Journal of Machine Learning Research 9 (2008) 1369-1398 Submitted 1/08; Revised 4/08; Published 7/08

# Coordinate Descent Method for Large-scale L2-loss Linear Support Vector Machines

**Kai-Wei Chang** B92084@CSIE.NTU.EDU.TW
**Cho-Jui Hsieh** B92085@CSIE.NTU.EDU.TW
**Chih-Jen Lin** CJLIN@CSIE.NTU.EDU.TW
*Department of Computer Science*
*National Taiwan University*
*Taipei 106, Taiwan*

**Editor:** Leon Bottou

## Abstract

Linear support vector machines (SVM) are useful for classifying large-scale sparse data. Problems with sparse features are common in applications such as document classification and natural language processing. In this paper, we propose a novel coordinate descent algorithm for training linear SVM with the L2-loss function. At each step, the proposed method minimizes a one-variable sub-problem while fixing other variables. The sub-problem is solved by Newton steps with the line search technique. The procedure globally converges at the linear rate. As each sub-problem involves only values of a corresponding feature, the proposed approach is suitable when accessing a feature is more convenient than accessing an instance. Experiments show that our method is more efficient and stable than state of the art methods such as Pegasos and TRON.

**Keywords:** linear support vector machines, document classification, coordinate descent

## 1. Introduction

Support vector machines (SVM) (Boser et al., 1992) are a popular data classification tool. Given a set of instance-label pairs $(\mathbf{x}_j, y_j), j = 1, \ldots, l,\ \mathbf{x}_j \in R^n,\ y_j \in \{-1, +1\}$, SVM solves the following unconstrained optimization problem:

$$\min_{\mathbf{w}} f(\mathbf{w}) = \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_{j=1}^{l} \xi(\mathbf{w}; \mathbf{x}_j, y_j), \tag{1}$$

where $\xi(\mathbf{w}; \mathbf{x}_j, y_j)$ is a loss function, and $C \in R$ is a penalty parameter. There are two common loss functions. L1-SVM uses the sum of losses and minimizes the following optimization problem:

$$f(\mathbf{w}) = \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_{j=1}^{l} \max(1 - y_j\mathbf{w}^T\mathbf{x}_j, 0), \tag{2}$$

while L2-SVM uses the sum of squared losses, and minimizes

$$f(\mathbf{w}) = \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_{j=1}^{l} \max(1 - y_j\mathbf{w}^T\mathbf{x}_j, 0)^2. \tag{3}$$

SVM is related to regularized logistic regression (LR), which solves the following problem:

$$\min_{\mathbf{w}} f(\mathbf{w}) = \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_{j=1}^{l}\log(1+e^{-y_j\mathbf{w}^T\mathbf{x}_j}). \tag{4}$$

In some applications, we include a bias term $b$ in SVM problems. For convenience, one may extend each instance with an additional dimension to eliminate this term:

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

SVM usually maps training vectors into a high-dimensional (and possibly infinite dimensional) space, and solves the dual problem of (1) with a nonlinear kernel. In some applications, data appear in a rich dimensional feature space, so that with/without nonlinear mapping obtain similar performances. If data are not mapped, we call such cases linear SVM, which are often encountered in applications such as document classification. While one can still solve the dual problem for linear SVM, directly solving (2) or (3) is possible. The objective function of L1-SVM (2) is nondifferentiable, so typical optimization methods cannot be directly applied. In contrast, L2-SVM (3) is a piecewise quadratic and strongly convex function, which is differentiable but not twice differentiable (Mangasarian, 2002). We focus on studying L2-SVM in this paper because of its differentiability.

In recent years, several optimization methods are applied to solve linear SVM in large-scale scenarios. For example, Keerthi and DeCoste (2005); Mangasarian (2002) propose modified Newton methods to train L2-SVM. As (3) is not twice differentiable, to obtain the Newton direction, they use the generalized Hessian matrix (i.e., generalized second derivative). A trust region Newton method (TRON) (Lin et al.) is proposed to solve logistic regression and L2-SVM. For large-scale L1-SVM, SVM$^{\text{perf}}$ (Joachims, 2006) uses a cutting plane technique to obtain the solution of (2). Smola et al. (2008) apply bundle methods, and view SVM$^{\text{perf}}$ as a special case. Zhang (2004) proposes a stochastic gradient method; Pegasos (Shalev-Shwartz et al., 2007) extends Zhang's work and develops an algorithm which alternates between stochastic gradient descent steps and projection steps. The performance is reported to be better than SVM$^{\text{perf}}$. Another stochastic gradient implementation similar to Pegasos is by Bottou (2007). All the above algorithms are iterative procedures, which update $\mathbf{w}$ at each iteration and generate a sequence $\{\mathbf{w}^k\}_{k=0}^{\infty}$. To distinguish these approaches, we consider the two extremes of optimization methods mentioned in the paper (Lin et al.):

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

Among methods discussed above, Pegasos randomly subsamples a few instances at a time, so the cost per iteration is low, but the number of iterations is high. In contrast, Newton methods such as TRON take significant efforts at each iteration, but converge at fast rates. In large-scale scenarios, usually an approximate solution of the optimization problem is enough to produce a good model. Thus, methods with a low-cost iteration may be preferred as they can quickly generate a reasonable model. However, if one specifies an unsuitable stopping condition, such methods may fall into the situation of lengthy iterations. A recent overview on the tradeoff between learning accuracy and optimization cost is by Bottou and Bousquet (2008).

Coordinate descent is a common unconstrained optimization technique, but its use for large linear SVM has not been exploited much.[1] In this paper, we aim at applying it to L2-SVM. A coor-

1. For SVM with kernels, decomposition methods are popular, and they are related to coordinate descent methods. Since we focus on linear SVM, we do not discuss decomposition methods in this paper.

dinate descent method updates one component of $\mathbf{w}$ at a time by solving a one-variable sub-problem. It is competitive if one can exploit efficient ways to solve the sub-problem. For L2-SVM, the sub-problem is to minimize a single-variable piecewise quadratic function, which is differentiable but not twice differentiable. An earlier paper using coordinate descents for L2-SVM is by Zhang and Oles (2001). The algorithm, called CMLS, applies a modified Newton method to approximately solve the one-variable sub-problem. Here, we propose another modified Newton method, which obtains an approximate solution by line searches. Two key properties differentiate our method and CMLS:

1. Our proposed method attempts to use the full Newton step if possible, while CMLS takes a more conservative step. Our setting thus leads to faster convergence.

2. CMLS maintains the strict decrease of the function value, but does not prove the convergence. We prove that our method globally converges to the unique minimum.

We say $\hat{\mathbf{w}}$ is an $\varepsilon$-accurate solution if

$$f(\hat{\mathbf{w}}) \leq \min_{\mathbf{w}} f(\mathbf{w}) + \varepsilon.$$

We prove that our process obtains an $\varepsilon$-accurate solution in $O\left(nC^3P^6(\#\text{nz})^3\log(1/\varepsilon)\right)$ iterations, where the definitions of #nz and $P$ can be found in the end of this section. Experiments show that our proposed method is more efficient and stable than existing algorithms.

Subsequent to this work, we and some collaborators propose a dual coordinate descent method for linear SVM (Hsieh et al., 2008). The method performs very well on document data (generally better than the primal-based method here). However, the dual method is not be stable for some non-document data with a small number of features. Clearly, if the number of features is much smaller than the number of instances, one should solve the primal form, which has less variables. In addition, the primal method uses the column format to store data (see Section 3.1). It is thus suitable for data stored as some form of inverted index in a very large database.

The organization of this paper is as follows. In Section 2, we describe and analyze our algorithm. Several implementation issues are discussed in Section 3. In Sections 4 and 5, we describe existing methods such as Pegasos, TRON and CMLS, and compare them with our approach. Results show that the proposed method is efficient and stable. Finally, we give discussions and conclusions in Section 6.

All sources used in this paper are available at
`http://www.csie.ntu.edu.tw/~cjlin/liblinear/exp.html`.

**Notation** The following notations are used in this paper. The input vectors are $\{\mathbf{x}_j\}_{j=1,\ldots,l}$, and $x_{ji}$ is the $i$th feature of $\mathbf{x}_j$. For the problem size, $l$ is the number of instances, $n$ is number of features, and #nz is total number of nonzero values of training data.

$$m = \frac{\#\text{nz}}{n} \tag{5}$$

is the average number of nonzero values per feature, and

$$P = \max_{ji} |x_{ji}| \tag{6}$$

represents the upper bound of $x_{ji}$. We use $\|\cdot\|$ to represent the 2-norm of a vector.

**Algorithm 1** Coordinate descent algorithm for L2-SVM

1. Start with any initial $\mathbf{w}^0$.
2. For $k = 0, 1, \ldots$ (outer iterations)
   (a) For $i = 1, 2, \ldots, n$ (inter iterations)
      i. Fix $w_1^{k+1}, \ldots, w_{i-1}^{k+1}, w_{i+1}^k, \ldots, w_n^k$ and approximately solve the sub-problem (7) to obtain $w_i^{k+1}$.

## 2. Solving Linear SVM via Coordinate Descent

In this section, we describe our coordinate descent method for solving L2-SVM given in (3). The algorithm starts from an initial point $\mathbf{w}^0$, and produces a sequence $\{\mathbf{w}^k\}_{k=0}^{\infty}$. At each iteration, $\mathbf{w}^{k+1}$ is constructed by sequentially updating each component of $\mathbf{w}^k$. This process generates vectors $\mathbf{w}^{k,i} \in R^n$, $i = 1, \ldots, n$, such that $\mathbf{w}^{k,1} = \mathbf{w}^k$, $\mathbf{w}^{k,n+1} = \mathbf{w}^{k+1}$, and

$$\mathbf{w}^{k,i} = [w_1^{k+1}, \ldots, w_{i-1}^{k+1}, w_i^k, \ldots, w_n^k]^T \text{ for } i = 2, \ldots, n.$$

For updating $\mathbf{w}^{k,i}$ to $\mathbf{w}^{k,i+1}$, we solve the following one-variable sub-problem:

$$\begin{aligned} &\min_z f(w_1^{k+1}, \ldots, w_{i-1}^{k+1}, w_i^k + z, w_{i+1}^k, \ldots, w_n^k) \\ \equiv &\min_z f(\mathbf{w}^{k,i} + z\mathbf{e}_i), \end{aligned} \tag{7}$$

where $\mathbf{e}_i = [\underbrace{0, \ldots, 0}_{i-1}, 1, 0, \ldots, 0]^T$. A description of the coordinate descent algorithm is in Algorithm 1. The function in (7) can be rewritten as

$$\begin{aligned} D_i(z) &= f(\mathbf{w}^{k,i} + z\mathbf{e}_i) \\ &= \frac{1}{2}(\mathbf{w}^{k,i} + z\mathbf{e}_i)^T(\mathbf{w}^{k,i} + z\mathbf{e}_i) + C \sum_{j \in I(\mathbf{w}^{k,i} + z\mathbf{e}_i)} (b_j(\mathbf{w}^{k,i} + z\mathbf{e}_i))^2, \end{aligned} \tag{8}$$

where

$$b_j(\mathbf{w}) = 1 - y_j \mathbf{w}^T \mathbf{x}_j \text{ and } I(\mathbf{w}) = \{j \mid b_j(\mathbf{w}) > 0\}.$$

In any interval of $z$ where the set $I(\mathbf{w}^{k,i} + z\mathbf{e}_i)$ does not change, $D_i(z)$ is quadratic. Therefore, $D_i(z), z \in R$, is a piecewise quadratic function. As Newton method is suitable for quadratic optimization, here we apply it for minimizing $D_i(z)$. If $D_i(z)$ is twice differentiable, then the Newton direction at a given $\bar{z}$ would be

$$\frac{-D_i'(\bar{z})}{D_i''(\bar{z})}.$$

The first derivative of $D_i(z)$ is:

$$D_i'(z) = w_i^{k,i} + z - 2C \sum_{j \in I(\mathbf{w}^{k,i} + z\mathbf{e}_i)} y_j x_{ji} (b_j(\mathbf{w}^{k,i} + z\mathbf{e}_i)). \tag{9}$$

Unfortunately, $D_i(z)$ is not twice differentiable as the last term of $D_i'(z)$ is not differentiable at $\{z \mid b_j(\mathbf{w}^{k,i}+z\mathbf{e}_i)=0 \text{ for some } j\}$. We follow Mangasarian (2002) to define the generalized second derivative:

$$\begin{aligned} D_i''(z) &= 1+2C \sum_{j\in I(\mathbf{w}^{k,i}+z\mathbf{e}_i)} y_j^2 x_{ji}^2 \\ &= 1+2C \sum_{j\in I(\mathbf{w}^{k,i}+z\mathbf{e}_i)} x_{ji}^2. \end{aligned} \tag{10}$$

A simple Newton method to solve (7) begins with $z^0=0$ and iteratively updates $z$ by the following way until $D_i'(z)=0$:

$$z^{t+1} = z^t - D_i'(z^t)/D_i''(z^t) \text{ for } t=0,1,\ldots. \tag{11}$$

Mangasarian (2002) proved that under an assumption, this procedure terminates in finite steps and solves (7). Coordinate descent methods are known to converge if at each inner iteration we uniquely attain the minimum of the sub-problem (Bertsekas, 1999, Proposition 2.7.1). Unfortunately, the assumption by Mangasarian (2002) may not hold in real cases, so taking the full Newton step (11) may not decrease the function $D_i(z)$. Furthermore, solving the sub-problem exactly is too expensive.

