Distributed Box-Constrained Quadratic Optimization for Dual Linear SVM

Ching-pei Lee LEECHINGPEI@GMAIL.COM
Dan Roth DANR@ILLINOIS.EDU
University of Illinois at Urbana-Champaign, 201 N. Goodwin Avenue, Urbana, IL 61801 USA

Abstract

Training machine learning models sometimes needs to be done on large amounts of data that exceed the capacity of a single machine, motivating recent works on developing algorithms that train in a distributed fashion. This paper proposes an efficient box-constrained quadratic optimization algorithm for distributedly training linear support vector machines (SVMs) with large data. Our key technical contribution is an analytical solution to the problem of computing the optimal step size at each iteration, using an efficient method that requires only $O(1)$ communication cost to ensure fast convergence. With this optimal step size, our approach is superior to other methods by possessing global linear convergence, or, equivalently, $O(\log(1/\epsilon))$ iteration complexity for an ϵ-accurate solution, for distributedly solving the non-strongly-convex linear SVM dual problem. Experiments also show that our method is significantly faster than state-of-the-art distributed linear SVM algorithms including DSVM-AVE, DisDCA and TRON.

1. Introduction

With the rapid growth of data volume, distributed machine learning gains more importance as a way to address training with massive data sets that could not fit the capacity of a single machine. Linear support vector machine (SVM) (Boser et al., 1992; Vapnik, 1995) is a widely adopted model for large-scale linear classification. Given training instances $\{(\boldsymbol{x}_i, y_i) \in \mathbf{R}^n \times \{-1, 1\}\}_{i=1}^l$, linear SVM solves the following problem:

$$\min_{\boldsymbol{w}\in\mathbf{R}^n} \quad f^P(\boldsymbol{w}) \equiv \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w} + C\sum_{i=1}^{l}\xi(\boldsymbol{w}, \boldsymbol{x}_i; y_i), \quad (1)$$

where $C > 0$ is a specified parameter, and ξ is a loss function. Two common choices of ξ are

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

which we refer to as L1-SVM and L2-SVM, respectively.

Linear SVM has been used in many applications, and its single-machine training has been studied extensively. It is well-known that instead of directly solving (1), optimizing its dual problem shown below is sometimes faster (Hsieh et al., 2008; Yuan et al., 2012), especially when $l < n$, because there are fewer variables to optimize.

$$\begin{aligned} \min_{\boldsymbol{\alpha}\in\mathbf{R}^l} \quad & f(\boldsymbol{\alpha}) \equiv \frac{1}{2}\boldsymbol{\alpha}^T\bar{Q}\boldsymbol{\alpha} - \boldsymbol{e}^T\boldsymbol{\alpha} \\ \text{subject to} \quad & 0 \le \alpha_i \le U, i = 1, \ldots, l, \end{aligned} \quad (2)$$

where $\boldsymbol{e}$ is the vector of ones, $\bar{Q} = Q + sI$, $Q = YX(YX)^T$, $X = [\boldsymbol{x}_1, \ldots, \boldsymbol{x}_l]^T$, Y is a diagonal matrix such that $Y_{i,i} = y_i$, I is the identity matrix, and

$$(s, U) = \begin{cases} (0, C) & \text{if L1-SVM,} \\ (1/2C, \infty) & \text{if L2-SVM.} \end{cases}$$

Because the main computations in batch solvers for (1) are matrix-vector products that can be naively parallelized, several works have successfully adapted these solvers to distributed environments (Agarwal et al., 2014; Zhang et al., 2012; Zhuang et al., 2015; Lin et al., 2014). However, state-of-the-art single-machine dual algorithms are all sequential and cannot be easily parallelized. Moreover, in a distributed setting, because each machine only has a fraction of the training data, and the cost of communication and synchronization is relatively high, it is important to consider algorithms with low communication cost. Consequently, algorithms with a faster convergence rate are desirable because a smaller number of iterations implies a smaller number of communication rounds. We observe that without careful consideration of this issue, existing distributed dual solvers do not achieve satisfactory training speed. However, as mentioned above, given that a dual solver might be more suitable for high-dimensional data, it is important to develop better distributed dual linear SVM

Proceedings of the 32^{nd} International Conference on Machine Learning, Lille, France, 2015. JMLR: W&CP volume 37.

algorithms for extremely large data sets of high dimensionality.

In this work, we propose an efficient box-constrained quadratic optimization framework for solving (2) when the training instances are distributed among K machines, where the instances in machine k are $\{(\boldsymbol{x}_i, y_i)\}_{i\in J_k}$. In our setting, J_k are disjoint index sets such that $\cup_{k=1}^K J_k = \{1,\ldots,l\}$. By considering a carefully designed positive definite block-diagonal approximation of $\bar{Q}$, at each iteration our algorithm efficiently obtains a descent direction that ensures fast convergence for (2) with low communication cost. We then conduct a line search method that only requires negligible computation cost and $O(1)$ communication to establish a global linear convergence rate. In other words, our algorithm only requires $O(\log(1/\epsilon))$ iterations to obtain an ϵ-accurate solution for (2). This convergence rate is better than that of existing distributed dual solvers for training L1-SVM, whose dual problem is not strongly convex. We also discuss the main differences between our algorithm and existing distributed algorithms for (2) and point out the key differences.

Experiments show that our algorithm is significantly faster than existing distributed solvers for (2). Our framework can be easily extended to solve other similar problems.

This paper is organized as follows. Our algorithm and its convergence analysis are described in Section 2. Section 3 discusses related works. We present experimental results in Section 4. Section 5 concludes this work.

2. Distributed Box-Constrained Quadratic Optimization

Before discussing our method, we summarize notations frequently used in this paper. $\pi(i) = k$ indicates that $i \in J_k$. We will frequently use the following notation.

$$\mathcal{P}(\boldsymbol{\alpha}) \equiv [-\alpha_1, U-\alpha_1] \times \cdots \times [-\alpha_l, U-\alpha_l].$$

We denote $\|\boldsymbol{u}\|_A^2 \equiv \boldsymbol{u}^T A \boldsymbol{u}$ for $\boldsymbol{u} \in \mathbf{R}^t$, and $A \in \mathbf{R}^{t\times t}$, where t is a positive integer. For any $\boldsymbol{v} \in \mathbf{R}^l$, $\boldsymbol{v}_{J_k}$ denotes the sub-vector of $\boldsymbol{v}$ that contains the coordinates in J_k. Similarly, $\mathcal{P}(\boldsymbol{\alpha})_{J_k}$ is the constraint subset of $\mathcal{P}(\boldsymbol{\alpha})$ that only has the coordinates indexed by J_k. For $A \in \mathbf{R}^{l\times l}$, $A_{J_k,J_m} \in \mathbf{R}^{|J_k|\times|J_m|}$ denotes the sub-matrix of A corresponding to the entries $A_{i,j}$ with $i \in J_k, j \in J_m$. If $J_k = J_m$, we simplify it to A_{J_k}. We denote by X_{J_k} the sub-matrix of X containing the instances in J_k, and, similarly Y_{J_k} is the corresponding sub-diagonal matrix of Y.

2.1. Algorithm

Our algorithm starts with a feasible point $\boldsymbol{\alpha}^0$ for (2) and iteratively generates a sequence of solutions $\{\boldsymbol{\alpha}^t\}_{t=0}^{\infty}$. For model evaluation at each iteration, we transform the current $\boldsymbol{\alpha}^t$ to a primal solution by the KKT condition of (1).

$$\boldsymbol{w}^t = X^T Y \boldsymbol{\alpha}^t. \tag{3}$$

At the t-th iteration, we update the current $\boldsymbol{\alpha}^t$ by

$$\boldsymbol{\alpha}^{t+1} = \boldsymbol{\alpha}^t + \eta_t \Delta\boldsymbol{\alpha}^t, \tag{4}$$

where $\eta_t \in \mathbf{R}$ is the step size and $\Delta\boldsymbol{\alpha}^t \in \mathbf{R}^l$ is the update direction. We first describe our method for computing $\Delta\boldsymbol{\alpha}^t$, and then present two efficient methods for calculating a good η_t for $\Delta\boldsymbol{\alpha}^t$.

Computing the Update Direction: In a distributed environment, usually communication and synchronization are expensive. Therefore, in order to reduce the training time, we should consider high-order optimization methods that require fewer iterations to converge. Therefore, we try to obtain a good $\Delta\boldsymbol{\alpha}^t$ by solving the following quadratic problem that approximates (2).

$$\Delta\boldsymbol{\alpha}^t \equiv \operatorname*{arg\,min}_{\boldsymbol{d}\in\mathcal{P}(\boldsymbol{\alpha}^t)} \quad \nabla f(\boldsymbol{\alpha}^t)^T \boldsymbol{d} + \frac{1}{2}\boldsymbol{d}^T H \boldsymbol{d}, \tag{5}$$

where H is an approximation of $\bar{Q}$. Because machine k only has access to those instances in J_k, it is natural to consider the following H to avoid frequent communication.

$$H = \tilde{Q} + (s+\tilde{\tau})I, \text{ where } \tilde{\tau} = \begin{cases} \tau & \text{if L1-SVM,} \\ 0 & \text{if L2-SVM,} \end{cases} \tag{6}$$

$\tau > 0$ is a small value to ensure positive definiteness, and

$$\tilde{Q}_{i,j} = \begin{cases} Q_{i,j} & \text{if } \pi(i) = \pi(j), \\ 0 & \text{otherwise.} \end{cases}$$

