Stochastic Primal-Dual Coordinate Method for Regularized Empirical Risk Minimization

Yuchen Zhang* Lin Xiao†

September 2014

Abstract

We consider a generic convex optimization problem associated with regularized empirical risk minimization of linear predictors. The problem structure allows us to reformulate it as a convex-concave saddle point problem. We propose a stochastic primal-dual coordinate (SPDC) method, which alternates between maximizing over a randomly chosen dual variable and minimizing over the primal variable. An extrapolation step on the primal variable is performed to obtain accelerated convergence rate. We also develop a mini-batch version of the SPDC method which facilitates parallel computing, and an extension with weighted sampling probabilities on the dual variables, which has a better complexity than uniform sampling on unnormalized data. Both theoretically and empirically, we show that the SPDC method has comparable or better performance than several state-of-the-art optimization methods.

1 Introduction

We consider a generic convex optimization problem that arises often in machine learning: regularized empirical risk minimization (ERM) of linear predictors. More specifically, let $a_1, \ldots, a_n \in \mathbb{R}^d$ be the feature vectors of n data samples, $\phi_i : \mathbb{R} \to \mathbb{R}$ be a convex loss function associated with the linear prediction $a_i^T x$, for $i = 1, \ldots, n$, and $g : \mathbb{R}^d \to \mathbb{R}$ be a convex regularization function for the predictor $x \in \mathbb{R}^d$. Our goal is to solve the following optimization problem:

$$\underset{x \in \mathbb{R}^d}{\text{minimize}} \quad \left\{ P(x) \stackrel{\text{def}}{=} \frac{1}{n} \sum_{i=1}^{n} \phi_i(a_i^T x) + g(x) \right\}. \tag{1}$$

Examples of the above formulation include many well-known classification and regression problems. For binary classification, each feature vector a_i is associated with a label $b_i \in \{\pm 1\}$. We obtain the linear SVM (support vector machine) by setting $\phi_i(z) = \max\{0, 1 - b_i z\}$ (the hinge loss) and $g(x) = (\lambda/2)\|x\|_2^2$, where $\lambda > 0$ is a regularization parameter. Regularized logistic regression is obtained by setting $\phi_i(z) = \log(1 + \exp(-b_i z))$. For linear regression problems, each feature vector a_i is associated with a dependent variable $b_i \in \mathbb{R}$, and $\phi_i(z) = (1/2)(z - b_i)^2$. Then we get ridge regression with $g(x) = (\lambda/2)\|x\|_2^2$, and the Lasso with $g(x) = \lambda\|x\|_1$. Further backgrounds on regularized ERM in machine learning and statistics can be found, e.g., in the book [13].

*Department of Electrical Engineering and Computer Science, University of California, Berkekey, CA 94720, USA. Email: yuczhang@eecs.berkeley.edu. (This work was perfomed during an internship at Microsoft Research.)

†Machine Learning Groups, Microsoft Research, Redmond, WA 98053, USA. Email: lin.xiao@microsoft.com.

We are especially interested in developing efficient algorithms for solving problem (1) when the number of samples n is very large. In this case, evaluating the full gradient or subgradient of the function $P(x)$ is very expensive, thus incremental methods that operate on a single component function ϕ_i at each iteration can be very attractive. There have been extensive research on incremental (sub)gradient methods (e.g. [40, 4, 21, 2, 3]) as well as variants of the stochastic gradient method (e.g., [46, 5, 11, 17, 43]). While the computational cost per iteration of these methods is only a small fraction, say $1/n$, of that of the batch gradient methods, their iteration complexities are much higher (it takes many more iterations for them to reach the same precision). In order to better quantify the complexities of various algorithms and position our contributions, we need to make some concrete assumptions and introduce the notion of condition number and batch complexity.

1.1 Condition number and batch complexity

Let γ and λ be two positive real parameters. We make the following assumption:

Assumption A. *Each ϕ_i is convex and differentiable, and its derivative is $(1/\gamma)$-Lipschitz continuous (same as ϕ_i being $(1/\gamma)$-smooth), i.e.,*

$$|\phi_i'(\alpha) - \phi_i'(\beta)| \leq (1/\gamma)|\alpha - \beta|, \quad \forall\, \alpha, \beta \in \mathbb{R}, \quad i = 1, \ldots, n.$$

In addition, the regularization function g is λ-strongly convex, i.e.,

$$g(y) \geq g(x) + g'(y)^T(x - y) + \frac{\lambda}{2}\|x - y\|_2^2, \quad \forall\, g'(y) \in \partial g(y), \quad x, y \in \mathbb{R}^n.$$

For example, the logistic loss $\phi_i(z) = \log(1 + \exp(-b_i z))$ is $(1/4)$-smooth, the squared error $\phi_i(z) = (1/2)(z - b_i)^2$ is 1-smooth, and the squared ℓ_2-norm $g(x) = (\lambda/2)\|x\|_2^2$ is λ-strongly convex. The hinge loss $\phi_i(z) = \max\{0, 1 - b_i z\}$ and the ℓ_1-regularization $g(x) = \lambda\|x\|_1$ do not satisfy Assumption A. Nevertheless, we can treat them using smoothing and strongly convex perturbations, respectively, so that our algorithm and theoretical framework still apply (see Section 3.1).

Under Assumption A, the gradient of each component function, $\nabla\phi_i(a_i^T x)$, is also Lipschitz continuous, with Lipschitz constant $L_i = \|a_i\|_2^2/\gamma \leq R^2/\gamma$, where $R = \max_i \|a_i\|_2$. In other words, each $\phi_i(a_i^T x)$ is (R^2/γ)-smooth. We define a *condition number* $\kappa = R^2/(\lambda\gamma)$, and focus on ill-conditioned problems where $\kappa \gg 1$. In the statistical learning context, the regularization parameter λ is usually on the order of $1/\sqrt{n}$ or $1/n$ (e.g., [6]), thus κ is on the order of $\sqrt{n}$ or n. It can be even larger if the strong convexity in g is added purely for numerical regularization purposes (see Section 3.1). We note that the actual conditioning of problem (1) may be better than κ, if the empirical loss function $(1/n)\sum_{i=1}^n \phi_i(a_i^T x)$ by itself is strongly convex. In those cases, our complexity estimates in terms of κ can be loose (upper bounds), but they are still useful in comparing different algorithms for solving the same given problem.

Let $P^\star$ be the optimal value of problem (1), i.e., $P^\star = \min_{x \in \mathbb{R}^d} P(x)$. In order to find an approximate solution $\hat{x}$ satisfying $P(\hat{x}) - P^\star \leq \epsilon$, the classical full gradient method and its proximal variants require $\mathcal{O}((1 + \kappa)\log(1/\epsilon))$ iterations (e.g., [24, 26]). Accelerated full gradient (AFG) methods [24, 41, 1, 26] enjoy the improved iteration complexity $\mathcal{O}((1 + \sqrt{\kappa})\log(1/\epsilon))$.[1] However, each iteration of these batch methods requires a full pass over the dataset, computing the gradient

[1] For the analysis of full gradient methods, we should use $(R^2/\gamma + \lambda)/\lambda = 1 + \kappa$ as the condition number of problem (1); see [26, Section 5.1]. Here we used the upper bound $\sqrt{1 + \kappa} < 1 + \sqrt{\kappa}$ for easy comparison. When $\kappa \gg 1$, the additive constant 1 can be dropped.

of each component function and forming their average, which cost $\mathcal{O}(nd)$ operations (assuming the features vectors $a_i \in \mathbb{R}^d$ are dense). In contrast, the stochastic gradient method and its proximal variants operate on one single component $\phi_i(a_i^T x)$ (chosen randomly) at each iteration, which only costs $\mathcal{O}(d)$. But their iteration complexities are far worse. Under Assumption A, it takes them $\mathcal{O}(\kappa/\epsilon)$ iterations to find an $\hat{x}$ such that $\mathbb{E}[P(\hat{x}) - P^\star] \leq \epsilon$, where the expectation is with respect to the random choices made at all the iterations (see, e.g., [30, 23, 11, 17, 43]).

To make fair comparisons with batch methods, we measure the complexity of stochastic or incremental gradient methods in terms of the number of equivalent passes over the dataset required to reach an expected precision ϵ. We call this measure the *batch complexity*, which are usually obtained by dividing their iteration complexities by n. For example, the batch complexity of the stochastic gradient method is $\mathcal{O}(\kappa/(n\epsilon))$. The batch complexities of full gradient methods are the same as their iteration complexities.

By carefully exploiting the finite average structure in (1) and other similar problems, several recent work [32, 36, 16, 44] proposed new variants of the stochastic gradient or dual coordinate ascent methods and obtained the iteration complexity $\mathcal{O}((n+\kappa)\log(1/\epsilon))$. Since their computational cost per iteration is $\mathcal{O}(d)$, the equivalent batch complexity is $\mathcal{O}((1+\kappa/n)\log(1/\epsilon))$. This complexity has much weaker dependence on n than the full gradient methods, and also much weaker dependence on ϵ than the stochastic gradient methods. In this paper, we present a new algorithm that has the batch complexity

$$\mathcal{O}\big((1+\sqrt{\kappa/n})\log(1/\epsilon)\big), \tag{2}$$

which is more efficient when $\kappa > n$.

1.2 Outline of the paper

Our approach is based on reformulating problem (1) as a convex-concave saddle point problem, and then devising a primal-dual algorithm to approximate the saddle point. More specifically, we replace each component function $\phi_i(a_i^T x)$ through convex conjugation, i.e.,

$$\phi_i(a_i^T x) = \sup_{y_i \in \mathbb{R}} \left\{ y_i \langle a_i, x\rangle - \phi_i^*(y_i) \right\},$$

where $\phi_i^*(y_i) = \sup_{\alpha\in\mathbb{R}}\{\alpha y_i - \phi_i(\alpha)\}$, and $\langle a_i, x\rangle$ denotes the inner product of a_i and x (which is the same as $a_i^T x$, but is more convenient for later presentation). This leads to a convex-concave saddle point problem

$$\min_{x\in\mathbb{R}^d} \max_{y\in\mathbb{R}^n} \left\{ f(x,y) \overset{\text{def}}{=} \frac{1}{n}\sum_{i=1}^n \big(y_i\langle a_i, x\rangle - \phi_i^*(y_i)\big) + g(x) \right\}. \tag{3}$$

Under Assumption A, each ϕ_i^* is γ-strongly convex (since ϕ_i is $(1/\gamma)$-smooth; see, e.g., [14, Theorem 4.2.2]) and g is λ-strongly convex. As a consequence, the saddle point problem (3) has a unique solution, which we denote by $(x^\star, y^\star)$.

