend our trust region method for L2-SVM. We then compare it with an earlier Newton method for L2-SVM (Keerthi and DeCoste, 2005).

6.1 Trust Region Method

Let $f_2(\mathbf{w})$ be the L2-SVM function. It is strictly convex, so a proof similar to Theorem 1 shows that a unique global minimum exists. From Mangasarian (2002), $f_2(\mathbf{w})$ is continuously differentiable with the gradient

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

where

$$I = \{i \mid 1 - y_i\mathbf{w}^T\mathbf{x}_i > 0\} \tag{23}$$

is an index set depending on $\mathbf{w}$ and $X_{I,:}$ includes X's rows corresponding to the set I. Unfortunately, L2-SVM is not twice differentiable, so one cannot use Newton directions. However, as shown by Mangasarian (2002), this function is almost twice differentiable. The gradient $\nabla f_2(\mathbf{w})$ is Lipschitz continuous, so one can define the *generalized* Hessian matrix

$$B(\mathbf{w}) = I + 2CX^T DX,$$

where

$$D_{ii} = \begin{cases} 1 & \text{if } 1 - y_i \mathbf{w}^T \mathbf{x}_i > 0, \\ \text{any element in } [0,1] & \text{if } 1 - y_i \mathbf{w}^T \mathbf{x}_i = 0, \\ 0 & \text{if } 1 - y_i \mathbf{w}^T \mathbf{x}_i < 0. \end{cases}$$

Then the trust region method (Lin and Moré, 1999) can be applied by replacing $\nabla^2 f(\mathbf{w})$ in Section 2.2 with $B(\mathbf{w})$. In other words, we use the generalized Hessian matrix $B(\mathbf{w})$ to obtain Newton-like directions. As $B(\mathbf{w})$ is uniformly bounded:

$$1 \le \|B(\mathbf{w})\| \le 1 + 2C\|X^T\|\|X\|, \quad \forall \mathbf{w},$$

Theorem 2.1 of Lin and Moré (1999) implies the global convergence. However, we cannot apply Theorem 5.4 of Lin and Moré (1999) to have quadratic convergence. This result requires the twice continuous differentiability.

For experiments here, we set $D_{ii} = 0$ if $1 - y_i \mathbf{w}^T \mathbf{x}_i = 0$. The Hessian-vector product in the conjugate gradient procedure is then

$$B(\mathbf{w})\mathbf{s} = \mathbf{s} + 2C \cdot X_{I,:}^T (D_{I,I}(X_{I,:}\mathbf{s})). \tag{24}$$

6.2 Modified Newton Method for L2-SVM

The method by Keerthi and DeCoste (2005) is currently one of the most efficient methods to train large-scale linear L2-SVM. Its key idea is that for any given index set $I \subset \{1, \ldots, l\}$, if the optimal solution $\mathbf{w}^*$ of the following problem

$$\min_{\mathbf{w}} \quad \frac{1}{2}\mathbf{w}^T \mathbf{w} + C \sum_{i \in I} (1 - y_i \mathbf{w}^T \mathbf{x}_i)^2 \tag{25}$$

satisfies

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

then $\mathbf{w}^*$ is an optimal solution of the L2-SVM problem (22). Once I is fixed, (25) is a simple regularized least square problem and can be solved by the following linear system:

$$(I + 2CX_{I,:}^T X_{I,:})\mathbf{w} = 2CX_{I,:}^T \mathbf{y}_I. \tag{26}$$

One then guesses this set I by (23) and solves (26). The matrix in (26) is a generalized Hessian at $\mathbf{w}$, so (26) intends to obtain a Newton-like direction. Keerthi and DeCoste (2005) use conjugate gradient methods to solve (26), and the procedure is described in Algorithm 5. They prove that Algorithm 5 converges to the optimal solution of (22) in a finite number of iterations. This convergence result assumes that at each iteration, (26) is exactly solved. However, they use a relative stopping condition in practical implementations, so the convergence remains an issue. In contrast, the convergence of our trust region method holds when the conjugate gradient procedure only approximately minimizes the trust-region sub-problem.

