A Dual Coordinate Descent Method for Large-scale Linear SVM

Cho-Jui Hsieh B92085@CSIE.NTU.EDU.TW
Kai-Wei Chang B92084@CSIE.NTU.EDU.TW
Chih-Jen Lin CJLIN@CSIE.NTU.EDU.TW
Department of Computer Science, National Taiwan University, Taipei 106, Taiwan
S. Sathiya Keerthi SELVARAK@YAHOO-INC.COM
Yahoo! Research, Santa Clara, USA
S. Sundararajan SSRAJAN@YAHOO-INC.COM
Yahoo! Labs, Bangalore, India

Abstract

In many applications, data appear with a huge number of instances as well as features. Linear Support Vector Machines (SVM) is one of the most popular tools to deal with such large-scale sparse data. This paper presents a novel dual coordinate descent method for linear SVM with L1- and L2-loss functions. The proposed method is simple and reaches an ϵ-accurate solution in $O(\log(1/\epsilon))$ iterations. Experiments indicate that our method is much faster than state of the art solvers such as Pegasos, TRON, SVM$^{\text{perf}}$, and a recent primal coordinate descent implementation.

1. Introduction

Support vector machines (SVM) (Boser et al., 1992) are useful for data classification. Given a set of instance-label pairs $(\boldsymbol{x}_i, y_i), i = 1, \ldots, l,\ \boldsymbol{x}_i \in R^n,\ y_i \in \{-1, +1\}$, SVM requires the solution of the following unconstrained optimization problem:

$$\min_{\boldsymbol{w}} \quad \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w} + C\sum_{i=1}^{l}\xi(\boldsymbol{w}; \boldsymbol{x}_i, y_i), \tag{1}$$

where $\xi(\boldsymbol{w}; \boldsymbol{x}_i, y_i)$ is a loss function, and $C \geq 0$ is a penalty parameter. Two common loss functions are:

$$\max(1 - y_i\boldsymbol{w}^T\boldsymbol{x}_i, 0) \text{ and } \max(1 - y_i\boldsymbol{w}^T\boldsymbol{x}_i, 0)^2. \tag{2}$$

The former is called L1-SVM, while the latter is L2-SVM. In some applications, an SVM problem appears with a bias term b. One often deal with this term by appending each instance with an additional dimension:

$$\boldsymbol{x}_i^T \leftarrow [\boldsymbol{x}_i^T, 1] \qquad \boldsymbol{w}^T \leftarrow [\boldsymbol{w}^T, b]. \tag{3}$$

Problem (1) is often referred to as the primal form of SVM. One may instead solve its dual problem:

$$\begin{aligned} \min_{\boldsymbol{\alpha}} \quad & f(\boldsymbol{\alpha}) = \frac{1}{2}\boldsymbol{\alpha}^T\bar{Q}\boldsymbol{\alpha} - \boldsymbol{e}^T\boldsymbol{\alpha} \\ \text{subject to} \quad & 0 \leq \alpha_i \leq U, \forall i, \end{aligned} \tag{4}$$

where $\bar{Q} = Q + D$, D is a diagonal matrix, and $Q_{ij} = y_i y_j \boldsymbol{x}_i^T\boldsymbol{x}_j$. For L1-SVM, $U = C$ and $D_{ii} = 0,\ \forall i$. For L2-SVM, $U = \infty$ and $D_{ii} = 1/(2C),\ \forall i$.

An SVM usually maps training vectors into a high-dimensional space via a nonlinear function $\phi(\boldsymbol{x})$. Due to the high dimensionality of the vector variable $\boldsymbol{w}$, one solves the dual problem (4) by the kernel trick (i.e., using a closed form of $\phi(\boldsymbol{x}_i)^T\phi(\boldsymbol{x}_j)$). We call such a problem as a nonlinear SVM. In some applications, data appear in a rich dimensional feature space, the performances are similar with/without nonlinear mapping. If data are not mapped, we can often train much larger data sets. We indicate such cases as linear SVM; these are often encountered in applications such as document classification. In this paper, we aim at solving very large linear SVM problems.

Recently, many methods have been proposed for linear SVM in large-scale scenarios. For L1-SVM, Zhang (2004), Shalev-Shwartz et al. (2007), Bottou (2007) propose various stochastic gradient descent methods. Collins et al. (2008) apply an exponentiated gradient method. SVM$^{\text{perf}}$ (Joachims, 2006) uses a cutting plane technique. Smola et al. (2008) apply bundle methods, and view SVM$^{\text{perf}}$ as a special case. For L2-SVM, Keerthi and DeCoste (2005) propose modified Newton methods. A trust region Newton method (TRON) (Lin et al., 2008) is proposed for logistic re-

Appearing in *Proceedings of the 25th International Conference on Machine Learning*, Helsinki, Finland, 2008.

gression and L2-SVM. These algorithms focus on different aspects of the training speed. Some aim at quickly obtaining a usable model, but some achieve fast final convergence of solving the optimization problem in (1) or (4). Moreover, among these methods, Joachims (2006), Smola et al. (2008) and Collins et al. (2008) solve SVM via the dual (4). Others consider the primal form (1). The decision of using primal or dual is of course related to the algorithm design.

Very recently, Chang et al. (2007) propose using coordinate descent methods for solving primal L2-SVM. Experiments show that their approach more quickly obtains a useful model than some of the above methods. Coordinate descent, a popular optimization technique, updates one variable at a time by minimizing a single-variable sub-problem. If one can efficiently solve this sub-problem, then it can be a competitive optimization method. Due to the non-differentiability of the primal L1-SVM, Chang et al's work is restricted to L2-SVM. Moreover, as primal L2-SVM is differentiable but not twice differentiable, certain considerations are needed in solving the single-variable sub-problem.