An earlier approach of using coordinate descents for L2-SVM without exactly solving the sub-problem is by Zhang and Oles (2001). In their algorithm CMLS, the approximate solution is restricted within a region. By evaluating the upper bound of generalized second-order derivatives in this region, one replaces the denominator of the Newton step (11) with that upper bound. This setting guarantees the decrease of $D_i(z)$. However, there are two problems. First, function decreasing does not imply that $\{\mathbf{w}^k\}$ converges to the global optimum. Secondly, the step size generated by evaluating the upper bound of generalized second derivatives may be too conservative. We describe details of CMLS in Section 4.3.

While coordinate descent methods have been well studied in optimization, most convergence analyses assume that the one-variable sub-problem is exactly solved. We consider the result by Grippo and Sciandrone (1999), which establishes the convergence by requiring only the following sufficient decrease condition:

$$D_i(z) - D_i(0) \le -\sigma z^2, \tag{12}$$

where $z$ is the step taken and $\sigma$ is any constant in (0, 1/2). Since we intend to take the Newton direction

$$d = \frac{-D_i'(0)}{D_i''(0)}, \tag{13}$$

it is important to check if $z=d$ satisfies (12). The discussion below shows that in general the condition hold. If the function $D_i(z)$ is quadratic around 0, then

$$D_i(z) - D_i(0) = D_i'(0)z + \frac{1}{2}D_i''(0)z^2.$$

Using $D_i''(0) > 1$ in (10), $z=d=-D_i'(0)/D_i''(0)$ leads to

$$-\frac{D_i'(0)^2}{2D_i''(0)} \le -\sigma \frac{D_i'(0)^2}{D_i''(0)^2},$$

so (12) holds. As $D_i(z)$ is only piecewise quadratic, (12) may not hold using $z=d$. However, we can conduct a simple line search. The following theorem shows that there is a $\lambda \in (0,1)$ such that $z=\lambda d$ satisfies the sufficient decrease condition:

**Algorithm 2** Solving the sub-problem using Newton direction with the line search.

1. Given $\mathbf{w}^{k,i}$. Choose $\beta \in (0,1)$ (e.g., $\beta = 0.5$).
2. Calculate the Newton direction $d = -D_i'(0)/D_i''(0)$.
3. Compute $\lambda = \max\{1, \beta, \beta^2, \ldots\}$ such that $z = \lambda d$ satisfies (12).

**Theorem 1** *Given the Newton direction $d$ as in* (13)*. Then $z = \lambda d$ satisfies* (12) *for all $0 \le \lambda \le \bar{\lambda}$, where*

$$\bar{\lambda} = \frac{D_i''(0)}{H_i/2 + \sigma} \text{ and } H_i = 1 + 2C \sum_{j=1}^{l} x_{ji}^2. \tag{14}$$

The proof is in Appendix A.1. Therefore, at each inner iteration of Algorithm 1, we take the Newton direction $d$ as in (13), and then sequentially check $\lambda = 1, \beta, \beta^2, \ldots$, where $\beta \in (0,1)$, until $\lambda d$ satisfies (12). Algorithm 2 lists the details of a line search procedure. We did not specify how to approximately solve sub-problems in Algorithm 1. From now on, we assume that it uses Algorithm 2.

Calculating $D_i(\lambda d)$ is the main cost of checking (12). We can use a trick to reduce the number of $D_i(\lambda d)$ calculations. Theorem 1 indicates that if

$$0 \le \lambda \le \bar{\lambda} = \frac{D_i''(0)}{H_i/2 + \sigma}, \tag{15}$$

then $z = \lambda d$ satisfies the sufficient decrease condition (12). $H_i$ is independent of $\mathbf{w}$, so it can be precomputed before training. Furthermore, we already evaluate $D_i''(0)$ in computing the Newton step, so it takes only constant time to check (15). At Step 3 of Algorithm 2, we sequentially use $\lambda = 1, \beta, \beta^2, \ldots$, etc. Before calculating (12) using a smaller $\lambda$, we check if $\lambda$ satisfies (15). If it does, then there is no need to evaluate the new $D_i(\lambda d)$. If $\lambda = 1$ already satisfies (15), the line search procedure is essentially waived. Thus the computational time is effectively reduced.

We discuss parameters in our algorithm. First, as $\lambda = 1$ is often successful, our algorithm is insensitive to $\beta$. We choose $\beta$ as 0.5. Secondly, there is a parameter $\sigma$ in (12). The smaller value of $\sigma$ leads to a looser sufficient decrease condition, which reduces the time of line search, but increases the number of outer iterations. A common choice of $\sigma$ is 0.01 in unconstrained optimization algorithms.

It is important to study the convergence properties of Algorithm 1. An excellent study on the convergence rate of coordinate descent methods is by Luo and Tseng (1992). They assume that each sub-problem is exactly solved, so we cannot apply their results here. The following theorem proves the convergence results of Algorithm 1.

**Theorem 2** *The sequence $\{\mathbf{w}^k\}$ generated by Algorithm 1 linearly converges. That is, there is a constant $\mu \in (0,1)$ such that*

$$f(\mathbf{w}^{k+1}) - f(\mathbf{w}^*) \le (1-\mu)(f(\mathbf{w}^k) - f(\mathbf{w}^*)), \forall k.$$

*Moreover, the sequence $\{\mathbf{w}^k\}$ globally converges to $\mathbf{w}^*$. The algorithm obtains an $\varepsilon$-accurate solution in*

$$O\left(nC^3P^6(\#nz)^3 \log(1/\varepsilon)\right) \tag{16}$$

*iterations.*

The proof is in Appendix A.2. Note that as data are usually scaled before training, $P \leq 1$ in most practical cases.

Next, we investigate the computational complexity per outer iteration of Algorithm 1. The main cost comes from solving the sub-problem by Algorithm 2. At Step 2 of Algorithm 2, to evaluate $D_i'(0)$ and $D_i''(0)$, we need $b_j(\mathbf{w}^{k,i})$ for all $j$. Here we consider sparse data instances. Calculating $b_j(\mathbf{w}), j = 1,\ldots,l$ takes $O(\#\text{nz})$ operations, which are large. However, one can use the following trick to save the time:

$$b_j(\mathbf{w}+z\mathbf{e}_i) = b_j(\mathbf{w}) - zy_jx_{ji}, \tag{17}$$

If $b_j(\mathbf{w}), j = 1,\ldots,l$ are available, then obtaining $b_j(\mathbf{w}+z\mathbf{e}_i)$ involves only nonzero $x_{ji}$'s of the $i$th feature. Using (17), obtaining all $b_j(\mathbf{w}+z\mathbf{e}_i)$ costs $O(m)$, where $m$, the average number of nonzero values per feature, is defined in (5). To have $b_j(\mathbf{w}^0)$, we can start with $\mathbf{w}^0 = \mathbf{0}$, so $b_j(\mathbf{w}^0) = 1, \forall j$. With $b_j(\mathbf{w}^{k,i})$ available, the cost of evaluating $D_i'(0)$ and $D_i''(0)$ is $O(m)$. At Step 3 of Algorithm 2, we need several line search steps using $\lambda = 1, \beta, \beta^2, \ldots$, etc. For each $\lambda$, the main cost is on calculating

$$\begin{aligned} D_i(\lambda d) - D_i(0) = &\frac{1}{2}(w_i^{k,i}+\lambda d)^2 - \frac{1}{2}(w_i^{k,i})^2 \\ &+ C\Big(\sum_{j\in I(\mathbf{w}^{k,i}+\lambda d\mathbf{e}_i)} (b_j(\mathbf{w}^{k,i}+\lambda d\mathbf{e}_i))^2 - \sum_{j\in I(\mathbf{w}^{k,i})} (b_j(\mathbf{w}^{k,i}))^2\Big). \end{aligned} \tag{18}$$

Note that from (17), if $x_{ji} = 0$,

$$b_j(\mathbf{w}^{k,i}+\lambda d\mathbf{e}_i) = b_j(\mathbf{w}^{k,i}).$$

Hence, (18) involves no more than $O(m)$ operations. In summary, Algorithm 2 costs

$$\begin{aligned} & O(m) \text{ for evaluating } D_i'(0) \text{ and } D_i''(0) \\ + \quad & O(m) \times \#\text{ line search steps.} \end{aligned}$$

From the explanation earlier and our experiments, in general the sufficient decrease condition holds when $\lambda = 1$. Then the cost of Algorithm 2 is about $O(m)$. Therefore, in general the complexity per outer iteration is:

$$O(nm) = O(\#\text{nz}). \tag{19}$$

## 3. Implementation Issues

In this section, we discuss some techniques for a fast implementation of Algorithm 1. First, we aim at suitable data representations. Secondly, we show that the order of sub-problems at each iteration can be any permutation of $\{1,\ldots,n\}$. Experiments in Section 5 indicate that the performance of using a random permutation is superior to that of using the fixed order $1,\ldots,n$. Finally, we present an online version of our algorithm.

### 3.1 Data Representation

For sparse data, we use a sparse matrix

$$X = \begin{bmatrix} \mathbf{x}_1^T \\ \vdots \\ \mathbf{x}_l^T \end{bmatrix} \tag{20}$$

to store the training instances. There are several ways to implement a sparse matrix. Two common ones are "row format" and "column format" (Duff et al., 1989). For data classification, using column (row) format allows us to easily access any particular feature (instance). In our case, as we decompose the problem (3) into sub-problems over features, the column format is more suitable.

### 3.2 Random Permutation of Sub-problems

In Section 2, we propose a coordinate descent algorithm which solves the one-variable sub-problems in the order of $w_1, \ldots, w_n$. As the features may be correlated, the order of features may affect the training speed. One can even use an arbitrary order of sub-problems. To prove the convergence, we require that each sub-problem is solved once at one outer iteration. Therefore, at the $k$th iteration, we construct a random permutation $\pi_k$ of $\{1, \ldots, n\}$, and sequentially minimize with respect to variables $w_{\pi(1)}, w_{\pi(2)}, \ldots, w_{\pi(n)}$. Similar to Algorithm 1, the algorithm generates a sequence $\{\mathbf{w}^{k,i}\}$ such that $\mathbf{w}^{k,1} = \mathbf{w}^k$, $\mathbf{w}^{k,n+1} = \mathbf{w}^{k+1,1}$ and

$$w_t^{k,i} = \begin{cases} w_t^{k+1} & \text{if } \pi_k^{-1}(t) < i, \\ w_t^k & \text{if } \pi_k^{-1}(t) \geq i. \end{cases}$$

The update from $\mathbf{w}^{k,i}$ to $\mathbf{w}^{k,i+1}$ is by

$$w_t^{k,i+1} = w_t^{k,i} + \arg\min_z f(\mathbf{w}^{k,i} + z\mathbf{e}_{\pi_k(i)}) \quad \text{if } \pi_k^{-1}(t) = i.$$

We can prove the same convergence result:

**Theorem 3** *Results in Theorem 2 hold for Algorithm 1 with random permutations $\pi_k$.*

The proof is in Appendix A.3. Experiments in Section 5 show that a random permutation of sub-problems leads to faster training.