In Section 2, we propose a stochastic primal-dual coordinate (SPDC) method, which alternates between maximizing f over a randomly chosen dual coordinate y_i and minimizing f over the primal variable x. We also apply an extrapolation step to the primal variable x to accelerate the convergence. The SPDC method has iteration complexity $\mathcal{O}((n+\sqrt{\kappa n}))\log(1/\epsilon))$. Since each iteration of SPDC only operates on a single dual coordinate y_i, its batch complexity is given by (2). We also present a mini-batch SPDC algorithm which is well suited for distributed computing.

Algorithm 1: The SPDC method

Input: parameters $\tau, \sigma, \theta \in \mathbb{R}_+$, number of iterations T, and initial points $x^{(0)}$ and $y^{(0)}$.

Initialize: $\overline{x}^{(0)} = x^{(0)}$, $u^{(0)} = (1/n)\sum_{i=1}^n y_i^{(0)} a_i$.

for $t = 0, 1, 2, \ldots, T-1$ **do**

Pick an index $k \in \{1, 2, \ldots, n\}$ uniformly at random, and execute the following updates:

$$y_i^{(t+1)} = \begin{cases} \arg\max_{\beta \in \mathbb{R}} \left\{ \beta \langle a_i, \overline{x}^{(t)} \rangle - \phi_i^*(\beta) - \frac{1}{2\sigma}(\beta - y_i^{(t)})^2 \right\} & \text{if } i = k, \\ y_i^{(t)} & \text{if } i \neq k, \end{cases} \tag{4}$$

$$x^{(t+1)} = \arg\min_{x \in \mathbb{R}^d} \left\{ g(x) + \left\langle u^{(t)} + (y_k^{(t+1)} - y_k^{(t)}) a_k,\ x \right\rangle + \frac{\|x - x^{(t)}\|_2^2}{2\tau} \right\}, \tag{5}$$

$$u^{(t+1)} = u^{(t)} + \frac{1}{n}(y_k^{(t+1)} - y_k^{(t)}) a_k, \tag{6}$$

$$\overline{x}^{(t+1)} = x^{(t+1)} + \theta(x^{(t+1)} - x^{(t)}). \tag{7}$$

end

Output: $x^{(T)}$ and $y^{(T)}$

In Section 3, we present two extensions of the SPDC method. We first explain how to solve problem (1) when Assumption A does not hold. The idea is to apply small regularizations to the saddle point function so that SPDC can still be applied, which results in accelerated sublinear rates. The second extension is a SPDC method with non-uniform sampling. The batch complexity of this algorithm has the same form as (2), but κ is defined as $\kappa = \bar{R}/(\lambda\gamma)$, where $\bar{R} = \frac{1}{n}\sum_{i=1}^n \|a_i\|$, which can be much smaller than $R = \max_i \|a_i\|$ if there is considerable variation in the norms $\|a_i\|$.

In Section 4, we discuss related work. In particular, the SPDC method can be viewed as a coordinate-update extension of the batch primal-dual algorithm developed by Chambolle and Pock [8]. We also discuss two very recent work [34, 18] which achieve the same batch complexity (2).

In Section 5, we discuss efficient implementation of the SPDC method when the feature vectors a_i are sparse. We focus on two popular cases: when g is a squared ℓ_2-norm penalty and when g is an $\ell_1 + \ell_2$ penalty. We show that the computational cost per iteration of SPDC only depends on the number of non-zero elements in the feature vectors.

In Section 6, we present experiment results comparing SPDC with several state-of-the-art optimization methods, including two efficient batch methods (AFG [24] and L-BFGS [27, Section 7.2]), the stochastic average gradient (SAG) method [32, 33], and the stochastic dual coordinate ascent (SDCA) method [36]. On all scenarios we tested, SPDC has comparable or better performance.

2 The SPDC method

In this section, we describe and analyze the Stochastic Primal-Dual Coordinate (SPDC) method. The basic idea of SPDC is quite simple: to approach the saddle point of $f(x, y)$ defined in (3), we alternatively maximize f with respect to y, and minimize f with respect to x. Since the dual vector y has n coordinates and each coordinate is associated with a feature vector $a_i \in \mathbb{R}^d$, maximizing f with respect to y takes $\mathcal{O}(nd)$ computation, which can be very expensive if n is large. We reduce the computational cost by randomly picking a single coordinate of y at a time, and

Algorithm 2: The Mini-Batch SPDC method

Input: mini-batch size m, parameters $\tau, \sigma, \theta \in \mathbb{R}_+$, number of iterations T, and $x^{(0)}$ and $y^{(0)}$.

Initialize: $\overline{x}^{(0)} = x^{(0)}$, $u^{(0)} = (1/n)\sum_{i=1}^n y_i^{(0)} a_i$.

for $t = 0, 1, 2, \ldots, T-1$ **do**

Randomly pick a subset of indices $K \subset \{1, 2, \ldots, n\}$ of size m, such that the probability of each index being picked is equal to m/n. Execute the following updates:

$$y_i^{(t+1)} = \begin{cases} \arg\max_{\beta \in \mathbb{R}} \left\{ \beta \langle a_i, \overline{x}^{(t)} \rangle - \phi_i^*(\beta) - \frac{1}{2\sigma}(\beta - y_i^{(t)})^2 \right\} & \text{if } i \in K, \\ y_i^{(t)} & \text{if } i \notin K, \end{cases} \tag{8}$$

$$x^{(t+1)} = \arg\min_{x \in \mathbb{R}^d} \left\{ g(x) + \left\langle u^{(t)} + \frac{1}{m} \sum_{k \in K} (y_k^{(t+1)} - y_k^{(t)}) a_k,\ x \right\rangle + \frac{\|x - x^{(t)}\|_2^2}{2\tau} \right\}, \tag{9}$$

$$u^{(t+1)} = u^{(t)} + \frac{1}{n} \sum_{k \in K} (y_k^{(t+1)} - y_k^{(t)}) a_k,$$

$$\overline{x}^{(t+1)} = x^{(t+1)} + \theta(x^{(t+1)} - x^{(t)}).$$

end

Output: $x^{(T)}$ and $y^{(T)}$

maximizing f only with respect to this coordinate. Consequently, the computational cost of each iteration is $\mathcal{O}(d)$.

We give the details of the SPDC method in Algorithm 1. The dual coordinate update and primal vector update are given in equations (4) and (5) respectively. Instead of maximizing f over y_k and minimizing f over x directly, we add two quadratic regularization terms to penalize $y_k^{(t+1)}$ and $x^{(t+1)}$ from deviating from $y_k^{(t)}$ and $x^{(t)}$. The parameters σ and τ control their regularization strength, which we will specify in the convergence analysis (Theorem 1). Moreover, we introduce two auxiliary variables $u^{(t)}$ and $\overline{x}^{(t)}$. From the initialization $u^{(0)} = (1/n)\sum_{i=1}^n y_i^{(0)} a_i$ and the update rules (4) and (6), we have

$$u^{(t)} = \frac{1}{n} \sum_{i=1}^n y_i^{(t)} a_i, \qquad t = 0, \ldots, T.$$

Equation (7) obtains $\overline{x}^{(t+1)}$ based on extrapolation from $x^{(t)}$ and $x^{(t+1)}$. This step is similar to Nesterov's acceleration technique [24, Section 2.2], and yields faster convergence rate.

Before presenting the theoretical results, we introduce a Mini-Batch SPDC method in Algorithm 2, which is a natural extension of SPDC in Algorithm 1. The difference between these two algorithms is that, the Mini-Batch SPDC method may simultaneously select more than one dual coordinates to update. Let m be the mini-batch size. During each iteration, the Mini-Batch SPDC method randomly picks a subset of indices $K \subset \{1, \ldots, n\}$ of size m, such that the probability of each index being picked is equal to m/n. The following is a simple procedure to achieve this. First, partition the set of indices into m disjoint subsets, so that the cardinality of each subset is equal to n/m (assuming m divides n). Then, during each iteration, randomly select a single index from each subset and add it to K. Other approaches for mini-batch selection are also possible.

With a single processor, each iteration of Algorithm 2 takes $\mathcal{O}(md)$ time to accomplish. Since

the updates of each coordinate y_k are independent of each other, we can use parallel computing to accelerate the Mini-Batch SPDC method. Concretely, we can use m processors to update the m coordinates in the subset K in parallel, then aggregate them to update $x^{(t+1)}$. Such a procedure can be achieved by a single round of communication, for example, using the `Allreduce` operation in MPI [20] or MapReduce [10]. If we ignore the communication delay, then each iteration takes $\mathcal{O}(d)$ time, which is the same as running one iteration of the basic SPDC algorithm. Not surprisingly, we will show that the Mini-Batch SPDC algorithm converges faster than SPDC in terms of the iteration complexity, because it processes multiple dual coordinates in a single iteration.

2.1 Convergence analysis

Since the basic SPDC algorithm is a special case of Mini-Batch SPDC with $m = 1$, we only present a convergence theorem for the mini-batch version.