While the dual form (4) involves bound constraints $0 \le \alpha_i \le U$, its objective function is twice differentiable for both L1- and L2-SVM. In this paper, we investigate coordinate descent methods for the dual problem (4). We prove that an ϵ-optimal solution is obtained in $O(\log(1/\epsilon))$ iterations. We propose an implementation using a random order of sub-problems at each iteration, which leads to very fast training. Experiments indicate that our method is more efficient than the primal coordinate descent method. As Chang et al. (2007) solve the primal, they require the easy access of a feature's corresponding data values. However, in practice one often has an easier access of values per instance. Solving the dual takes this advantage, so our implementation is simpler than Chang et al. (2007).

Early SVM papers (Mangasarian & Musicant, 1999; Friess et al., 1998) have discussed coordinate descent methods for the SVM dual form. However, they do not focus on large data using the linear kernel. Crammer and Singer (2003) proposed an online setting for multi-class SVM without considering large sparse data. Recently, Bordes et al. (2007) applied a coordinate descent method to multi-class SVM, but they focus on nonlinear kernels. In this paper, we point out that *dual coordinate descent methods make crucial advantage of the linear kernel and outperform other solvers when the numbers of data and features are both large.*

Coordinate descent methods for (4) are related to the popular decomposition methods for training nonlinear SVM. In this paper, we show their key differences and explain why earlier studies on decomposition methods failed to modify their algorithms in an efficient way like ours for large-scale linear SVM. We also discuss the connection to other linear SVM works such as (Crammer & Singer, 2003; Collins et al., 2008; Shalev-Shwartz et al., 2007).

This paper is organized as follows. In Section 2, we describe our proposed algorithm. Implementation issues are investigated in Section 3. Section 4 discusses the connection to other methods. In Section 5, we compare our method with state of the art implementations for large linear SVM. Results show that the new method is more efficient. Proofs can be found at `http://www.csie.ntu.edu.tw/~cjlin/papers/cddual.pdf`.

2. A Dual Coordinate Descent Method

In this section, we describe our coordinate descent method for L1- and L2-SVM. The optimization process starts from an initial point $\boldsymbol{\alpha}^0 \in R^l$ and generates a sequence of vectors $\{\boldsymbol{\alpha}^k\}_{k=0}^{\infty}$. We refer to the process from $\boldsymbol{\alpha}^k$ to $\boldsymbol{\alpha}^{k+1}$ as an outer iteration. In each outer iteration we have l inner iterations, so that sequentially $\alpha_1, \alpha_2, \ldots, \alpha_l$ are updated. Each outer iteration thus generates vectors $\boldsymbol{\alpha}^{k,i} \in R^l$, $i = 1, \ldots, l+1$, such that $\boldsymbol{\alpha}^{k,1} = \boldsymbol{\alpha}^k$, $\boldsymbol{\alpha}^{k,l+1} = \boldsymbol{\alpha}^{k+1}$, and

$$\boldsymbol{\alpha}^{k,i} = [\alpha_1^{k+1}, \ldots, \alpha_{i-1}^{k+1}, \alpha_i^k, \ldots, \alpha_l^k]^T, \ \forall i = 2, \ldots, l.$$

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

$$\min_d \ f(\boldsymbol{\alpha}^{k,i} + d\boldsymbol{e}_i) \quad \text{subject to} \quad 0 \le \alpha_i^k + d \le U, \quad (5)$$

where $\boldsymbol{e}_i = [0, \ldots, 0, 1, 0, \ldots, 0]^T$. The objective function of (5) is a simple quadratic function of d:

$$f(\boldsymbol{\alpha}^{k,i} + d\boldsymbol{e}_i) = \frac{1}{2}\bar{Q}_{ii}d^2 + \nabla_i f(\boldsymbol{\alpha}^{k,i})d + \text{constant}, \quad (6)$$

where $\nabla_i f$ is the ith component of the gradient ∇f. One can easily see that (5) has an optimum at $d = 0$ (i.e., no need to update α_i) if and only if

$$\nabla_i^P f(\boldsymbol{\alpha}^{k,i}) = 0, \quad (7)$$

where $\nabla^P f(\boldsymbol{\alpha})$ means the projected gradient

$$\nabla_i^P f(\boldsymbol{\alpha}) = \begin{cases} \nabla_i f(\boldsymbol{\alpha}) & \text{if } 0 < \alpha_i < U, \\ \min(0, \nabla_i f(\boldsymbol{\alpha})) & \text{if } \alpha_i = 0, \\ \max(0, \nabla_i f(\boldsymbol{\alpha})) & \text{if } \alpha_i = U. \end{cases} \quad (8)$$

If (7) holds, we move to the index $i+1$ without updating $\alpha_i^{k,i}$. Otherwise, we must find the optimal solution of (5). If $\bar{Q}_{ii} > 0$, easily the solution is:

$$\alpha_i^{k,i+1} = \min\left(\max\left(\alpha_i^{k,i} - \frac{\nabla_i f(\boldsymbol{\alpha}^{k,i})}{\bar{Q}_{ii}}, 0\right), U\right). \quad (9)$$

Algorithm 1 A dual coordinate descent method for Linear SVM

- Given $\boldsymbol{\alpha}$ and the corresponding $\boldsymbol{w} = \sum_i y_i \alpha_i \boldsymbol{x}_i$.
- While $\boldsymbol{\alpha}$ is not optimal

  For $i = 1, \ldots, l$

  (a) $\bar{\alpha}_i \leftarrow \alpha_i$

  (b) $G = y_i \boldsymbol{w}^T \boldsymbol{x}_i - 1 + D_{ii}\alpha_i$

  (c)
  $$PG = \begin{cases} \min(G, 0) & \text{if } \alpha_i = 0, \\ \max(G, 0) & \text{if } \alpha_i = U, \\ G & \text{if } 0 < \alpha_i < U \end{cases}$$

  (d) If $|PG| \neq 0$,
  $\alpha_i \leftarrow \min(\max(\alpha_i - G/\bar{Q}_{ii}, 0), U)$
  $\boldsymbol{w} \leftarrow \boldsymbol{w} + (\alpha_i - \bar{\alpha}_i) y_i \boldsymbol{x}_i$

