An Improved GLMNET for L1-regularized Logistic Regression

Guo-Xun Yuan
Dept. of Computer Science
National Taiwan University
Taipei 106, Taiwan
r96042@csie.ntu.edu.tw

Chia-Hua Ho
Dept. of Computer Science
National Taiwan University
Taipei 106, Taiwan
b95082@csie.ntu.edu.tw

Chih-Jen Lin
Dept. of Computer Science
National Taiwan University
Taipei 106, Taiwan
cjlin@csie.ntu.edu.tw

ABSTRACT

GLMNET proposed by Friedman et al. [1] is an algorithm for generalized linear model with elastic net. It has been widely applied to solve L1-regularized logistic regression. However, recent experiments indicated that the existing GLMNET implementation may not be stable for large-scale problems. In this paper, we propose an improved GLMNET to address some theoretical and implementation issues. In particular, as a Newton-type method, GLMNET achieves fast local convergence, but may fail to quickly obtain a useful solution. By a careful design to adjust the effort for each iteration, our method is efficient regardless of loosely or strictly solving the optimization problem. Experiments demonstrate that the improved GLMNET is more efficient than a state-of-the-art coordinate descent method.

Categories and Subject Descriptors

I.5.2 [**Pattern Recognition**]: Design Methodology—*Classifier design and evaluation*

General Terms

Algorithms, Performance, Experimentation

1. INTRODUCTION

Logistic regression is a popular classification method in machine learning. Recently, L1-regularized logistic regression is widely used because it can generate a sparse model. Given a set of instance-label pairs $(\boldsymbol{x}_i, y_i),\ i = 1, \ldots, l,\ \boldsymbol{x}_i \in \mathbf{R}^n,\ y_i \in \{-1, +1\}$, L1-regularized logistic regression solves the following unconstrained optimization problem:

$$\min_{\boldsymbol{w}} \quad f(\boldsymbol{w}) \equiv \|\boldsymbol{w}\|_1 + C \sum_{i=1}^{l} \log(1 + e^{-y_i \boldsymbol{w}^T \boldsymbol{x}_i}), \quad (1)$$

where $\|\cdot\|_1$ denotes the 1-norm. The regularization term $\|\boldsymbol{w}\|_1$ is used to avoid overfitting the training data, while the other term is the sum of training losses. The user-defined parameter $C > 0$ is used to balance the regularization and loss terms. Different from the 2-norm regularization, the 1-norm regularization gives a sparse solution of (1).

A difficulty for solving (1) is that $\|\boldsymbol{w}\|_1$ is not differentiable. Many optimization approaches have been proposed; for example, [2, 3, 4, 5, 6]. Recently, Yuan et al. [7] made a comprehensive comparison among state-of-the-art algorithms and software for L1-regularized logistic regression and SVM. They conclude that coordinate descent (CD) methods perform better than others for large sparse data (e.g., document data). In particular, CD methods can quickly obtain a useful solution.

In [7], the method GLMNET by Friedman et al. [1] is included for comparison and the performance is reasonably well. GLMNET is a Newton-type approach by iteratively minimizing the second-order approximation. The result in [7] indicates that GLMNET has fast local convergence, but may spend too much time in early iterations. Thus, different from CD methods, GLMNET could not quickly obtain a useful model. Further, in [7], the existing GLMNET implementation failed to solve two large problems.

In this paper, we propose an improved GLMNET called newGLMNET. Our contributions are as follows. First, we modify GLMNET to be under a general framework in [8] for establishing the theoretical convergence. Second, we address some implementation concerns in GLMNET. In particular, by a careful design to adjust the effort for each iteration, our method is efficient for both loosely and strictly solving the optimization problem. Third, we show that the CD method recommended by [7] may be inefficient for dense data (i.e., each instance contains many non-zero feature values). The reason is that compared to newGLMNET and GLMNET, in a similar number of operations, CD methods need much more exponential/logarithmic operations.

We define notation and discuss some basic properties of (1). The logistic loss is twice differentiable, so most existing approaches use the gradient and the Hessian of the second term in (1): By defining

$$L(\boldsymbol{w}) \equiv C \sum_{i=1}^{l} \log\left(1 + e^{-y_i \boldsymbol{w}^T \boldsymbol{x}_i}\right),$$

the gradient and Hessian of $L(\boldsymbol{w})$ are respectively

$$\begin{aligned} &\nabla L(\boldsymbol{w}) = C \sum_{i=1}^{l} (\tau(y_i \boldsymbol{w}^T \boldsymbol{x}_i) - 1) y_i \boldsymbol{x}_i \quad \text{and} \\ &\nabla^2 L(\boldsymbol{w}) = C X^T D X, \end{aligned} \quad (2)$$

where $\tau(s)$ is the derivative of the logistic loss $\log(1 + e^s)$:

$$\tau(s) = \frac{1}{1 + e^{-s}},$$

KDD'11, August 21–24, 2011, San Diego, CA, USA.

$D \in \mathbf{R}^{l\times l}$ is a diagonal matrix with

$$D_{ii} = \tau(y_i \boldsymbol{w}^T \boldsymbol{x}_i)\left(1 - \tau(y_i \boldsymbol{w}^T \boldsymbol{x}_i)\right),$$

and

$$X \equiv \begin{bmatrix} \boldsymbol{x}_1^T \\ \vdots \\ \boldsymbol{x}_l^T \end{bmatrix} \in \mathbf{R}^{l\times n}.$$

From standard convex analysis, $\boldsymbol{w}^*$ is optimal for problem (1) if and only if $\boldsymbol{w}^*$ satisfies the optimality conditions:

$$\begin{cases} \nabla_j L(\boldsymbol{w}^*) + 1 = 0 & \text{if } w_j^* > 0, \\ \nabla_j L(\boldsymbol{w}^*) - 1 = 0 & \text{if } w_j^* < 0, \\ -1 \le \nabla_j L(\boldsymbol{w}^*) \le 1 & \text{if } w_j^* = 0. \end{cases} \tag{3}$$

Equivalently, $\boldsymbol{w}^*$ is optimal if and only if the minimum-norm sub-gradient at $\boldsymbol{w}^*$ is zero:

$$\nabla^S f(\boldsymbol{w}^*) = \mathbf{0}, \tag{4}$$

where

$$\nabla_j^S f(\boldsymbol{w}) \equiv \begin{cases} \nabla_j L(\boldsymbol{w}) + 1 & \text{if } w_j > 0, \\ \nabla_j L(\boldsymbol{w}) - 1 & \text{if } w_j < 0, \\ \operatorname{sgn}(\nabla_j L(\boldsymbol{w}))\max(|\nabla_j L(\boldsymbol{w})| - 1, 0) & \text{if } w_j = 0. \end{cases}$$

This paper is organized as follows. Section 2 discusses a framework of decomposition methods by [8]. The CD method recommended by [7] and GLMNET by [1] can be considered as examples of this framework. We also point out some theoretical and implementation issues of GLMNET. In Section 3, we propose an improved GLMNET called newGLMNET. Experiments in Section 4 shows the superiority of newGLMNET. Section 5 concludes this work.

2. DECOMPOSITION METHODS

Decomposition methods are a commonly-used optimization approach by iteratively solving smaller sub-problems. At each iteration of a decomposition method, a set J of working variables is chosen and the following sub-problem is solved:

$$\min_{\boldsymbol{d}} \quad f(\boldsymbol{w}+\boldsymbol{d}) - f(\boldsymbol{w}) \quad \text{subject to} \quad d_j = 0 \quad \forall j \notin J, \tag{5}$$

where

$$f(\boldsymbol{w}+\boldsymbol{d}) - f(\boldsymbol{w}) = L(\boldsymbol{w}+\boldsymbol{d}) - L(\boldsymbol{w}) + \|\boldsymbol{w}+\boldsymbol{d}\|_1 - \|\boldsymbol{w}\|_1.$$

Only variables in J are updated at each iteration. If $\boldsymbol{d}$ is an optimal solution of (5), then $\boldsymbol{w}$ is updated by

$$w_j \leftarrow w_j + d_j, \forall j \in J.$$

There are two key issues of a decomposition method: one is the selection of the working set J; the other is how to solve the sub-problem (5). Practically, because the smaller sub-problem (5) may be still hard, most existing decomposition methods only approximately solve (5).