Theorem 1. *Assume that each ϕ_i is $(1/\gamma)$-smooth and g is λ-strongly convex (Assumption A). Let $R = \max\{\|a_i\|_2 : i = 1, \dots, n\}$. If the parameters τ, σ and θ in Algorithm 2 are chosen such that*

$$\tau = \frac{1}{2R}\sqrt{\frac{m\gamma}{n\lambda}}, \qquad \sigma = \frac{1}{2R}\sqrt{\frac{n\lambda}{m\gamma}}, \qquad \theta = 1 - \frac{1}{(n/m) + R\sqrt{(n/m)/(\lambda\gamma)}}, \tag{10}$$

then for each $t \geq 1$, the Mini-Batch SPDC algorithm achieves

$$\begin{aligned} &\Big(\frac{1}{2\tau} + \lambda\Big)\mathbb{E}\big[\|x^{(t)} - x^\star\|_2^2\big] + \Big(\frac{1}{4\sigma} + \gamma\Big)\frac{\mathbb{E}\big[\|y^{(t)} - y^\star\|_2^2\big]}{m} \\ \leq\, &\theta^t\left(\Big(\frac{1}{2\tau} + \lambda\Big)\|x^{(0)} - x^\star\|_2^2 + \Big(\frac{1}{2\sigma} + \gamma\Big)\frac{\|y^{(0)} - y^\star\|_2^2}{m}\right). \end{aligned}$$

The proof of Theorem 1 is given in Appendix A. The following corollary establishes the expected iteration complexity of Mini-Batch SPDC for obtaining an ϵ-accurate solution.

Corollary 1. *Suppose Assumption A holds and the parameters τ, σ and θ are set as in* (10)*. In order for Algorithm 2 to obtain*

$$\mathbb{E}[\|x^{(T)} - x^\star\|_2^2] \leq \epsilon, \qquad \mathbb{E}[\|y^{(T)} - y^\star\|_2^2] \leq \epsilon, \tag{11}$$

it suffices to have the number of iterations T satisfy

$$T \geq \left(\frac{n}{m} + R\sqrt{\frac{n}{m\lambda\gamma}}\right)\log\left(\frac{C}{\epsilon}\right),$$

where

$$C = \frac{\big(1/(2\tau) + \lambda\big)\|x^{(0)} - x^\star\|_2^2 + \big(1/(2\sigma) + \gamma\big)\|y^{(0)} - y^\star\|_2^2/m}{\min\big\{1/(2\tau) + \lambda,\ (1/(4\sigma) + \gamma)/m\big\}}.$$

Proof. By Theorem 1, we have $\mathbb{E}[\|x^{(T)} - x^\star\|_2^2] \leq \theta^T C$ and $\mathbb{E}[\|y^{(T)} - y^\star\|_2^2] \leq \theta^T C$. To obtain (11), it suffices to ensure that $\theta^T C \leq \epsilon$, which is equivalent to

$$T \;\geq\; \frac{\log(C/\epsilon)}{-\log(\theta)} \;=\; \frac{\log(C/\epsilon)}{-\log\Big(1 - \Big((n/m) + R\sqrt{(n/m)/(\lambda\gamma)}\Big)^{-1}\Big)}.$$

Applying the inequality $-\log(1 - x) \geq x$ to the denominator above completes the proof. □

Recall the definition of the condition number $\kappa = R^2/(\lambda\gamma)$ in Section 1.1. Corollary 1 establishes that the iteration complexity of the Mini-Batch SPDC method for achieving (11) is

$$\mathcal{O}\left(\left((n/m) + \sqrt{\kappa(n/m)}\right)\log(1/\epsilon)\right).$$

So a larger batch size m leads to less number of iterations. In the extreme case of $n = m$, we obtain a full batch algorithm, which has iteration or batch complexity $\mathcal{O}((1+\sqrt{\kappa})\log(1/\epsilon))$. This complexity is also shared by the AFG methods [24, 26] (see Section 1.1), as well as the batch primal-dual algorithm of Chambolle and Pock [8] (see discussions on related work in Section 4).

Since an equivalent pass over the dataset corresponds to n/m iterations, the batch complexity (the number of equivalent passes over the data) of Mini-Batch SPDC is

$$\mathcal{O}\left(\left(1 + \sqrt{\kappa(m/n)}\right)\log(1/\epsilon)\right).$$

The above expression implies that a smaller batch size m leads to less number of passes through the data. In this sense, the basic SPDC method with $m = 1$ is the most efficient one. However, if we prefer the least amount of wall-clock time, then the best choice is to choose a mini-batch size m that matches the number of parallel processors available.

2.2 Convergence of primal objective

In the previous subsection, we established iteration complexity of the Mini-Batch SPDC method in terms of approximating the saddle point of the minimax problem (3), more specifically, to meet the requirement in (11). Next we show that it has the same order of complexity in reducing the primal objective gap $P(x^{(T)}) - P(x^\star)$. But we need an extra assumption.

Assumption B. *There exist constants $G \geq 0$ and $H \geq 0$ such that for any $x \in \mathbb{R}^d$,*

$$g(x) - g(x^\star) \leq G\|x - x^\star\|_2 + \frac{H}{2}\|x - x^\star\|_2^2.$$

We note that Assumption B is weaker than either G-Lipschitz continuity or H-smoothness. It is satisfied by the ℓ_1 norm, the squared ℓ_2-norm, and mixed $\ell_1 + \ell_2$ regularizations.

Corollary 2. *Suppose both Assumptions A and B hold, and the parameters τ, σ and θ are set as in (10). To guarantee $\mathbb{E}[P(x^{(T)}) - P(x^\star)] \leq \epsilon \leq 1$, it suffices to run Algorithm 2 for T iterations, with*

$$T \geq \left(\frac{n}{m} + R\sqrt{\frac{n}{m\lambda\gamma}}\right)\log\left(\frac{C(4G^2 + H + 1/\gamma)}{\epsilon^2}\right),$$

where

$$C = \|x^{(0)} - x^\star\|_2^2 + \left(\frac{1/(2\sigma) + \gamma}{1/(2\tau) + \lambda}\right)\frac{\|y^{(0)} - y^\star\|_2^2}{m}.$$

Proof. Using the $(1/\gamma)$-smoothness of $P - g$ and Assumption B, we have

$$\begin{aligned} P(x^{(T)}) - P(x^\star) &\leq \langle (P-g)'(x^\star), x^{(T)} - x^\star\rangle + \frac{1}{2\gamma}\|x^{(T)} - x^\star\|_2^2 + g(x^{(T)}) - g(x^\star) \\ &\leq (\|(P-g)'(x^\star)\|_2 + G)\|x^{(T)} - x^\star\|_2 + \frac{H + 1/\gamma}{2}\|x^{(T)} - x^\star\|_2^2. \end{aligned}$$

Since $x^\star$ minimizes P, we have $-(P-g)'(x^\star) \in \partial g(x^\star)$. Hence, Assumption B implies that $\|(P-g)'(x^\star)\|_2 \le G$. Substituting this relation into the above inequality, and using Hölder's inequality, we have

$$\mathbb{E}[P(x^{(T)}) - P(x^\star)] \le 2G\big(\mathbb{E}[\|x^{(T)} - x^\star\|_2^2]\big)^{1/2} + \frac{H + 1/\gamma}{2}\mathbb{E}[\|x^{(T)} - x^\star\|_2^2].$$

To make $\mathbb{E}[P(x^{(T)}) - P(x^\star)] \le \epsilon$, it suffices to let the right-hand side of the above inequality bounded by ϵ. Since $\epsilon \le 1$, this is guaranteed by

$$\mathbb{E}[\|x^{(T)} - x^\star\|_2^2] \le \frac{\epsilon^2}{4G^2 + H + 1/\gamma}. \tag{12}$$

By Theorem 1, we have $\mathbb{E}[\|x^{(T)} - x^\star\|_2^2] \le \theta^T C$. To secure inequality (12), it is sufficient to make $\theta^T \le \frac{\epsilon^2}{C(4G^2+H+1/\gamma)}$, which is equivalent to

$$T \ge \frac{\log(C(4G^2 + H + 1/\gamma)/\epsilon^2)}{-\log(\theta)} = \frac{\log(C(4G^2 + H + 1/\gamma)/\epsilon^2)}{-\log\Big(1 - \Big((n/m) + R\sqrt{(n/m)/(\lambda\gamma)}\Big)^{-1}\Big)}.$$

Applying $-\log(1-x) \ge x$ to the denominator above completes the proof. □

3 Extensions of SPDC

In this section, we derive two extensions of the SPDC method. The first one handles problems for which Assumption A does not hold. The second one employs a non-uniform sampling scheme to improve the iteration complexity when the feature vectors a_i are unnormalized.

3.1 Non-smooth or non-strongly convex functions

The complexity bounds established in Section 2 require each ϕ_i^* to be γ-strongly convex, which corresponds to the condition that the first derivative of ϕ_i is $(1/\gamma)$-Lipschitz continuous. In addition, the function g needs to be λ-strongly convex. For general loss functions where either or both of these conditions fail (e.g., the hinge loss and ℓ_1-regularization), we can slightly perturb the saddle-point function $f(x,y)$ so that the SPDC method can still be applied.

For simplicity, here we consider the case where neither ϕ_i is smooth nor g is strongly convex. Formally, we assume that each ϕ_i and g are convex and Lipschitz continuous, and $f(x,y)$ has a saddle point $(x^\star, y^\star)$. We choose a scalar $\delta > 0$ and consider the modified saddle-point function:

$$f_\delta(x,y) \overset{\text{def}}{=} \frac{1}{n}\sum_{i=1}^n \left(y_i\langle a_i, x\rangle - \left(\phi_i^*(y_i) + \frac{\delta y_i^2}{2}\right)\right) + g(x) + \frac{\delta}{2}\|x\|_2^2. \tag{13}$$

Denote by $(x_\delta^\star, y_\delta^\star)$ the saddle-point of f_δ. We employ the Mini-Batch SPDC method (Algorithm 2) to approximate $(x_\delta^\star, y_\delta^\star)$, treating $\phi_i^* + \frac{\delta}{2}(\cdot)^2$ as ϕ_i^* and $g + \frac{\delta}{2}\|\cdot\|_2^2$ as g, which now are all δ-strongly convex. We note that adding strongly convex perturbation on ϕ_i^* is equivalent to smoothing ϕ_i, which becomes $(1/\delta)$-smooth. Letting $\gamma = \lambda = \delta$, the parameters τ, σ and θ in (10) become

$$\tau = \frac{1}{2R}\sqrt{\frac{m}{n}}, \qquad \sigma = \frac{1}{2R}\sqrt{\frac{n}{m}}, \qquad \text{and} \quad \theta = 1 - \left(\frac{n}{m} + \frac{R}{\delta}\sqrt{\frac{n}{m}}\right)^{-1}.$$

Although $(x_\delta^\star, y_\delta^\star)$ is not exactly the saddle point of f, the following corollary shows that applying the SPDC method to the perturbed function f_δ effectively minimizes the original loss function P.