### 3.3 An Online Algorithm

If the number of features is very large, we may not need to go through all $\{w_1, \ldots, w_n\}$ at each iteration. Instead, one can have an online setting by arbitrarily choosing a feature at a time. That is, from $\mathbf{w}^k$ to $\mathbf{w}^{k+1}$ we only modify one component. A description is in Algorithm 3. The following theorem indicates the convergence rate in expectation:

**Theorem 4** *Let $\delta \in (0,1)$. Algorithm 3 requires $O\left(nl^2C^3P^6(\#nz)\log(\frac{1}{\delta\varepsilon})\right)$ iterations to obtain an $\varepsilon$-accurate solution with confidence $1-\delta$.*

The proof is in Appendix A.4.

## 4. Related Methods

In this section, we discuss three existing schemes for large-scale linear SVM. They will be compared in Section 5. The first one is Pegasos (Shalev-Shwartz et al., 2007), which is notable for its efficiency in training linear L1-SVM. The second one is a trust region Newton method (Lin et al.). It is one of the fastest implementations for L2-SVM. The last one is CMLS, which is a coordinate descent method proposed by Zhang and Oles (2001).

**Algorithm 3** An online coordinate descent algorithm

1. Start with any initial $\mathbf{w}^0$.
2. For $k = 0, 1, \ldots$
   (a) Randomly choose $i_k \in \{1, 2, \ldots, n\}$.
   (b) Fix $w_1^k, \ldots, w_{i_k-1}^k, w_{i_k+1}^k, \ldots, w_n^k$ and approximately solve the sub-problem (7) to obtain $w_{i_k}^{k+1}$.

Coordinate descent methods have been used in other machine learning problems. For example, Rätsch et al. (2002) discuss the connection between boosting/logistic regression and coordinate descent methods. Their strategies for selecting coordinates at each outer iteration are different from ours. We do not discuss details here.

### 4.1 Pegasos for L1-SVM

We briefly introduce the Pegasos algorithm (Shalev-Shwartz et al., 2007). It is an efficient method to solve the following L1-SVM problem:

$$\min_{\mathbf{w}} g(\mathbf{w}) = \frac{\lambda}{2}\mathbf{w}^T\mathbf{w} + \frac{1}{l}\sum_{j=1}^{l} \max(1 - y_j \mathbf{w}^T \mathbf{x}_j, 0). \tag{21}$$

By setting $\lambda = \frac{1}{Cl}$, we have

$$g(\mathbf{w}) = f(\mathbf{w})/Cl, \tag{22}$$

where $f(\mathbf{w})$ is the objective function of (2). Thus (21) and (2) are equivalent. Pegasos has two parameters. One is the subsample size $K$, and the other is the penalty parameter $\lambda$. It begins with an initial $\mathbf{w}^0$ whose norm is at most $1/\sqrt{\lambda}$. At each iteration $k$, it randomly selects a set $A_k \subset \{\mathbf{x}_j, y_j\}_{j=1,\ldots,l}$ of size $K$ as the subsamples of training instances and sets a learning rate

$$\eta_k = \frac{1}{\lambda k}. \tag{23}$$

Then it updates $\mathbf{w}^k$ with the following rules:

$$\begin{aligned}
\mathbf{w}^{k+1} &= \min\left(1, \frac{1/\sqrt{\lambda}}{\|\mathbf{w}^{k+\frac{1}{2}}\|}\right)\mathbf{w}^{k+\frac{1}{2}}, \\
\mathbf{w}^{k+\frac{1}{2}} &= \mathbf{w}^k - \eta_k \nabla_k, \\
\nabla_k &= \lambda \mathbf{w}^k - \frac{1}{K}\sum_{j \in A_k^+(\mathbf{w}^k)} y_j \mathbf{x}_j, \\
A_k^+(\mathbf{w}) &= \{j \in A_k \mid 1 - y_j \mathbf{w}^T \mathbf{x}_j > 0\},
\end{aligned} \tag{24}$$

where $\nabla_k$ is considered as a sub-gradient of the approximate objective function:

$$\frac{\lambda}{2}\mathbf{w}^T\mathbf{w} + \frac{1}{K}\sum_{j \in A_k} \max(1 - y_j \mathbf{w}^T \mathbf{x}_j, 0).$$

**Algorithm 4** Pegasos algorithm for solving L1-SVM.

1. Given $\lambda, K$, and $\mathbf{w}^0$ with $\|\mathbf{w}^0\| \leq 1/\sqrt{\lambda}$.
2. For $k = 0, 1, \ldots$
   (a) Select a set $A_k \in \{\mathbf{x}_j, y_j \mid j = 1 \ldots l\}$, and the learning rate $\eta$ by (23).
   (b) Obtain $\mathbf{w}^{k+1}$ by (24).

Here $\mathbf{w}^{k+1/2}$ is a vector obtained by the stochastic gradient descent step, and $\mathbf{w}^{k+1}$ is the projection of $\mathbf{w}^{k+1/2}$ to the set $\{\mathbf{w} \mid \|\mathbf{w}\| \leq 1/\sqrt{\lambda}\}$. Algorithm 4 lists the detail of Pegasos. The parameter $K$ decides the number of training instances involved at an iteration. If $K = l$, Pegasos considers all examples at each iteration, and becomes a subgradient projection method. In this case the cost per iteration is $O(\#\text{nz})$. If $K < l$, Pegasos is a randomized algorithm. For the extreme case of $K = 1$, Pegasos chooses only one training instance for updating. Thus the average cost per iteration is $O(\#\text{nz}/l)$. In subsequent experiments, we set the subsample size $K$ to one as Shalev-Shwartz et al. (2007) suggested.

Regarding the complexity of Pegasos, we first compare Algorithm 1 with Pegasos ($K = l$). Both algorithms are deterministic and cost $O(\#\text{nz})$ per iteration. Shalev-Shwartz et al. (2007) prove that Pegasos with $K = l$ needs $\tilde{O}(R^2/(\varepsilon_g \lambda))$ iterations to achieve an $\varepsilon_g$-accurate solution, where $R = \max_j \|\mathbf{x}_j\|$, and $\tilde{O}(h(n))$ is shorthand for $O(h(n)\log^k h(n))$, for some $k \geq 0$. We use $\varepsilon_g$ as Pegasos considers $g(\mathbf{w})$ in (22), a scaled form of $f(\mathbf{w})$. From (1), an $\varepsilon_g$-accurate solution for $g(\mathbf{w})$ is equivalent to an $(\varepsilon/Cl)$-accurate solution for $f(\mathbf{w})$. With $\lambda = 1/Cl$ and $R^2 = O(P^2(\#\text{nz})/l)$, where $P$ is defined in (6), Pegasos takes

$$\tilde{O}\left(\frac{C^2P^2l(\#\text{nz})}{\varepsilon}\right)$$

iterations to achieve an $\varepsilon$-accurate solution. One can compare this value with (16), the number of iterations by Algorithm 1.

Next, we compare two random algorithms: Pegasos with $K = 1$ and our Algorithm 3. Shalev-Shwartz et al. (2007) prove that Pegasos takes $\tilde{O}(\frac{R^2}{\lambda\delta\varepsilon_g})$ iterations to obtain an $\varepsilon_g$-accurate solution with confidence $1 - \delta$. Using a similar derivation in the last paragraph, we can show that this is equivalent to $\tilde{O}(C^2P^2l(\#\text{nz})/\delta\varepsilon)$. As the cost per iteration is $O(\#\text{nz}/l)$, the overall complexity is

$$\tilde{O}\left(\frac{C^2P^2(\#\text{nz})^2}{\delta\varepsilon}\right).$$

For our Algorithm 3, each iteration costs $O(m)$, so following Theorem 4 the overall complexity is $O\left(l^2C^3P^6(\#\text{nz})^2\log(\frac{1}{\delta\varepsilon})\right)$.

Based on the above analysis, the number of iterations required for our algorithm is proportional to $O(\log(1/\varepsilon))$, while that for Pegasos is $O(1/\varepsilon)$. Therefore, our algorithm tends to have better final convergence than Pegasos for both deterministic and random settings. However, for the dependence on the size of data (number of instances and features), our algorithm is worse.

Regarding the stopping condition, as at each iteration Pegasos only takes one sample for updating $\mathbf{w}$, neither function nor gradient information is available. This keeps Pegasos from designing a suitable stopping condition. Shalev-Shwartz et al. (2007) suggest to set a maximal number of

iterations. However, deciding a suitable value may be difficult. We will discuss stopping conditions of Pegasos and other methods in Section 5.3.

### 4.2 Trust Region Newton Method (TRON) for L2-SVM

Recently, Lin et al. introduced a trust region Newton method for logistic regression. Their proposed method can be extended to L2-SVM. In this section, we briefly discuss their approach. For convenience, in subsequent sections, we use TRON to indicate the trust region Newton method for L2-SVM, and TRON-LR for logistic regression

The optimization procedure of TRON has two layers of iterations. At each outer iteration $k$, TRON sets a size $\Delta_k$ of the trust region, and builds a quadratic model

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

as the approximation of the value $f(\mathbf{w}^k+\mathbf{s}) - f(\mathbf{w}^k)$, where $f(\mathbf{w})$ is the objective function in (3) and $\nabla^2 f(\mathbf{w})$ is the generalized Hessian (Mangasarian, 2002) of $f(\mathbf{w})$. Then an inner conjugate gradient procedure approximately finds the Newton direction by minimizing the following optimization problem:

$$\begin{aligned} \min_{\mathbf{s}} \quad & q_k(\mathbf{s}) \\ \text{subject to} \quad & \|\mathbf{s}\| \le \Delta_k. \end{aligned} \tag{25}$$

TRON updates $\mathbf{w}^k$ and $\Delta_k$ by the following rules:

$$\begin{aligned} \mathbf{w}^{k+1} &= \begin{cases} \mathbf{w}^k + \mathbf{s}^k & \text{if } \rho_k > \eta_0, \\ \mathbf{w}^k & \text{if } \rho_k \le \eta_0, \end{cases} \\ \Delta_{k+1} &\in \begin{cases} [\sigma_1 \min\{\|\mathbf{s}^k\|, \Delta_k\}, \sigma_2\Delta_k] & \text{if } \rho_k \le \eta_1, \\ [\sigma_1\Delta_k, \sigma_3\Delta_k] & \text{if } \rho_k \in (\eta_1, \eta_2), \\ [\Delta_k, \sigma_3\Delta_k] & \text{if } \rho_k \ge \eta_2, \end{cases} \\ \rho_k &= \frac{f(\mathbf{w}^k + \mathbf{s}^k) - f(\mathbf{w}^k)}{q_k(\mathbf{s}^k)}, \end{aligned} \tag{26}$$

where $\rho_k$ is the ratio of the actual reduction in the objective function to the approximation model $q_k(\mathbf{s})$. Users pre-specify parameters $\eta_0 > 0$, $1 > \eta_2 > \eta_1 > 0$, and $\sigma_3 > 1 > \sigma_2 > \sigma_1 > 0$. We use

$$\begin{aligned} \eta_0 &= 10^{-4}, \eta_1 = 0.25, \eta_2 = 0.75, \\ \sigma_1 &= 0.25, \sigma_2 = 0.5, \sigma_3 = 4, \end{aligned}$$

as suggested by Lin et al.. The procedure is listed in Algorithm 5.