Algorithm 5 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 (26) using

$$I_k = \{i \mid 1 - y_i(\mathbf{w}^k)^T \mathbf{x}_i > 0\}.$$

Solve (26) 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 $\mathbf{w}^{k+1} = \mathbf{w}^k + \alpha_k \mathbf{s}^k$.

6.3 Comparisons

We compare our proposed trust region implementation (TRON) in Section 6.1 with SVMlin

$$\texttt{http://people.cs.uchicago.edu/~vikass/svmlin.html,}$$

an implementation of the method by Keerthi and DeCoste (2005). To solve (26), SVMlin considers a relative stopping condition for the conjugate gradient procedure. Following their convergence result, we modify SVMlin to quite accurately solve the linear system (26): Recall in Algorithm 5 that we sequentially obtain the following items:

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

We then use

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

as the stopping condition of the conjugate gradient procedure in SVMlin.

    Figure 3 presents the result of time versus the difference to the optimal function value. Both approaches spend most of their time on the operation (24) in the conjugate gradient procedure. Clearly, TRON more quickly reduces the function value. SVMlin is slower because it accurately solves (26) at early iterations. Hence, many conjugate gradient iterations are wasted. In contrast, trust region methods are effective on using only approximate directions in the early stage of the procedure.

7. Discussion and Conclusions

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 (2) becomes

$$\min_{\mathbf{w}} \quad \|\mathbf{w}\|_1 + C \sum_{i=1}^{l} \log(1 + e^{-y_i \mathbf{w}^T \mathbf{x}_i}). \tag{27}$$

This formula has been used for some applications. See (Balakrishnan and Madigan, 2005) and Koh et al. (2007) and references therein. Unfortunately, (27) is not differentiable on $\mathbf{w}$. We can transform it to a twice-differentiable bound-constrained problem by using $\mathbf{w} \equiv \mathbf{w}^+ - \mathbf{w}^-$:

$$\min_{\mathbf{w}^+, \mathbf{w}^-} \quad \sum_{j=1}^{n} w_j^+ + \sum_{j=1}^{n} w_j^- + C \sum_{i=1}^{l} \log(1 + e^{-y_i(\mathbf{w}^+ - \mathbf{w}^-)^T \mathbf{x}_i})$$
$$\text{subject to} \quad w_j^+ \geq 0, w_j^- \geq 0, \quad j = 1, \ldots, n. \tag{28}$$

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 (28). 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 as well as L2-SVM. 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

Appendix 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:

$$\{\mathbf{w} \mid f(\mathbf{w}) \leq f(\mathbf{w}^0)\}, \tag{29}$$

where $\mathbf{w}^0$ is any vector. If this property is wrong, there is a sequence $\{\mathbf{w}^k\}$ in the set (29) 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 $f(\mathbf{w}^k) \leq f(\mathbf{w}^0), \forall k$.

References

Arthur Asuncion and David J. Newman. UCI machine learning repository, 2007. URL `http://www.ics.uci.edu/$\sim$mlearn/{MLR}epository.html`.

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*, 8:1519–1555, 2007. URL `http://www.stanford.edu/~boyd/l1_logistic_reg.html`.

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.

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.

Shai Shalev-Shwartz, Yoram Singer, and Nathan Srebro. Pegasos: primal estimated sub-gradient solver for SVM. In *Proceedings of the 24th International Conference on Machine Learning (ICML)*, 2007.

Alex J. Smola, S V N Vishwanathan, and Quoc Le. Bundle methods for machine learning. In J.C. Platt, D. Koller, Y. Singer, and S. Roweis, editors, *Advances in Neural Information Processing Systems 20*. MIT Press, Cambridge, MA, 2008.

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.

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.