After re-indexing the instances, we observe that the choice of H in (6) is a symmetric, block-diagonal matrix with K blocks, where the k-th block is

$$H_{J_k} = Y_{J_k} X_{J_k} (Y_{J_k} X_{J_k})^T + (s+\tilde{\tau})I.$$

Because H is block-diagonal, (5) can be decomposed into the following independent sub-problems for $k = 1,\ldots,K$.

$$\Delta\boldsymbol{\alpha}^t_{J_k} = \operatorname*{arg\,min}_{\boldsymbol{d}_{J_k}\in\mathcal{P}(\boldsymbol{\alpha}^t)_{J_k}} \nabla f(\boldsymbol{\alpha}^t)_{J_k}^T \boldsymbol{d}_{J_k} + \frac{1}{2}\|\boldsymbol{d}_{J_k}\|_{H_{J_k}}^2. \tag{7}$$

If $\boldsymbol{w}^t$ is available to all machines, both H_{J_k} and

$$\nabla f(\boldsymbol{\alpha}^t)_{J_k} = Y_{J_k} X_{J_k} \boldsymbol{w}^t + s\boldsymbol{\alpha}^t_{J_k} - \boldsymbol{e}$$

can be obtained using only instances in machine k. Thus the only communication required in this procedure is making $\boldsymbol{w}^t$ available to all machines. We see that obtaining $\boldsymbol{w}^t$ requires gathering information from all machines.

$$\boldsymbol{w}^t = \bigoplus_{k=1}^K X_{J_k}^T Y_{J_k} \boldsymbol{\alpha}^t_{J_k}. \tag{8}$$

The symbol $\bigoplus$ represents the operation of receiving the information from all machines and broadcasting the results back to all machines. In practice, this can be achieved by the *allreduce* operation in MPI.

With the availability of $\boldsymbol{w}^t$, (7) is in the same form as (2), with $\bar{Q}, \boldsymbol{e}$, and $\mathcal{P}(\mathbf{0})$ being replaced by $\bar{Q}_{J_k} + \tilde{\tau} I, \nabla f(\boldsymbol{\alpha}^t)_{J_k}$, and $\mathcal{P}(\boldsymbol{\alpha}^t)_{J_k}$, respectively. Therefore, the smaller per machine problem can be solved by any existing single machine dual linear SVM optimization methods. We will discuss this issue in Section 2.3.

After $\Delta\boldsymbol{\alpha}^t$ is computed, we need to calculate the step size η_t to ensure sufficient function decrease. We discuss two methods to obtain a good η_t. The first approach gives an approximate solution while the second one computes the optimal step size. Note that these methods can be applied to any descent direction, and are not restricted to the direction obtained by the above method.

Approximate approach for step size: We first consider backtracking line search with the Armijo rule described in Tseng & Yun (2009). Given the direction $\Delta\boldsymbol{\alpha}^t$, and the parameters $\sigma, \beta \in (0,1)$, $\gamma \in [0,1)$, the Armijo rule assigns η_t to be the largest element in $\{\beta^k\}_{k=0}^{\infty}$ that satisfies

$$f(\boldsymbol{\alpha}^t + \beta^k \Delta\boldsymbol{\alpha}^t) - f(\boldsymbol{\alpha}^t) \le \beta^k \sigma \Delta_t, \tag{9}$$

where

$$\Delta_t \equiv \nabla f(\boldsymbol{\alpha}^t)^T \Delta\boldsymbol{\alpha}^t + \gamma \|\Delta\boldsymbol{\alpha}^t\|_H^2. \tag{10}$$

To obtain the desired β^k, a naive approach sequentially tries $k = 0, 1, \ldots,$ and goes through the whole training data to recompute $f(\boldsymbol{\alpha}^t + \beta^k \Delta\boldsymbol{\alpha}^t)$ for each k until (9) is satisfied. This approach is expensive because we do not know ahead of time what value of k satisfies (9). If the direction is not chosen carefully, it is likely that this approach will be very time-consuming. To deal with this problem, observe that, because f is a quadratic function with Hessian $\bar{Q}$,

$$\begin{aligned} &f(\boldsymbol{\alpha}^t + \beta^k \Delta\boldsymbol{\alpha}^t) \\ =&f(\boldsymbol{\alpha}^t) + \beta^k \nabla f(\boldsymbol{\alpha}^t)^T \Delta\boldsymbol{\alpha}^t + \frac{\beta^{2k}}{2} \|\Delta\boldsymbol{\alpha}^t\|_{\bar{Q}}^2. \end{aligned} \tag{11}$$

Hence (9) can be simplified to

$$\frac{\beta^k}{2} \|\Delta\boldsymbol{\alpha}^t\|_{\bar{Q}}^2 \le (\sigma - 1) \nabla f(\boldsymbol{\alpha}^t)^T \Delta\boldsymbol{\alpha}^t + \sigma\gamma \|\Delta\boldsymbol{\alpha}^t\|_H^2,$$

which implies

$$\begin{aligned} \eta_t = \min(\beta^0, \beta^{\bar{k}}), \text{ where } \bar{k} \equiv \lceil \frac{1}{\log \beta} (\log \frac{2}{\|\Delta\boldsymbol{\alpha}^t\|_{\bar{Q}}^2} \\ + \log\left((\sigma - 1) \nabla f\left(\boldsymbol{\alpha}^t\right)^T \Delta\boldsymbol{\alpha}^t + \sigma\gamma \|\Delta\boldsymbol{\alpha}^t\|_H^2 \right)) \rceil. \end{aligned} \tag{12}$$

We can obtain $\|\Delta\boldsymbol{\alpha}^t\|_H^2$, $\|\Delta\boldsymbol{\alpha}^t\|_{\bar{Q}}^2$, and $\nabla f(\boldsymbol{\alpha}^t)^T \Delta\boldsymbol{\alpha}^t$ by

$$\|\Delta\boldsymbol{\alpha}^t\|_H^2 = \bigoplus_{k=1}^{K} (\|X_{J_k}^T Y_{J_k} \Delta\boldsymbol{\alpha}_{J_k}^t\|^2) + (s + \tilde{\tau}) \bigoplus_{k=1}^{K} (\|\Delta\boldsymbol{\alpha}_{J_k}\|^2), \tag{13}$$

$$\|\Delta\boldsymbol{\alpha}^t\|_{\bar{Q}}^2 = \|\bigoplus_{k=1}^{K} X_{J_k}^T Y_{J_k} \Delta\boldsymbol{\alpha}_{J_k}^t\|^2 + s \bigoplus_{k=1}^{K} (\|\Delta\boldsymbol{\alpha}_{J_k}\|^2), \tag{14}$$

and

$$\begin{aligned} \nabla f(\boldsymbol{\alpha}^t)^T \Delta\boldsymbol{\alpha}^t = (\boldsymbol{w}^t)^T (\bigoplus_{k=1}^{K} X_{J_k}^T Y_{J_k} \Delta\boldsymbol{\alpha}_{J_k}^t) \\ + s \bigoplus_{k=1}^{K} (\boldsymbol{\alpha}_{J_k}^t)^T \Delta\boldsymbol{\alpha}_{J_k}^t - \bigoplus_{k=1}^{K} \boldsymbol{e}^T \Delta\boldsymbol{\alpha}_{J_k}^t. \end{aligned} \tag{15}$$

To reduce the communication cost, we replace (8) with

$$\boldsymbol{w}^t = \boldsymbol{w}^{t-1} + \eta_t \Delta\boldsymbol{w}^{t-1}, \tag{16}$$

where

$$\Delta\boldsymbol{w}^t \equiv \bigoplus_{k=1}^{K} X_{J_k}^T Y_{J_k} \Delta\boldsymbol{\alpha}_{J_k}^t. \tag{17}$$

Thus obtaining $\boldsymbol{w}^t$ still has the same communication cost, but (14) and (15) can respectively be simplified to

$$\|\Delta\boldsymbol{\alpha}^t\|_{\bar{Q}}^2 = \|\Delta\boldsymbol{w}^t\|^2 + s \bigoplus_{k=1}^{K} \|(\Delta\boldsymbol{\alpha}_{J_k}^t)\|^2, \tag{18}$$

$$\begin{aligned} \nabla f(\boldsymbol{\alpha}^t)^T \Delta\boldsymbol{\alpha}^t = (\boldsymbol{w}^t)^T \Delta\boldsymbol{w}^t + s \bigoplus_{k=1}^{K} (\boldsymbol{\alpha}_{J_k}^t)^T \Delta\boldsymbol{\alpha}_{J_k}^t \\ - \bigoplus_{k=1}^{K} \boldsymbol{e}^T \Delta\boldsymbol{\alpha}_{J_k}^t. \end{aligned} \tag{19}$$

Thus, the two size-n *allreduce* of (8) and $\bigoplus X_{J_k} Y_{J_k} \Delta\boldsymbol{\alpha}_{J_k}^t$ required in (15) can be achieved using only one size-n *allreduce*. When $\Delta\boldsymbol{w}^t$ is available, obtaining (18)-(19) and (13) requires only $O(l+n)$ computation and $O(1)$ additional communication. This additional $O(l+n)$ computation is negligible in comparison with the $O(ln)$ cost of solving (5), and the new communication can be combined into the *allreduce* operation for $\Delta\boldsymbol{w}^t$ by augmenting the vector being communicated. With this method, backtracking line search can be done very efficiently.