For the computational complexity, the main cost per TRON iteration is

$$O(\#\text{nz}) \times (\#\text{ conjugate gradient iterations}). \tag{27}$$

Compared to our approach or Pegasos, the cost per TRON iteration is high. It keeps TRON from quickly obtaining a usable model. However, when $\mathbf{w}$ gets close to the minimum, TRON takes the Newton step to achieve fast convergence. We give more observations in the experiment section.

**Algorithm 5** Trust region Newton method for L2-SVM.

1. Given $\mathbf{w}^0$.
2. For $k = 0, 1, \ldots$
   (a) Find an approximate solution $\mathbf{s}^k$ of the trust region sub-problem (25).
   (b) Update $\mathbf{w}^k$ and $\Delta_k$ according to (26).

### 4.3 CMLS: A Coordinate Descent Method for L2-SVM

In Sections 2 and 3, we introduced our coordinate descent method for solving L2-SVM. Here, we discuss the previous work (Zhang and Oles, 2001), which also applies the coordinate descent technique. Zhang and Oles refer to their method as CMLS. At each outer iteration $k$, it sequentially minimizes sub-problems (8) by updating one variable of (3) at a time. In solving the sub-problem, Zhang and Oles (2001) mention that using line searches may result in small step sizes. Hence, CMLS applies a technique similar to the trust region method. It sets a size $\Delta^{k,i}$ of the trust region, evaluates the first derivative (9) of (8), and calculates the upper bound of the generalized second derivative subject to $|z| \le \Delta^{k,i}$:

$$U_i(z) = 1 + \sum_{j=1}^{l} \beta_j(\mathbf{w}^{k,i} + z\mathbf{e}_i),$$

$$\beta_j(\mathbf{w}) = \begin{cases} 2C & \text{if } y_j \mathbf{w}^T \mathbf{x}_j \le 1 + |\Delta^{k,i} x_{ij}|, \\ 0 & \text{otherwise.} \end{cases}$$

Then we obtain the step $z$ as:

$$z = \min(\max(-\frac{D_i'(z)}{U_i(z)}, -\Delta^{k,i}), \Delta^{k,i}). \tag{28}$$

The updating rule of $\Delta$ is:

$$\Delta^{k+1,i} = 2|z| + \varepsilon, \tag{29}$$

where $\varepsilon$ is a small constant.

In order to speed up the process, Zhang and Oles (2001) smooth the objective function of sub-problems with a parameter $c_k \in [0, 1]$:

$$D_i(z) = \frac{1}{2}(\mathbf{w}^{k,i} + z\mathbf{e}_i)^T(\mathbf{w}^{k,i} + z\mathbf{e}_i) + C\sum_{j=1}^{l}(b_j(\mathbf{w}^{k,i} + z\mathbf{e}_i))^2,$$

where

$$b_j(\mathbf{w}) = \begin{cases} 1 - y_j \mathbf{w}^T \mathbf{x}_j & \text{if } 1 - y_j \mathbf{w}^T \mathbf{x}_j > 0, \\ c_k(1 - y_j \mathbf{w}^T \mathbf{x}_j) & \text{otherwise.} \end{cases}$$

Following the setting by (Zhang and Oles, 2001), we choose

$$c_k = \max(0, 1 - k/50), \tag{30}$$

**Algorithm 6** CMLS algorithm for solving L2-SVM.

1. Given $\mathbf{w}^0$ and set initial $\Delta^{0,i} = 10, \forall i$.
2. For $k = 0, 1, \ldots$
   (a) Set $c_k$ by (30). Let $\mathbf{w}^{k,1} = \mathbf{w}^k$.
   (b) For $i = 1, 2, \ldots, n$
      i. Evaluate $z$ by (28).
      ii. $\mathbf{w}^{k,i+1} = \mathbf{w}^{k,i} + z\mathbf{e}_i$.
      iii. Update $\Delta$ by (29).
   (c) Let $\mathbf{w}^{k+1} = \mathbf{w}^{k,n+1}$.

| Problem | $l$ | $n$ | #nz |
|---|---|---|---|
| astro-physic | 62,369 | 99,757 | 4,834,550 |
| real-sim | 72,309 | 20,958 | 3,709,083 |
| news20 | 19,996 | 1,355,191 | 9,097,916 |
| yahoo-japan | 176,203 | 832,026 | 23,506,415 |
| rcv1 | 677,399 | 47,236 | 49,556,258 |
| yahoo-korea | 460,554 | 3,052,939 | 156,436,656 |

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

and set the initial $\mathbf{w} = \mathbf{0}$ and $\Delta^{0,i} = 10, \forall i$. We find that the result is insensitive to these parameters. The detail of CMLS algorithm is listed in Algorithm 6.

Zhang and Oles (2001) prove that if $c_k = 0, \forall k$, then the objective function of (3) is decreasing after each inner iteration. However, such a property may not imply that Algorithm 6 converges to the minimum. In addition, CMLS updates $\mathbf{w}$ by (28), which is more conservative than Newton steps. In Section 5, we show that CMLS takes more time and iterations than ours to obtain a solution.

## 5. Experiments and Analysis

In this section, we conduct two experiments to investigate the performance of our proposed coordinate descent algorithm. The first experiment compares our method with other L2-SVM solvers in terms of the speed to reduce function/gradient values. The second experiment evaluates various state of the art linear classifiers for L1-SVM, L2-SVM, and logistic regression. We also discuss the stopping condition of these methods.

### 5.1 Data Sets

Table 1 lists the number of instances ($l$), features ($n$), and non-zero elements (#nz) of six data sets. All sets are from document classification. Past studies show that linear SVM performs as good as kernelized ones for such data. Details of astro-physic are mentioned in Joachims (2006), while others are in Lin et al.. Three data sets real-sim, news20 and rcv1 are publicly available

| Data set | CDPER | CD | TRON | CMLS |
|---|---|---|---|---|
| astro-physic | **0.5** | 1.2 | 1.2 | 2.6 |
| real-sim | **0.2** | 0.3 | 0.9 | 2.0 |
| news20 | 2.4 | **1.0** | 5.2 | 5.3 |
| yahoo-japan | **2.9** | 9.3 | 38.2 | 13.5 |
| rcv1 | **5.1** | 10.8 | 18.6 | 54.8 |
| yahoo-korea | **18.4** | 58.1 | 286.1 | 146.3 |

Table 2: The training time for an L2-SVM solver to reduce the objective value to within 1% of the optimal value. Time is in seconds. We use $C = 1$. The approach with the shortest running time is boldfaced.

at `http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets`. A brief reminder for each data set can be found below.

- astro-physic: This set is a classification problem of scientific papers from Physics ArXiv.
- real-sim: This set includes some Usenet articles.
- news20: This is a collection of news documents, and was preprocessed by Keerthi and DeCoste (2005).
- yahoo-japan: 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.
- yahoo-korea: Similar to yahoo-japan, we use binary term frequencies and normalize each instance to unit length.

To examine the testing accuracy, we use a stratified selection to split each set to $4/5$ training and $1/5$ testing.

### 5.2 Comparisons

We compare the following six implementations. TRON-LR is for logistic regression, Pegasos is for L1-SVM, and all others are for L2-SVM.

1. CD: the coordinate descent method described in Section 2. We choose $\sigma$ in (12) as 0.01.
2. CDPER: the method modified from CD by permuting sub-problems at each outer step. See the discussion in Section 3.2.
3. CMLS: a coordinate descent method for L2-SVM (Zhang and Oles, 2001, Algorithm 3). It is discussed in Section 4.3.
4. TRON: the trust region Newton method (Lin et al.) for L2-SVM. See the discussion in Section 4.2. We use the L2-loss linear SVM implementation in the software LIBLINEAR (version 1.21 with option `-s 2`; `http://www.csie.ntu.edu.tw/~cjlin/liblinear`).

| Data set | L2-SVM | | L1-SVM | | LR | |
|---|---|---|---|---|---|---|
| | $C$ | Accuracy | $C$ | Accuracy | $C$ | Accuracy |
| astro-physic | 0.5 | 97.14 | 1.0 | 97.09 | 8.0 | 97.03 |
| real-sim | 1.0 | 97.59 | 1.0 | 97.52 | 8.0 | 97.57 |
| news20 | 4.0 | 96.85 | 2.0 | 96.70 | 64.0 | 96.17 |
| yahoo-japan | 0.5 | 92.91 | 1.0 | 92.97 | 4.0 | 92.76 |
| rcv1 | 0.5 | 97.77 | 1.0 | 97.77 | 8.0 | 97.76 |
| yahoo-korea | 2.0 | 87.51 | 4.0 | 87.42 | 64.0 | 87.31 |

Table 3: The best parameter $C$ and the corresponding testing accuracy of L1-SVM, L2-SVM and logistic regression (LR). We conduct five-fold cross validation to select $C$.

5. TRON-LR: the trust region Newton method for logistic regression introduced by Lin et al.. Similar to TRON, we use the implementation in the software LIBLINEAR with option `-s 0`.

6. Pegasos: the primal estimated sub-gradient solver for L1-SVM (Shalev-Shwartz et al., 2007). See the discussion in Section 4.1. The source code is available online at `http://ttic.uchicago.edu/~shai/code`.

We do not include the bias term in all the solvers. All the above algorithms are implemented in C++ with double-precision floating-point numbers. Using single precision (e.g., Bottou, 2007) may reduce the computational time in some situations, but this setting may cause numerical inaccuracy. We conduct experiments on an Intel 2.66GHz processor with 8GB of main memory under Linux.

In our first experiment, we compare L2-SVM solvers (with $C = 1$) in term of the speed to reduce function/gradient values. In Table 2, we check their CPU time of reducing the relative difference of the function value to the optimum,

$$\frac{f(\mathbf{w}^k) - f(\mathbf{w}^*)}{|f(\mathbf{w}^*)|}, \tag{31}$$

to within 0.01. We run TRON with the stopping condition $\|\nabla f(\mathbf{w}^k)\| \leq 0.01$ to obtain the reference solutions. Since objective values are stable under such strict stopping conditions, these solutions are seen to be very close to the optima. Overall, our proposed algorithms CDPER and CD perform well on all the data sets. For the large data sets (rcv1, yahoo-japan, yahoo-korea), CD is significantly better than TRON and CMLS. With the permutation of sub-problems, CDPER is even better than CD. To show more detailed comparisons, Figure 1 presents time versus relative difference (31). As a reference, we draw a horizontal dotted line to indicate the relative difference 0.01. Consistent with the observation in Table 2, CDPER is more efficient and stable than others.

In addition, we are interested in how fast these methods decrease the norm of gradients. Figure 2 shows the result. Overall, CDPER converges faster in the beginning, while TRON is the best for final convergence.