Tseng and Yun [8] study a general decomposition method CGD (Coordinate Gradient Descent) for L1-regularized problems. They assume that at each iteration, a second-order approximation to $L(\boldsymbol{w}+\boldsymbol{d}) - L(\boldsymbol{w})$ is used. Specifically, given a symmetric positive-definite matrix $H^k \in \mathbf{R}^{n\times n}$ approximating the Hessian matrix $\nabla^2 L(\boldsymbol{w}^k)$ and a working set

Algorithm 1 A decomposition framework by [8] for L1-regularized logistic regression

1. Given $\boldsymbol{w}^1$. Choose a strategy for selecting working sets. Given $0 < \beta, \sigma < 1$ and $0 \le \gamma < 1$.
2. For $k = 1, 2, 3, \dots$
   - Choose an H^k and a working set J^k.
   - Get $\boldsymbol{d}^k$ by solving the sub-problem (6).
   - Compute $\lambda = \max\{1, \beta, \beta^2, \dots\}$ such that $\lambda \boldsymbol{d}^k$ satisfies (8).
   - $\boldsymbol{w}^{k+1} = \boldsymbol{w}^k + \lambda \boldsymbol{d}^k$.

$J^k \subseteq \mathcal{N} = \{1, \dots, n\}$ at the kth iteration, CGD solves the following quadratic sub-problem:

$$\min_{\boldsymbol{d}} \quad q_k(\boldsymbol{d}) \equiv \nabla L(\boldsymbol{w}^k)^T \boldsymbol{d} + \frac{1}{2}\boldsymbol{d}^T H^k \boldsymbol{d} + \|\boldsymbol{w}^k + \boldsymbol{d}\|_1 - \|\boldsymbol{w}^k\|_1$$
$$\text{subject to} \quad d_j = 0 \quad \forall j \notin J^k, \tag{6}$$

where $\boldsymbol{w}^k$ is the current iterate.

Many methods are available to select the working set J. A common rule is to choose working variables in a cyclic manner. That is, elements of the sequence $\{\{1\}, \{2\}, \dots, \{n\}\}$ repeatedly take turns to serve as the working set J. The cyclic rule for choosing working variables is often called the Gauss-Seidel rule. Besides, because the gradient reveals the closeness to an optimal solution, some methods, called Gauss-Southwell rules, select J based on the gradient information. The concept is extended in [8] by checking the relationship between $\boldsymbol{d}(J)$ and $\boldsymbol{d}(\mathcal{N})$, which are optimal solutions of (6) using J and $\mathcal{N}$ as working sets, respectively. For example, one rule selects J satisfying

$$\|\boldsymbol{d}(J)\|_\infty \ge v\|\boldsymbol{d}(\mathcal{N})\|_\infty, \tag{7}$$

where $v \in (0, 1)$.[1]

To ensure the convergence, [8] conducts line search after obtaining an optimal solution $\boldsymbol{d}$ of (6). A step size $\lambda \in \{\beta^i \mid i = 0, 1, \dots\}$ is found such that $\lambda\boldsymbol{d}$ satisfies the following sufficient decrease condition:

$$\begin{aligned} &f(\boldsymbol{w}^k + \lambda\boldsymbol{d}) - f(\boldsymbol{w}^k) \\ \le &\sigma\lambda\big(\nabla L(\boldsymbol{w}^k)^T\boldsymbol{d} + \gamma\boldsymbol{d}^T H^k \boldsymbol{d} + \|\boldsymbol{w}^k + \boldsymbol{d}\|_1 - \|\boldsymbol{w}^k\|_1\big), \end{aligned} \tag{8}$$

where $0 < \beta < 1$, $0 < \sigma < 1$, and $0 \le \gamma < 1$. The overall procedure is in Algorithm 1. Under some assumptions, [9] proved the asymptotic convergence and the local linear convergence of Algorithm 1.

In the rest of this section, we discuss two examples of Algorithm 1. One is a CD method recommended by [7] and the other is GLMNET [1]. The latter is what we would like to improve upon in this paper, while the former is used for comparison.

2.1 Size of Working Set is One: Cyclic Coordinate Descent Method

This method cyclically selects a singleton working set $J = \{j\}$, where $j = 1, \dots, n$. We describe the specific version in [7]. The sub-problem (6) has only one variable:

$$\min_z \quad q_k(z\boldsymbol{e}_j), \tag{9}$$

[1] We explain why (7) indirectly uses gradient information. If the 1-norm term is not considered and H^k is diagonal, then $\boldsymbol{d}(\mathcal{N})$ is a "scaled" gradient with $d_j(\mathcal{N}) = -\nabla_j L(\boldsymbol{w}^k)/H_{jj}^k$. Thus, (7) selects indices with larger scaled gradient values.

where

$$q_k(z\boldsymbol{e}_j) = \nabla_j L(\boldsymbol{w}^k)z + \frac{1}{2}H^k_{jj}z^2 + |w^k_j + z| - |w^k_j| \quad (10)$$

and $\boldsymbol{e}_j = [\underbrace{0,\dots,0}_{j-1}, 1, 0, \dots, 0]^T$ is an indicator vector.

Because of only one variable in (9), [7] considers the exact Hessian matrix to have $H^k_{jj} = \nabla^2_{jj}L(\boldsymbol{w}^k)$. It is well known that (9) has a simple closed-form solution

$$d = \begin{cases} -\frac{\nabla_j L(\boldsymbol{w}^k)+1}{H^k_{jj}} & \text{if } \nabla_j L(\boldsymbol{w}^k) + 1 \le H^k_{jj} w^k_j, \\ -\frac{\nabla_j L(\boldsymbol{w}^k)-1}{H^k_{jj}} & \text{if } \nabla_j L(\boldsymbol{w}^k) - 1 \ge H^k_{jj} w^k_j, \\ -w^k_j & \text{otherwise.} \end{cases} \quad (11)$$

Because (11) gives a Newton direction, [7] refers to this setting as CDN (CD method using one-dimensional Newton directions). By setting $\gamma = 0$, the sufficient decrease condition (8) in the line search procedure is reduced to

$$f(\boldsymbol{w}^k + \lambda d\boldsymbol{e}_j) - f(\boldsymbol{w}^k) \le \sigma\lambda\left(\nabla_j L(\boldsymbol{w}^k)d + |w^k_j + d| - |w^k_j|\right). \quad (12)$$

Because CDN is a special case of Algorithm 1, any limit point of the sequence $\{\boldsymbol{w}^1, \boldsymbol{w}^2, \dots\}$ is an optimum of (1).