We thus need to calculate $\bar{Q}_{ii}$ and $\nabla_i f(\boldsymbol{\alpha}^{k,i})$. First, $\bar{Q}_{ii} = \boldsymbol{x}_i^T \boldsymbol{x}_i + D_{ii}$ can be precomputed and stored in the memory. Second, to evaluate $\nabla_i f(\boldsymbol{\alpha}^{k,i})$, we use

$$\nabla_i f(\boldsymbol{\alpha}) = (\bar{Q}\boldsymbol{\alpha})_i - 1 = \sum_{j=1}^{l} \bar{Q}_{ij}\alpha_j - 1. \quad (10)$$

$\bar{Q}$ may be too large to be stored, so one calculates $\bar{Q}$'s ith row when doing (10). If $\bar{n}$ is the average number of nonzero elements per instance, and $O(\bar{n})$ is needed for each kernel evaluation, then calculating the ith row of the kernel matrix takes $O(l\bar{n})$. Such operations are expensive. However, for a linear SVM, we can define

$$\boldsymbol{w} = \sum_{j=1}^{l} y_j \alpha_j \boldsymbol{x}_j, \quad (11)$$

so (10) becomes

$$\nabla_i f(\boldsymbol{\alpha}) = y_i \boldsymbol{w}^T \boldsymbol{x}_i - 1 + D_{ii}\alpha_i. \quad (12)$$

To evaluate (12), the main cost is $O(\bar{n})$ for calculating $\boldsymbol{w}^T \boldsymbol{x}_i$. This is much smaller than $O(l\bar{n})$. To apply (12), $\boldsymbol{w}$ must be maintained throughout the coordinate descent procedure. Calculating $\boldsymbol{w}$ by (11) takes $O(l\bar{n})$ operations, which are too expensive. Fortunately, if $\bar{\alpha}_i$ is the current value and α_i is the value after the updating, we can maintain $\boldsymbol{w}$ by

$$\boldsymbol{w} \leftarrow \boldsymbol{w} + (\alpha_i - \bar{\alpha}_i) y_i \boldsymbol{x}_i. \quad (13)$$

The number of operations is only $O(\bar{n})$. To have the first $\boldsymbol{w}$, one can use $\boldsymbol{\alpha}^0 = \boldsymbol{0}$ so $\boldsymbol{w} = \boldsymbol{0}$. In the end, we obtain the optimal $\boldsymbol{w}$ of the primal problem (1) as the primal-dual relationship implies (11).

If $\bar{Q}_{ii} = 0$, we have $D_{ii} = 0$, $Q_{ii} = \boldsymbol{x}_i^T \boldsymbol{x}_i = 0$, and hence $\boldsymbol{x}_i = \boldsymbol{0}$. This occurs only in L1-SVM without the bias term by (3). From (12), if $\boldsymbol{x}_i = \boldsymbol{0}$, then $\nabla_i f(\boldsymbol{\alpha}^{k,i}) = -1$. As $U = C < \infty$ for L1-SVM, the solution of (5) makes the new $\alpha_i^{k,i+1} = U$. We can easily include this case in (9) by setting $1/\bar{Q}_{ii} = \infty$.

Briefly, our algorithm uses (12) to compute $\nabla_i f(\boldsymbol{\alpha}^{k,i})$, checks the optimality of the sub-problem (5) by (7), updates α_i by (9), and then maintains $\boldsymbol{w}$ by (13). A description is in Algorithm 1. The cost per iteration (i.e., from $\boldsymbol{\alpha}^k$ to $\boldsymbol{\alpha}^{k+1}$) is $O(l\bar{n})$. The main memory requirement is on storing $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_l$. For the convergence, we prove the following theorem using techniques in (Luo & Tseng, 1992):

Theorem 1 *For L1-SVM and L2-SVM, $\{\boldsymbol{\alpha}^{k,i}\}$ generated by Algorithm 1 globally converges to an optimal solution $\boldsymbol{\alpha}^*$. The convergence rate is at least linear: there are $0 < \mu < 1$ and an iteration k_0 such that*

$$f(\boldsymbol{\alpha}^{k+1}) - f(\boldsymbol{\alpha}^*) \leq \mu(f(\boldsymbol{\alpha}^k) - f(\boldsymbol{\alpha}^*)), \forall k \geq k_0. \quad (14)$$

The global convergence result is quite remarkable. Usually for a convex but not strictly convex problem (e.g., L1-SVM), one can only obtain that any limit point is optimal. We define an ϵ-accurate solution $\boldsymbol{\alpha}$ if $f(\boldsymbol{\alpha}) \leq f(\boldsymbol{\alpha}^*) + \epsilon$. By (14), our algorithm obtains an ϵ-accurate solution in $O(\log(1/\epsilon))$ iterations.

3. Implementation Issues

3.1. Random Permutation of Sub-problems

In Algorithm 1, the coordinate descent algorithm solves the one-variable sub-problems in the order of $\alpha_1, \ldots, \alpha_l$. Past results such as (Chang et al., 2007) show that solving sub-problems in an arbitrary order may give faster convergence. This inspires us to randomly permute the sub-problems at each outer iteration. Formally, at the kth outer iteration, we permute $\{1, \ldots, l\}$ to $\{\pi(1), \ldots, \pi(l)\}$, and solve sub-problems in the order of $\alpha_{\pi(1)}, \alpha_{\pi(2)}, \ldots, \alpha_{\pi(l)}$. Similar to Algorithm 1, the algorithm generates a sequence $\{\boldsymbol{\alpha}^{k,i}\}$ such that $\boldsymbol{\alpha}^{k,1} = \boldsymbol{\alpha}^k$, $\boldsymbol{\alpha}^{k,l+1} = \boldsymbol{\alpha}^{k+1,1}$ and

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

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

$$\alpha_t^{k,i+1} = \alpha_t^{k,i} + \arg\min_{0 \leq \alpha_t^{k,i} + d \leq U} f(\boldsymbol{\alpha}^{k,i} + d\boldsymbol{e}_t) \text{ if } \pi_k^{-1}(t) = i.$$

We prove that Theorem 1 is still valid. Hence, the new setting obtains an ϵ-accurate solution in $O(\log(1/\epsilon))$ iterations. A simple experiment reveals that this setting of permuting sub-problems is much faster than Algorithm 1. The improvement is also bigger than that observed in (Chang et al., 2007) for primal coordinate descent methods.