The second experiment is to check the relationship between training time and testing accuracy using our implementation and other solvers: CMLS, TRON (L2-SVM and logistic regression), and Pegasos. That is, we investigate which method achieves reasonable testing accuracy more quickly. To have a fair evaluation, we conduct five-fold cross validation to select the best parameter $C$ for each learning method. Using the selected $C$, we then train the whole training set and predict the

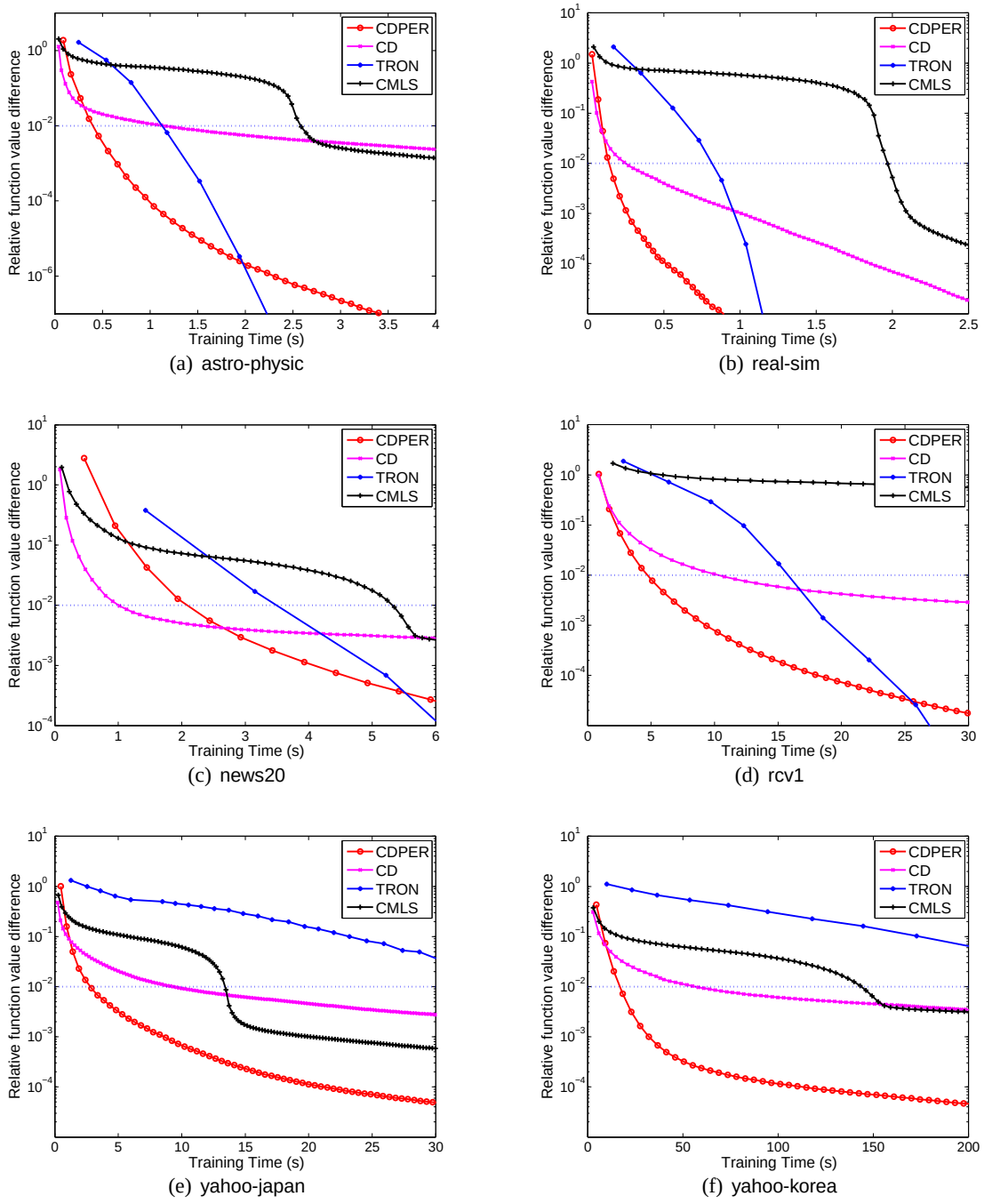


Figure 1: Time versus the relative difference of the objective value to the minimum. The dotted line indicates the relative difference 0.01. We show the training time for each solver to reach this ratio in Table 2. Time is in seconds. $C = 1$ is used.

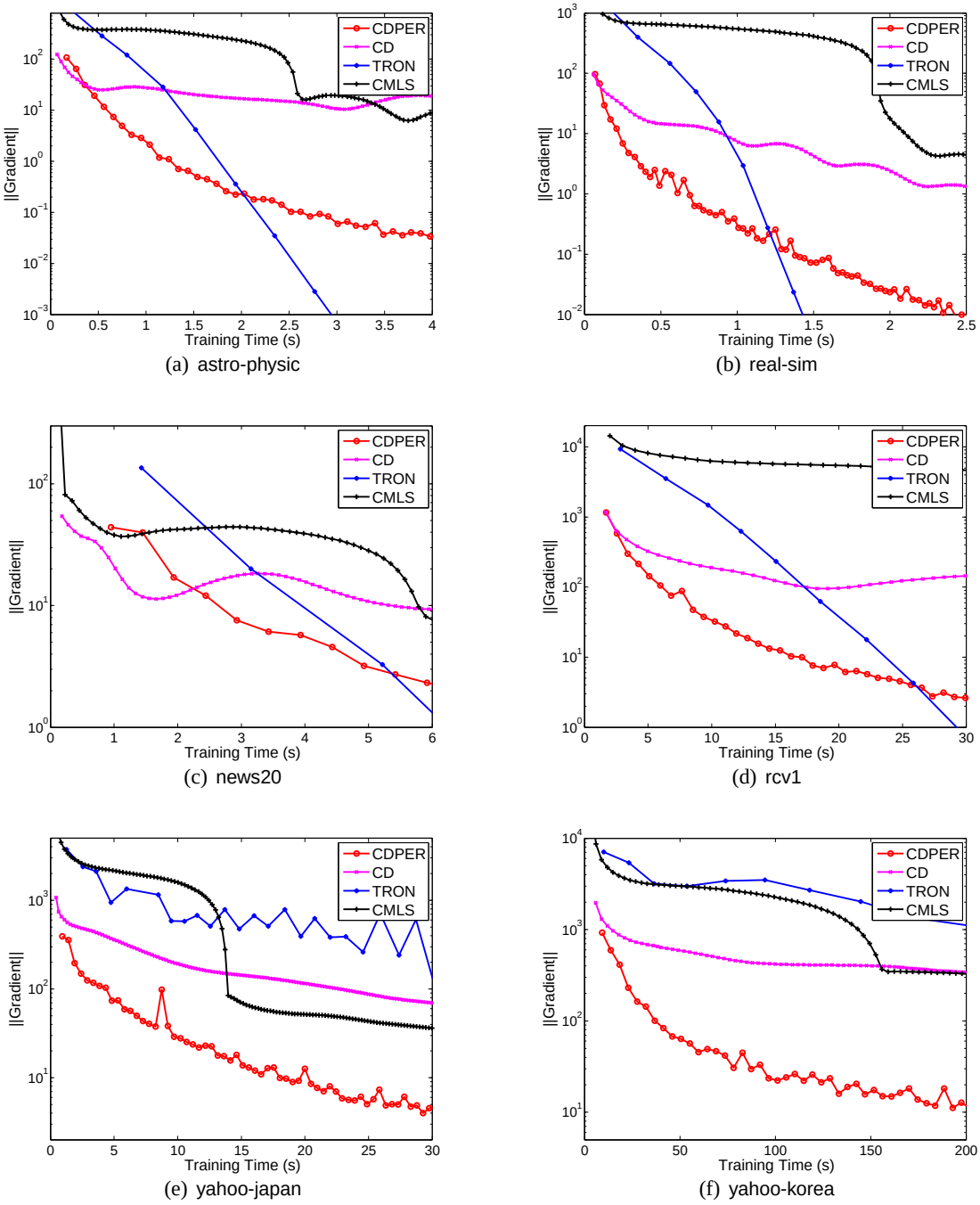


Figure 2: The two-norm of gradient versus the training time. Time is in seconds. $C = 1$ is used.

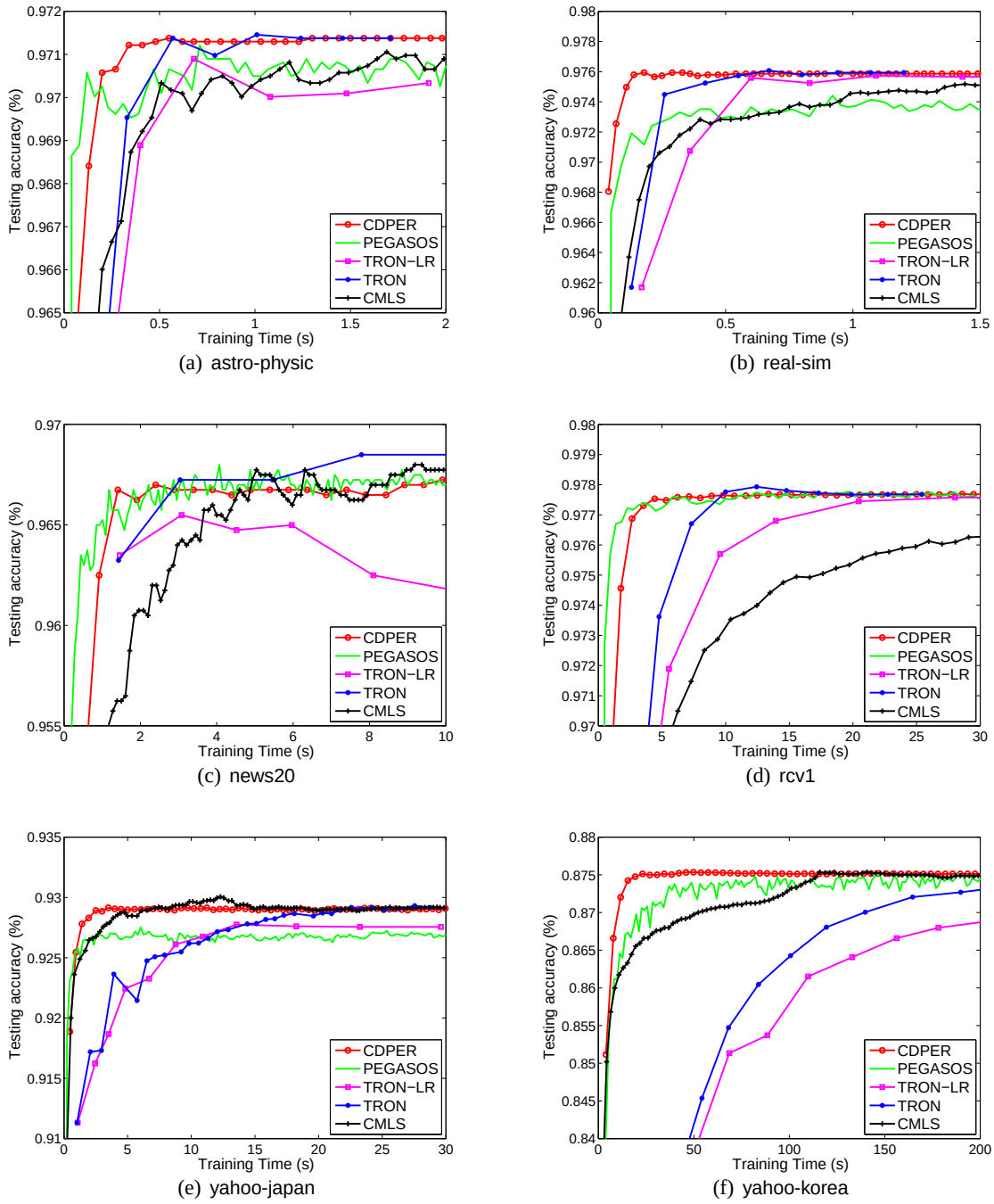


Figure 3: Testing accuracy versus the training time. Time is in seconds. We train each data set using the best $C$ from cross validation. (see Table 3 for details.)

testing set. Table 3 presents the testing accuracy. Notice that some solvers scale the SVM formulation, so we adjust their regularization parameter $C$ accordingly.[2] With the best parameter setting, SVM (L1 and L2) and logistic regression give comparable generalization performances. In Figure 3, we present the testing accuracy along the training time. As an accurate solution of the SVM optimization problem does not imply the best testing accuracy, some implementations achieve higher accuracy before reaching the minimal function value. Below we give some observations of the experiments.

We do not include CD in Figure 3, because CDPER is better than it in almost all situations. One may ask if simply shuffling features once in the beginning can give similar performances to CDPER. Moreover, we can apply permutation schemes to CMLS as well. In Section 6.1, we give a detailed discussion on the issue of feature permutations.

Regarding the online setting of randomly selecting only one feature at each step (Algorithm 3), we find that results are similar to those of CDPER.