We discuss the computational complexity of CDN. While solving (9) by (11) takes a constant number of operations, calculating $\nabla_j L(\boldsymbol{w}^k)$ and $\nabla^2_{jj}L(\boldsymbol{w}^k)$ for constructing the sub-problem (9) is expensive. From (2),we need $O(nl)$ operations for obtaining $\boldsymbol{w}^T\boldsymbol{x}_i, \forall i$. A common trick to make CD methods viable for classification problems is to store and maintain $\boldsymbol{w}^T\boldsymbol{x}_i, \forall i$. [7] stores $e^{\boldsymbol{w}^T\boldsymbol{x}_i}$ instead and updates the values by

$$e^{\boldsymbol{w}^T\boldsymbol{x}_i} \leftarrow e^{\boldsymbol{w}^T\boldsymbol{x}_i} \cdot e^{\bar{\lambda}dx_{ij}}, \forall i, \quad (13)$$

where $\bar{\lambda}$ is the step size decided by the line search procedure and d is the optimal solution of (9). If $e^{\boldsymbol{w}^T\boldsymbol{x}_i}, \forall i$ are available, the evaluation of $\nabla_j L(\boldsymbol{w}^k)$ and $\nabla^2_{jj}L(\boldsymbol{w}^k)$ in (10), and $f(\boldsymbol{w}^k + \lambda d\boldsymbol{e}_j)$ in (12) takes $O(l)$ operations. Therefore, the complexity of one cycle of n CD steps is:

$$n \times (1+ \text{ \# steps of line search}) \times O(l). \quad (14)$$

We call the n CD steps to update $w_1, \dots, w_n$ as a cycle, which will be frequently used in our subsequent analysis and experiments.

We discuss two implementation techniques in [7]. A conventional CD method sequentially minimizes $q_k(z\boldsymbol{e}_j)$ in the order of $\{1, \dots, n\}$. Many past works have indicated that solving n sub-problems in a random order leads to faster convergence. Thus, [7] uses a randomly permuted sequence $\pi(1), \dots, \pi(n)$ as the order of solving n sub-problems.

The second technique is shrinking. Because problem (1) has a sparse solution, [7] heuristically removes some zero elements during the optimization procedure. They use the following property to conjecture zero elements in an optimal solution: From (3), if $\boldsymbol{w}^*$ is optimal for (1), then

$$-1 < \nabla_j L(\boldsymbol{w}^*) < 1 \quad \text{implies} \quad w^*_j = 0. \quad (15)$$

Assume the previous CD cycle contains iterates $\boldsymbol{w}^{k-|\bar{J}|+1}$, $\dots$, $\boldsymbol{w}^k$, where elements in $\bar{J} = \{\bar{j}_1, \dots, \bar{j}_{|\bar{J}|}\}$ were updated. Then, $\bar{J}$ also corresponds to variables remained in the beginning of the current cycle. Sequentially, $j \in \bar{J}$ is checked at the current cycle. It is removed if

$$w^{k+t-1}_j = 0 \quad \text{and} \quad -1 + \frac{M}{l} < \nabla_j L(\boldsymbol{w}^{k+t-1}) < 1 - \frac{M}{l}, \quad (16)$$

where t is the iteration index for the current cycle and

$$M \equiv \max\left(\left|\nabla^S_{\bar{j}_1} f(\boldsymbol{w}^{k-|\bar{J}|})\right|, \dots, \left|\nabla^S_{\bar{j}_{|\bar{J}|}} f(\boldsymbol{w}^{k-1})\right|\right). \quad (17)$$

If (16) does not hold, then a CD iteration is conducted to update $\boldsymbol{w}$. After all elements in $\bar{J}$ have been processed, a smaller subset J is obtained and passed to the next cycle. Note that in (16), we have $t = 1, \dots, |J|$ instead of $1, \dots, |\bar{J}|$ because a new iterate is generated only if j remains.

Because (15) holds only for an optimal solution, in (16), an interval slightly smaller than $(-1, 1)$ is used. Further, this interval is adjusted according to M, which uses the minimum-norm sub-gradient to measure the maximum violation of the optimality condition in the previous cycle. The calculation of M is in a sequential manner because for updating w_j, we only calculate $\nabla_j L(\boldsymbol{w})$ and can easily obtain $\nabla^S_j f(\boldsymbol{w})$. We never need the whole vector $\nabla L(\boldsymbol{w})$.

2.2 Size of Working Set is Full

An extreme setting of Tseng and Yun's framework [8] is to update all variables at every iteration. That is, we consider a working set $J = \mathcal{N} = \{1, \dots, n\}$. Because (7) holds if $J = \mathcal{N}$, this selection is a special Gauss-Southwell rule. The method GLMNET by Friedman et al. [1] can be considered as an example. GLMNET supports different choices of H^k, but here we discuss the setting of using $H^k = \nabla^2 L(\boldsymbol{w}^k)$. Then, (6) becomes the second-order approximation of (5) and a Newton direction is obtained.

For the sake of computational efficiency, GLMNET does not conduct line search after solving (6). Therefore, seriously speaking it is not an example of Algorithm 1. Also, the convergence is in question. In Section 3, we will add line search back and conduct proper changes to ensure the convergence. Further, we experimentally show in Section 4 that these changes do not cause any inefficiency.

Many optimization methods can be applied to solve the sub-problem (6). Friedman et al. [1] consider a cyclic coordinate descent method, so the procedure for solving sub-problem (6) is almost the same as CDN in Section 2.1. Indeed, it is simpler because (6) is a quadratic problem. Sequentially, $\boldsymbol{d}$'s values are updated by minimizing the following one-variable function:

$$\begin{aligned} &q_k(\boldsymbol{d} + z\boldsymbol{e}_j) - q_k(\boldsymbol{d}) \quad (18)\\ =&|w^k_j + d_j + z| - |w^k_j + d_j| + \nabla_j \bar{q}_k(\boldsymbol{d})z + \frac{1}{2}\nabla^2_{jj}\bar{q}_k(\boldsymbol{d})z^2, \end{aligned}$$

where

$$\bar{q}_k(\boldsymbol{d}) \equiv \nabla L(\boldsymbol{w}^k)^T\boldsymbol{d} + \frac{1}{2}\boldsymbol{d}^T H^k \boldsymbol{d}$$

represents the smooth terms of $q_k(\boldsymbol{d})$ and plays a similar role to $L(\boldsymbol{w})$ for (1). We have

$$\nabla_j \bar{q}_k(\boldsymbol{d}) = \nabla_j L(\boldsymbol{w}^k) + (\nabla^2 L(\boldsymbol{w}^k)\boldsymbol{d})_j, \nabla^2_{jj}\bar{q}_k(\boldsymbol{d}) = \nabla^2_{jj}L(\boldsymbol{w}^k). \quad (19)$$

Since (18) is in the same form as (10), it can be easily solved by (11). Regarding the line search required in Algorithm 1, because (18) is exactly solved, the sufficient decrease condi-

tion (8) holds with $\lambda = 1$ and $\gamma = 1/2$.[2] Thus, line search is not needed.

Because an iterative procedure (CD method) is used to solve the sub-problem (6), GLMNET contains two levels of iterations. An "outer iteration" corresponds to the process from $\boldsymbol{w}^k$ to $\boldsymbol{w}^{k+1}$, while the "inner" level consists of CD iterations for solving (6). For an algorithm involving two levels of iterations, the stopping condition of the inner iterations must be carefully designed. A strict inner stopping condition may cause the algorithm to take a prohibitive amount of time at the first outer iteration. Alternatively, a loose inner condition leads to an inaccurate solution of (6) and possibly lengthy outer iterations. GLMNET terminates the CD procedure by checking if $\boldsymbol{d}$ is still significantly changed. That is, in a cycle of n CD steps to update $d_1, \ldots, d_n$, the corresponding changes $z_1, \ldots, z_n$ satisfy

$$\max_j (\nabla^2_{jj} L(\boldsymbol{w}^k) \cdot z_j^2) \le \epsilon_{\text{in}}, \tag{20}$$

where ϵ_{in} is the inner stopping tolerance. For the outer stopping condition, similarly, GLMNET checks if $\boldsymbol{w}$ is still significantly changed. Let $\boldsymbol{w}^{k+1} = \boldsymbol{w}^k + \boldsymbol{d}^k$. GLMNET stops if the following condition holds:

$$\max_j (\nabla^2_{jj} L(\boldsymbol{w}^{k+1}) \cdot (d_j^k)^2) \le \epsilon_{\text{out}}, \tag{21}$$

where ϵ_{out} is the outer stopping tolerance. GLMNET uses the same value for inner and outer tolerances; i.e., $\epsilon_{\text{in}} = \epsilon_{\text{out}}$. We will propose better settings in Section 3 and show experiments in Section 4.