Algorithm 2 Coordinate descent algorithm with randomly selecting one instance at a time

- Given $\boldsymbol{\alpha}$ and the corresponding $\boldsymbol{w} = \sum_i y_i \alpha_i \boldsymbol{x}_i$.
- While $\boldsymbol{\alpha}$ is not optimal
  - Randomly choose $i \in \{1, \ldots, l\}$.
  - Do steps (a)-(d) of Algorithm 1 to update α_i.

3.2. Shrinking

Eq. (4) contains constraints $0 \le \alpha_i \le U$. If an α_i is 0 or U for many iterations, it may remain the same. To speed up decomposition methods for nonlinear SVM (discussed in Section 4.1), the shrinking technique (Joachims, 1998) reduces the size of the optimization problem without considering some bounded variables. Below we show it is much easier to apply this technique to linear SVM than the nonlinear case.

If A is the subset after removing some elements and $\bar{A} = \{1, \ldots, l\} \setminus A$, then the new problem is

$$\begin{aligned} \min_{\boldsymbol{\alpha}_A} \quad & \frac{1}{2}\boldsymbol{\alpha}_A^T \bar{Q}_{AA}\boldsymbol{\alpha}_A + (\bar{Q}_{A\bar{A}}\boldsymbol{\alpha}_{\bar{A}} - \boldsymbol{e}_A)^T\boldsymbol{\alpha}_A \\ \text{subject to} \quad & 0 \le \alpha_i \le U, i \in A, \end{aligned} \tag{15}$$

where $\bar{Q}_{AA}, \bar{Q}_{A\bar{A}}$ are sub-matrices of $\bar{Q}$, and $\boldsymbol{\alpha}_{\bar{A}}$ is considered as a constant vector. Solving this smaller problem consumes less time and memory. Once (15) is solved, we must check if the vector $\boldsymbol{\alpha}$ is optimal for (4). This check needs the whole gradient $\nabla f(\boldsymbol{\alpha})$. Since

$$\nabla_i f(\boldsymbol{\alpha}) = \bar{Q}_{i,A}\boldsymbol{\alpha}_A + \bar{Q}_{i,\bar{A}}\boldsymbol{\alpha}_{\bar{A}} - 1,$$

if $i \in A$, and one stores $\bar{Q}_{i,\bar{A}}\boldsymbol{\alpha}_{\bar{A}}$ before solving (15), we already have $\nabla_i f(\boldsymbol{\alpha})$. However, for all $i \notin A$, we must calculate the corresponding rows of $\bar{Q}$. This step, referred to as the reconstruction of gradients in training nonlinear SVM, is very time consuming. It may cost up to $O(l^2\bar{n})$ if each kernel evaluation is $O(\bar{n})$.

For linear SVM, in solving the smaller problem (15), we still have the vector

$$\boldsymbol{w} = \sum\nolimits_{i\in A} y_i\alpha_i\boldsymbol{x}_i + \sum\nolimits_{i\in\bar{A}} y_i\alpha_i\boldsymbol{x}_i$$

though only the first part $\sum_{i\in A} y_i\alpha_i\boldsymbol{x}_i$ is updated. Therefore, using (12), $\nabla f(\boldsymbol{\alpha})$ is easily available. Below we demonstrate a shrinking implementation so that reconstructing the whole $\nabla f(\boldsymbol{\alpha})$ is never needed.

Our method is related to what LIBSVM (Chang & Lin, 2001) uses. From the optimality condition of bound-constrained problems, $\boldsymbol{\alpha}$ is optimal for (4) if and only if $\nabla^P f(\boldsymbol{\alpha}) = \boldsymbol{0}$, where $\nabla^P f(\boldsymbol{\alpha})$ is the projected gradient defined in (8). We then prove the following result:

Theorem 2 *Let $\boldsymbol{\alpha}^*$ be the convergent point of $\{\boldsymbol{\alpha}^{k,i}\}$.*

1. *If $\alpha_i^* = 0$ and $\nabla_i f(\boldsymbol{\alpha}^*) > 0$, then $\exists k_i$ such that $\forall k \ge k_i,\ \forall s, \alpha_i^{k,s} = 0$.*
2. *If $\alpha_i^* = U$ and $\nabla_i f(\boldsymbol{\alpha}^*) < 0$, then $\exists k_i$ such that $\forall k \ge k_i,\ \forall s,\ \alpha_i^{k,s} = U$.*
3. $\lim_{k\to\infty} \max_j \nabla_j^P f(\boldsymbol{\alpha}^{k,j}) = \lim_{k\to\infty} \min_j \nabla_j^P f(\boldsymbol{\alpha}^{k,j}) = 0$.