Corollary 3. *Assume that each ϕ_i is convex and G_ϕ-Lipschitz continuous, and g is convex and G_g-Lipschitz continuous. Define two constants:*

$$C_1 = (\|x^\star\|_2^2 + G_\phi^2), \qquad C_2 = (G_\phi R + G_g)^2 \left(\|x^{(0)} - x_\delta^\star\|_2^2 + \left(\frac{1/(2\sigma)+\delta}{1/(2\tau)+\delta}\right) \frac{\|y^{(0)} - y_\delta^\star\|_2^2}{m} \right).$$

If we choose $\delta \le \epsilon / C_1$, and run the Mini-Batch SPDC algorithm for T iterations where

$$T \ge \left(\frac{n}{m} + \frac{R}{\delta}\sqrt{\frac{n}{m}} \right) \log\left(\frac{4C_2}{\epsilon^2} \right),$$

then $\mathbb{E}[P(x^{(T)}) - P(x^\star)] \le \epsilon$.

Proof. Let $\widetilde{y} = \arg\max_y f(x_\delta^\star, y)$ be a shorthand notation. We have

$$\begin{aligned} P(x_\delta^\star) &\overset{(i)}{=} f(x_\delta^\star, \widetilde{y}) \overset{(ii)}{\le} f_\delta(x_\delta^\star, \widetilde{y}) + \frac{\delta\|\widetilde{y}\|_2^2}{2n} \overset{(iii)}{\le} f_\delta(x_\delta^\star, y_\delta^\star) + \frac{\delta\|\widetilde{y}\|_2^2}{2n} \overset{(iv)}{\le} f_\delta(x^\star, y_\delta^\star) + \frac{\delta\|\widetilde{y}\|_2^2}{2n} \\ &\overset{(v)}{\le} f(x^\star, y_\delta^\star) + \frac{\delta\|x^\star\|_2^2}{2} + \frac{\delta\|\widetilde{y}\|_2^2}{2n} \overset{(vi)}{\le} f(x^\star, y^\star) + \frac{\delta\|x^\star\|_2^2}{2} + \frac{\delta\|\widetilde{y}\|_2^2}{2n} \\ &\overset{(vii)}{=} P(x^\star) + \frac{\delta\|x^\star\|_2^2}{2} + \frac{\delta\|\widetilde{y}\|_2^2}{2n}. \end{aligned}$$

Here, equations (i) and (vii) use the definition of the function f, inequalities (ii) and (v) use the definition of the function f_δ, inequalities (iii) and (iv) use the fact that $(x_\delta^\star, y_\delta^\star)$ is the saddle point of f_δ, and inequality (vi) is due to the fact that $(x^\star, y^\star)$ is the saddle point of f.

Since ϕ_i is G_ϕ-Lipschitz continuous, the domain of ϕ_i^* is in the interval $[-G_\phi, G_\phi]$, which implies $\|\widetilde{y}\|_2^2 \le nG_\phi^2$ (see, e.g., [34, Lemma 1]). Thus, we have

$$P(x_\delta^\star) - P(x^\star) \le \frac{\delta}{2}(\|x^\star\|_2^2 + G_\phi^2) = \frac{\delta}{2} C_1. \tag{14}$$

On the other hand, since P is $(G_\phi R + G_g)$-Lipschitz continuous, Theorem 1 implies

$$\mathbb{E}[P(x^{(T)}) - P(x_\delta^\star)] \;\le\; (G_\phi R + G_g)\mathbb{E}[\|x^{(T)} - x_\delta^\star\|_2] \;\le\; \sqrt{C_2}\left(1 - \left(\frac{n}{m} + \frac{R}{\delta}\sqrt{\frac{n}{m}}\right)^{-1}\right)^{T/2}. \tag{15}$$

Combining inequality (14) and inequality (15), to guarantee $\mathbb{E}[P(x^{(T)}) - P(x^\star)] \le \epsilon$, it suffices to have $C_1\delta \le \epsilon$ and

$$\sqrt{C_2}\left(1 - \left(\frac{n}{m} + \frac{R}{\delta}\sqrt{\frac{n}{m}}\right)^{-1}\right)^{T/2} \le \frac{\epsilon}{2}. \tag{16}$$

The corollary is established by finding the smallest T that satisfies inequality (16). □

There are two other cases that can be considered: when ϕ_i is not smooth but g is strongly convex, and when ϕ_i is smooth but g is not strongly convex. They can be handled with the same technique described above, and we omit the details here. (Alternatively, it is possible to use the techniques described in [8, Section 5.1] to obtain accelerated sublinear convergence rates without using strongly convex perturbations.) In Table 1, we list the complexities of the Mini-Batch SPDC method for finding an ϵ-optimal solution of problem (1) under various assumptions. Similar results are also obtained in [34].

ϕ_i	g	iteration complexity $\widetilde{\mathcal{O}}(\cdot)$
$(1/\gamma)$-smooth	λ-strongly convex	$n/m + \sqrt{(n/m)/(\lambda\gamma)}$
$(1/\gamma)$-smooth	non-strongly convex	$n/m + \sqrt{(n/m)/(\epsilon\gamma)}$
non-smooth	λ-strongly convex	$n/m + \sqrt{(n/m)/(\epsilon\lambda)}$
non-smooth	non-strongly convex	$n/m + \sqrt{n/m}/\epsilon$

Table 1: Iteration complexities of the SPDC method under different assumptions on the functions ϕ_i and g. For the last three cases, we solve the perturbed saddle-point problem with $\delta = \epsilon/C_1$.

3.2 SPDC with non-uniform sampling

One potential drawback of the SPDC algorithm is that, its convergence rate depends on a problem-specific constant R, which is the largest ℓ_2-norm of the feature vectors a_i. As a consequence, the algorithm may perform badly on unnormalized data, especially if the ℓ_2-norms of some feature vectors are substantially larger than others. In this section, we propose an extension of the SPDC method to mitigate this problem, which is given in Algorithm 3.

The basic idea is to use non-uniform sampling in picking the dual coordinate to update at each iteration. In Algorithm 3, we pick coordinate k with the probability

$$p_k = \frac{1}{2n} + \frac{\|a_k\|_2}{2\sum_{i=1}^n \|a_i\|_2}, \qquad k = 1, \ldots, n. \tag{17}$$

Therefore, instances with large feature norms are sampled more frequently. Simultaneously, we adopt an adaptive regularization in step (18), imposing stronger regularization on such instances. In addition, we adjust the weight of a_k in (19) for updating the primal variable. As a consequence, the convergence rate of Algorithm 3 depends on the average norm of feature vectors. This is summarized by the following theorem.

Theorem 2. *Suppose Assumption A holds. Let $\bar{R} = \frac{1}{n}\sum_{i=1}^n \|a_i\|_2$. If the parameters τ, σ, θ in Algorithm 3 are chosen such that*

$$\tau = \frac{1}{4\bar{R}}\sqrt{\frac{\gamma}{n\lambda}}, \qquad \sigma = \frac{1}{4\bar{R}}\sqrt{\frac{n\lambda}{\gamma}}, \qquad \theta = 1 - \frac{1}{2n + 2\bar{R}\sqrt{n/(\lambda\gamma)}},$$

then for each $t \geq 1$, we have

$$\begin{aligned}&\Big(\frac{1}{2\tau} + \lambda\Big)\mathbb{E}\big[\|x^{(t)} - x^\star\|_2^2\big] + \Big(\frac{7}{16\sigma} + \frac{2\gamma}{n}\Big)\mathbb{E}\big[\|y^{(t)} - y^\star\|_2^2\big] \\ \leq\, &\theta^t\left(\Big(\frac{1}{2\tau} + \lambda\Big)\|x^{(0)} - x^\star\|_2^2 + \Big(\frac{1}{2\sigma} + 2\gamma\Big)\|y^{(0)} - y^\star\|_2^2\right).\end{aligned}$$

Comparing the constant θ in Theorem 2 to that of Theorem 1, we can find two differences. First, there is an additional factor of 2 multiplied to the denominator $2n + 2\bar{R}\sqrt{n/(\lambda\gamma)}$, making the value of θ larger. Second, the constant $\bar{R}$ here is determined by the average norm of features, instead of the largest one, which makes the value of θ smaller. The second difference makes the algorithm more robust to unnormalized feature vectors. For example, if the a_i's are sampled i.i.d.

Algorithm 3: SPDC method with weighted sampling

Input: parameters $\tau, \sigma, \theta \in \mathbb{R}_+$, number of iterations T, and initial points $x^{(0)}$ and $y^{(0)}$.

Initialize: $\overline{x}^{(0)} = x^{(0)}$, $u^{(0)} = (1/n)\sum_{i=1}^n y_i^{(0)} a_i$.

for $t = 0, 1, 2, \ldots, T-1$ **do**

Randomly pick $k \in \{1, 2, \ldots, n\}$, with probability $p_k = \frac{1}{2n} + \frac{\|a_k\|_2}{2\sum_{i=1}^n \|a_i\|_2}$.