We discuss the computational complexity of GLMNET. At each CD step, most operations are spent on calculating $\nabla_j \bar{q}_k(\boldsymbol{d})$ and $\nabla^2_{jj} \bar{q}_k(\boldsymbol{d})$ in (19). Note that $\nabla^2_{jj} \bar{q}_k(\boldsymbol{d}) = \nabla^2_{jj} L(\boldsymbol{w}^k), \forall j$ can be pre-calculated before the CD procedure. For $\nabla_j \bar{q}_k(\boldsymbol{d})$, from (2),

$$(\nabla^2 L(\boldsymbol{w}^k)\boldsymbol{d})_j = C \sum_{t=1}^{n} \sum_{i=1}^{l} X_{ji}^T D_{ii} X_{it} d_t = C \sum_{i=1}^{l} X_{ji}^T D_{ii} (X\boldsymbol{d})_i.$$

Thus, if $X\boldsymbol{d}$ (i.e., $\boldsymbol{x}_i^T \boldsymbol{d}, \forall i$) is maintained and updated by

$$(Xd)_i \leftarrow (Xd)_i + X_{ij} z, \ \forall i,$$

then calculating $\nabla_j \bar{q}_k(\boldsymbol{d})$ costs $O(l)$ operations.[3] Therefore, the CD method for (6) requires

$$O(nl) \text{ operations for one cycle of } n \text{ CD steps.} \tag{22}$$

GLMNET applies a shrinking technique to obtain a smaller sub-problem (6). That is, some $\boldsymbol{w}$'s elements are conjectured to be already optimal, so the working set J becomes only a subset of $\mathcal{N}$. At the kth iteration, GLMNET conducts a loop to sequentially solve some smaller sub-problems:

While (`TRUE`)

- Conduct one cycle of n CD steps. Let J include those d_j's which still need to be changed.
- If (20) holds, then `break`.
- Use CD to solve a sub-problem with the working set J.

The working set J chosen in the above procedure excludes the following elements satisfying the optimality conditions of $\min_z q_k(\boldsymbol{d} - z\boldsymbol{e}_j) - q_k(\boldsymbol{d})$:

$$\nabla_j \bar{q}_k(\boldsymbol{d}) + 1 = 0 \quad \text{if } w_j^k + d_j > 0, \tag{23}$$

$$\nabla_j \bar{q}_k(\boldsymbol{d}) - 1 = 0 \quad \text{if } w_j^k + d_j < 0, \tag{24}$$

$$-1 \le \nabla_j \bar{q}_k(\boldsymbol{d}) \le 1 \quad \text{if } w_j^k + d_j = 0. \tag{25}$$

This shrinking method differs from (16) of CDN in several aspects. First, (16) shrinks only zero elements. Second, while (25) is similar to (16), CDN uses an interval slightly smaller than $(-1, 1)$. Thus, (23)–(25) are more aggressive in removing variables. In Section 3.5, we will propose a setting similar to (16) for GLMNET.

3. AN IMPROVED GLMNET: NEWGLMNET

As discussed in Section 2.2, GLMNET lacks theoretical convergence properties. In this section, we propose an improved GLMNET to have the asymptotic convergence. Moreover, we carefully address some implementation concerns. We refer to the improved algorithm as newGLMNET.

3.1 Positive Definite H^k

The convergence proof in [8] requires that H^k in (6) is positive definite. However, from (2), $H^k = \nabla^2 L(\boldsymbol{w}^k)$ in GLMNET is only positive semi-definite. For example, if X has dependent columns, then $\nabla^2 L(\boldsymbol{w}^k)$ is not positive definite. To make GLMNET a case in the framework [8], we slightly enlarge the diagonal elements of $\nabla^2 L(\boldsymbol{w}^k)$:

$$H^k \equiv \nabla^2 L(\boldsymbol{w}^k) + \nu \mathcal{I}, \tag{26}$$

where $\nu > 0$ is a small positive value and $\mathcal{I} \in \mathbf{R}^{n \times n}$ is an identity matrix.

3.2 Line Search and Asymptotic Convergence

For computational efficiency, GLMNET omits the line search procedure after obtaining the Newton direction. Thus, the function value may not be decreasing. To use the convergence result in [8], we include the line search procedure. Experiments in Section 4 show that in almost all cases, $\lambda = 1$ satisfies the sufficient decrease condition (8). Thus, the line search procedure does not introduce extra cost. We prove that all conditions needed in [8] are satisfied, so any limit point of $\{\boldsymbol{w}^k\}$ generated by newGLMNET is an optimal solution of (1). The proof is omitted due to space limit.

3.3 CD Method for Solving Sub-problem (6)

As shown in Section 2.2, the CD method for the quadratic problem (6) is simpler than CDN for (1) because each one-variable function can be exactly minimized and line search is not needed. Thus, following an analysis similar to that in Section 3.2, any limit point of the CD procedure is an optimum of (6).

The similarity to CDN implies that two implementation tricks discussed in Section 2.1 can be applied. We discuss the shrinking implementation in detail in Section 3.5.

3.4 An Adaptive Inner Stopping Condition

We mentioned in Section 2.2 that the inner tolerance must be carefully decided. For example, under GLMNET's current setting of $\epsilon_{\text{in}} = \epsilon_{\text{out}}$, if users specify a small ϵ_{out}, a huge amount of time is needed for the first outer iteration.

For newGLMNET, we propose an adaptive inner stopping condition. The design principle is that in the early stage,

[2] Now $q_k(\boldsymbol{d})$ is minimized, so in (8) $f(\cdot)$ is replaced by $q_k(\cdot)$ and $L(\cdot)$ is replaced by $\bar{q}_k(\cdot)$.

[3] This is like how $e^{\boldsymbol{w}^T \boldsymbol{x}_i}, \ \forall i$ are handled in Section 2.1.

newGLMNET should behave like CDN to quickly obtain a reasonable model, while in the final stage, newGLMNET should achieve fast local convergence by using Newton-like directions. In a cycle of updating $\boldsymbol{d}$'s n variables, we assume that $\boldsymbol{d}^1, \ldots, \boldsymbol{d}^n$ are sequentially generated and from $\boldsymbol{d}^j$ to $\boldsymbol{d}^{j+1}$, the jth element is changed. Then, we propose the following inner stopping condition by checking the minimum-norm sub-gradient:

$$\sum_{j=1}^{n} |\nabla_j^S q_k(\boldsymbol{d}^j)| \le \epsilon_{\text{in}}. \tag{27}$$

Note that we do not need to calculate the whole $\nabla^S q_k(\boldsymbol{d}^j)$. Instead, $\nabla_j^S q_k(\boldsymbol{d}^j)$ is easily available via $\nabla_j \bar{q}_k(\boldsymbol{d}^j)$ in (19).

If at one outer iteration, the condition (27) holds after only one cycle of n CD steps, then we reduce ϵ_{in} by

$$\epsilon_{\text{in}} \leftarrow \epsilon_{\text{in}}/4. \tag{28}$$

Therefore, we can choose a large ϵ_{in} in the beginning. The program automatically adjusts ϵ_{in} if it finds that too few CD steps are conducted for minimizing $q_k(\boldsymbol{d})$.