From the experimental results, CDPER converges faster than CMLS. Both are coordinate descent methods, and the cost per iteration is similar. However, CMLS suffers from lengthy iterations because its modified Newton method takes a conservative step size. In Figure 3(d), the testing accuracy even does not reach a reasonable value after 30 seconds. Conversely, CDPER usually uses full Newton steps, so it converges faster. For example, CDPER takes the full Newton step in 99.997% inner iterations for solving rcv1 (we check up to 5.96 seconds).

Though Pegasos is efficient for several data sets, the testing accuracy is sometimes unstable (see Figure 3(c)). As Pegasos only subsamples one training data to update $\mathbf{w}^k$, it is influenced more by noisy data. We also observe slow final convergence on the function value. This slow convergence may make the selection of stopping conditions (maximal number of iterations for Pegasos) more difficult.

Finally, compared to TRON and TRON-LR, CDPER is more efficient to yield good testing accuracy (See Table 2 and Figure 3); however, if we check the value $||\nabla f(\mathbf{w}^k)||$, Figure 2 shows that TRON converges faster in the end. This result is consistent with what we discussed in Section 1 on distinguishing various optimization methods. We indicated that a Newton method (where TRON is) has fast final convergence. Unfortunately, since the cost per TRON iteration is high, and the Newton direction is not effective in the beginning, TRON is less efficient in the early stage of the optimization procedure.

### 5.3 Stopping Conditions

In this section, we discuss stopping conditions of our algorithm and other existing methods. In solving a strictly convex optimization problem, the norm of gradients is often considered in the stopping condition. The reason is that

$$\|\nabla f(\mathbf{w})\| = 0 \iff \mathbf{w} \text{ is the global minimum.}$$

For example, TRON checks whether the norm of gradient is small enough for stopping. However, as our coordinate descent method updates one component of $\mathbf{w}$ at each inner iteration, we have only $D_i'(0) = \nabla f(\mathbf{w}^{k,i})_i, i = 1,\ldots,n$. Theorem 2 shows that $\mathbf{w}^{k,i} \to \mathbf{w}^*$, so we have

$$D_i'(0) = \nabla f(\mathbf{w}^{k,i})_i \to 0, \forall i.$$

2. The objective function of Pegasos and (2) are equivalent by setting $\lambda = 1/(C \times \text{number of instances})$, where $\lambda$ is the penalty parameter used by Shalev-Shwartz et al. (2007).

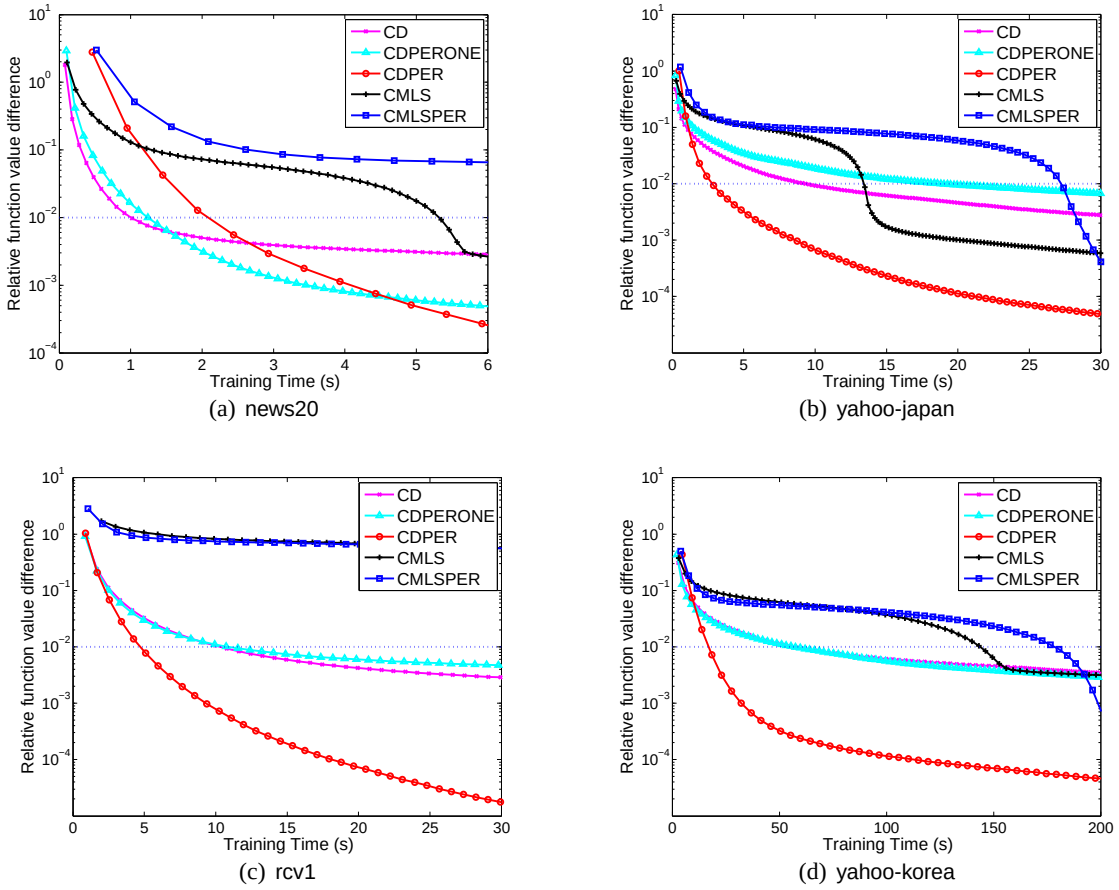


Figure 4: Results of different orders of sub-problems at each outer iteration. We present time versus the relative difference of the objective value to the minimum. Time is in second.

Therefore, by storing $D_i(0), \forall i$, at the end of the $k$th iteration, one can check if $\sum_{i=1}^{n} D_i'(0)^2$ or $\max_i |D_i'(0)|$ is small enough. For Pegasos, we mentioned in Section 4.1 that one may need a maximal number of iterations as the stopping condition due to the lack of function/gradient information. Another possible condition is to check the validation accuracy. That is, the training procedure terminates after reaching a stable validation accuracy value.

## 6. Discussion and Conclusions

In this section, we discuss some related issues and give conclusions.

### 6.1 Order of Sub-problems at Each Outer Iteration

In Section 5.2, we show that a random order of the sub-problems helps our coordinate descent method to converge faster in most cases. In this section, we give detailed experiments. Following the same setting in Figure 1, we compare our coordinate descent method with/without permutation

of sub-problems (CDPER and CD), with permutation only once before training (CDPERONE), and CMLS with/without permuting sub-problems (CMLS and CMLSPER). Figure 4 shows the relative difference of the objective value to the minimum along time. Overall, CDPER converges faster than CDPERONE and CD, but CMLSPER does not improve over CMLS much.

With the permutation of features at each iteration, the cost per CDPER iteration is slightly higher than CD, but CDPER requires much fewer iterations to achieve a similar accuracy value. This result seems to indicate that if the sub-problem order is fixed, the update of variables becomes slower. However, as CD sequentially accesses features, it has better data locality in the computer memory hierarchy. An example is news20 in Figure 4(a). As the number of features is much larger than the number of instances, two adjacent sub-problems of CDPER may access two very far away features. Then the cost per CD iteration is only 1/5 of CDPER, so CD is better in the beginning. CDPER catches up in the end due to its faster convergence.

For CMLS and CMLSPER, the latter is only faster in the final stage (see the right end of Figures 4(b) and 4(d)). Since the function reduction of CMLS (or CMLSPER) is slow, the advantage of doing permutations appears after long training time.

The main difference between CDPERONE and CDPER is that the former only permutes features once in the beginning. Figure 4 clearly shows that CDPER is better than CDPERONE, so permuting features only once is not enough. If we compare CD and CDPERONE, there is no definitive winner. This result seems to indicate that feature ordering affects the performance of the coordinate descent method. By using various feature orders, CDPER avoids taking a bad one throughout all iterations.

### 6.2 Coordinate Descents for Logistic Regression

We can apply the proposed coordinate descent method to solve logistic regression, which is twice differentiable. An earlier study of using coordinate decent methods for logistic regression/maximum entropy is by Miroslav et al. (2004). We compare an implementation with TRON-LR. Surprisingly, our method is not better in most cases. Experiments show that for training rcv1, our coordinate descent method takes 93.1 seconds to reduce the objective value to within 1% of the optimal value, while TRON-LR takes 27.9 seconds. Only for yahoo-japan and yahoo-korea, where TRON-LR is slow (see Figure 3), the coordinate descent method is competitive. This result is contrast to earlier experiments for L2-SVM, where the coordinate descent method more quickly obtains a useful model than TRON. We give some explanations below.

With the logistic loss, the objective function is (4). The single-variable function $D_i(z)$ is non-linear, so we use Algorithm 2 to obtain an approximate minimum. To use the Newton direction, similar to $D_i'(z)$ and $D_i''(z)$ in (9) and (10), we need

$$D_i'(0) = \nabla_i f(\mathbf{w}) = w_i + C \sum_{j:x_{ji}\neq 0} \frac{-y_j x_{ji} e^{-y_j \mathbf{w}^T \mathbf{x}_j}}{1+e^{-y_j \mathbf{w}^T \mathbf{x}_j}},$$

$$D_i''(0) = \nabla_{ii}^2 f(\mathbf{w}) = 1 + C \sum_{j:x_{ji}\neq 0} \frac{x_{ji}^2 e^{-y_j \mathbf{w}^T \mathbf{x}_j}}{(1+e^{-y_j \mathbf{w}^T \mathbf{x}_j})^2},$$

where we abbreviate $\mathbf{w}^{k,i}$ to $\mathbf{w}$, and use $y_j = \pm 1$. Then $|\{j \mid x_{ji} \neq 0\}|$ exponential operations are conducted. If we assume that $\lambda = 1$ satisfies the sufficient decrease condition (12), then the cost per outer iteration is

$$O(\#\text{nz}) + (\#\text{nz exponential operations}). \tag{32}$$

This complexity is the same as (19) for L2-SVM. However, since each exponential operation is expensive (equivalent to tens of multiplications/divisions), in practice (32) is much more time consuming. For TRON-LR, a trust region Newton method, it calculates $\nabla f(\mathbf{w})$ and $\nabla^2 f(\mathbf{w})$ at the beginning of each iteration. Hence $l$ exponential operations are needed for $\exp(-y_j \mathbf{w}^T \mathbf{x}_j), j = 1, \ldots, l$. From (27), the cost per iteration is

$$O(\#\text{nz}) \times (\#\text{ conjugate gradient iterations}) + (l \text{ exponential operations}). \tag{33}$$

Since $l \ll \#\text{nz}$, exponential operations are not significant in (33). Therefore, the cost per iteration of applying trust region Newton methods to L2-SVM and logistic regression does not differ much. In contrast, (32) shows that coordinate descent methods are less suitable for logistic regression than L2-SVM. However, we may avoid expensive exponential operations if all the elements of $\mathbf{x}_j$ are either 0 or the same constant. By storing $\exp(-y_j \mathbf{w}^T \mathbf{x}_j), j = 1, \ldots, l$, one updates $\exp(-y_j (\mathbf{w}^{k,i})^T \mathbf{x}_j)$ by multiplying it by $\exp(-z y_j x_{ji})$. Using $y_j = \pm 1$, $\exp(-z y_j x_{ji}) = (\exp(-z x_{ji}))^{y_j}$. As $x_{ji}$ is zero or a constant for all $j$, the number of exponential operations per inner iteration is reduced to one. In addition, applying fast approximations of exponential operations such as Schraudolph (1999) may speed up the coordinate descent method for logistic regression.

### 6.3 Conclusions

In summary, we propose and analyze a coordinate descent method for large-scale linear L2-SVM. The new method possesses sound optimization properties. The method is suitable for data with an easy access of any feature. Experiments indicate that our method is more stable and efficient than most existing algorithms. We plan to extend our work to other challenging problems such as training large data which can not fit into memory.

## Acknowledgments

This work was supported in part by the National Science Council of Taiwan grant 95-2221-E-002-205-MY3. The authors thank Associate Editor and reviewers for helpful comments.