During the optimization procedure, $\nabla^P f(\boldsymbol{\alpha}^k) \ne \boldsymbol{0}$, so in general $\max_j \nabla_j^P f(\boldsymbol{\alpha}^k) > 0$ and $\min_j \nabla_j^P f(\boldsymbol{\alpha}^k) < 0$. These two values measure how the current solution violates the optimality condition. In our iterative procedure, what we have are $\nabla_i f(\boldsymbol{\alpha}^{k,i})$, $i = 1, \ldots, l$. Hence, at the $(k-1)$st iteration, we obtain

$$M^{k-1} \equiv \max_j \nabla_j^P f(\boldsymbol{\alpha}^{k-1,j}), m^{k-1} \equiv \min_j \nabla_j^P f(\boldsymbol{\alpha}^{k-1,j}).$$

Then at each inner step of the kth iteration, before updating $\alpha_i^{k,i}$ to $\alpha_i^{k,i+1}$, this element is shrunken if one of the following two conditions holds:

$$\begin{aligned} &\alpha_i^{k,i} = 0 \text{ and } \nabla_i f(\boldsymbol{\alpha}^{k,i}) > \bar{M}^{k-1}, \\ &\alpha_i^{k,i} = U \text{ and } \nabla_i f(\boldsymbol{\alpha}^{k,i}) < \bar{m}^{k-1}, \end{aligned} \tag{16}$$

where

$$\bar{M}^{k-1} = \begin{cases} M^{k-1} & \text{if } M^{k-1} > 0, \\ \infty & \text{otherwise,} \end{cases}$$

$$\bar{m}^{k-1} = \begin{cases} m^{k-1} & \text{if } m^{k-1} < 0 \\ -\infty & \text{otherwise.} \end{cases}$$

In (16), $\bar{M}^{k-1}$ must be strictly positive, so we set it be ∞ if $M^{k-1} = 0$. From Theorem 2, elements satisfying the "if condition" of properties 1 and 2 meet (16) after certain iterations, and are then correctly removed for optimization. To have a more aggressive shrinking, one may multiply both $\bar{M}^{k-1}$ and $\bar{m}^{k-1}$ in (16) by a threshold smaller than one.

Property 3 of Theorem 2 indicates that with a tolerance ϵ,

$$M^k - m^k < \epsilon \tag{17}$$

is satisfied after a finite number of iterations. Hence (17) is a valid stopping condition. We also use it for smaller problems (15). If at the kth iteration, (17) for (15) is reached, we enlarge A to $\{1, \ldots, l\}$, set $\bar{M}^k = \infty, \bar{m}^k = -\infty$ (so no shrinking at the $(k+1)$st iteration), and continue regular iterations. Thus, we do shrinking without reconstructing gradients.

3.3. An Online Setting

In some applications, the number of instances is huge, so going over all $\alpha_1, \ldots, \alpha_l$ causes an expensive outer iteration. Instead, one can randomly choose an index i_k at a time, and update only α_{i_k} at the kth outer iteration. A description is in Algorithm 2. The setting is related to (Crammer & Singer, 2003; Collins et al., 2008). See also the discussion in Section 4.2.

Table 1. A comparison between decomposition methods (Decomp.) and dual coordinate descent (DCD). For both methods, we consider that one α_i is updated at a time. We assume Decomp. maintains gradients, but DCD does not. The average number of nonzeros per instance is $\bar{n}$.

	Nonlinear SVM		Linear SVM	
	Decomp.	DCD	Decomp.	DCD
Update α_i	$O(1)$	$O(l\bar{n})$	$O(1)$	$O(\bar{n})$
Maintain $\nabla f(\boldsymbol{\alpha})$	$O(l\bar{n})$	NA	$O(l\bar{n})$	NA

4. Relations with Other Methods

4.1. Decomposition Methods for Nonlinear SVM

Decomposition methods are one of the most popular approaches for training nonlinear SVM. As the kernel matrix is dense and cannot be stored in the computer memory, decomposition methods solve a sub-problem of few variables at each iteration. Only a small number of corresponding kernel columns are needed, so the memory problem is resolved. If the number of variables is restricted to one, a decomposition method is like the online coordinate descent in Section 3.3, but it differs in the way it selects variables for updating. It has been shown (Keerthi & DeCoste, 2005) that, for linear SVM decomposition methods are inefficient. On the other hand, here we are pointing out that dual coordinate descent is efficient for linear SVM. Therefore, it is important to discuss the relationship between decomposition methods and our method.

In early decomposition methods that were first proposed (Osuna et al., 1997; Platt, 1998), variables minimized at an iteration are selected by certain heuristics. However, subsequent developments (Joachims, 1998; Chang & Lin, 2001; Keerthi et al., 2001) all use gradient information to conduct the selection. The main reason is that maintaining the whole gradient does not introduce extra cost. Here we explain the detail by assuming that one variable of $\boldsymbol{\alpha}$ is chosen and updated at a time[1]. To set-up and solve the sub-problem (6), one uses (10) to calculate $\nabla_i f(\boldsymbol{\alpha})$. If $O(\bar{n})$ effort is needed for each kernel evaluation, obtaining the ith row of the kernel matrix takes $O(l\bar{n})$ effort. If instead one maintains the whole gradient, then $\nabla_i f(\boldsymbol{\alpha})$ is directly available. After updating $\alpha_i^{k,i}$ to $\alpha_i^{k,i+1}$, we obtain $\bar{Q}$'s ith column (same as the ith row due to the symmetry of $\bar{Q}$), and calculate the new whole gradient:

$$\nabla f(\boldsymbol{\alpha}^{k,i+1}) = \nabla f(\boldsymbol{\alpha}^{k,i}) + \bar{Q}_{:,i}(\alpha_i^{k,i+1} - \alpha_i^{k,i}), \quad (18)$$

where $\bar{Q}_{:,i}$ is the ith column of $\bar{Q}$. The cost is $O(l\bar{n})$ for $\bar{Q}_{:,i}$ and $O(l)$ for (18). Therefore, maintaining the whole gradient does not cost more. As using the whole gradient implies fewer iterations (i.e., faster convergence due to the ability to choose for updating the variable that violates optimality most), one should take this advantage. However, the situation for linear SVM is very different. With the different way (12) to calculate $\nabla_i f(\boldsymbol{\alpha})$, the cost to update one α_i is only $O(\bar{n})$. If we still maintain the whole gradient, evaluating (12) l times takes $O(l\bar{n})$ effort. We gather this comparison of different situations in Table 1. Clearly, for nonlinear SVM, one should use decomposition methods by maintaining the whole gradient. However, for linear SVM, if l is large, the cost per iteration without maintaining gradients is much smaller than that with. Hence, the coordinate descent method can be faster than the decomposition method by using many cheap iterations.