We use an outer stopping condition similar to (27):

$$\sum_{j=1}^{n} |\nabla_j^S f(\boldsymbol{w}^k)| \le \epsilon_{\text{out}}. \tag{29}$$

3.5 A Two-level Shrinking Scheme

Sections 2.1 and 2.2 have discussed shrinking schemes for CDN and GLMNET, respectively. Following the nature of newGLMNET, we propose a two-level shrinking scheme: the outer level removes some $\boldsymbol{w}$'s elements so that a smaller sub-problem (6) is solved; the inner level is applied to remove elements in $\boldsymbol{d}$ so that (6) becomes an even smaller problem.

In the beginning of each outer iteration, we remove w_j if

$$w_j^k = 0 \quad \text{and} \quad -1 + \frac{M^{\text{out}}}{l} < \nabla_j L(\boldsymbol{w}^k) < 1 - \frac{M^{\text{out}}}{l},$$

where

$$M^{\text{out}} \equiv \max\left(\left|\nabla_1^S f(\boldsymbol{w}^{k-1})\right|, \ldots, \left|\nabla_n^S f(\boldsymbol{w}^{k-1})\right|\right).$$

This setting is different from (23)–(25) in several aspects. First, because we must calculate $\nabla_j L(\boldsymbol{w}^k)$ in the beginning of the kth iteration, we do not need a special cycle of n CD steps in GLMNET. Note that $\nabla^S f(\boldsymbol{w}^k)$ can be easily obtained via $\nabla L(\boldsymbol{w}^k)$ and then used for M^{out} of the next iteration.[4] Second, like (17), M^{out} is used to adjust the shrinking scheme from a conservative setting in the beginning to an aggressive setting in the end.

As indicated in Section 2.2, the CD procedure in each GLMNET's outer iteration for solving (6) is very similar to CDN for (1). Thus, we can apply CDN's shrinking scheme to the inner loop of newGLMNET. Assume the previous cycle of CD steps contains points $\boldsymbol{d}^{r-|\bar{J}|+1}, \ldots, \boldsymbol{d}^r$, where elements in the set $\bar{J} = \{\bar{j}_1, \ldots, \bar{j}_{|\bar{J}|}\}$ were updated. Then, $\bar{J}$ corresponds to variables remained in the beginning of the current cycle. Sequentially, $j \in \bar{J}$ is checked at the current cycle. It is removed if

$$w_j^k + d_j^{r+t-1} = 0 \quad \text{and} \quad -1 + \frac{M^{\text{in}}}{l} < \nabla_j \bar{q}_k(\boldsymbol{d}^{r+t-1}) < 1 - \frac{M^{\text{in}}}{l}, \tag{30}$$

[4] If $k = 1$, $\nabla^S f(\boldsymbol{w}^{k-1})$ is not available. We set $M^{\text{out}} = \infty$, so no variables are shrunk at the first outer iteration.

Algorithm 2 newGLMNET with two-level shrinking

1. Given $\boldsymbol{w}^1$, ϵ_{in}, and ϵ_{out}. Choose a small positive number ν. Choose $\beta \in (0,1)$, $\gamma \in [0,1)$, and $\sigma \in (0,1)$.
2. Let $M^{\text{out}} \leftarrow \infty$.
3. For $k = 1, 2, 3, \ldots$
   - Let $J \leftarrow \{1, \ldots, n\}$, $M \leftarrow 0$, and $\bar{M} \leftarrow 0$.
   - For $j = 1, \ldots, n$
     - Calculate H_{jj}^k, $\nabla_j L(\boldsymbol{w}^k)$ and $\nabla_j^S f(\boldsymbol{w}^k)$.
     - If $w_j^k = 0$ (outer-level shrinking)
       - If $|\nabla_j L(\boldsymbol{w}^k)| < 1 - M^{\text{out}}/l$
         $J \leftarrow J \backslash \{j\}$.
     - $M \leftarrow \max(M, |\nabla_j^S f(\boldsymbol{w}^k)|)$.
     - $\bar{M} \leftarrow \bar{M} + |\nabla_j^S f(\boldsymbol{w}^k)|$.
   - If $\bar{M} \le \epsilon_{\text{out}}$
     `return` $\boldsymbol{w}^k$.
   - Let $M^{\text{out}} \leftarrow M$ and $M^{\text{in}} \leftarrow \infty$.
   - Let $T \leftarrow J$ and $\boldsymbol{d} \leftarrow \boldsymbol{0}$.
   - For $p = 1, 2, 3, \ldots, 1000$
     - Let $m \leftarrow 0$ and $\bar{m} \leftarrow 0$.
     - For $j \in T$
       - Let $\nabla_{jj}^2 \bar{q}_k(\boldsymbol{d}) = H_{jj}^k$. Calculate $\nabla_j \bar{q}_k(\boldsymbol{d})$ and $\nabla_j^S q_k(\boldsymbol{d})$.
       - If $w_j^k + d_j = 0$ (inner-level shrinking)
         - If $|\nabla_j \bar{q}(\boldsymbol{d})| < 1 - M^{\text{in}}/l$
           $T \leftarrow T \backslash \{j\}$.
       - Else
         - $m \leftarrow \max(m, |\nabla_j^S q(\boldsymbol{d})|)$.
         - $\bar{m} \leftarrow \bar{m} + |\nabla_j^S q(\boldsymbol{d})|$.
         - $d_j \leftarrow d_j + \arg\min_z q(\boldsymbol{d} + z\boldsymbol{e}_j) - q(\boldsymbol{d})$.
     - If $\bar{m} \le \epsilon_{\text{in}}$
       - If $T = J$ (inner stopping)
         `break`.
       - Else (active set reactivation)
         $T \leftarrow J$ and $M^{\text{in}} \leftarrow \infty$.
     - Else
       $M^{\text{in}} \leftarrow m$.
   - If $p = 1$, then $\epsilon_{\text{in}} \leftarrow \epsilon_{\text{in}}/4$.
   - Compute $\lambda = \max\{1, \beta, \beta^2, \ldots\}$ such that $\lambda \boldsymbol{d}$ satisfies (8). $\boldsymbol{w}^{k+1} = \boldsymbol{w}^k + \lambda \boldsymbol{d}$.

where t is the iteration index for the current cycle and

$$M^{\text{in}} \equiv \max\left(\left|\nabla_{\bar{j}_1}^S q_k(\boldsymbol{d}^{r-|\bar{J}|})\right|, \ldots, \left|\nabla_{\bar{j}_{|\bar{J}|}}^S q_k(\boldsymbol{d}^{r-1})\right|\right).$$

If (30) does not hold, element j remains.[5] Then, a regular CD iteration is conducted by solving (18) to obtain $\bar{z}$ and generating $\boldsymbol{d}^{r+t} = \boldsymbol{d}^{r+t-1} + \bar{z}\boldsymbol{e}_j$. After the whole set $\bar{J}$ has been processed, a smaller subset J is obtained and we move to the next cycle.[6] The overall procedure of newGLMNET with two-level shrinking is shown in Algorithm 2.

3.6 Analysis on the Number of Exponential and Logarithmic Operations

Algorithms for logistic regression involve exponential and logarithmic operations, each of which is much more expensive than one multiplication/division. It is important to

[5] Note that in (30), $t = 1, \ldots, |J|$ instead of $1, \ldots, |\bar{J}|$; see the explanation in Section 2.1.

[6] Similar to the way to initialize M^{out}, for the first CD cycle, we set $M^{\text{in}} = \infty$.

Table 1: Number of exp/log operations in a cycle of n coordinate descent steps.