Exact solution of the best step size: By substituting β^k with η_t in (11), we observe that $f(\boldsymbol{\alpha}^t + \eta_t \Delta\boldsymbol{\alpha}^t)$ is a quadratic convex function of η_t. Therefore, we can obtain η_t^* that minimizes $f(\boldsymbol{\alpha}^t + \eta_t \Delta\boldsymbol{\alpha}^t)$ by

$$\frac{\partial f(\boldsymbol{\alpha}^t + \eta_t \Delta\boldsymbol{\alpha}^t)}{\partial \eta_t} = 0 \Rightarrow \eta_t^* = \frac{-\nabla f(\boldsymbol{\alpha}^t)^T \Delta\boldsymbol{\alpha}^t}{(\Delta\boldsymbol{\alpha}^t)^T \bar{Q} \Delta\boldsymbol{\alpha}}. \tag{20}$$

Algorithm 1 A box-constrained quadratic optimization algorithm for distributedly solving (2)

1. $t \leftarrow 0$, given $\boldsymbol{\alpha}^0$, computes $\boldsymbol{w}^0$ by (8).
2. While $\boldsymbol{\alpha}^t$ is not optimal:
   2.1. Obtain $\Delta\boldsymbol{\alpha}^t$ by distributedly solving (7) on K machines in parallel.
   2.2. Compute $\Delta\boldsymbol{w}^t = \bigoplus_{k=1}^K X_{J_k}^T Y_{J_k} \Delta\boldsymbol{\alpha}_{J_k}^t$.
   2.3. Compute $(\boldsymbol{\alpha}^t)^T\Delta\boldsymbol{\alpha}^t$, $\boldsymbol{e}^T\Delta\boldsymbol{\alpha}^t$, $\bigoplus_{k=1}^K \|\Delta\boldsymbol{\alpha}_{J_k}^t\|^2$, $\bigoplus_{k=1}^K \|X_{J_k}^T Y_{J_k}\Delta\boldsymbol{\alpha}_{J_k}^t\|^2$ by *allreduce*.
   2.4. Obtain η_t by (12) or (21).
   2.5. $\boldsymbol{\alpha}^{t+1} \leftarrow \boldsymbol{\alpha}^t + \eta_t\Delta\boldsymbol{\alpha}^t$, $\boldsymbol{w}^{t+1} \leftarrow \boldsymbol{w}^t + \eta_t\Delta\boldsymbol{w}^t$.
   2.6. $t \leftarrow t+1$.

Using (14) and (19), the computation of (20) has identical cost to the approximate approach. However, unlike the previous method that guarantees $\eta_t \leq 1$ and thus $\eta_t\Delta\boldsymbol{\alpha}^t \in \mathcal{P}(\boldsymbol{\alpha}^t)$, it is possible that $\eta_t^* > 1$ in (20). In this case, $\boldsymbol{\alpha}^t + \eta_t^*\Delta\boldsymbol{\alpha}^t$ may not be a feasible point for (2). To avoid the infeasibility, we consider

$$\lambda_i = \begin{cases} -\alpha_i^t/\Delta\alpha_i^t & \text{if } \Delta\alpha_i^t < 0, \\ (U-\alpha_i^t)/(\Delta\alpha_i^t) & \text{if } \Delta\alpha_i^t > 0, \\ \infty & \text{if } \Delta\alpha_i^t = 0. \end{cases}$$

and let

$$\eta_t = \min(\eta_t^*, \min_{1\leq i\leq l} \lambda_i). \tag{21}$$

We can see from convexity that (21) is the optimal feasible solution for η_t. The minimum λ_i over $\{1,\ldots,l\}$ is obtained by an $O(1)$ communication.

In L1-SVM, $\bar{Q}$ is only positive semidefinite. Therefore, $\|\Delta\boldsymbol{\alpha}^t\|_{\bar{Q}}^2$ can be zero for a nonzero $\Delta\boldsymbol{\alpha}^t$. But this zero denominator does not cause any problem in (12) or (21). We will show in Lemma 2.1 in the next section that $\nabla f(\boldsymbol{\alpha}^t)^T\Delta\boldsymbol{\alpha}^t$ is negative provided $\Delta\boldsymbol{\alpha}^t \neq \mathbf{0}$. Consequently, the last log term of $\bar{k}$ in (12) is finite. If $(\Delta\boldsymbol{\alpha}^t)^T\bar{Q}\Delta\boldsymbol{\alpha}^t = 0$, $\bar{k} = -\infty$ because $\log\beta < 0$. We thus have $\eta_t = \beta^0 = 1$ when $\|\Delta\boldsymbol{\alpha}^t\|_{\bar{Q}}^2 = 0$. In (21), $\nabla f(\boldsymbol{\alpha}^t)\Delta\boldsymbol{\alpha}^t < 0$ ensures that $\eta_t^* > 0$, hence when $(\Delta\boldsymbol{\alpha}^t)^T\bar{Q}\Delta\boldsymbol{\alpha}^t = 0$, η_t^* is ∞. Because of the min operation in (20) and since U is finite in L1-SVM, η_t is still finite unless $\Delta\boldsymbol{\alpha}^t = \mathbf{0}$. We will show in the next section that this only happens when $\boldsymbol{\alpha}^t$ is optimal.

In summary, our algorithm uses (16)-(17) to synchronize information between machines, solves (7) to obtain the update direction $\Delta\boldsymbol{\alpha}^t$, adopts (12) or (21) to decide the step size η_t, and then updates $\boldsymbol{\alpha}^t$ using (4). A detailed description appears in Algorithm 1. In practice, steps 2.2 and 2.3 can be done in one *allreduce* operation.

2.2. Convergence Analysis

To establish the convergence of Algorithm (1), we first show that the direction obtained from (5) is always a descent direction and thus the line search computations in (12) and (20) do not face the problem of $0/0$ or $\eta_t = 0$.

Lemma 2.1 *If H is positive definite, then $\boldsymbol{\alpha}^t$ is optimal if and only if $\Delta\boldsymbol{\alpha}^t = \mathbf{0}$. Moreover, $\Delta\boldsymbol{\alpha}^t$ obtained from (5) is a descent direction for $f(\boldsymbol{\alpha}^t)$ such that $\nabla f(\boldsymbol{\alpha}^t)^T\Delta\boldsymbol{\alpha}^t < 0$.*

Proof: $\mathbf{0} \leq \boldsymbol{\alpha}^t \leq U\boldsymbol{e}$ indicates $\Delta\boldsymbol{\alpha}^t = \mathbf{0}$ is feasible for (5). Because $f(\boldsymbol{\alpha})$ is convex, $\boldsymbol{\alpha}^t$ is optimal if and only if

$$\nabla f(\boldsymbol{\alpha}^t)^T\boldsymbol{d} \geq 0, \forall \boldsymbol{d} \in \mathcal{P}(\boldsymbol{\alpha}^t). \tag{22}$$

Because H is positive definite, by strong convexity of (5), (22) holds if and only if for all nonzero $\boldsymbol{d} \in \mathcal{P}(\boldsymbol{\alpha}^t)$,

$$\nabla f(\boldsymbol{\alpha}^t)^T\boldsymbol{d} + \frac{1}{2}\boldsymbol{d}^T H\boldsymbol{d} > \nabla f(\boldsymbol{\alpha}^t)^T\mathbf{0} + \frac{1}{2}\mathbf{0}^T H\mathbf{0}. \tag{23}$$

Now if $\boldsymbol{\alpha}^t$ is not optimal, there exists $\boldsymbol{d} \in \mathcal{P}(\boldsymbol{\alpha}^t)$ such that (23) does not hold. Thus,

$$\nabla f(\boldsymbol{\alpha}^t)^T\Delta\boldsymbol{\alpha}^t < \nabla f(\boldsymbol{\alpha}^t)^T\Delta\boldsymbol{\alpha}^t + \frac{1}{2}(\Delta\boldsymbol{\alpha}^t)^T H\Delta\boldsymbol{\alpha}^t \leq 0.$$

That H is positive definite implies the strict inequality. ■

Lemma 2.1 can also be viewed as an application of Lemmas 1-2 in Tseng & Yun (2009).

The following two theorems show that Algorithm 1 possesses global linear convergence for problem (2). Namely, for any $\epsilon > 0$, Algorithm 1 requires at most $O(\log(1/\epsilon))$ iterations to obtain a solution of $\boldsymbol{\alpha}^t$ such that

$$(f(\boldsymbol{\alpha}^t) - f(\boldsymbol{\alpha}^*)) \leq \epsilon, \tag{24}$$

where $\boldsymbol{\alpha}^*$ is the optimal solution of (2).

Theorem 2.2 *Algorithm 1 with the Armijo rule backtracking line search (12) has at least global linear convergence rate for solving problem (2).*

Proof Sketch: If we rewrite problem (2) as

$$\min_{\boldsymbol{\alpha}\in\mathbf{R}^l} \quad F(\boldsymbol{\alpha}) \equiv f(\boldsymbol{\alpha}) + P(\boldsymbol{\alpha}), \tag{25}$$

where

$$P(\boldsymbol{\alpha}) \equiv \sum_{i=1}^l P_i(\boldsymbol{\alpha}), \quad P_i(\boldsymbol{\alpha}) \equiv \begin{cases} 0 & \text{if } 0 \leq \alpha_i \leq U, \\ \infty & \text{otherwise,} \end{cases}$$

then (5) can also be written as

$$\Delta\boldsymbol{\alpha}^t = \arg\min_{\boldsymbol{d}} \quad \nabla f(\boldsymbol{\alpha}^t)^T\boldsymbol{d} + \frac{1}{2}\|\boldsymbol{d}\|_H^2 + P(\boldsymbol{\alpha}^t + \boldsymbol{d}).$$

Because we optimize over all instances at each iteration, our algorithm can be seen as a cyclic block coordinate descent method satisfying the Gauss-Seidel rule with only one block. For L2-SVM, both $\bar{Q}$ and H are positive definite. For L1-SVM, $\bar{Q}$ is only positive semidefinite, but the $\tilde{\tau}I$ term in (6) ensures H is positive definite. Thus Algorithm 1 is a special case of the algorithm in Tseng & Yun (2009) and satisfies their Assumption 1. Theorem 2 in Tseng & Yun (2009) then provides local linear convergence for our algorithm such that for some $k_0 \geq 0$,

$$f(\boldsymbol{\alpha}^{t+1}) - f(\boldsymbol{\alpha}^*) \leq \theta(f(\boldsymbol{\alpha}^t) - f(\boldsymbol{\alpha}^*)), \forall t \geq k_0, \quad (26)$$

where $\theta < 1$ is a constant. This k_0 indicates the number of iterations required to make their local error bound assumption hold. This local error bound has been shown to hold from the first iteration of the algorithm for both L1-SVM (Pang, 1987, Theorem 3.1) and L2-SVM (Wang & Lin, 2014, Theorem 4.6). This implies $k_0 = 0$. We thus can obtain from (26) with $k_0 = 0$ the desired $O(\log(1/\epsilon))$ rate for solving both L1-SVM and L2-SVM.