An earlier attempt to speed up decomposition methods for linear SVM is (Kao et al., 2004). However, it failed to derive our method here because it does not give up maintaining gradients.

[1]Solvers like LIBSVM update at least two variables due to a linear constraint in their dual problems. Here (4) has no such a constraint, so selecting one variable is possible.

4.2. Existing Linear SVM Methods

We discussed in Section 1 and other places the difference between our method and a primal coordinate descent method (Chang et al., 2007). Below we describe the relations with other linear SVM methods.

We mentioned in Section 3.3 that our Algorithm 2 is related to the online mode in (Collins et al., 2008). They aim at solving multi-class and structured problems. At each iteration an instance is used; then a sub-problem of several variables is solved. They approximately minimize the sub-problem, but for two-class case, one can exactly solve it by (9). For the batch setting, our approach is different from theirs. The algorithm for multi-class problems in (Crammer & Singer, 2003) is also similar to our online setting. For the two-class case, it solves (1) with the loss function $\max(-y_i \boldsymbol{w}^T \boldsymbol{x}_i, 0)$, which is different from (2). They do not study data with a large number of features.

Next, we discuss the connection to stochastic gradient descent (Shalev-Shwartz et al., 2007; Bottou, 2007). The most important step of this method is the following update of $\boldsymbol{w}$:

$$\boldsymbol{w} \leftarrow \boldsymbol{w} - \eta \nabla_{\boldsymbol{w}}(y_i, \boldsymbol{x}_i), \quad (19)$$

where $\nabla_{\boldsymbol{w}}(y_i, \boldsymbol{x}_i)$ is the sub-gradient of the approximate objective function:

$$\boldsymbol{w}^T \boldsymbol{w}/2 + C \max(1 - y_i \boldsymbol{w}^T \boldsymbol{x}_i, 0),$$

and η is the learning rate (or the step size). While our method is dual-based, throughout the iterations we

Table 2. On the right training time for a solver to reduce the primal objective value to within 1% of the optimal value; see (20). Time is in seconds. The method with the shortest running time is boldfaced. Listed on the left are the statistics of data sets: l is the number of instances and n is the number of features.

Data set	Data statistics			Linear L1-SVM			Linear L2-SVM		
	l	n	# nonzeros	DCDL1	Pegasos	SVM$^{\text{perf}}$	DCDL2	PCD	TRON
a9a	32,561	123	451,592	**0.2**	1.1	6.0	0.4	**0.1**	0.1
astro-physic	62,369	99,757	4,834,550	**0.2**	2.8	2.6	**0.2**	0.5	1.2
real-sim	72,309	20,958	3,709,083	**0.2**	2.4	2.4	**0.1**	0.2	0.9
news20	19,996	1,355,191	9,097,916	**0.5**	10.3	20.0	**0.2**	2.4	5.2
yahoo-japan	176,203	832,026	23,506,415	**1.1**	12.7	69.4	**1.0**	2.9	38.2
rcv1	677,399	47,236	49,556,258	**2.6**	21.9	72.0	**2.7**	5.1	18.6
yahoo-korea	460,554	3,052,939	156,436,656	**8.3**	79.7	656.8	**7.1**	18.4	286.1

maintain $\boldsymbol{w}$ by (13). Both (13) and (19) use one single instance $\boldsymbol{x}_i$, but they take different directions $y_i\boldsymbol{x}_i$ and $\nabla_{\boldsymbol{w}}(y_i, \boldsymbol{x}_i)$. The selection of the learning rate η may be the subtlest thing in stochastic gradient descent, but for our method this is never a concern. The step size $(\alpha_i - \bar{\alpha}_i)$ in (13) is governed by solving a sub-problem from the dual.

5. Experiments

In this section, we analyze the performance of our dual coordinate descent algorithm for L1- and L2-SVM. We compare our implementation with state of the art linear SVM solvers. We also investigate how the shrinking technique improves our algorithm.

Table 2 lists the statistics of data sets. Four of them (a9a, real-sim, news20, rcv1) are at `http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets`. The set astro-physic is available upon request from Thorsten Joachims. Except a9a, all others are from document classification. Past results show that linear SVM performs as well as kernelized ones for such data. To estimate the testing accuracy, we use a stratified selection to split each set to 4/5 training and 1/5 testing. We briefly describe each set below. Details can be found in (Joachims, 2006) (astro-physic) and (Lin et al., 2008) (others). a9a is from the UCI "adult" data set. real-sim includes Usenet articles. astro-physic includes documents from Physics Arxiv. news20 is a collection of news documents. yahoo-japan and yahoo-korea are obtained from Yahoo!. rcv1 is an archive of manually categorized newswire stories from Reuters.

We compare six implementations of linear SVM. Three solve L1-SVM, and three solve L2-SVM.

DCDL1 and DCDL2: the dual coordinate descent method with sub-problems permuted at each outer iteration (see Section 3.1). DCDL1 solves L1-SVM while DCDL2 solves L2-SVM. We *omit the shrinking setting.*

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

SVM$^{\text{perf}}$ (Joachims, 2006): a cutting plane method for L1-SVM. We use the latest version 2.1. The source is at `http://svmlight.joachims.org/svm_perf.html`.

TRON: a trust region Newton method (Lin et al., 2008) for L2-SVM. We use the software LIBLINEAR version 1.22 with option `-s 2` (`http://www.csie.ntu.edu.tw/~cjlin/liblinear`).

PCD: a primal coordinate descent method for L2-SVM (Chang et al., 2007).

Since (Bottou, 2007) is related to Pegasos, we do not present its results. We do not compare with another online method Vowpal Wabbit (Langford et al., 2007) either as it has been made available only very recently. Though a code for the bundle method (Smola et al., 2008) is available, we do not include it for comparison due to its closeness to SVM$^{\text{perf}}$. All sources used for our comparisons are available at `http://csie.ntu.edu.tw/~cjlin/liblinear/exp.html`.