One cycle of n CD steps	CDN	newGLMNET
Total # of operations (dense data)	$O(nl)$	$O(nl)$
# exp/log (dense data)	$O(nl)$	$\leq O(l)$
Total # of operations (sparse data)	$O(\text{nnz})$	$O(\text{nnz})$
# of exp/log (sparse data)	$O(\text{nnz})$	$\leq O(l)$

check how many exp/log operations are required in an algorithm because a high proportion of exp/log operations may substantially influence the performance. We analyze both CDN and newGLMNET by checking each cycle of n CD steps. We make an assumption that the number of line search steps in CDN is small (e.g., one or two). Table 1 shows the number of total operations and the number of exp/log operations. From (14) and (22), CDN and newGLMNET both require $O(nl)$ total operations in a cycle. However, their numbers of exp/log operations are very different. In CDN, because $e^{\boldsymbol{w}^T\boldsymbol{x}_i}$ $\forall i$ must be updated in (13), $O(l)$ exp operations are needed at each CD iteration. In line search, calculating $f(\boldsymbol{w}^k + \lambda d\boldsymbol{e}_j)$ in (12) takes $O(l)$ log operations. Thus, $O(nl)$ exp/log operations are needed in one cycle. In contrast, because $q_k(\boldsymbol{d})$ in (6) has no exp/log terms, the CD procedure in newGLMNET does not require any exp/log operations. To construct (6) in the beginning of each outer iteration, from (2), $O(l)$ exp operations are required. Because several cycles are used to solve (6), we can say that each cycle shares no more than $O(l)$ exp operations.

Therefore, CDN's $O(nl)$ exp/log operations are much more than newGLMNET's $O(l)$. The difference becomes smaller for sparse data because CDN's $O(nl)$ is replaced by $O(\text{nnz})$, where nnz is the total number of non-zero elements in X (i.e., training data). From this analysis, we expect that CDN suffers from many slow exp/log operations if data are dense and n is not small. This result will be clearly observed in Section 4. A related discussion on exp/log operations for L2-regularized problems is in [10].

Besides, newGLMNET is also superior to CDN in terms of the cost of line search. More analysis appears in the long version of the paper [11, Section 4].

4. EXPERIMENTS

In this section, we investigate the performance of CDN, GLMNET, and newGLMNET. All these methods can be easily extended to solve logistic regression with a bias term:

$$\min_{\boldsymbol{w},b} \quad \|\boldsymbol{w}\|_1 + C\sum_{i=1}^{l} \log(1 + e^{-y_i(\boldsymbol{w}^T\boldsymbol{x}_i+b)}). \qquad (31)$$

Because the GLMNET implementation solves (31) instead of (1), in our comparison, (31) is used. We do not consider other methods because in [7], CDN is shown to be the best for sparse data.

4.1 Data Sets and Experimental Settings

We use eight data sets in our experiments. Five of them (news20, rcv1, yahoo-japan, yahoo-korea, and webspam) are document data sets. The other three data sets come from different learning problems: gisette is a handwriting digit recognition problem from NIPS 2003 feature selection challenge; epsilon is an artificial data set for Pascal large scale learning challenge in 2008; KDD2010-b includes student performance prediction data for a tutoring system and is used for the data mining competition KDD Cup 2010. All document data sets are instance-wisely normalized, so the length of each instance is one. For non-document data, gisette is feature-wisely scaled to the $[-1, 1]$ interval, and epsilon is feature-wisely scaled to $N(1, 0)$ and then instance-wisely normalized. KDD2010-b is preprocessed in [12].

We prepare training and testing sets for each problem. For gisette and KDD2010-b, we use their original training and test sets. For others, we randomly split data to one fifth for testing and the remaining for training.

We choose the parameter C in (31) by five-fold cross validation on the training set. All methods then solve (31) with the best C to obtain the model for prediction. Table 2 shows the statistics and the best C of all data sets. We can clearly see that two data sets (epsilon and gisette) are very dense, while others are sparse.

Next, we describe software information and parameter settings in our experiments.

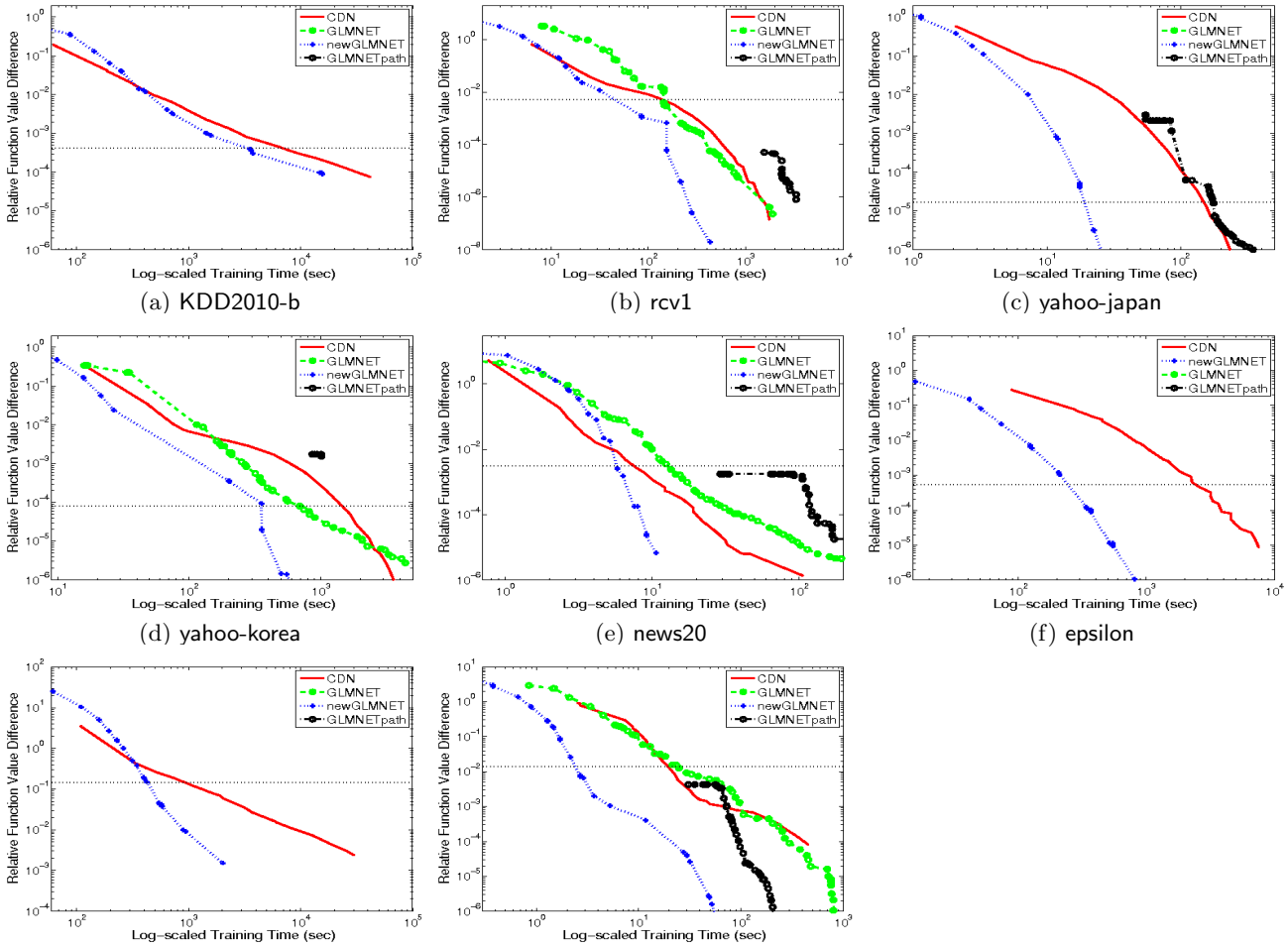
- CDN: this coordinate descent method is described in Section 2.1. In the line search procedure, we use $\sigma = 0.01$ and $\beta = 0.5$. The C/C++ implementation is included in LIBLINEAR (version 1.7), which is available at `http://www.csie.ntu.edu.tw/~cjlin/liblinear/`.
- newGLMNET: an improved GLMNET is described in Section 3. For the positive definiteness of H^k, we set $\nu = 10^{-12}$ in (26). To check the sufficient decrease condition (8), we use $\beta = 0.5$, $\gamma = 0$, and $\sigma = 0.01$. We choose the initial $\epsilon_{\text{in}} = \|\nabla^S f(\boldsymbol{w}^1)\|_1$. newGLMNET is implemented in C/C++.
- GLMNET: this method is described in Section 2.2. GLMNET is implemented in Fortran with an R interface. The source code (version 1.5.3) is available at `http://cran.r-project.org/web/packages/glmnet/`. GLMNET uses the regularization parameter $\lambda = 1/(Cl)$ instead of C. We ensure that the equivalent settings have been made in our experiments.