Alternatively, the same result can also be obtained by combining the analysis in Yun (2014) and Tseng & Yun (2009). A more detailed proof is shown in the supplementary materials. ■

Corollary 2.3 *Algorithm 1 with exact line search (21) converges at least as fast as Algorithm 1 with the Armijo rule backtracking line search. Thus, it has at least global linear convergence rate for solving problem (2).*

Proof: Note that the global linear convergence in Theorem 2.2 is obtained by (26) with $k_0 = 0$. If we denote by $\bar{\eta}_t$ the step size obtained from (12) and take $\hat{\eta}_t$ as the step size obtained from solving (21), we have from the convexity of (2) that

$$f(\boldsymbol{\alpha}^t + \hat{\eta}_t \Delta\boldsymbol{\alpha}^t) \leq f(\boldsymbol{\alpha}^t + \bar{\eta}_t \Delta\boldsymbol{\alpha}^t).$$

Therefore, (26) still holds because

$$\begin{aligned} &f(\boldsymbol{\alpha}^{t+1}) - f(\boldsymbol{\alpha}^*) = f(\boldsymbol{\alpha}^t + \hat{\eta}_t \Delta\boldsymbol{\alpha}^t) - f(\boldsymbol{\alpha}^*) \\ \leq &f(\boldsymbol{\alpha}^t + \bar{\eta}_t \Delta\boldsymbol{\alpha}^t) - f(\boldsymbol{\alpha}^*) \leq \theta(f(\boldsymbol{\alpha}^t) - f(\boldsymbol{\alpha}^*)). \end{aligned}$$ ■

2.3. Practical Issues

As mentioned earlier, we can use any existing linear SVM dual solver to solve (7) locally. In our implementation, we consider dual coordinate descent methods (Hsieh et al., 2008; Shalev-Shwartz & Zhang, 2013) that are reported to work well in single-core training. To ensure (5) is minimized, we can adopt the approach of Hsieh et al. (2008), which cyclically goes through all training instances several times until a stopping condition related to sub-optimality is satisfied. Alternatively, we can also use the stochastic approach in Shalev-Shwartz & Zhang (2013) with enough iterations to have a high probability that the solution is close enough to the optimum. An advantage of the cyclic approach is that it comes with an easily computable stopping condition for sub-optimality that can prevent redundant inner iterations. On the other hand, even when each machine contains the same amount of data, the cyclic method could not guarantee that each machine will finish optimizing the sub-problems simultaneously, and hence some machines might be idle for a long time. We thus consider a setting as follows. For solving each sub-problem, we use the cyclic approach, but at each inner iteration[1], we follow the approach in Hsieh et al. (2008); Yu et al. (2012); Chang & Roth (2011) to randomly shuffle the instance order, to have faster empirical convergence. We also assimilate the idea of stochastic solvers to set the number of inner iterations for solving (5) identical for all machines throughout the whole training procedure of Algorithm 1 to ensure that each machine stops solving (7) at roughly the same time. This hybrid method assimilates advantages of both approaches.

During the procedure of minimizing the dual problem, it is possible that a descent direction for the dual problem might not correspond to a descent direction for the primal problem if we have not reached a solution that is close enough to the optimum. This may happen in any algorithms optimizing the dual problem. When (2) is properly optimized, strong duality of SVM problems ensure that the corresponding primal objective value is also minimized. However, in case where we need to stop our algorithm before reaching an accurate enough solution, a smaller primal objective value is desirable. We can easily deal with this problem by adopting the pocket algorithm for perceptrons (Gallant, 1990). The idea is to compute the primal function value at each iteration and maintain the $\boldsymbol{w}^t$ that has the smallest primal objective value as the current model. Now that the primal objective value is available, we can use the relative duality gap as our stopping condition.

$$(f^P(\boldsymbol{w}^t) - (-f(\boldsymbol{\alpha}^t))) \leq \epsilon(f^P(\mathbf{0}) - (-f(\mathbf{0}))) \quad (27)$$

Computing the primal function value is expensive in the single-machine setting, but it is relatively cheap in distributed settings because fewer instances are processed by each machine, and usually the training cost is dominated by communication and synchronization.

3. Related Works

Our algorithm is closely related to the methods proposed by Pechyony et al. (2011); Yang (2013). Even though these approaches were originally discussed in different ways, they can be described using our framework. The discussion that follows indicates that our approach provides better theoretical guarantees than these related approaches.

[1]Here one inner iteration means passing through the data once.

Pechyony et al. (2011) proposed two methods for solving L1-SVM. The first one, DSVM-AVE, iteratively solves (5) with $H = \tilde{Q}$ to obtain $\Delta\boldsymbol{\alpha}^t$, while η_t is fixed to be $1/K$. Its iteration complexity is $O(1/\epsilon)$ for L1-SVM and $O(\log(1/\epsilon))$ for L2-SVM (Ma et al., 2015).

The second approach in Pechyony et al. (2011), called DSVM-LS, conducts line search for $\Delta\boldsymbol{\alpha}^t$ in the primal problem. However, line search in the primal objective function is more expensive because it requires passing through the whole training data to evaluate the function value for each backtracking conducted. Also, DSVM-LS does not have convergence guarantee. On the other hand, our approach conducts efficient line searches in the dual problem with a very low cost. The line search approach in our algorithm is the essential step to guaranteeing global linear convergence for solving L1-SVM. The multi-core parallel primal coordinate descent method in Bian et al. (2014) is similar to DSVM-LS in conducting primal line search. They solve L1-regularized logistic regression problems in the primal by an approach similar to solving (25) with a diagonal H and conducting Armijo line search. Their method has guaranteed convergence and can be adopted to L2-SVM. However, as mentioned above, line search for the primal problem is expensive. Also, their method requires differentiable loss term and is not applicable to L1-SVM.

DisDCA (Yang, 2013) can optimize a class of dual problems including (2). This approach iteratively solves (5) with a stochastic solver and the step size is fixed to $\eta_t = 1$. Yang (2013) proposed a basic and a practical variant, both of which use positive semi-definite H. In the basic variant, H is a diagonal matrix such that $H_{i,i} = mKQ_{i,i}$, where m is the number of instances used each time (5) is solved. The author showed that this variant requires $O(\log(1/\epsilon))$ iteration complexity to satisfy (27) in expected function values for smooth-loss problems like L2-SVM, and $O(1/\epsilon)$ for Lipschitz continuous losses such as L1-SVM. Note that an ϵ-accurate solution for (27) is roughly ϵCl-accurate for (24), so to compare the constants for convergence in different algorithms, we need to properly scale ϵ. In the practical variant of DisDCA, $H = K\tilde{Q} + sI$. Yang et al. (2014) provides convergence for this variant only under the unrealistic assumption $\bar{Q} = \tilde{Q}$. Recently, Ma et al. (2015) show that when $L2$ regularization is used, for both DSVM-AVE and the practical variant of DisDCA, the number of iterations required to satisfy (27) in expected function values are also $O(\log(1/\epsilon))$ and $O(1/\epsilon)$ for smooth-loss problems and problems with Lipschitz continuous losses, respectively. A key difference between these results and ours is that we only require $O(\log(1/\epsilon))$ iterations for L1-SVM, and our result is for deterministic function values, which is stronger than the expected function values.

Ma et al. (2015) proposed a framework CoCoA+, and this framework in their experimental setting reduces to the practical variant of DisDCA (Yang, 2013). Their experiments showed that the DisDCA practical variant is faster than other existing approaches.

Data set	l	n	#nonzeros
webspam	280,000	*16,609,143	1,044,393,506
url	1,916,904	3,231,961	221,663,296
epsilon	400,000	2,000	800,000,000

*: Among the feature dimensions, only 680,715 coordinates have nonzero entries, but we still use the original data to examine the situation that communication cost is extremely high.

Table 1. Data statistics. For webspam and url, test sets are not available so we randomly split the original data into 80%/20% as training set and test set, respectively.