## Appendix A. Proofs

In this section, we prove theorems appeared in the paper. First, we discuss some properties of our objective function $f(\mathbf{w})$. Consider the following piecewise quadratic strongly convex function:

$$g(\mathbf{s}) = \frac{1}{2}\mathbf{s}^T \mathbf{s} + C\|(A\mathbf{s} - \mathbf{h})_+\|^2, \tag{34}$$

where $(\cdot)_+$ is the operator that replaces negative components of a vector with zeros. Mangasarian (2002) proves the following inequalities for all $\mathbf{s}, \mathbf{v} \in R^n$:

$$(\nabla g(\mathbf{s}) - \nabla g(\mathbf{v}))^T (\mathbf{s} - \mathbf{v}) \geq \|\mathbf{s} - \mathbf{v}\|^2, \tag{35}$$

$$|g(\mathbf{v}) - g(\mathbf{s}) - \nabla g(\mathbf{s})^T (\mathbf{v} - \mathbf{s})| \leq \frac{K}{2}\|\mathbf{v} - \mathbf{s}\|^2, \tag{36}$$

where

$$K = 1 + 2C\|A\|_2^2.$$

Our objective function $f(\mathbf{w})$ is a special case of (34) with $A = YX$ ($X$ is defined in Eq. 20) and $\mathbf{h} = -\mathbf{1}$, where $Y$ is a diagonal matrix with $Y_{jj} = y_j$, $j = 1,\ldots,l$ and $\mathbf{1}$ is the vector of all ones. With $y_i = \pm 1$, $f(\mathbf{w})$ satisfies (35) and (36) with

$$K = 1 + 2C\|X\|_2^2. \tag{37}$$

To derive properties of the subproblem $D_i(z)$ (defined in Eq. 8), we set

$$A = -\begin{bmatrix} y_1 x_{1i} \\ \vdots \\ y_l x_{li} \end{bmatrix} \text{ and } \mathbf{h} = \begin{bmatrix} y_1 \mathbf{w}^T \mathbf{x}_1 - y_1 w_i x_{1i} - 1 \\ \vdots \\ y_l \mathbf{w}^T \mathbf{x}_l - y_l w_i x_{li} - 1 \end{bmatrix},$$

where we abbreviate $\mathbf{w}^{k,i}$ to $\mathbf{w}$. Since

$$D_i(z) = \frac{1}{2}\mathbf{w}^T\mathbf{w} - \frac{1}{2}w_i^2 + \left(\frac{1}{2}(z+w_i)^2 + \|(A(z+w_i) - \mathbf{h})_+\|^2\right),$$

the first and second terms of the above form are constants. Hence, $D_i(z)$ satisfies (35) and (36) with

$$K = 1 + 2C\sum_{j=1}^{l} x_{ji}^2.$$

We use $H_i$ to denote $D_i(z)$'s corresponding $K$. This definition of $H_i$ is the same as the one in (14). We then derive several lemmas.

**Lemma 5** *For any $i \in \{1,\ldots,n\}$, and any $z \in R$,*

$$D_i'(0)z + \frac{1}{2}H_i z^2 \geq D_i(z) - D_i(0) \geq D_i'(0)z + \frac{1}{2}z^2. \tag{38}$$

**Proof** There are two inequalities in (38). The first inequality directly comes from (36) using $D_i(z)$ as $g(\mathbf{s})$ and $H_i$ as $K$. To derive the second inequality, if $z < 0$, using $D_i(z)$ as $g(\mathbf{s})$ in (35) yields

$$D_i'(z) \leq D_i'(0) + z.$$

Then,

$$D_i(z) - D_i(0) = -\int_{t=z}^{0} D_i'(t)dt \geq D_i'(0)z + \frac{1}{2}z^2.$$

The situation for $z \geq 0$ is similar.

■

**Lemma 6** *There exists a unique optimum solution for* (3).

**Proof** From Weierstrass' Theorem, any continuous function on a compact set attains its minimum. We consider the level set $A = \{\mathbf{w} \mid f(\mathbf{w}) \leq f(\mathbf{0})\}$. If $A$ is not bounded, there is a sub-sequence $\{\mathbf{w}^k\} \subset A$ such that $\|\mathbf{w}^k\| \to \infty$. Then

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

This contradicts $f(\mathbf{w}^k) \leq f(\mathbf{0})$, so $A$ is bounded. Thus there is at least one optimal solution for (3). Combining with the strict convexity of (3), a unique global optimum exists. ■

### A.1 Proof of Theorem 1

**Proof** By Lemma 5, we let $z = \lambda d$ and have

$$\begin{aligned} & D_i(\lambda d) - D_i(0) + \sigma\lambda^2 d^2 \\ \leq\ & D_i'(0)\lambda d + \frac{1}{2}H_i\lambda^2 d^2 + \sigma\lambda^2 d^2 \\ =\ & -\lambda\frac{D_i'(0)^2}{D_i''(0)} + \frac{1}{2}H_i\lambda^2\frac{D_i'(0)^2}{D_i''(0)^2} + \sigma\lambda^2\frac{D_i'(0)^2}{D_i''(0)^2} \\ =\ & \lambda\frac{D_i'(0)^2}{D_i''(0)}\left(\lambda(\frac{H_i/2+\sigma}{D_i''(0)}) - 1\right). \end{aligned} \tag{39}$$

If we choose $\bar{\lambda} = \frac{D_i''(0)}{H_i/2+\sigma}$, then for $\lambda \leq \bar{\lambda}$, (39) is non-positive. Therefore, (12) is satisfied for all $0 \leq \lambda \leq \bar{\lambda}$. ■

### A.2 Proof of Theorem 2 (convergence of Algorithm 1)

**Proof** By setting $\pi_k(i) = i$, this theorem is a special case of Theorem 3. ■

### A.3 Proof of Theorem 3 (Convergence of Generalized Algorithm 1)

**Proof** To begin, we define 1-norm and 2-norm of a vector $\mathbf{w} \in R^n$:

$$\|\mathbf{w}\|_1 = \sum_{i=1}^{n}|w_i|, \quad \|\mathbf{w}\|_2 = \sqrt{\sum_{i=1}^{n} w_i^2}.$$

The following inequality is useful:

$$\|\mathbf{w}\|_2 \leq \|\mathbf{w}\|_1 \leq \sqrt{n}\|\mathbf{w}\|_2, \quad \forall \mathbf{w} \in R^n. \tag{40}$$

By Theorem 1, any $\lambda \in [\beta\bar{\lambda}, \bar{\lambda}]$ satisfies the sufficient decrease condition (12), where $\beta \in (0,1)$ and $\bar{\lambda}$ is defined in (14). Since Algorithm 2 selects $\lambda$ by trying $\{1, \beta, \beta^2, \dots\}$, the value $\lambda$ selected by Algorithm 2 satisfies

$$\lambda \geq \beta\bar{\lambda} = \frac{\beta}{H_i/2+\sigma}D''_{\pi_k(i)}(0).$$

This and (13) suggest that the step size $z = \lambda d$ in Algorithm 2 satisfies

$$|z| = \lambda\left|\frac{-D'_{\pi_k(i)}(0)}{D''_{\pi_k(i)}(0)}\right| \geq \frac{\beta}{H_i/2+\sigma}|D'_{\pi_k(i)}(0)|. \tag{41}$$

Assume

$$H = \max(H_1, \dots, H_n) \text{ and } \gamma = \frac{\beta}{H/2+\sigma}. \tag{42}$$

We use $\mathbf{w}$ and $f(\mathbf{w})$ to rewrite (41):

$$|w^{k,i+1}_{\pi_k(i)} - w^{k,i}_{\pi_k(i)}| \geq \gamma |\nabla f(\mathbf{w}^{k,i})_{\pi_k(i)}|, \tag{43}$$

where we use the fact

$$D'_{\pi_k(i)}(0) = \nabla f(\mathbf{w}^{k,i})_{\pi_k(i)}.$$

Taking the summation of (43) from $i = 1$ to $n$, we have

$$\begin{aligned}
\|\mathbf{w}^{k+1} - \mathbf{w}^k\|_1 &\geq \gamma \sum_{i=1}^{n} |\nabla f(\mathbf{w}^{k,i})_{\pi_k(i)}| \\
&\geq \gamma \sum_{i=1}^{n} (|\nabla f(\mathbf{w}^{k,1})_{\pi_k(i)}| - |\nabla f(\mathbf{w}^{k,i})_{\pi_k(i)} - \nabla f(\mathbf{w}^{k,1})_{\pi_k(i)}|) \\
&= \gamma \left( \|\nabla f(\mathbf{w}^{k,1})\|_1 - \sum_{i=1}^{n} |\nabla f(\mathbf{w}^{k,i})_{\pi_k(i)} - \nabla f(\mathbf{w}^{k,1})_{\pi_k(i)}| \right).
\end{aligned} \tag{44}$$

By the definition of $f(\mathbf{w})$ in (3),

$$\nabla f(\mathbf{w}) = \mathbf{w} - 2C \sum_{j=1}^{l} y_j \mathbf{x}_j \max(1 - y_j \mathbf{w}^T \mathbf{x}_j, 0).$$

With $y_j = \pm 1$,

$$\begin{aligned}
&\sum_{i=1}^{n} |\nabla f(\mathbf{w}^{k,i})_{\pi_k(i)} - \nabla f(\mathbf{w}^{k,1})_{\pi_k(i)}| \\
\leq &\sum_{i=1}^{n} \left( |w^{k,i}_{\pi_k(i)} - w^{k,1}_{\pi_k(i)}| + 2C \sum_{j=1}^{l} |x_{j\pi_k(i)}| \, |(\mathbf{w}^{k,i})^T \mathbf{x}_j - (\mathbf{w}^{k,1})^T \mathbf{x}_j| \right) \\
\leq &\sum_{i=1}^{n} \left( |w^{k+1}_{\pi_k(i)} - w^{k}_{\pi_k(i)}| + 2C \sum_{j=1}^{l} |x_{j\pi_k(i)}| \sum_{q=1}^{n} |x_{jq}| \, |w^{k,i}_q - w^{k,1}_q| \right) \\
= &\|\mathbf{w}^{k+1} - \mathbf{w}^k\|_1 + 2C \sum_{i=1}^{n} \sum_{j=1}^{l} \sum_{q=1}^{n} |x_{j\pi_k(i)}| \, |x_{jq}| \, |w^{k,i}_q - w^{k,1}_q| \\
\leq &\|\mathbf{w}^{k+1} - \mathbf{w}^k\|_1 + 2C \sum_{q=1}^{n} |w^{k+1}_q - w^k_q| \sum_{i,j: x_{j\pi_k(i)} \neq 0} P^2 \\
= &(1 + 2CP^2(\#\text{nz})) \|\mathbf{w}^{k+1} - \mathbf{w}^k\|_1,
\end{aligned} \tag{45}$$

where $P$ is defined in (6). From (44) and (45), we have

$$\|\mathbf{w}^{k+1} - \mathbf{w}^k\|_1 \geq \frac{\gamma}{1 + \gamma + 2\gamma CP^2(\#\text{nz})} \|\nabla f(\mathbf{w}^{k,1})\|_1.$$

With (40),

$$\begin{aligned}
&\|\mathbf{w}^{k+1} - \mathbf{w}^k\|_2 \geq \frac{1}{\sqrt{n}} \|\mathbf{w}^{k+1} - \mathbf{w}^k\|_1 \\
\geq &\frac{\gamma}{\sqrt{n}(1 + \gamma + 2\gamma CP^2(\#\text{nz}))} \|\nabla f(\mathbf{w}^k)\|_1 \geq \frac{\gamma}{\sqrt{n}(1 + \gamma + 2\gamma CP^2(\#\text{nz}))} \|\nabla f(\mathbf{w}^k)\|_2.
\end{aligned} \tag{46}$$