GLMNET offers an option to find a solution path $\{\boldsymbol{w}^{C_1}, \ldots, \boldsymbol{w}^{C_*}\}$ of an increasing parameter sequence $\{C_1, \ldots, C_*\}$. It applies a warm start technique so that the optimal solution of the previous C_{i-1} is used as the initial point for the current C_i. The number of outer iterations should be small because of a more accurate initial point. GLMNET authors suggest that finding a solution path may be faster than solving a single optimization problem under a fixed C. [1, Section 2.5]. We refer to this approach as GLMNETpath and include it for comparison. By their default setting, we consider a parameter sequence of length 100 starting from the smallest C_1 such that $\boldsymbol{w}^{C_1} = \boldsymbol{0}$. Given our desired parameter C_*, a geometric sequence is generated by a fixed ratio between successive C values..

We set the initial $\boldsymbol{w}^1 = \boldsymbol{0}$ for all methods. All experiments are conducted on a 64-bit machine with Intel Xeon 2.0GHz CPU (E5504), 128KB L1 cache, 1GB L2 cache, and 32GB main memory. We use GNU C/C++/Fortran compilers (version 4.4.1) and the optimization flag is properly set.

4.2 Running Time Comparison

We begin with checking the change of function values along the running time. We show in Figure 1 pairs of (time, function value) by gradually reducing the stopping toler-

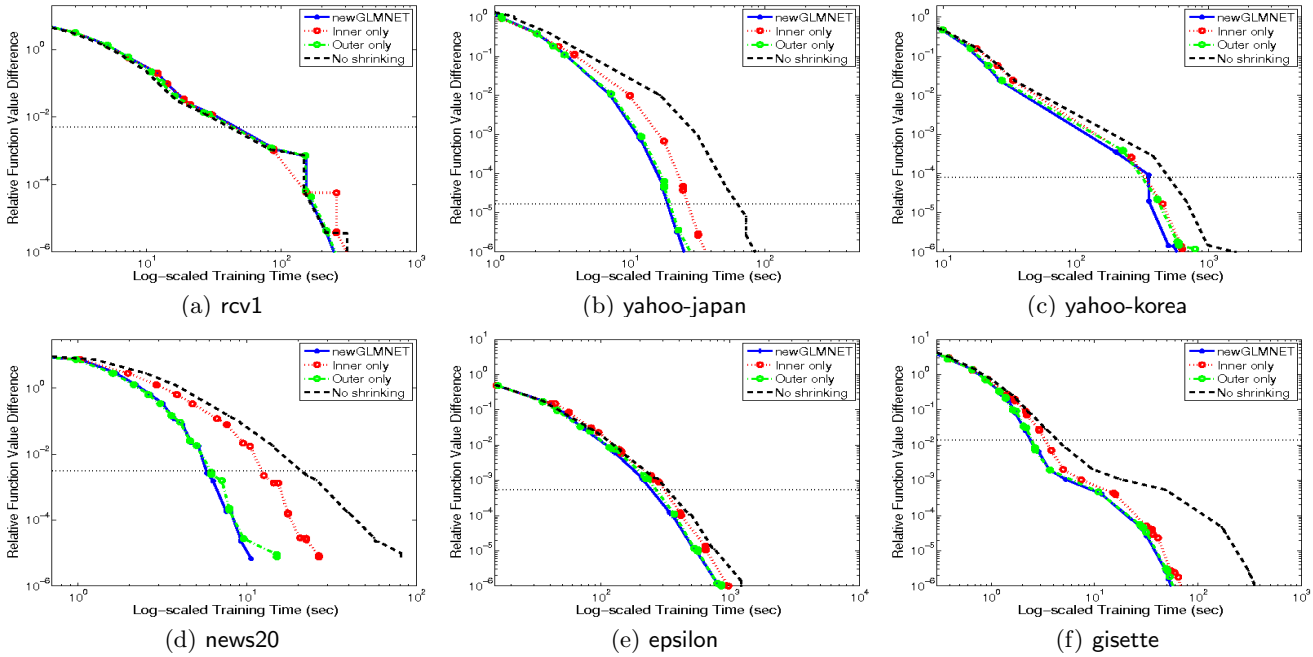


(a) KDD2010-b (b) rcv1 (c) yahoo-japan

(d) yahoo-korea (e) news20 (f) epsilon

(g) webspam (h) gisette

Figure 1: Relative difference to the optimal function value versus training time. Both x-axis and y-axis are log-scaled. GLMNET and GLMNETpath failed to generate some results because of either memory problems or lengthy running time.



(a) rcv1 (b) yahoo-japan (c) yahoo-korea

(d) news20 (e) epsilon (f) gisette

Figure 2: Effect of two-level shrinking. "Inner only" ("Outer only") indicates that only inner-level (outer-level) shrinking is conducted.

Table 2: Data statistics and the best parameter C. Data sets are sorted by the number of nonzero elements per instance in the training data. We conduct five-fold cross validation on the training set to select C in $\{2^k \mid k = -4, -3, \ldots, 6\}$. All data sets except yahoo-japan and yahoo-korea are available at `http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/`.

Data set	#data train	#data test	#features	#nonzeros in training	#nonzeros per instance	C
KDD2010-b	19,264,097	748,401	29,890,095	566,345,888	29	0.5
rcv1	541,920	135,479	47,236	39,625,144	73	4
yahoo-japan	140,963	35,240	832,026	18,738,315	133	4
yahoo-korea	368,444	92,110	3,052,939	125,190,807	340	4
news20	15,997	3,999	1,355,191	7,281,110	455	64
epsilon	400,000	100,000	2,000	800,000,000	2,000	0.5
webspam	280,000	70,000	16,609,143	1,043,724,776	3,727	64
gisette	6,000	1,000	5,000	29,729,997	4,955	0.25

ance.[7] The x-axis is the log-scaled training time and the y-axis is the relative difference to the optimal function value:

$$\frac{f(\boldsymbol{w}) - f^*}{f^*},$$

where $\boldsymbol{w}$ is the solution under the specified tolerance and f^* is the optimal function value. Because f^* is not available, we obtain an approximation by running newGLMNET with a small stopping tolerance

$$\epsilon_{\text{out}} = \epsilon \cdot \min(\#\text{pos}, \#\text{neg})/l \cdot \|\nabla^S f(\boldsymbol{w}^1)\|_1, \qquad (32)$$

where $\epsilon = 10^{-8}$, and #pos and #neg indicate the numbers of positive and negative labels in a data set, respectively. The horizontal dotted line in Figure 1 indicates the relative function difference by running CDN using LIBLINEAR's default stopping tolerance with $\epsilon = 0.01$ in (32). The point where a method's curve passes this horizontal line roughly indicates the time needed to obtain an accurate enough solution.

From Figure 1, if the optimization problem is loosely solved using a large ϵ_{out}, CDN is faster than newGLMNET and GLMNET. This result is reasonable because CDN uses a greedy setting to sequentially update variables. In contract, in each outer iteration, newGLMNET uses only a fixed H^k. If using a smaller ϵ_{out}, newGLMNET surpasses CDN and achieves fast local convergence. For dense data (epsilon and gisette), newGLMNET is always better than CDN. Take epsilon as an example. In Figure 1(f), to reach the horizontal line, newGLMNET is ten times faster than CDN. This huge difference is expected following the analysis on the number of exp/log operations in Section 3.6.