Most other distributed linear SVM algorithms optimize (1). Primal batch solvers that require computing the full gradient or Hessian-vector products are inherently parallelizable and can be easily extended to distributed environments because the main computations are matrix-vector products like $X\boldsymbol{w}$. Vowpal Wabbit (VW) by Agarwal et al. (2014) uses a distributed L-BFGS method and outperforms stochastic gradient approaches. Zhuang et al. (2015); Lin et al. (2014) propose a distributed trust region Newton method (TRON) that works well and is faster than VW. These second-order Newton-type algorithms only work on differentiable primal problems like L2-SVM, but could not be applied to L1-SVM. Another popular algorithm for distributedly solving (1) without requiring differentiability is the alternating direction method of multipliers (ADMM) (Boyd et al., 2011). Zhang et al. (2012) applied ADMM to solve linear SVM problems. However, ADMM is known to converge slowly and these works do not provide convergence rates. Using Fenchel dual, Hazan et al. (2008) derived an algorithm that is similar to ADMM, and showed that their algorithm possesses global linear convergence in their reformulated problem for L1-SVM. However, there are Kl variables in their problem, and thus the training speed should be slower. Experiment results in Yang (2013); Zhuang et al. (2015) also verify that ADMM approaches are empirically slower than other methods.

4. Experiments

The following algorithms are compared in our experiments.

- DisDCA (Yang, 2013): we consider the practical variant because it is shown to be faster than the basic variant.
- TRON (Zhuang et al., 2015): this method only works on L2-SVM. We use the package MPI-LIBLINEAR 1.96.[2]
- DSVM-AVE (Pechyony et al., 2011).
- **BQO-E**: our box-constrained quadratic optimization al-

[2]Downloaded from http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/distributed-liblinear/.

Data set	ϵ	L1-SVM solvers				L2-SVM solvers				
		BQO-E	**BQO-A**	DisDCA	DSVM-AVE	**BQO-E**	**BQO-A**	DisDCA	DSVM-AVE	TRON
url	0.3	**2,162.8**	3,566.5	6,624.0	8,277.2	**68.1**	133.0	248.8	299.0	314.2
epsilon	0.01	3.4	**2.6**	3.0	2.8	**2.3**	5.2	2.8	3.7	6.7
webspam	0.01	35.1	29.8	**27.4**	42.3	**16.9**	29.6	25.9	40.3	123.7

Table 2. Training time required for a solver to reach $(f^P(\boldsymbol{w}^t) - f^P(\boldsymbol{w}^*)) \leq \epsilon f^P(\boldsymbol{w}^*)$. url uses larger ϵ because of longer training time.

Data set	ϵ	L1-SVM solvers				L2-SVM solvers			
		BQO-E	**BQO-A**	DisDCA	DSVM-AVE	**BQO-E**	**BQO-A**	DisDCA	DSVM-AVE
url	0.04	**1,092.9**	1,801.2	3,524.2	4,343.4	**1,328.6**	2,201.1	4,251.2	5,126.6
epsilon	0.01	**2.8**	3.9	8.0	13.2	**6.3**	10.9	24.4	28.1
webspam	0.01	**21.7**	30.6	54.7	79.5	**19.8**	30.3	54.3	84.9

Table 3. Training time required for a solver to reach $(f(\boldsymbol{\alpha}^*) - f(\boldsymbol{\alpha}^t)) \leq \epsilon f(\boldsymbol{\alpha}^*)$. url uses larger ϵ because of longer training time.

gorithm with exact line search.

- **BQO-A**: our algorithm with Armijo line search. We follow Tseng & Yun (2009) to use $\beta = 0.5, \sigma = 0.1, \gamma = 0$.

In order to have a fair comparison between algorithms, all methods are implemented using the MPI-LIBLINEAR structure to prevent training time differences resulting from different implementations. Note that the recent work Ma et al. (2015) uses the same algorithm as DisDCA so our comparison already includes it.

In BQO-E and BQO-A, $\tau = 0.001$ is used. All methods are implemented in C++/MPI. ADMM is excluded because DisDCA is reported to outperform it, and its speed is dependent on additional parameters. The code used in the experiments is available at `http://github.com/leepei/distcd_exp/`.

We compare different dual algorithms in terms of the relative dual objective function value.

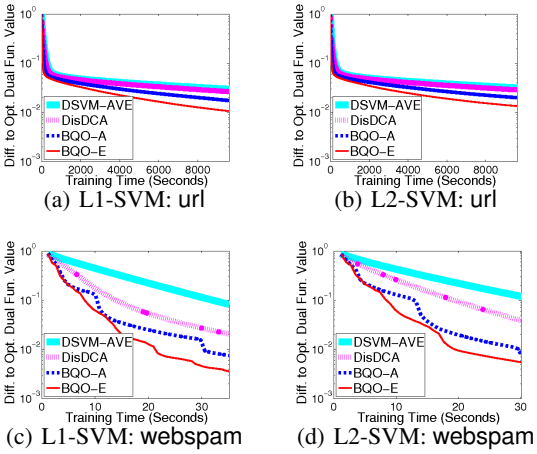
$$|(f(\boldsymbol{\alpha}^t) - f(\boldsymbol{\alpha}^*))/f(\boldsymbol{\alpha}^*)|, \tag{28}$$

$\boldsymbol{\alpha}^*$ is obtained approximately by running our method with a strict stopping condition. To have a fair comparison with TRON which solves the primal problem, we also report the relative primal objective function values and the accuracies relative to the accuracy at the optimum. The pocket approach discussed in Section 2.3 is adopted in all dual algorithms.

We use 16 nodes in a cluster. Each node has two Intel HP X5650 2.66GHZ 6C Processors, and one core per node is used. The MPI environment is MPICH2 (Gropp, 2002).

The statistics of the data sets in our experiments are shown in Table 1. All of them are publicly available.[3] Instances are split across machines in a uniform random fashion. We fix $C = 1$ in all experiments for a fair comparison in optimization. Therefore, some algorithms may have accuracies exceeding that of the optimal solution because the best C is not chosen. We note, though, that once parameter selection is conducted, the method that decreases the function value faster should also achieve the best accuracy faster.

[3] `http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets`.

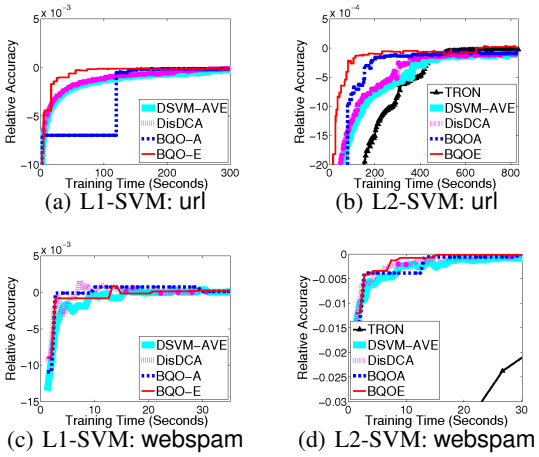


(a) L1-SVM: url (b) L2-SVM: url (c) L1-SVM: webspam (d) L2-SVM: webspam

Figure 1. Time versus relative dual objective value (28).

Since this work aims at a framework that can accommodate any local solvers, we use the same local solver setting for DisDCA, DSVM-AVE and our algorithm. In particular, at each time of solving (5), we randomly shuffle the instance order, and then let each machine pass through all local instances once. The training time could be improved for high-dimensional data if we use more inner iterations, but our setting provides a fair comparison by having computation-communication ratios similar to TRON.

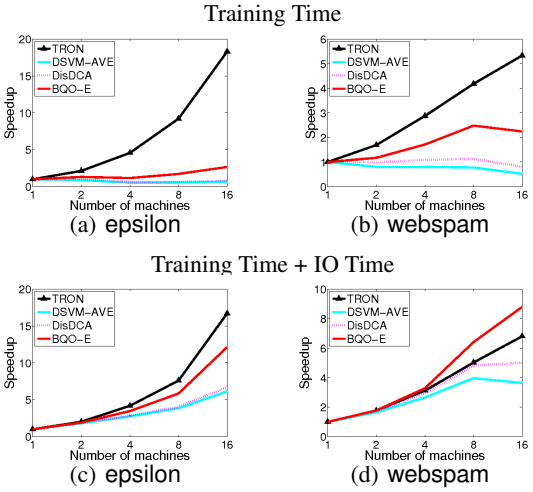
Tables 2-3 show the results of the primal and dual objective, respectively. More detailed examination of dual objective and relative accuracies are in Figures 1-2. Due to space limit, we only present results on two data sets. More results are discussed in the supplement. In most cases, BQO-E reduces the primal objective the fastest. Moreover, BQO-E always reduces the dual objective significantly faster than



(a) L1-SVM: url (b) L2-SVM: url

(c) L1-SVM: webspam (d) L2-SVM: webspam

Figure 2. Time versus relative accuracy.



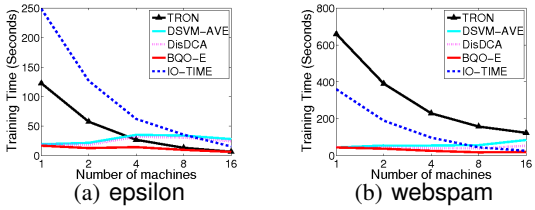
(a) epsilon (b) webspam

(c) epsilon (d) webspam

Figure 3. Speedup of training L2-SVM. Top: training time. Bottom: total running time including training and data loading time.

other solvers and almost always reaches stable accuracies the earliest. Because BQO-A is inferior to BQO-E in most cases, it is excluded from later comparisons.

We then examine the speedup of solving L2-SVM using different numbers of machines. In this comparison, we use the two largest data sets webspam and epsilon that represent the cases $l \ll n$ and $n \ll l$ respectively. The results are in Figure 3. We present both training time speedup and total running time speedup. In both data sets, the training time speedup of our algorithm is worse than TRON but better than other methods. But BQO-E has significantly better speedup of overall running time when the I/O time is included. A further investigation of the training time and data loading time in Figure 4 shows that this running time speedup of BQO-E results from the fact that BQO-E has shorter training time in all cases. Thus, the data loading



(a) epsilon (b) webspam

Figure 4. Training time (I/O time excluded) of L2-SVM using different numbers of machines.

time is the bottleneck of the running time in BQO-E, and decreasing the data loading time can significantly improve the running time, even if its training time does not improve much with more machines.