From Lemma 6, there is a unique global optimum $\mathbf{w}^*$ for (3). The optimality condition shows that

$$\nabla f(\mathbf{w}^*) = \mathbf{0}. \tag{47}$$

From (35) and (47),

$$\|\mathbf{w}^k - \mathbf{w}^*\|_2 \le \|\nabla f(\mathbf{w}^k) - \nabla f(\mathbf{w}^*)\|_2 = \|\nabla f(\mathbf{w}^k)\|_2. \tag{48}$$

With (46),

$$\|\mathbf{w}^{k+1} - \mathbf{w}^k\|_2 \ge \tau \|\mathbf{w}^k - \mathbf{w}^*\|_2, \quad \text{where } \tau = \frac{\gamma}{\sqrt{n}(1+\gamma+2\gamma CP^2(\#\text{nz}))}. \tag{49}$$

From (12) and (49),

$$\begin{aligned} f(\mathbf{w}^k) - f(\mathbf{w}^{k+1}) &= \sum_{i=1}^{n} (f(\mathbf{w}^{k,i}) - f(\mathbf{w}^{k,i+1})) \\ &\ge \sum_{i=1}^{n} \sigma (w^{k,i+1}_{\pi_k(i)} - w^{k,i}_{\pi_k(i)})^2 = \sigma \|\mathbf{w}^{k+1} - \mathbf{w}^k\|_2^2 \ge \sigma\tau^2 \|\mathbf{w}^k - \mathbf{w}^*\|_2^2. \end{aligned}$$

By (36) and (47),

$$f(\mathbf{w}^k) - f(\mathbf{w}^*) \le \frac{K}{2} \|\mathbf{w}^k - \mathbf{w}^*\|_2^2, \tag{50}$$

where $K$ is defined in (37). Therefore, we have

$$f(\mathbf{w}^k) - f(\mathbf{w}^{k+1}) \ge \frac{2\sigma\tau^2}{K} (f(\mathbf{w}^k) - f(\mathbf{w}^*)).$$

This is equivalent to

$$(f(\mathbf{w}^k) - f(\mathbf{w}^*)) + (f(\mathbf{w}^*) - f(\mathbf{w}^{k+1})) \ge \frac{2\sigma\tau^2}{K} (f(\mathbf{w}^k) - f(\mathbf{w}^*)).$$

Finally, we have

$$f(\mathbf{w}^{k+1}) - f(\mathbf{w}^*) \le (1 - \frac{2\sigma\tau^2}{K})(f(\mathbf{w}^k) - f(\mathbf{w}^*)). \tag{51}$$

With $\tau \le 1$ from (49), $K \ge 1$ from (37) and $\sigma < 1/2$, we have $2\sigma\tau^2/K < 1$. Hence, (51) ensures that $f(\mathbf{w}^k)$ approaches $f(\mathbf{w}^*)$.

From (51), $\{f(\mathbf{w}^k)\}$ converges to $f(\mathbf{w}^*)$. We can then prove that $\{\mathbf{w}^k\}$ globally converges to $\mathbf{w}^*$. If this result does not hold, there is a sub-sequence $\{\mathbf{w}^k\}_M$ converging to a point $\bar{\mathbf{w}} \ne \mathbf{w}^*$. However, Lemma 6 shows that $f(\bar{\mathbf{w}}) > f(\mathbf{w}^*)$, so $\lim_{k \in M} f(\mathbf{w}^k) > f(\mathbf{w}^*)$, a contradiction.

Let $\mu = 2\sigma\tau^2/K$, (51) implies

$$f(\mathbf{w}^k) - f(\mathbf{w}^*) \le (1-\mu)^k (f(\mathbf{w}^0) - f(\mathbf{w}^*)), \ \forall k.$$

To achieve an $\varepsilon$-accurate solution, we need the right-hand side to be smaller than $\varepsilon$. Thus,

$$k \ge \frac{\log(f(\mathbf{w}^0) - f(\mathbf{w}^*)) + \log(1/\varepsilon)}{-\log(1-\mu)}.$$

From the inequality

$$\log(1-x) \leq -x \text{ if } x < 1,$$

we have

$$k \geq \frac{\log(f(\mathbf{w}^0) - f(\mathbf{w}^*)) + \log(1/\varepsilon)}{\mu}. \tag{52}$$

In following we discuss the order of $\mu^{-1} = \frac{K}{2\sigma\tau^2}$. From (37),

$$K = 2C\|X\|_2^2 + 1 \leq 2C\|X\|_F^2 + 1 \leq 2CP^2(\#\text{nz}) + 1, \tag{53}$$

where $\|\cdot\|_F$ is the Frobenious norm. From (49),

$$\tau^{-1} = \sqrt{n}\left(\frac{1}{\gamma} + 2CP^2(\#\text{nz}) + 1\right).$$

Since $\gamma = \frac{\beta}{\sigma + H/2}$, from (14) and (42) we have

$$\gamma^{-1} = O(lCP^2 + 1) \leq O(CP^2(\#\text{nz}) + 1). \tag{54}$$

As #nz is usually large, we omit the constant term $O(1)$ in the following discussion. Then $\tau^{-1} = O(\sqrt{n}CP^2(\#\text{nz}))$. Thus,

$$\mu^{-1} = \frac{K}{2\sigma\tau^2} = O(nC^3P^6(\#\text{nz})^3). \tag{55}$$

From (52) and (55), Algorithm 1 obtains an $\varepsilon$-accurate solution in

$$O\left(nC^3P^6(\#\text{nz})^3\log(1/\varepsilon)\right)$$

iterations. ■

### A.4 Proof of Theorem 4 (Linear Convergence of the Online Setting)

**Proof** To begin, we denote the expectation value of a function $g$ of a random variable $y$ to be

$$E_y(g(y)) = \sum_y P(y)g(y).$$

Then for any vector $\mathbf{s} \in R^n$ and a random variable $I$ where

$$P(I = i) = \frac{1}{n},\ \forall i \in \{1, \ldots, n\},$$

we have

$$E_I(s_I^2) = \sum_{i=1}^{n} \frac{s_i^2}{n} = \frac{1}{n}\|\mathbf{s}\|_2^2. \tag{56}$$

At each iteration $k$ ($k = 0, 1, \ldots$) of Algorithm 3, we randomly choose one index $i_k$ and update $\mathbf{w}^k$ to $\mathbf{w}^{k+1}$. The expected function value after iteration $k$ can be represented as

$$E_{i_0,\ldots,i_{k-1},i_k}(f(\mathbf{w}^{k+1})).$$

From (12), (43), (56), (48), and (50), we have

$$\begin{aligned}
& E_{i_0,\ldots,i_{k-1}} E_{i_k}(f(\mathbf{w}^k) - f(\mathbf{w}^{k+1})) \\
\geq\ & \sigma E_{i_0,\ldots,i_{k-1}} E_{i_k}(|w_{i_k}^{k+1} - w_{i_k}^k|^2) \\
\geq\ & \sigma\gamma^2 E_{i_0,\ldots,i_{k-1}} E_{i_k}(|\nabla f(\mathbf{w}^k)_{i_k}|^2) \\
=\ & \frac{\sigma\gamma^2}{n} E_{i_0,\ldots,i_{k-1}}(\|\nabla f(\mathbf{w}^k)\|_2^2) \\
\geq\ & \frac{\sigma\gamma^2}{n} E_{i_0,\ldots,i_{k-1}}(\|\mathbf{w}^k - \mathbf{w}^*\|_2^2) \\
\geq\ & \frac{2\sigma\gamma^2}{nK} E_{i_0,\ldots,i_{k-1}}(f(\mathbf{w}^k) - f(\mathbf{w}^*)).
\end{aligned}$$

This is equivalent to

$$E_{i_0,\ldots,i_k}(f(\mathbf{w}^{k+1})) - f(\mathbf{w}^*) \leq \left(1 - \frac{2\sigma\gamma^2}{nK}\right)\left(E_{i_0,\ldots,i_{k-1}}(f(\mathbf{w}^k)) - f(\mathbf{w}^*)\right).$$

From Markov inequality $P(|Z| \geq a) \leq E(|Z|)/a$ for any random variable $Z$ and the fact $f(\mathbf{w}^{k+1}) \geq f(\mathbf{w}^*)$, we have

$$P\left(f(\mathbf{w}^k) - f(\mathbf{w}^*) \geq \varepsilon\right) \leq E\left(f(\mathbf{w}^k) - f(\mathbf{w}^*)\right)/\varepsilon. \tag{57}$$

To achieve an $\varepsilon$-accurate solution with confidence $1-\delta$, we need the right-hand side of (57) to be less than $\delta$. This indicates the iteration number $k$ must satisfy

$$E_{i_0,\ldots,i_{k-1}}(f(\mathbf{w}^k)) - f(\mathbf{w}^*) \leq (f(\mathbf{w}^0) - f(\mathbf{w}^*))\left(1 - \frac{2\sigma\gamma^2}{nK}\right)^k \leq \varepsilon\delta.$$

By a derivation similar to Theorem 3, we can show that after $O(\frac{nK}{\sigma\gamma^2}\log(\frac{1}{\delta\varepsilon}))$ iterations, with confidence $1-\delta$, we obtain an $\varepsilon$-accurate solution. From (53) and (54), we have

$$K \leq 2CP^2(\#\text{nz}) + 1 \text{ and } \gamma^{-1} = O(lCP^2 + 1).$$

Therefore,

$$O\left(\frac{nK}{\sigma\gamma^2}\log(\frac{1}{\delta\varepsilon})\right) = O\left(nl^2C^3P^6(\#\text{nz})\log(\frac{1}{\delta\varepsilon})\right).$$

■

## References

Dimitri P. Bertsekas. *Nonlinear Programming*. Athena Scientific, Belmont, MA 02178-9998, second edition, 1999.

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.

Leon Bottou. Stochastic gradient descent examples, 2007. http://leon.bottou.org/projects/sgd.

Leon Bottou and Olivier Bousquet. The tradeoffs of large scale 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.

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

Luigi Grippo and Marco Sciandrone. Globally convergent block-coordinate techniques for unconstrained optimization. *Optimization Methods and Software*, 10:587–637, 1999.

Cho-Jui Hsieh, Kai-Wei Chang, Chih-Jen Lin, S. Sathiya Keerthi, and Sellamanickam Sundararajan. A dual coordinate descent method for large-scale linear SVM. In *Proceedings of the Twenty Fifth International Conference on Machine Learning (ICML)*, 2008. URL http://www.csie.ntu.edu.tw/~cjlin/papers/cddual.pdf. Software available at http://www.csie.ntu.edu.tw/~cjlin/liblinear.

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

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.

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, Ruby C. Weng, and S. Sathiya Keerthi. Trust region Newton method for large-scale logistic regression. *Journal of Machine Learning Research*, 9:627–650. URL http://www.csie.ntu.edu.tw/~cjlin/papers/logistic.pdf. Software available at http://www.csie.ntu.edu.tw/~cjlin/liblinear.

Zhi-Quan Luo and Paul Tseng. On the convergence of coordinate descent method for convex differentiable minimization. *Journal of Optimization Theory and Applications*, 72(1):7–35, 1992.

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

Dukík Miroslav, Steven J. Phillips, and Robert E. Schapire. Performance guarantees for regularized maximum entropy density estimation. In *Proceedings of the 17th Annual Conference on Computational Learning Theory*, pages 655–662, New York, 2004. ACM press.

Gunnar Rätsch, Sebastian Mika, and Manfred K. Warmuth. On the convergence of leveraging. In T. G. Dietterich, S. Becker, and Z. Ghahramani, editors, *Advances in Neural Information Processing Systems 14*, pages 487–494. MIT Press, Cambridge, MA, 2002.

Nicol N. Schraudolph. A fast, compact approximation of the exponential function. *Neural Computation*, 11:853–862, 1999.

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.

Tong Zhang. Solving large scale linear prediction problems using stochastic gradient descent algorithms. In *Proceedings of the 21th International Conference on Machine Learning (ICML)*, 2004.

Tong Zhang and Frank J. Oles. Text categorization based on regularized linear classification methods. *Information Retrieval*, 4(1):5–31, 2001.