From results above the horizontal lines in Figure 1, we see that newGLMNET is faster than GLMNET in the early stage. Recall that GLMNET sets $\epsilon_{\text{in}} = \epsilon_{\text{out}}$, while newGLMNET uses an adaptive setting to adjust ϵ_{in}. Because a large ϵ_{in} is considered in the beginning, newGLMNET can compete with CDN in the early stage by loosely solving (6). We use an example to further illustrate the importance to properly choose ϵ_{in}. By running GLMNET with the default $\epsilon_{\text{out}} = 10^{-4}$ on news20 and rcv1, the training time is 218.15 and 4171.74 seconds, respectively. The first outer iteration already takes 108.62 seconds on news20 and 656.14 on rcv1. A quick fix is to enlarge the initial ϵ_{in}, but the local convergence in the later stage may be slow. A better inner stopping condition should be adaptive like ours so that the sub-problem (6) can be solved properly at each outer iteration.

In Figure 1, GLMNET and GLMNETpath failed to generate some results because of either memory problems or lengthy running time. This indicates that a careful implementation is very important for large-scale problems. We also observe that GLMNETpath is not faster than GLMNET. Another drawback of GLMNETpath is that it is hard to quickly obtain an approximate solution. That is, regardless of ϵ_{out} specified, a sequence of problems (1) still needs to be solved.

We have checked the relationship between the testing accuracy and the training time. The comparison result, not presented due to space limit, is similar to that in Figure 1.

Recall we added a line search procedure in newGLMNET. We find that the sufficient decrease condition (8) holds for $\lambda = 1$ in all cases. In contrast, CDN spends considerable time on line search.

In summary, because of the proposed adaptive inner stopping condition, newGLMNET takes both advantages of fast approximation in the early stage like CDN and of fast local convergence in the final stage like GLMNET.

4.3 Effect of Exp/log Operations

In Table 2, we sort data sets according to nnz/l (the average number of non-zero values per instance). The purpose is to see the effect of exp/log operations, where CDN needs $O(\text{nnz})$ and GLMNET needs $O(l)$ operations per cycle of n CD steps, respectively. Clearly, for the two dense data (epsilon and gisette), newGLMNET is much faster. Because many other issues affect the running time, we focus on the first CD cycle and present running time in Table 3. Clearly, for dense data, CDN's exp/log operations are much more expensive than other operations, so the total time is longer than that of newGLMNET. This result indicates that CDN is not competitive for data with a large nnz/l.

4.4 Effect of Shrinking

We compare newGLMNET implementations with/without shrinking in Figure 2, which shows that shrinking significantly improves the running time. The outer-level shrinking is particularly useful. Interestingly, from [7, Figure 9] and Figure 2, shrinking is more effective for newGLMNET than CDN.

5. DISCUSSIONS AND CONCLUSIONS

In newGLMNET, a CD method is applied to solve the sub-problem (6). Using the property that CD involves simple

[7]For GLMNET and newGLMNET, the tolerance means the outer tolerance ϵ_{out} in (21). For GLMNETpath, under any given ϵ_{out}, a sequence of problems (1) is solved.

Table 3: Timing analysis of the first cycle of n CD steps. Time is in seconds. (*: dense data)

Data set	CDN		newGLMNET	
	exp/log	Total	exp/log	Total
KDD2010-b	21.72 (30.7%)	70.80	3.88 (6.9%)	56.50
rcv1	4.61 (73.8%)	6.25	0.12 (5.3%)	2.20
yahoo-japan	1.47 (70.9%)	2.08	0.03 (4.2%)	0.71
yahoo-korea	10.65 (66.3%)	16.06	0.08 (1.2%)	6.66
news20	0.21 (27.3%)	0.76	0.003 (0.5%)	0.60
epsilon*	64.98 (73.0%)	89.07	0.09 (0.8%)	11.15
webspam	72.89 (66.6%)	109.39	0.06 (0.1%)	41.10
gisette*	1.66 (66.8%)	2.49	0.002 (0.6%)	0.27

and cheap updates, we carefully adjust the stopping condition for sub-problems. Then, newGLMNET is competitive with a CD method like CDN in the early stage, but becomes a Newton method in the end. This design is similar to "truncated Newton" methods in optimization. While CD seems to be a good choice for solving the sub-problem, whether there are better alternatives is an interesting future issue.

In this work, we not only improve GLMNET's theoretical properties, but also successfully address many implementation issues. The proposed newGLMNET is more stable and efficient for large-scale L1-regularized logistic regression. The implementation is included in LIBLINEAR (version 1.8).

6. REFERENCES

[1] J. Friedman, T. Hastie, and R. Tibshirani, "Regularization paths for generalized linear models via coordinate descent," *Journal of Statistical Software*, vol. 33, no. 1, pp. 1–22, 2010.

[2] A. Genkin, D. D. Lewis, and D. Madigan, "Large-scale Bayesian logistic regression for text categorization," *Technometrics*, vol. 49, no. 3, pp. 291–304, 2007.

[3] K. Koh, S.-J. Kim, and S. Boyd, "An interior-point method for large-scale l1-regularized logistic regression," *Journal of Machine Learning Research*, vol. 8, pp. 1519–1555, 2007.

[4] G. Andrew and J. Gao, "Scalable training of L1-regularized log-linear models," in *Proceedings of the Twenty Fourth International Conference on Machine Learning (ICML)*, 2007.

[5] J. Liu, J. Chen, and J. Ye, "Large-scale sparse logistic regression," in *Proceedings of The 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 547–556, 2009.

[6] M. Schmidt, G. Fung, and R. Rosales, "Fast optimization methods for l1 regularization: A comparative study and two new approaches," in *Proceedings of European Conference on Machine Learning*, pp. 286–297, 2007.

[7] G.-X. Yuan, K.-W. Chang, C.-J. Hsieh, and C.-J. Lin, "A comparison of optimization methods and software for large-scale l1-regularized linear classification," *Journal of Machine Learning Research*, vol. 11, pp. 3183–3234, 2010.

[8] P. Tseng and S. Yun, "A coordinate gradient descent method for nonsmooth separable minimization," *Mathematical Programming*, vol. 117, pp. 387–423, 2009.

[9] S. Yun and K.-C. Toh, "A coordinate gradient descent method for l1-regularized convex minimization," 2009. To appear in Computational Optimizations and Applications.

[10] K.-W. Chang, C.-J. Hsieh, and C.-J. Lin, "Coordinate descent method for large-scale L2-loss linear SVM," *Journal of Machine Learning Research*, vol. 9, pp. 1369–1398, 2008.

[11] G.-X. Yuan, C.-H. Ho, and C.-J. Lin, "An improved GLMNET for l1-regularized logistic regression and support vector machines," tech. rep., National Taiwan University, 2011.

[12] H.-F. Yu, H.-Y. Lo, H.-P. Hsieh, J.-K. Lou, T. G. McKenzie, J.-W. Chou, P.-H. Chung, C.-H. Ho, C.-F. Chang, Y.-H. Wei, J.-Y. Weng, E.-S. Yan, C.-W. Chang, T.-T. Kuo, Y.-C. Lo, P. T. Chang, C. Po, C.-Y. Wang, Y.-H. Huang, C.-W. Hung, Y.-X. Ruan, Y.-S. Lin, S.-D. Lin, H.-T. Lin, and C.-J. Lin, "Feature engineering and classifier ensemble for KDD cup 2010," in *JMLR Workshop and Conference Proceedings*, 2011. To appear.