Execute the following updates:

$$y_i^{(t+1)} = \begin{cases} \arg\max_{\beta \in \mathbb{R}} \left\{ \beta \langle a_i, \overline{x}^{(t)} \rangle - \phi_i^*(\beta) - \frac{p_i n}{2\sigma}(\beta - y_i^{(t)})^2 \right\} & i = k, \\ y_i^{(t)} & i \neq k, \end{cases} \tag{18}$$

$$x^{(t+1)} = \arg\min_{x \in \mathbb{R}^d} \left\{ g(x) + \left\langle u^{(t)} + \frac{1}{p_k n}(y_k^{(t+1)} - y_k^{(t)}) a_k,\ x \right\rangle + \frac{\|x - x^{(t)}\|_2^2}{2\tau} \right\}, \tag{19}$$

$$u^{(t+1)} = u^{(t)} + \frac{1}{n}(y_k^{(t+1)} - y_k^{(t)}) a_k,$$

$$\overline{x}^{(t+1)} = x^{(t+1)} + \theta(x^{(t+1)} - x^{(t)}).$$

end

Output: $x^{(T)}$ and $y^{(T)}$

from a multivariate normal distribution, then $\max_i\{\|a_i\|_2\}$ almost surely goes to infinity as $n \to \infty$, but the average norm $\frac{1}{n}\sum_{i=1}^n \|a_i\|_2$ converges to $\mathbb{E}[\|a_i\|_2]$.

For simplicity of presentation, we described in Algorithm 3 a weighted sampling SPDC method with single dual coordinate update, i.e., the case of $m = 1$. It is not hard to see that the non-uniform sampling scheme can also be extended to Mini-Batch SPDC with $m > 1$. Moreover, the non-uniform sampling scheme can also be applied to solve problems with non-smooth ϕ_i or non-strongly convex g, leading to similar conclusions as in Corollary 3. Here, we omit the technical details.

4 Related Work

Chambolle and Pock [8] considered a class of convex optimization problems with the following saddle-point structure:

$$\min_{x \in \mathbb{R}^d} \max_{y \in \mathbb{R}^n} \left\{ \langle Kx, y \rangle + G(x) - F^*(y) \right\}, \tag{20}$$

where $K \in \mathbb{R}^{m \times d}$, G and F^* are proper closed convex functions, with F^* itself being the conjugate of a convex function F. They developed the following first-order primal-dual algorithm:

$$y^{(t+1)} = \arg\max_{y \in \mathbb{R}^n} \left\{ \langle K\overline{x}^{(t)}, y \rangle - F^*(y) - \frac{1}{2\sigma}\|y - y^{(t)}\|_2^2 \right\}, \tag{21}$$

$$x^{(t+1)} = \arg\min_{x \in \mathbb{R}^d} \left\{ \langle K^T y^{(t+1)}, x \rangle + G(x) + \frac{1}{2\tau}\|x - x^{(t)}\|_2^2 \right\}, \tag{22}$$

$$\overline{x}^{(t+1)} = x^{(t+1)} + \theta(x^{(t+1)} - x^{(t)}). \tag{23}$$

When both F^* and G are strongly convex and the parameters τ, σ and θ are chosen appropriately, this algorithm obtains accelerated linear convergence rate [8, Theorem 3].

algorithm	τ	σ	θ	batch complexity
Chambolle-Pock [8]	$\frac{\sqrt{n}}{\|A\|_2}\sqrt{\frac{\gamma}{\lambda}}$	$\frac{\sqrt{n}}{\|A\|_2}\sqrt{\frac{\lambda}{\gamma}}$	$1-\frac{1}{1+\|A\|_2/(2\sqrt{n\lambda\gamma})}$	$\left(1+\frac{\|A\|_2}{2\sqrt{n\lambda\gamma}}\right)\log(1/\epsilon)$
SPDC with $m=n$	$\frac{1}{2R}\sqrt{\frac{\gamma}{\lambda}}$	$\frac{1}{2R}\sqrt{\frac{\lambda}{\gamma}}$	$1-\frac{1}{1+R/\sqrt{\lambda\gamma}}$	$\left(1+\frac{R}{\sqrt{\lambda\gamma}}\right)\log(1/\epsilon)$
SPDC with $m=1$	$\frac{1}{2R}\sqrt{\frac{\gamma}{n\lambda}}$	$\frac{1}{2R}\sqrt{\frac{n\lambda}{\gamma}}$	$1-\frac{1}{n+R\sqrt{n/\lambda\gamma}}$	$\left(1+\frac{R}{\sqrt{n\lambda\gamma}}\right)\log(1/\epsilon)$

Table 2: Comparing SPDC with Chambolle and Pock [8, Algorithm 3, Theorem 3].

We can map the saddle-point problem (3) into the form of (20) by letting $A=[a_1,\dots,a_n]^T$ and

$$K=\frac{1}{n}A, \qquad G(x)=g(x), \qquad F^*(y)=\frac{1}{n}\sum_{i=1}^n \phi_i^*(y_i). \tag{24}$$

The SPDC method developed in this paper can be viewed as an extension of the batch method (21)-(23), where the dual update step (21) is replaced by a single coordinate update (4) or a mini-batch update (8). However, in order to obtain accelerated convergence rate, more subtle changes are necessary in the primal update step. More specifically, we introduced the auxiliary variable $u^{(t)}=\frac{1}{n}\sum_{i=1}^n y_i^{(t)}a_i=K^Ty^{(t)}$, and replaced the primal update step (22) by (5) and (9). The primal extrapolation step (23) stays the same.

To compare the batch complexity of SPDC with that of (21)-(23), we use the following facts implied by Assumption A and the relations in (24):

$$\|K\|_2=\frac{1}{n}\|A\|_2, \quad G(x) \text{ is } \lambda\text{-strongly convex}, \quad \text{and } F^*(y) \text{ is } (\gamma/n)\text{-strongly convex}.$$

Based on these conditions, we list in Table 2 the equivalent parameters used in [8, Algorithm 3] and the batch complexity obtained in [8, Theorem 3], and compare them with SPDC.

The batch complexity of the Chambolle-Pock algorithm is $\widetilde{\mathcal{O}}(1+\|A\|_2/(2\sqrt{n\lambda\gamma}))$, where the $\widetilde{\mathcal{O}}(\cdot)$ notation hides the $\log(1/\epsilon)$ factor. We can bound the spectral norm $\|A\|_2$ by the Frobenius norm $\|A\|_F$ and obtain

$$\|A\|_2\le\|A\|_F\le\sqrt{n}\max_i\{\|a_i\|_2\}=\sqrt{n}R.$$

(Note that the second inequality above would be an equality if the columns of A are normalized.) So in the worst case, the batch complexity of the Chambolle-Pock algorithm becomes

$$\widetilde{\mathcal{O}}\left(1+R/\sqrt{\lambda\gamma}\right)=\widetilde{\mathcal{O}}\left(1+\sqrt{\kappa}\right), \qquad \text{where } \kappa=R^2/(\lambda\gamma),$$

which matches the worst-case complexity of the AFG methods [24, 26] (see Section 1.1 and also the discussions in [18, Section 5]). This is also of the same order as the complexity of SPDC with $m=n$ (see Section 2.1). When the condition number $\kappa\gg 1$, they can be $\sqrt{n}$ worse than the batch complexity of SPDC with $m=1$, which is $\widetilde{\mathcal{O}}(1+\sqrt{\kappa/n})$.

If either $G(x)$ or $F^*(y)$ in (20) is not strongly convex, Chambolle and Pock proposed variants of the primal-dual batch algorithm to achieve accelerated sublinear convergence rates [8, Section 5.1]. It is also possible to extend them to coordinate update methods for solving problem (1) when either ϕ_i^* or g is not strongly convex. Their complexities would be similar to those in Table 1.

4.1 Dual coordinate ascent methods

We can also solve the primal problem (1) via its dual:

$$\underset{y\in\mathbb{R}^n}{\text{maximize}} \quad \left\{ D(y) \stackrel{\text{def}}{=} \frac{1}{n}\sum_{i=1}^{n} -\phi_i^*(y_i) - g^*\left(-\frac{1}{n}\sum_{i=1}^{n} y_i a_i\right)\right\}, \tag{25}$$

where $g^*(u) = \sup_{x\in\mathbb{R}^d}\{x^T u - g(x)\}$ is the conjugate function of g. Here again, coordinate ascent methods (e.g., [29, 9, 15, 36]) can be more efficient than full gradient methods. In the stochastic dual coordinate ascent (SDCA) method [36], a dual coordinate y_i is picked at random during each iteration and updated to increase the dual objective value. Shalev-Shwartz and Zhang [36] showed that the iteration complexity of SDCA is $O\left((n+\kappa)\log(1/\epsilon)\right)$, which corresponds to the batch complexity $\widetilde{\mathcal{O}}(1+\kappa/n)$. Therefore, the SPDC method, which has batch complexity $\widetilde{\mathcal{O}}(1+\sqrt{\kappa/n})$, can be much better when $\kappa > n$, i.e., for ill-conditioned problems.

For more general convex optimization problems, there is a vast literature on coordinate descent methods. In particular, Nesterov's work on randomized coordinate descent [25] sparked a lot of recent activities on this topic. Richtárik and Takáč [31] extended the algorithm and analysis to composite convex optimization. When applied to the dual problem (25), it becomes one variant of SDCA studied in [36]. Mini-batch and distributed versions of SDCA have been proposed and analyzed in [39] and [45] respectively. Non-uniform sampling schemes similar to the one used in Algorithm 3 have been studied for both stochastic gradient and SDCA methods (e.g., [22, 44, 48]).

Shalev-Shwartz and Zhang [35] proposed an accelerated mini-batch SDCA method which incorporates additional primal updates than SDCA, and bears some similarity to our Mini-Batch SPDC method. They showed that its complexity interpolates between that of SDCA and AFG by varying the mini-batch size m. In particular, for $m = n$, it matches that of the AFG methods (as SPDC does). But for $m = 1$, the complexity of their method is the same as SDCA, which is worse than SPDC for ill-conditioned problems.

In addition, Shalev-Shwartz and Zhang [34] developed an accelerated proximal SDCA method which achieves the same batch complexity $\widetilde{\mathcal{O}}\big(1+\sqrt{\kappa/n}\big)$ as SPDC. Their method is an inner-outer iteration procedure, where the outer loop is a full-dimensional accelerated gradient method in the primal space $x \in \mathbb{R}^d$. At each iteration of the outer loop, the SDCA method [36] is called to solve the dual problem (25) with customized regularization parameter and precision. In contrast, SPDC is a straightforward single-loop coordinate optimization methods.