From the results above, our method is significantly faster than the state-of-the-art primal solver TRON and all existing distributed dual linear SVM algorithms.

Note that the comparison between DSVM-AVE and DisDCA accords the results in the results in Ma et al. (2015) that when smaller λ (equivalent to larger weight on the loss term) is used, the difference between the two algorithms is less significant. This can also be verified by the result on the url data set that has larger l and thus a larger loss term with a fixed C, which is also equivalent to a λ smaller than that being considered in Ma et al. (2015). Additional experiments in the supplement show that for smaller C, the differences are significant and DisDCA is superior. But in these cases, most algorithms finish training in a very short time and thus the setting of smaller C does not provide meaningful comparisons.

5. Conclusions

In this paper, we present an efficient method for distributedly solving linear SVM dual problems. Our algorithm is shown to have better theoretical convergence rate and faster empirical training time than state-of-the-art algorithms. We plan to extend this work to problems like structured SVM (Tsochantaridis et al., 2005) where optimizing the primal problem is difficult. Based on this work, we have extended the package MPI-LIBLINEAR (after version 1.96) at `http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/distributed-liblinear/` to include the proposed implementation.

Acknowledgment

This material is based on research sponsored by DARPA under agreement number FA8750-13-2-0008. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of DARPA or the U.S. Government.

The authors thank the Illinois Campus Cluster Program for providing computing resources required to conduct experiments in this work. We also thank the anonymous reviewers for their comments, and thank Eric Horn for proofreading the paper. Ching-Pei Lee thanks Po-Wei Wang for fruitful discussion and great help, thanks Chih-Jen Lin, Hsiang-Fu Yu, and Hsuan-Tien Lin for their valuable suggestions, thanks Martin Jaggi for the pointer to the convergence proof of the practical variant of DisDCA, and thanks Tianbao Yang for discussion on DisDCA.

References

Agarwal, Alekh, Chapelle, Olivier, Dudik, Miroslav, and Langford, John. A reliable effective terascale linear learning system. *Journal of Machine Learning Research*, 15:1111–1133, 2014.

Bian, Yatao, Li, Xiong, and Liu, Yuncai. Parallel coordinate descent Newton for large-scale L1-regularized minimization. Technical report, 2014. arXiv:1306.4080v3.

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

Boyd, Stephen, Parikh, Neal, Chu, Eric, Peleato, Borja, and Eckstein, Jonathan. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends in Machine Learning*, 3(1):1–122, 2011.

Chang, Kai-Wei and Roth, Dan. Selective block minimization for faster convergence of limited memory large-scale linear models. In *Proceedings of the Seventeenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2011.

Gallant, Stephen I. Perceptron-based learning algorithms. *Neural Networks, IEEE Transactions on*, 1(2):179–191, 1990.

Gropp, William. MPICH2: A new start for MPI implementations. In *Recent Advances in Parallel Virtual Machine and Message Passing Interface*, pp. 7–7. Springer, 2002.

Hazan, Tamir, Man, Amit, and Shashua, Amnon. A parallel decomposition solver for SVM: Distributed dual ascend using Fenchel duality. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1–8. IEEE, 2008.

Hsieh, Cho-Jui, Chang, Kai-Wei, Lin, Chih-Jen, Keerthi, S. Sathiya, and Sundararajan, Sellamanickam. A dual coordinate descent method for large-scale linear SVM. In *Proceedings of the Twenty Fifth International Conference on Machine Learning (ICML)*, 2008.

Lin, Chieh-Yen, Tsai, Cheng-Hao, pei Lee, Ching, and Lin, Chih-Jen. Large-scale logistic regression and linear support vector machines using Spark. In *Proceedings of the IEEE International Conference on Big Data*, pp. 519–528, 2014.

Ma, Chenxin, Smith, Virginia, Jaggi, Martin, Jordan, Michael I, Richtárik, Peter, and Takáč, Martin. Adding vs. averaging in distributed primal-dual optimization. In *Proceedings of the 32nd International Conference on Machine Learning (ICML)*, 2015.

Pang, Jong-Shi. A posteriori error bounds for the linearly-constrained variational inequality problem. *Mathematics of Operations Research*, 12(3):474–484, 1987.

Pechyony, Dmitry, Shen, Libin, and Jones, Rosie. Solving large scale linear SVM with distributed block minimization. In *Neural Information Processing Systems Workshop on Big Learning: Algorithms, Systems, and Tools for Learning at Scale*, 2011.

Shalev-Shwartz, Shai and Zhang, Tong. Stochastic dual coordinate ascent methods for regularized loss minimization. *Journal of Machine Learning Research*, 14:567–599, 2013.

Tseng, Paul and Yun, Sangwoon. A coordinate gradient descent method for nonsmooth separable minimization. *Mathematical Programming*, 117:387–423, 2009.

Tsochantaridis, Ioannis, Joachims, Thorsten, Hofmann, Thomas, and Altun, Yasemin. Large margin methods for structured and interdependent output variables. *Journal of Machine Learning Research*, 6:1453–1484, 2005.

Vapnik, Vladimir. *The Nature of Statistical Learning Theory*. Springer-Verlag, New York, NY, 1995.

Wang, Po-Wei and Lin, Chih-Jen. Iteration complexity of feasible descent methods for convex optimization. *Journal of Machine Learning Research*, 15:1523–1548, 2014.

Yang, Tianbao. Trading computation for communication: Distributed stochastic dual coordinate ascent. In *Advances in Neural Information Processing Systems 26*, pp. 629–637, 2013.

Yang, Tianbao, Zhu, Shenghuo, Jin, Rong, and Lin, Yuanqing. On theoretical analysis of distributed stochastic dual coordinate ascent. Technical report, 2014. arXiv preprint arXiv:1312.1031.

Yu, Hsiang-Fu, Hsieh, Cho-Jui, Chang, Kai-Wei, and Lin, Chih-Jen. Large linear classification when data cannot fit in memory. *ACM Transactions on Knowledge Discovery from Data*, 5(4):23:1–23:23, February 2012.

Yuan, Guo-Xun, Ho, Chia-Hua, and Lin, Chih-Jen. Recent advances of large-scale linear classification. *Proceedings of the IEEE*, 100(9):2584–2603, 2012.

Yun, Sangwoon. On the iteration complexity of cyclic coordinate gradient descent methods. *SIAM Journal on Optimization*, 24(3):1567–1580, 2014.

Zhang, Caoxie, Lee, Honglak, and Shin, Kang G. Efficient distributed linear classification algorithms via the alternating direction method of multipliers. In *Proceedings of the 15th International Conference on Artificial Intelligence and Statistics*, 2012.

Zhuang, Yong, Chin, Wei-Sheng, Juan, Yu-Chin, and Lin, Chih-Jen. Distributed Newton method for regularized logistic regression. In *Proceedings of The Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD)*, 2015.

Supplementary Materials

A. More Details on Convergence Analysis

In this section, we will establish a detailed convergence analysis for our methods. The main idea follows from the framework of Tseng & Yun (2009) and substitutes the local error bound with the global error bound given in Wang & Lin (2014). Note that the result in Yun (2014) does not directly apply here because $H = I$ is assumed in that work, but their analysis can also be modified for other symmetric, positive definite H by applying lemmas in Section 3 of Tseng & Yun (2009) to get the same result as ours.

For any matrix A, let $\lambda_{\min}(A)$ denotes the smallest eigenvalue of A and $\lambda_{\max}(A)$ denotes the largest eigenvalue of A.

A.1. L2-SVM

For L2-SVM, we have that $\lambda_{\min}(\bar{Q}) \geq 1/2C$, so clearly f is $1/2C$ strongly convex. In other words, $\forall \boldsymbol{\alpha}_1, \boldsymbol{\alpha}_2 \geq \mathbf{0}$,

$$\begin{aligned}\frac{1}{2C}\|\boldsymbol{\alpha}_1 - \boldsymbol{\alpha}_2\|^2 \leq (\boldsymbol{\alpha}_1 - \boldsymbol{\alpha}_2)^T \bar{Q}(\boldsymbol{\alpha}_1 - \boldsymbol{\alpha}_2) \\ =(\nabla f(\boldsymbol{\alpha}_1) - \nabla f(\boldsymbol{\alpha}_2))^T(\boldsymbol{\alpha}_1 - \boldsymbol{\alpha}_2).\end{aligned}$$

Also the gradient of f is $\lambda_{\max}(\bar{Q})$ Lipschitz continuous

$$\begin{aligned}\|\nabla f(\boldsymbol{\alpha}_1) - \nabla f(\boldsymbol{\alpha}_2)\| = \sqrt{(\boldsymbol{\alpha}_1 - \boldsymbol{\alpha}_2)^T \bar{Q}^2(\boldsymbol{\alpha}_1 - \boldsymbol{\alpha}_2)} \\ \leq \lambda_{\max}(\bar{Q})\|\boldsymbol{\alpha}_1 - \boldsymbol{\alpha}_2\|. \qquad (29)\end{aligned}$$