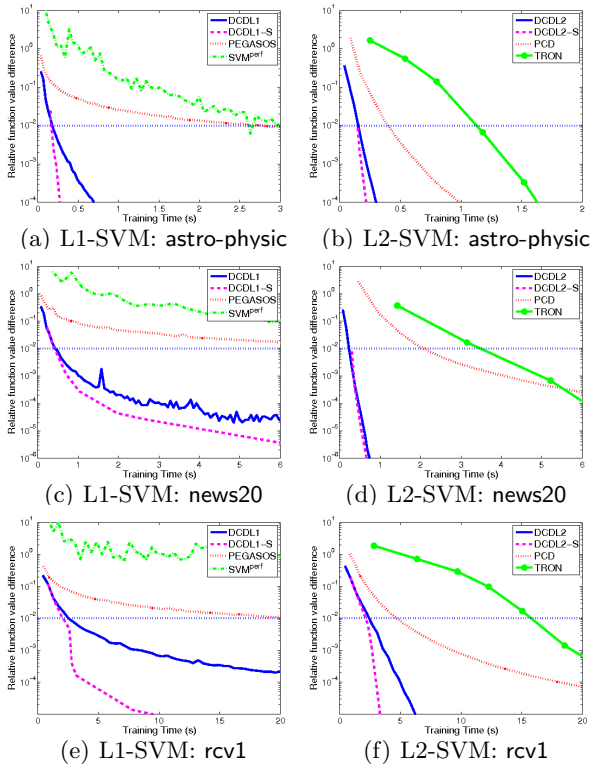
We set the penalty parameter $C = 1$ for comparison[2]. For all data sets, the testing accuracy does not increase after $C \geq 4$. All the above methods are implemented in C/C++ with double precision. Some implementations such as (Bottou, 2007) use single precision to reduce training time, but numerical inaccuracy may occur. We do not include the bias term by (3).

To compare these solvers, we consider the CPU time of reducing the relative difference between the primal objective value and the optimum to within 0.01:

$$|f^P(\boldsymbol{w}) - f^P(\boldsymbol{w}^*)|/|f^P(\boldsymbol{w}^*)| \leq 0.01, \qquad (20)$$

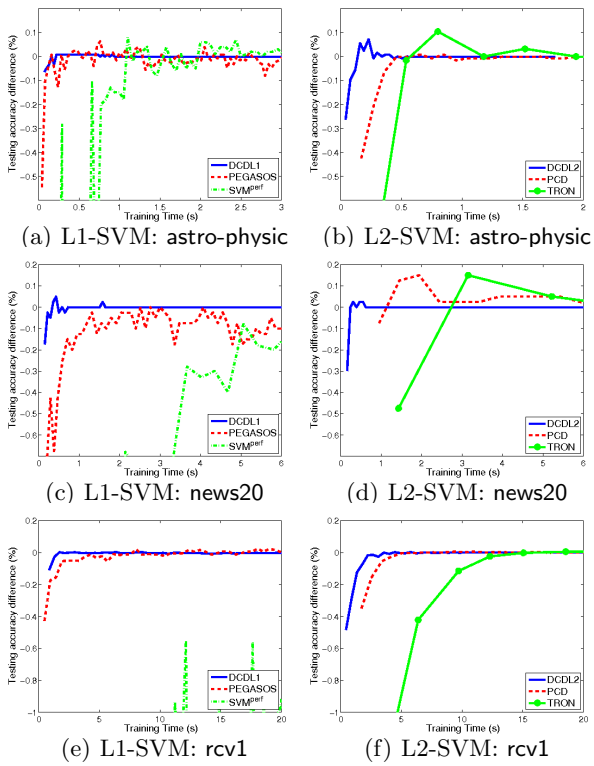
where f^P is the objective function of (1), and $f^P(\boldsymbol{w}^*)$ is the optimal value. Note that for consistency, we use primal objective values even for dual solvers. The reference solutions of L1- and L2-SVM are respectively obtained by solving DCDL1 and DCDL2 until the duality gaps are less than 10^{-6}. Table 2 lists the results. Clearly, our dual coordinate descent method

[2]The equivalent setting for Pegasos is $\lambda = 1/(Cl)$. For SVM$^{\text{perf}}$, its penalty parameter is $C_{perf} = 0.01Cl$.



(a) L1-SVM: astro-physic (b) L2-SVM: astro-physic
(c) L1-SVM: news20 (d) L2-SVM: news20
(e) L1-SVM: rcv1 (f) L2-SVM: rcv1

Figure 1. Time versus the relative error (20). DCDL1-S, DCDL2-S are DCDL1, DCDL2 with shrinking. The dotted line indicates the relative error 0.01. Time is in seconds.



(a) L1-SVM: astro-physic (b) L2-SVM: astro-physic
(c) L1-SVM: news20 (d) L2-SVM: news20
(e) L1-SVM: rcv1 (f) L2-SVM: rcv1

Figure 2. Time versus the difference of testing accuracy between the current model and the reference model (obtained using strict stopping conditions). Time is in seconds.

for both L1- and L2-SVM is significantly faster than other solvers. To check details, we choose astro-physic, news20, rcv1, and show the relative error along time in Figure 1. In Section 3.2, we pointed out that the shrinking technique is very suitable for DCD. In Figure 1, we also include them (DCDL1-S and DCDL2-S) for comparison. Like in Table 2, our solvers are efficient for both L1- and L2-SVM. With shrinking, its performance is even better.

Another evaluation is to consider how fast a solver obtains a model with reasonable testing accuracy. Using the optimal solutions from the above experiment, we generate the reference models for L1- and L2-SVM. We evaluate the testing accuracy difference between the current model and the reference model along the training time. Figure 2 shows the results. Overall, DCDL1 and DCDL2 are more efficient than other solvers. Note that we omit DCDL1-S and DCDL2-S in Figure 2, as the performances with/without shrinking are similar.