More recently, Lin et al. [18] developed an accelerated proximal coordinate gradient (APCG) method for solving a more general class of composite convex optimization problems. When applied to the dual problem (25), APCG enjoys the same batch complexity $\widetilde{\mathcal{O}}\big(1+\sqrt{\kappa/n}\big)$ as of SPDC. However, it needs an extra primal proximal-gradient step to have theoretical guarantees on the convergence of primal-dual gap [18, Section 5.1]. The computational cost of this additional step is equivalent to one pass of the dataset, thus it does not affect the overall complexity.

4.2 Other related work

Another way to approach problem (1) is to reformulate it as a constrained optimization problem

$$\begin{aligned} \text{minimize} \quad & \frac{1}{n}\sum_{i=1}^{n}\phi_i(z_i) + g(x) \\ \text{subject to} \quad & a_i^T x = z_i, \quad , i = 1,\ldots,n, \end{aligned} \tag{26}$$

and solve it by ADMM type of operator-splitting methods (e.g., [19]). In fact, as shown in [8], the batch primal-dual algorithm (21)-(23) is equivalent to a pre-conditioned ADMM (or inexact Uzawa method; see, e.g., [47]). Several authors [42, 28, 37, 49] have considered a more general formulation than (26), where each ϕ_i is a function of the whole vector $z \in \mathbb{R}^n$. They proposed online or stochastic versions of ADMM which operate on only one ϕ_i in each iteration, and obtained sublinear convergence rates. However, their cost per iteration is $\mathcal{O}(nd)$ instead of $\mathcal{O}(d)$.

Suzuki [38] considered a problem similar to (1), but with more complex regularization function g, meaning that g does not have a simple proximal mapping. Thus primal updates such as step (5) or (9) in SPDC and similar steps in SDCA cannot be computed efficiently. He proposed an algorithm that combines SDCA [36] and ADMM (e.g., [7]), and showed that it has linear rate of convergence under similar conditions as Assumption A. It would be interesting to see if the SPDC method can be extended to their setting to obtain accelerated linear convergence rate.

5 Efficient Implementation with Sparse Data

During each iteration of the SPDC methods, the updates of primal variables (i.e., computing $x^{(t+1)}$) require full d-dimensional vector operations; see the step (5) of Algorithm 1, the step (9) of Algorithm 2 and the step (19) of Algorithm 3. So the computational cost per iteration is $\mathcal{O}(d)$, and this can be too expensive if the dimension d is very high. In this section, we show how to exploit problem structure to avoid high-dimensional vector operations when the feature vectors a_i are sparse. We illustrate the efficient implementation for two popular cases: when g is an squared-ℓ_2 penalty and when g is an $\ell_1 + \ell_2$ penalty. For both cases, we show that the computation cost per iteration only depends on the number of non-zero components of the feature vector.

5.1 Squared ℓ_2-norm penalty

Suppose that $g(x) = \frac{\lambda}{2}\|x\|_2^2$. For this case, the updates for each coordinate of x are independent of each other. More specifically, $x^{(t+1)}$ can be computed coordinate-wise in closed form:

$$x_j^{(t+1)} = \frac{1}{1+\lambda\tau}(x_j^{(t)} - \tau u_j^{(t)} - \tau \Delta u_j), \quad j = 1, \dots, n, \tag{27}$$

where Δu denotes $(y_k^{(t+1)} - y_k^{(t)})a_k$ in Algorithm 1, or $\frac{1}{m}\sum_{k\in K}(y_k^{(t+1)} - y_k^{(t)})a_k$ in Algorithm 2, or $(y_k^{(t+1)} - y_k^{(t)})a_k/(p_k n)$ in Algorithm 3, and Δu_j represents the j-th coordinate of Δu.

Although the dimension d can be very large, we assume that each feature vector a_k is sparse. We denote by $J^{(t)}$ the set of non-zero coordinates at iteration t, that is, if for some index $k \in K$ picked at iteration t we have $a_{kj} \neq 0$, then $j \in J^{(t)}$. If $j \notin J^{(t)}$, then the SPDC algorithm (and its variants) updates $y^{(t+1)}$ without using the value of $x_j^{(t)}$ or $\overline{x}_j^{(t)}$. This can be seen from the updates in (4), (8) and (18), where the value of the inner product $\langle a_k, \overline{x}^{(t)}\rangle$ does not depend on the value of $\overline{x}_j^{(t)}$. As a consequence, we can delay the updates on x_j and $\overline{x}_j$ whenever $j \notin J^{(t)}$ without affecting the updates on $y^{(t)}$, and process all the missing updates at the next time when $j \in J^{(t)}$.

Such a delayed update can be carried out very efficiently. We assume that t_0 is the last time when $j \in J^{(t)}$, and t_1 is the current iteration where we want to update x_j and $\overline{x}_j$. Since $j \notin J^{(t)}$ implies $\Delta u_j = 0$, we have

$$x_j^{t+1} = \frac{1}{1+\lambda\tau}(x_j^{(t)} - \tau u_j^{(t)}), \qquad t = t_0 + 1, t_0 + 2, \dots, t_1 - 1. \tag{28}$$

Notice that $u_j^{(t)}$ is updated only at iterations where $j \in J^{(t)}$. The value of $u_j^{(t)}$ doesn't change during iterations $[t_0+1, t_1]$, so we have $u_j^{(t)} \equiv u_j^{(t_0+1)}$ for $t \in [t_0+1, t_1]$. Substituting this equation into the recursive formula (28), we obtain

$$x_j^{(t_1)} = \frac{1}{(1+\lambda\tau)^{t_1-t_0-1}} \left(x_j^{(t_0+1)} + \frac{u_j^{(t_0+1)}}{\lambda} \right) - \frac{u_j^{(t_0+1)}}{\lambda}. \tag{29}$$

The update (29) takes $\mathcal{O}(1)$ time to compute. Using the same formula, we can compute $x_j^{(t_1-1)}$ and subsequently compute $\overline{x}_j^{(t_1)} = x_j^{(t_1)} + \theta(x_j^{(t_1)} - x_j^{(t_1-1)})$. Thus, the computational complexity of a single iteration in SPDC is proportional to $|J^{(t)}|$, independent of the dimension d.

5.2 $(\ell_1 + \ell_2)$-norm penalty

Suppose that $g(x) = \lambda_1\|x\|_1 + \frac{\lambda_2}{2}\|x\|_2^2$. Since both the ℓ_1-norm and the squared ℓ_2-norm are decomposable, the updates for each coordinate of $x^{(t+1)}$ are independent. More specifically,

$$x_j^{(t+1)} = \arg\min_{\alpha\in\mathbb{R}} \left\{ \lambda_1|\alpha| + \frac{\lambda_2\alpha^2}{2} + (u_j^{(t)} + \Delta u_j)\alpha + \frac{(\alpha - x_j^{(t)})^2}{2\tau} \right\}, \tag{30}$$

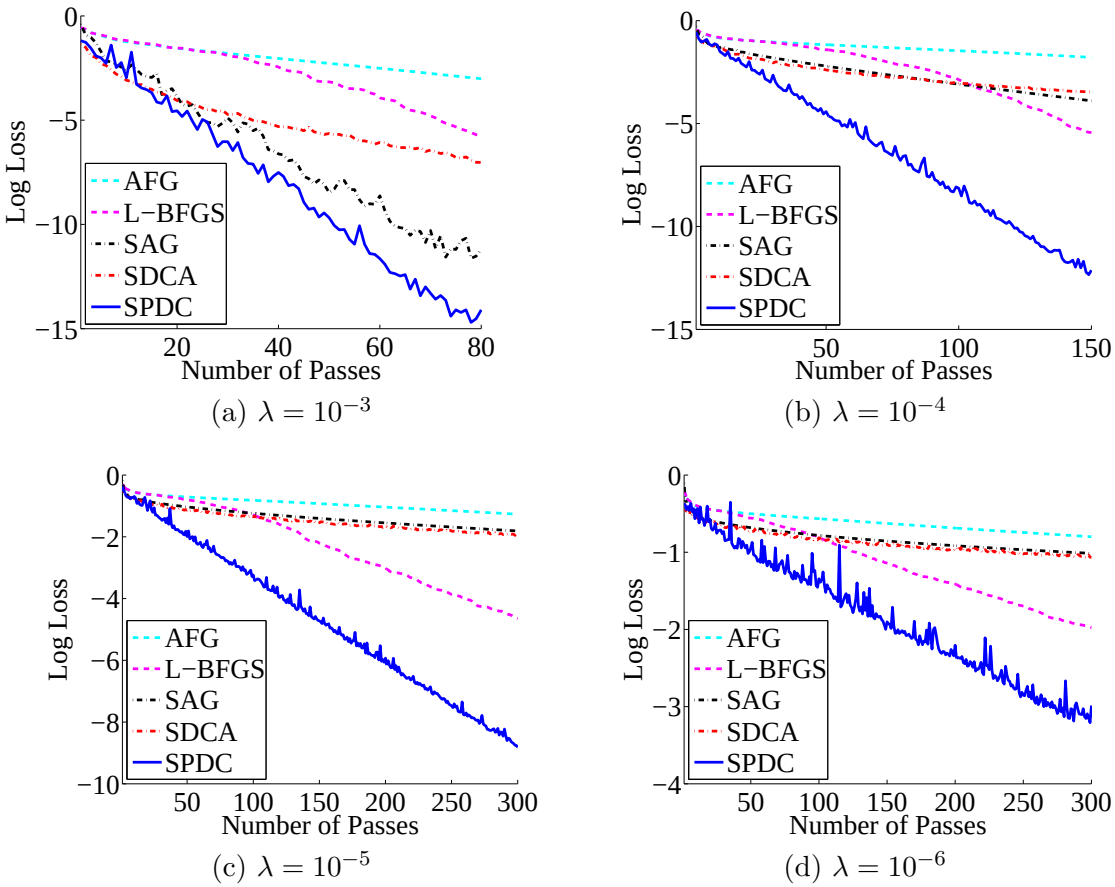
where Δu_j follows the definition in Section 5.1. If $j \notin J^{(t)}$, then $\Delta u_j = 0$ and equation (30) can be simplified as

$$x_j^{(t+1)} = \begin{cases} \frac{1}{1+\lambda_2\tau}(x_j^{(t)} - \tau u_j^{(t)} - \tau\lambda_1) & \text{if } x_j^{(t)} - \tau u_j^{(t)} > \tau\lambda_1, \\ \frac{1}{1+\lambda_2\tau}(x_j^{(t)} - \tau u_j^{(t)} + \tau\lambda_1) & \text{if } x_j^{(t)} - \tau u_j^{(t)} < -\tau\lambda_1, \\ 0 & \text{otherwise.} \end{cases} \tag{31}$$