Thus by Theorem 3.1 in Pang (1987), we have

$$\|\boldsymbol{\alpha} - \boldsymbol{\alpha}^*\| \leq 2C(1 + \lambda_{\max}(\bar{Q}))\|\nabla^+ f(\boldsymbol{\alpha})\|,$$

where

$$\begin{aligned}\nabla^+ f(\boldsymbol{\alpha}) &\equiv \boldsymbol{\alpha} - [\boldsymbol{\alpha} - \nabla f(\boldsymbol{\alpha})]^+_{\mathcal{P}(\mathbf{0})}, \\ [\boldsymbol{\alpha}]^+_{\mathcal{P}(\mathbf{0})} &\equiv \operatorname*{arg\,min}_{\boldsymbol{\beta} \in \mathcal{P}(\mathbf{0})} \|\boldsymbol{\alpha} - \boldsymbol{\beta}\|.\end{aligned}$$

Note that in our case, the definition of $\nabla^+ f(\boldsymbol{\alpha})$ is equivalent to that of $\boldsymbol{d}_I(\boldsymbol{\alpha})$ in Assumption 2(a) in Tseng & Yun (2009). Thus this assumption is satisfied with $\tau = 2C(1 + \lambda_{\max}(\bar{Q})), \epsilon = \infty$ for any ξ. Also note that Assumption 2(b) of Tseng & Yun (2009) always holds in convex optimization problems. Therefore, $\bar{k} = 0, \tau' = \tau$ in Equation (36), and $\hat{k} = 0$ in Equation (37) of Tseng & Yun (2009). Following their analysis and their Lemma 5(b), we then have

$$f(\boldsymbol{\alpha}^{t+1}) - f(\boldsymbol{\alpha}^*) \leq \frac{C_2}{1 + C_2}(f(\boldsymbol{\alpha}^t) - f(\boldsymbol{\alpha}^*)), \forall t \in \mathbf{N} \cup \{0\}, \qquad (30)$$

where

$$C_2 = C_1/(\sigma\beta \min\{1, \frac{1 - \sigma + \sigma\gamma}{C\lambda_{\max}(\bar{Q})}\}), \qquad (31)$$

and C_1 is a constant that only depends on $\lambda_{\max}(\bar{Q}), \lambda_{\min}(\bar{Q}), \lambda_{\max}(H), \lambda_{\min}(H)$, and C.

A.2. L1-SVM

For the case of L1-SVM, we see that the problem is of the form

$$\begin{aligned}\min_{\boldsymbol{\alpha} \in \mathbf{R}^l} \quad & f(\boldsymbol{\alpha}) = g((YX)^T\boldsymbol{\alpha}) - \boldsymbol{e}^T\boldsymbol{\alpha} \\ \text{subject to} \quad & \boldsymbol{\alpha} \in \mathcal{P}(\mathbf{0}),\end{aligned}$$

where

$$g(\cdot) = \frac{1}{2}\|\cdot\|^2$$

is 1 strongly convex. From (29), we know that the gradient of f is $\lambda_{\max}(\bar{Q})$ Lipschitz continuous. In addition, it is clear that $\mathcal{P}(\mathbf{0})$ is a polyhedral set. Thus, according to Theorem 4.6 in Wang & Lin (2014), Assumption 2(a) of Tseng & Yun (2009) is also satisfied with a τ that depends on $C, \lambda_{\max}(\bar{Q}), \lambda_{\max}(\bar{H}), f(\boldsymbol{\alpha}^0) - f(\boldsymbol{\alpha}^*), \|\nabla f(\boldsymbol{\alpha}^*)\|, (\boldsymbol{\alpha}^*)^T Q \boldsymbol{\alpha}^*$ and $\tilde{\tau}$ in (6). We can then substitute this result into (31) to get a similar result to (30). Since the rates are related to the eigenvalues of H, it will be interesting to consider this property to construct H that has a better convergence rate upper bound. We leave this as a future research direction.

A.3. Discussion

Our analysis indicates that the convergence speed of our method in (30) is independent of both l and n. Thus the iteration complexity depends on l only via the optimal function value, which is upper bounded by $f^P(\mathbf{0}) = Cl$. If we consider the scaled problem $f^p(\boldsymbol{w})/Cl$ and $f(\boldsymbol{\alpha})/Cl$ used in other works including Yang (2013); Ma et al. (2015), then the iteration complexity of our method is totally independent of l and n.

This result is not surprising, because we are considering the rounds of communication and outer iteration, while the overall training time might still be dependent on l and n. Note that the data dimension n does not affect the number of variables being optimized, and thus does not contribute to the iteration complexity. However, note that n affects the training time for solving the local-sub-problems at each iteration, and the communication cost is linear to n as long as a method synchronizes $\boldsymbol{w}$. Also, in the analysis of Wang & Lin (2014) for using cyclic coordinate descent method to train linear SVM, the training time depends on l because in this method, the definition of one iteration is passing through all instances once and thus the running time will be at least l times the iteration complexity. But since our framework can use any sub-problem solver, this is might be avoided by considering a solver whose training time is independent of l.

B. Additional Experiment Results

In this section, we provide more experimental results.

B.1. Results Appeared Partially in Section 4

We present more results under the same setting of Section 4. Figure 5 presents the dual objective values and accuracies of epsilon. Figure 6 shows the primal objective values of all data sets.

B.2. Training Time Profile

To better understand the bottleneck of the training time, we investigate the fraction of the total training time taken by computation, synchronization and communication. The result is presented in Figure 7. By synchronization cost, we mean the time amount spent when one machine finished solving its corresponding sub-problem (7) but has not started communication because it is idle to wait for other machines to finish solving (7). Note that since in our experimental setting, each solver conducts one round of size-n communication every time it goes through the whole data once, the behaviors in communication and synchronization of all solvers should be similar. Therefore, we only report the result of BQO-E. We can see that for sparse data sets url and webspam, the proportion of computation time is rather low and synchronization time is larger. This means that each machine spent a different amount of time to solve the local sub-problem. Thus, one might consider setting the number of instances being examined at each iteration to be identical for all machines to reduce the synchronization time. This will be another advantage over primal batch solvers that require exactly one pass through all instances at each round of communication.

B.3. Speedup On Primal Objective Value

In Section 4, the training time of the speedup experiments is obtained by the following. We reported the time for TRON to reach $\boldsymbol{w}^t$ satisfying

$$|\frac{f^P(\boldsymbol{w}^t) - f^P(\boldsymbol{w}^*)}{f^P(\boldsymbol{w}^*)}| \leq 0.01, \tag{32}$$

where $\boldsymbol{w}^*$ is the optimal solution of (1). For DSVM-AVE and BQO-E that minimize (2), we report the time for them to obtain $\boldsymbol{\alpha}^t$ such that

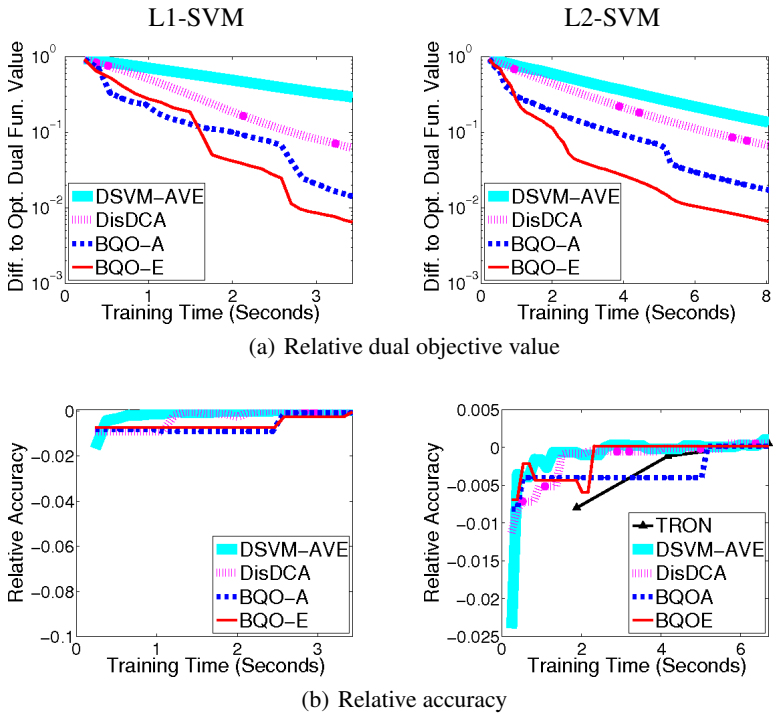
$$|\frac{f(\boldsymbol{\alpha}^t) - f(\boldsymbol{\alpha}^*)}{f(\boldsymbol{\alpha}^*)}| \leq 0.01.$$

Here we report the result of considering (32) for DSVM-AVE and BQO-E in Figures 8-9. In this setting, both DSVM-AVE and BQO-E have better speedups. Also, DSVM-AVE and BQO-E have similar training time in epsilon, but BQO-E is significantly better in webspam in terms of both training time and speedup.