Among L1-SVM solvers, SVM$^{\text{perf}}$ is competitive with Pegasos for small data. But in the case of a huge number of instances, Pegasos outperforms SVM$^{\text{perf}}$. However, Pegasos has slower convergence than DCDL1. As discussed in Section 4.2, the learning rate of stochastic gradient descent may be the cause, but for DCDL1 we exactly solve sub-problems to obtain the step size in updating $\boldsymbol{w}$. Also, Pegasos has a jumpy test set performance while DCDL1 gives a stable behavior.

In the comparison of L2-SVM solvers, DCDL2 and PCD are both coordinate descent methods. The former one is applied to the dual, but the latter one to the primal. DCDL2 has a closed form solution for each sub-problem, but PCD has not. The cost per PCD outer iteration is thus higher than that of DCDL2. Therefore, while PCD is very competitive (only second to DCDL1/DCDL2 in Table 2), DCDL2 is even better. Regarding TRON, as a Newton method, it possesses fast final convergence. However, since it takes significant effort at each iteration, it hardly generates a reasonable model quickly. From the experiment results, DCDL2 converges as fast as TRON, but also performs well in early iterations.

Due to the space limitation, we give the following observations without details. First, Figure 1 indicates that our coordinate descent method converges faster for L2-SVM than L1-SVM. As L2-SVM has the diagonal matrix D with $D_{ii} = 1/(2C)$, we suspect that its $\bar{Q}$ is better conditioned, and hence leads to faster convergence. Second, all methods have slower convergence when C is large. However, small C's are usually enough as the accuracy is stable after a threshold. In practice, one thus should try from a small C. More-

over, if $n \ll l$ and C is too large, then our DCDL2 is slower than TRON or PCD (see problem a9a in Table 2, where the accuracy does not change after $C \geq 0.25$). If $n \ll l$, clearly one should solve the primal, whose number of variables is just n. Such data are not our focus. Indeed, with a small number of features, one usually maps data to a higher space and train a nonlinear SVM. Third, we have checked the online Algorithm 2. Its performance is similar to DCDL1 and DCDL2 (i.e., batch setting without shrinking). Fourth, we have investigated real document classification which involves many two-class problems. Using the proposed method as the solver is more efficient than using others.

6. Discussion and Conclusions

We can apply the proposed method to solve regularized least square problems, which have the loss function $(1-y_i \boldsymbol{w}^T \boldsymbol{x}_i)^2$ in (1). The dual is simply (4) without constraints, so the implementation is simpler.

In summary, we present and analyze an efficient dual coordinate decent method for large linear SVM. It is very simple to implement, and possesses sound optimization properties. Experiments show that our method is faster than state of the art implementations.

References

Bordes, A., Bottou, L., Gallinari, P., & Weston, J. (2007). Solving multiclass support vector machines with LaRank. *ICML*.

Boser, B. E., Guyon, I., & Vapnik, V. (1992). A training algorithm for optimal margin classifiers. *COLT*.

Bottou, L. (2007). Stochastic gradient descent examples. `http://leon.bottou.org/projects/sgd`.

Chang, C.-C., & Lin, C.-J. (2001). *LIBSVM: a library for support vector machines*. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

Chang, K.-W., Hsieh, C.-J., & Lin, C.-J. (2007). *Coordinate descent method for large-scale L2-loss linear SVM* (Technical Report). `http://www.csie.ntu.edu.tw/~cjlin/papers/cdl2.pdf`.

Collins, M., Globerson, A., Koo, T., Carreras, X., & Bartlett, P. (2008). Exponentiated gradient algorithms for conditional random fields and max-margin markov networks. *JMLR*. To appear.

Crammer, K., & Singer, Y. (2003). Ultraconservative online algorithms for multiclass problems. *JMLR*, *3*, 951–991.

Friess, T.-T., Cristianini, N., & Campbell, C. (1998). The kernel adatron algorithm: a fast and simple learning procedure for support vector machines. *ICML*.

Joachims, T. (1998). Making large-scale SVM learning practical. *Advances in Kernel Methods - Support Vector Learning*. Cambridge, MA: MIT Press.

Joachims, T. (2006). Training linear SVMs in linear time. *ACM KDD*.

Kao, W.-C., Chung, K.-M., Sun, C.-L., & Lin, C.-J. (2004). Decomposition methods for linear support vector machines. *Neural Comput.*, *16*, 1689–1704.

Keerthi, S. S., & DeCoste, D. (2005). A modified finite Newton method for fast solution of large scale linear SVMs. *JMLR*, *6*, 341–361.

Keerthi, S. S., Shevade, S. K., Bhattacharyya, C., & Murthy, K. R. K. (2001). Improvements to Platt's SMO algorithm for SVM classifier design. *Neural Comput.*, *13*, 637–649.

Langford, J., Li, L., & Strehl, A. (2007). Vowpal Wabbit. `http://hunch.net/~vw`.

Lin, C.-J., Weng, R. C., & Keerthi, S. S. (2008). Trust region Newton method for large-scale logistic regression. *JMLR*, *9*, 623–646.

Luo, Z.-Q., & Tseng, P. (1992). On the convergence of coordinate descent method for convex differentiable minimization. *J. Optim. Theory Appl.*, *72*, 7–35.

Mangasarian, O. L., & Musicant, D. R. (1999). Successive overrelaxation for support vector machines. *IEEE Trans. Neural Networks*, *10*, 1032–1037.

Osuna, E., Freund, R., & Girosi, F. (1997). Training support vector machines: An application to face detection. *CVPR*.

Platt, J. C. (1998). Fast training of support vector machines using sequential minimal optimization. *Advances in Kernel Methods - Support Vector Learning*. Cambridge, MA: MIT Press.

Shalev-Shwartz, S., Singer, Y., & Srebro, N. (2007). Pegasos: primal estimated sub-gradient solver for SVM. *ICML*.

Smola, A. J., Vishwanathan, S. V. N., & Le, Q. (2008). Bundle methods for machine learning. *NIPS*.

Zhang, T. (2004). Solving large scale linear prediction problems using stochastic gradient descent algorithms. *ICML*.