Similar to the approach of Section 5.1, we delay the update of x_j until $j \in J^{(t)}$. We assume t_0 to be the last iteration when $j \in J^{(t)}$, and let t_1 be the current iteration when we want to update x_j. During iterations $[t_0+1, t_1]$, the value of $u_j^{(t)}$ doesn't change, so we have $u_j^{(t)} \equiv u_j^{(t_0+1)}$ for $t \in [t_0+1, t_1]$. Using equation (31) and the invariance of $u_j^{(t)}$ for $t \in [t_0+1, t_1]$, we have an $\mathcal{O}(1)$ time algorithm to calculate $x_j^{(t_1)}$, which we detail in Appendix C. The vector $\overline{x}_j^{(t_1)}$ can be updated by the same algorithm since it is a linear combination of $x_j^{(t_1)}$ and $x_j^{(t_1-1)}$. As a consequence, the computational complexity of each iteration in SPDC is proportional to $|J^{(t)}|$, independent of the dimension d.

6 Experiments

In this section, we compare the basic SPDC method (Algorithm 1) with several state-of-the-art optimization algorithms for solving problem (1). They include two batch-update algorithms: the accelerated full gradient (FAG) method [24, Section 2.2], and the limited-memory quasi-Newton method L-BFGS [27, Section 7.2]). For the AFG method, we adopt an adaptive line search scheme (e.g., [26]) to improve its efficiency. For the L-BFGS method, we use the memory size 30 as suggested by [27]. We also compare SPDC with two stochastic algorithms: the stochastic average gradient (SAG) method [32, 33], and the stochastic dual coordinate descent (SDCA) method [36]. We conduct experiments on a synthetic dataset and three real datasets.



(a) $\lambda = 10^{-3}$ (b) $\lambda = 10^{-4}$

(c) $\lambda = 10^{-5}$ (d) $\lambda = 10^{-6}$

Figure 1: Comparing SPDC with other methods on synthetic data, with the regularization coefficient $\lambda \in \{10^{-3}, 10^{-4}, 10^{-5}, 10^{-6}\}$. The horizontal axis is the number of passes through the entire dataset, and the vertical axis is the logarithmic gap $\log(P(x^{(T)}) - P(x^\star))$.

6.1 Ridge regression with synthetic data

We first compare SPDC with other algorithms on a simple quadratic problem using synthetic data. We generate $n = 500$ i.i.d. training examples $\{a_i, b_i\}_{i=1}^n$ according to the model

$$b = \langle a, x^* \rangle + \varepsilon, \quad a \sim \mathcal{N}(0, \Sigma), \quad \varepsilon \sim \mathcal{N}(0, 1),$$

where $a \in \mathbb{R}^d$ and $d = 500$, and x^* is the all-ones vector. To make the problem ill-conditioned, the covariance matrix Σ is set to be diagonal with $\Sigma_{jj} = j^{-2}$, for $j = 1, \ldots, d$. Given the set of examples $\{a_i, b_i\}_{i=1}^n$, we then solved a standard ridge regression problem

$$\underset{x \in \mathbb{R}^d}{\text{minimize}} \quad \left\{ P(x) \stackrel{\text{def}}{=} \frac{1}{n} \sum_{i=1}^n \frac{1}{2}(a_i^T x - b_i)^2 + \frac{\lambda}{2}\|x\|_2^2 \right\}.$$

In the form of problem (1), we have $\phi_i(z) = z^2/2$ and $g(x) = (1/2)\|x\|_2^2$. As a consequence, the derivative of ϕ_i is 1-Lipschitz continuous and g is λ-strongly convex.

Dataset name	number of samples n	number of features d	sparsity
Covtype	581,012	54	22%
RCV1	20,242	47,236	0.16%
News20	19,996	1,355,191	0.04%

Table 3: Characteristics of three real datasets obtained from LIBSVM data [12].

We evaluate the algorithms by the logarithmic optimality gap $\log(P(x^{(t)}) - P(x^\star))$, where $x^{(t)}$ is the output of the algorithms after t passes over the entire dataset, and $x^\star$ is the global minimum. When the regularization coefficient is relatively large, e.g., $\lambda = 10^{-1}$ or 10^{-2}, the problem is well-conditioned and we observe fast convergence of the stochastic algorithms SAG, SDCA and SPDC, which are substantially faster than the two batch methods AFG and L-BFGS.

Figure 1 shows the convergence of the five different algorithms when we varied λ from 10^{-3} to 10^{-6}. As the plot shows, when the condition number is greater than n, the SPDC algorithm also converges substantially faster than the other two stochastic methods SAG and SDCA. It is also notably faster than L-BFGS. These results support our theory that SPDC enjoys a faster convergence rate on ill-conditioned problems. In terms of their batch complexities, SPDC is up to $\sqrt{n}$ times faster than AFG, and $(\lambda n)^{-1/2}$ times faster than SAG and SDCA.

6.2 Binary classification with real data

Finally we show the results of solving the binary classification problem on three real datasets. The datasets are obtained from LIBSVM data [12] and summarized in Table 3. The three datasets are selected to reflect different relations between the sample size n and the feature dimensionality d, which cover $n \gg d$ (Covtype), $n \approx d$ (RCV1) and $n \ll d$ (News20). For all tasks, the data points take the form of (a_i, b_i), where $a_i \in \mathbb{R}^d$ is the feature vector, and $b_i \in \{-1, 1\}$ is the binary class label. Our goal is to minimize the regularized empirical risk:

$$P(x) = \frac{1}{n}\sum_{i=1}^{n} \phi_i(a_i^T x) + \frac{\lambda}{2}\|x\|_2^2 \quad \text{where} \quad \phi_i(z) = \begin{cases} 0 & \text{if } b_i z \geq 1 \\ \frac{1}{2} - b_i z & \text{if } b_i z \leq 0 \\ \frac{1}{2}(1 - b_i z)^2 & \text{otherwise.} \end{cases}$$

Here, ϕ_i is the smoothed hinge loss (see, e.g., [36]). It is easy to verify that the conjugate function of ϕ_i is $\phi_i^*(\beta) = b_i\beta + \frac{1}{2}\beta^2$ for $b_i\beta \in [-1, 0]$ and ∞ otherwise.

The performance of the five algorithms are plotted in Figure 2 and Figure 3. In Figure 2, we compare SPDC with the two batch methods: AFG and L-BFGS. The results show that SPDC is substantially faster than AFG and L-BFGS for relatively large λ, illustrating the advantage of stochastic methods over batch methods on well-conditioned problems. As λ decreases to 10^{-8}, the batch methods (especially L-BFGS) become comparable to SPDC.

In Figure 3, we compare SPDC with the two stochastic methods: SAG and SDCA. Here, the observations are just the opposite to that of Figure 2. The three stochastic algorithms have comparable performance on relatively large λ, but SPDC becomes substantially faster when λ gets closer to zero. Summarizing Figure 2 and Figure 3, the performance of SPDC are always comparable or better than the other methods in comparison.