B.4. Experiments on Different Values of C

We also conduct additional experiments using different values of C. For the data sets we considered, $C = 1$ is already large enough because the number of instances is large and the loss term is summation instead of average over all instances. Thus we consider smaller C. Tables 4-5 present the result of $C = 1e-2$ while Tables 6-7 present the result of $C = 1e-4$. We can see that when the problems are

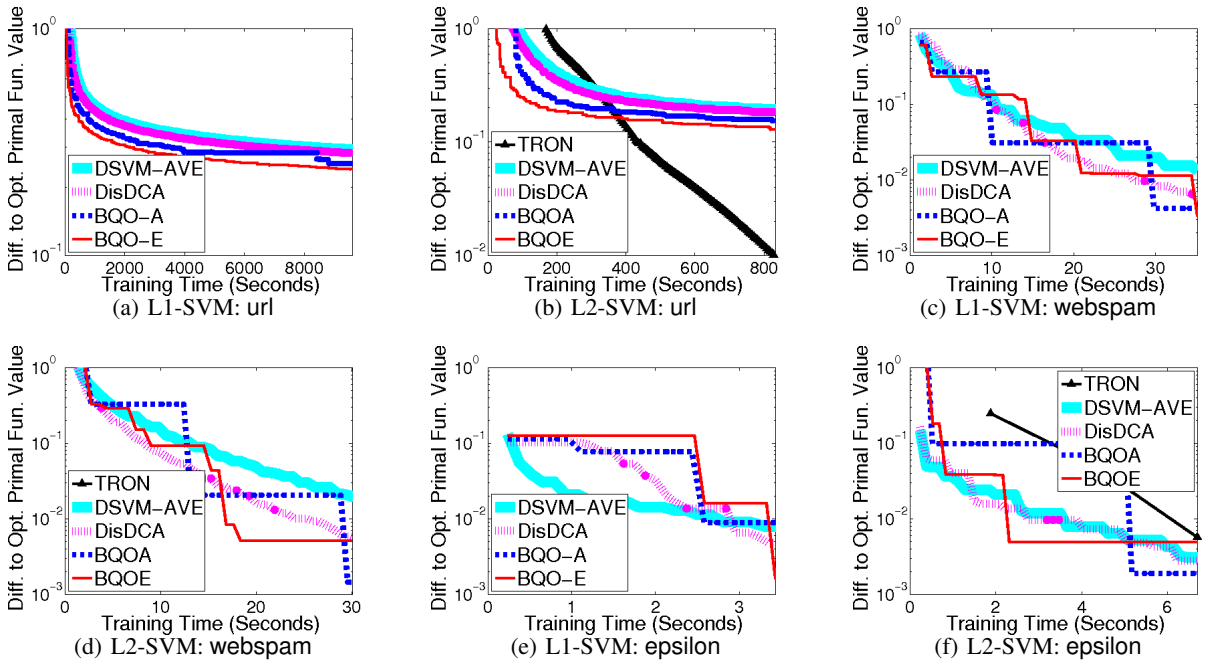
easier to solve, i.e., when the training time of all methods is short, DisDCA is usually the fastest, while our method is still faster than DSVM-AVE and TRON in most cases. However, in these situations, the training time does not affect the total running time too much, because in this case the data loading time is the bottleneck. Moreover, note that our line search methods are also applicable to DisDCA to further improve the training time in this situation.



(a) Relative dual objective value

(b) Relative accuracy

Figure 5. Experiment results on epsilon. Top: time versus relative dual objective value. Bottom: time versus relative accuracy.



(a) L1-SVM: url (b) L2-SVM: url (c) L1-SVM: webspam

(d) L2-SVM: webspam (e) L1-SVM: epsilon (f) L2-SVM: epsilon

Figure 6. Time versus relative primal objective value. Time is in seconds. Top: L1-SVM, bottom: L2-SVM. Note that in webspam, the function values of TRON for L2-SVM are too large to appear in the figure.

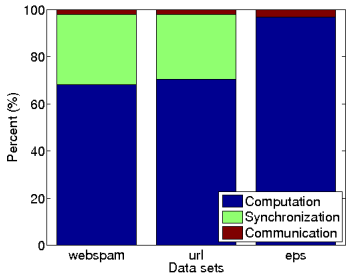


Figure 7. Percentage of computation, synchronization, and communication in the total training time.

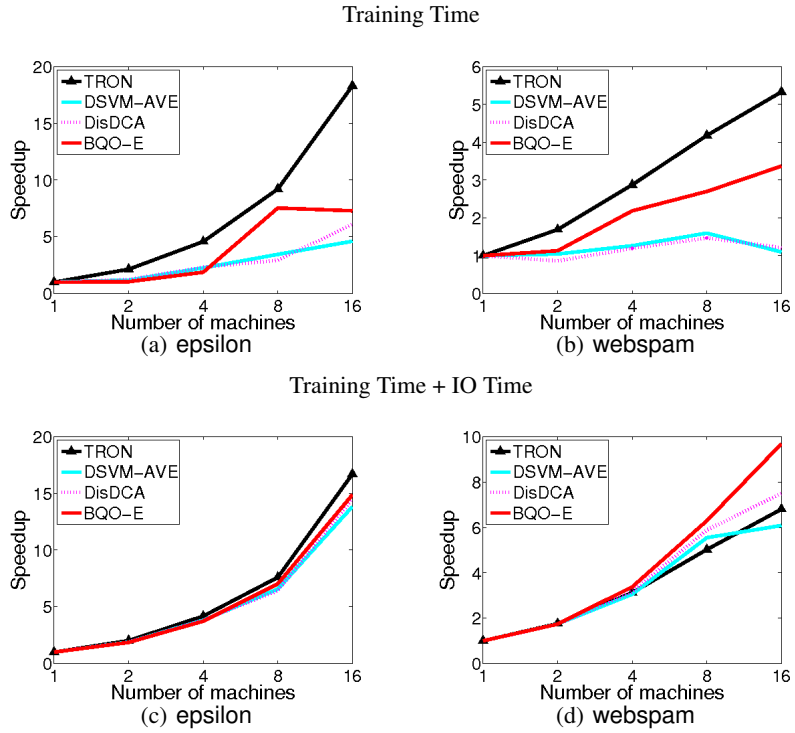


Figure 8. Speedup of different methods training L2-SVM. All algorithms use (32) to decide training time. Top: speedup of training time. Bottom: speedup of the total running time including training and data loading time.

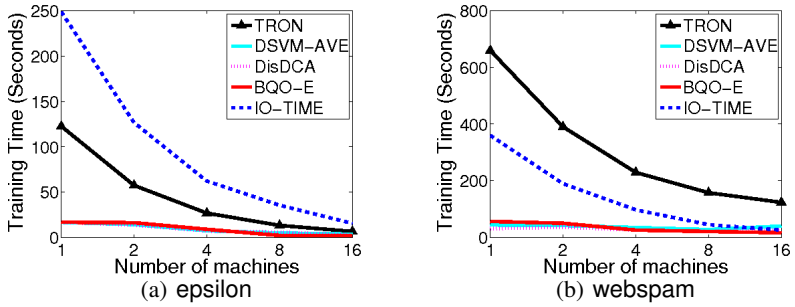


Figure 9. Training time of L2-SVM using different numbers of machines. All algorithms use (32) to decide training time.

Data set	ϵ	L1-SVM solvers				L2-SVM solvers				
		BQO-E	**BQO-A**	DisDCA	DSVM-AVE	**BQO-E**	**BQO-A**	DisDCA	DSVM-AVE	TRON
url	0.01	21.4	**21.4**	70.5	79.3	**42.7**	146.8	98.3	115.5	46.9
epsilon	0.01	2.1	1.2	**0.4**	2.2	1.6	3.3	**0.4**	1.6	3.1
webspam	0.01	17.1	14.1	**1.8**	15.2	6.6	9.0	**1.8**	12.0	35.9

Table 4. Training time required for a solver to reach $(f^P(\boldsymbol{w}^t) - f^P(\boldsymbol{w}^*)) \leq \epsilon f^P(\boldsymbol{w}^*)$. We present results of $C = 0.01$.

Data set	ϵ	L1-SVM solvers				L2-SVM solvers			
		BQO-E	**BQO-A**	DisDCA	DSVM-AVE	**BQO-E**	**BQO-A**	DisDCA	DSVM-AVE
url	0.01	**10.8**	11.3	36.4	41.3	**296.6**	593.1	1,025.6	1,203.0
epsilon	0.01	2.0	1.6	**0.5**	12.2	1.8	2.5	**0.6**	6.4
webspam	0.01	11.3	16.4	**2.4**	56.2	8.4	10.7	**2.6**	28.6

Table 5. Training time required for a solver to reach $(f(\boldsymbol{\alpha}^*) - f(\boldsymbol{\alpha}^t)) \leq \epsilon f(\boldsymbol{\alpha}^*)$. We present results of $C = 0.01$.

Data set	ϵ	L1-SVM solvers				L2-SVM solvers				
		BQO-E	**BQO-A**	DisDCA	DSVM-AVE	**BQO-E**	**BQO-A**	DisDCA	DSVM-AVE	TRON
url	0.01	3.3	5.9	**1.1**	3.1	1.3	9.1	**0.8**	2.7	9.2
epsilon	0.01	**0.3**	0.3	0.3	0.9	0.3	0.3	**0.3**	2.1	0.9
webspam	0.01	5.9	7.5	**1.9**	19.4	10.1	8.1	**2.1**	15.3	13.4

Table 6. Training time required for a solver to reach $(f^P(\boldsymbol{w}^t) - f^P(\boldsymbol{w}^*)) \leq \epsilon f^P(\boldsymbol{w}^*)$. We present results of $C = 0.0001$.

Data set	ϵ	L1-SVM solvers				L2-SVM solvers			
		BQO-E	**BQO-A**	DisDCA	DSVM-AVE	**BQO-E**	**BQO-A**	DisDCA	DSVM-AVE
url	0.01	2.7	6.1	**0.8**	11.9	**2.2**	3.6	5.2	8.9
epsilon	0.01	**0.3**	0.3	0.3	12.3	0.3	0.4	**0.3**	6.0
webspam	0.01	6.7	7.5	**1.9**	65.7	10.1	7.1	**2.1**	30.1

Table 7. Training time required for a solver to reach $(f(\boldsymbol{\alpha}^*) - f(\boldsymbol{\alpha}^t)) \leq \epsilon f(\boldsymbol{\alpha}^*)$. We present results of $C = 0.0001$.