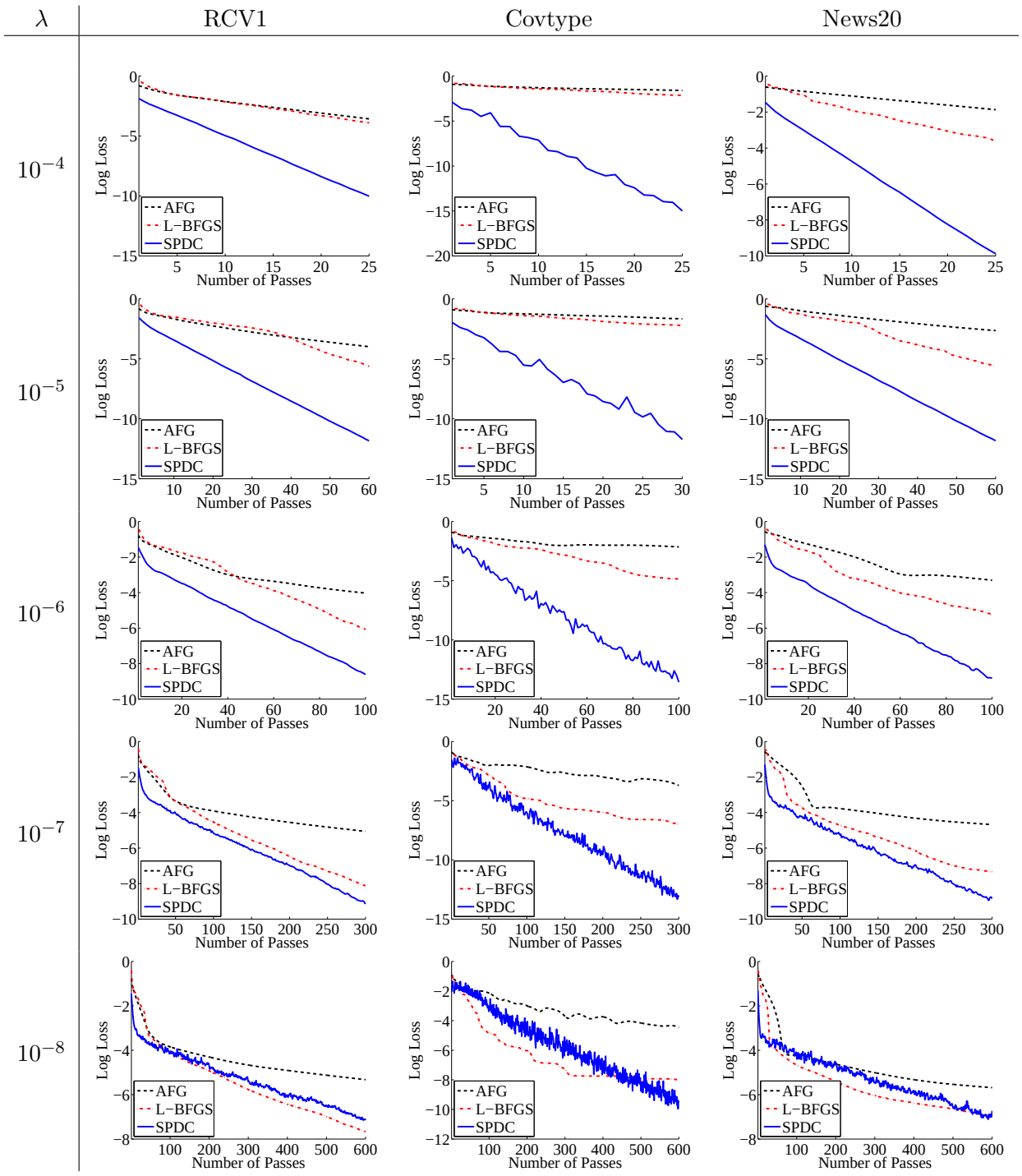


Figure 2: Comparing SPDC with AFG and L-BFGS on three real datasets with smoothed hinge loss. The horizontal axis is the number of passes through the entire dataset, and the vertical axis is the logarithmic optimality gap $\log(P(x^{(t)}) - P(x^\star))$. The SPDC algorithm is faster than the two batch methods when λ is relatively large.

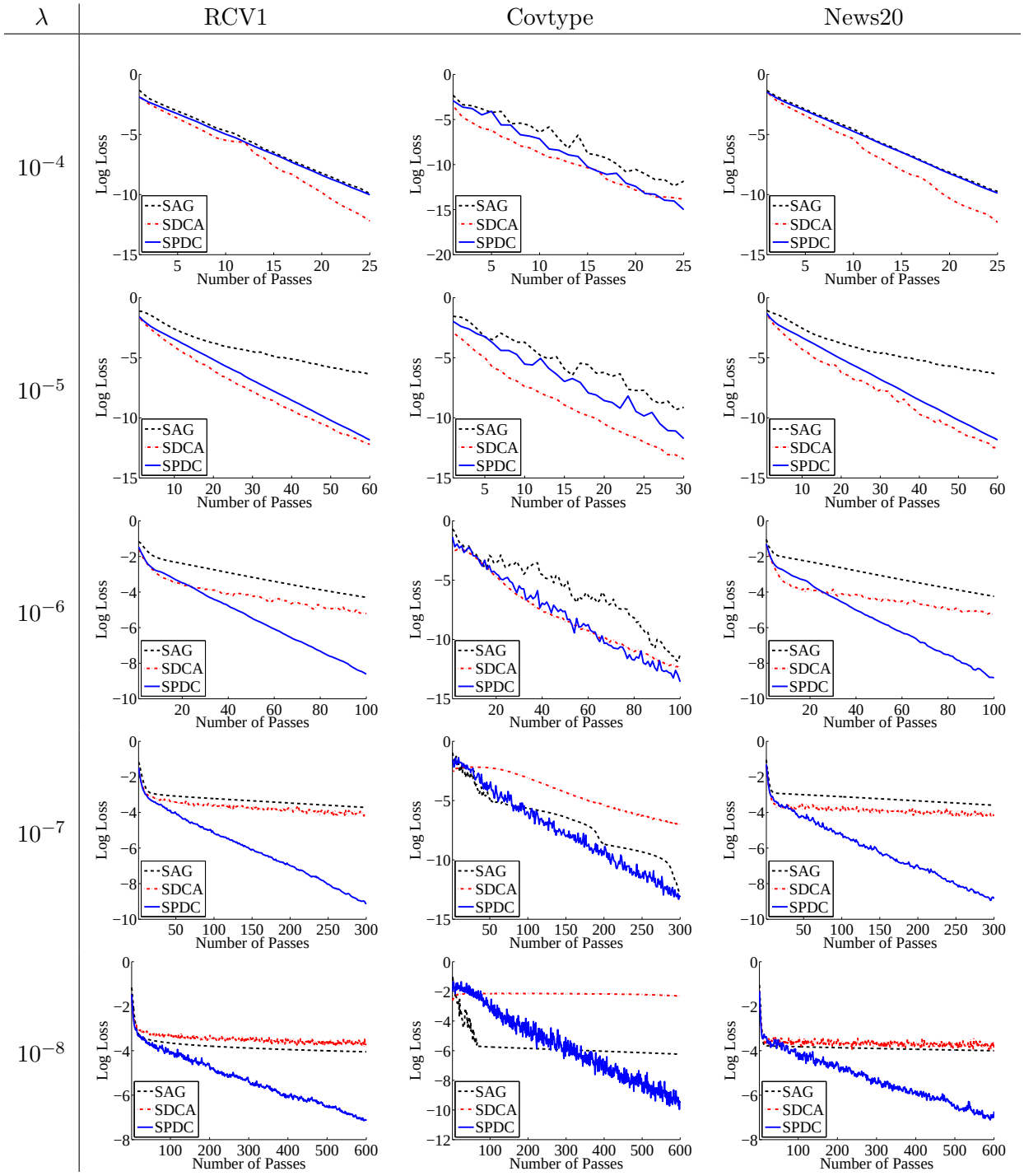


Figure 3: Comparing SPDC with SAG and SDCA on three real datasets with smoothed hinge loss. The horizontal axis is the number of passes through the entire dataset, and the vertical axis is the logarithmic optimality gap $\log(P(x^{(T)}) - P(x^\star))$. The SPDC algorithm is faster than the other two stochastic methods when λ is small.

A Proof of Theorem 1

We focus on characterizing the values of x and y after the t-th update in Algorithm 2. For any $i \in \{1, \ldots, n\}$, let $\widetilde{y}_i$ be the value of $y_i^{(t+1)}$ if $i \in K$, i.e.,

$$\widetilde{y}_i = \arg\max_{y \in \mathbb{R}} \left\{ y\langle a_i, \overline{x}^{(t)} \rangle - \phi_i^*(\beta) - \frac{(y - y_i^{(t)})^2}{2\sigma} \right\}.$$

Since ϕ_i is $(1/\gamma)$-smooth by assumption, its conjugate ϕ_i^* is γ-strongly convex (e.g., [14, Theorem 4.2.2]). Thus the function being maximized above is $(1/\sigma + \gamma)$-strongly concave. Therefore,

$$\begin{aligned} -y_i^\star \langle a_i, \overline{x}^{(t)} \rangle + \phi_i^*(y_i^\star) + \frac{(y_i^\star - y_i^{(t)})^2}{2\sigma} \geq & -\widetilde{y}_i \langle a_i, \overline{x}^{(t)} \rangle + \phi_i^*(\widetilde{y}_i) + \frac{(\widetilde{y}_i - y_i^{(t)})^2}{2\sigma} \\ & + \Big(\frac{1}{\sigma} + \gamma\Big) \frac{(\widetilde{y}_i - y_i^\star)^2}{2}. \end{aligned}$$

On the other hand, since $y_i^\star$ minimizes $\phi_k^*(y) - y\langle a_i, x^\star \rangle$ (by property of the saddle-point), we have

$$\phi_i^*(\widetilde{y}_i) - \widetilde{y}_i \langle a_i, x^\star \rangle \geq \phi_i^*(y_i^\star) - y_i^\star \langle a_i, x^\star \rangle + \frac{\gamma}{2}(\widetilde{y}_i - y_i^\star)^2.$$

Summing up the above two inequalities, we obtain

$$\frac{(y_i^{(t)} - y_i^\star)^2}{2\sigma} \geq \Big(\frac{1}{2\sigma} + \gamma\Big)(\widetilde{y}_i - y_i^\star)^2 + \frac{(\widetilde{y}_i - y_i^{(t)})^2}{2\sigma} + (\widetilde{y}_i - y_i^\star)\langle a_i, x^\star - \overline{x}^{(t)} \rangle. \tag{32}$$

According to Algorithm 2, the set K of indices to be updated are chosen randomly. For every specific index i, the event $i \in K$ happens with probability m/n. If $i \in K$, then $y_i^{(t+1)}$ is updated to the value $\widetilde{y}_i$, which satisfies inequality (32). Otherwise, $y_i^{(t+1)}$ is assigned by its old value $y_i^{(t)}$. Let $\mathcal{F}_t$ be the sigma field generated by all random variables defined before round t, and taking expectation conditioned on $\mathcal{F}_t$, we have

$$\begin{aligned} \mathbb{E}[(y_i^{(t+1)} - y_i^\star)^2 | \mathcal{F}_t] &= \frac{m(\widetilde{y}_i - y_i^\star)^2}{n} + \frac{(n-m)(y_i^{(t)} - y_i^\star)^2}{n}, \\ \mathbb{E}[(y_i^{(t+1)} - y_i^{(t)})^2 | \mathcal{F}_t] &= \frac{m(\widetilde{y}_i - y_i^{(t)})^2}{n}, \\ \mathbb{E}[y_i^{(t+1)} | \mathcal{F}_t] &= \frac{m\widetilde{y}_i}{n} + \frac{(n-m)y_i^{(t)}}{n}. \end{aligned}$$

As a result, we can represent $(\widetilde{y}_i - y_i^\star)^2$, $(\widetilde{y}_i - y_i^{(t)})^2$ and $\widetilde{y}_i$ in terms of the conditional expectations on $(y_i^{(t+1)} - y_i^\star)^2$, $(y_i^{(t+1)} - y_i^{(t)})^2$ and $y_i^{(t+1)}$. Plugging these representations into inequality (32), we have

$$\begin{aligned} \left(\frac{n}{2m\sigma} + \frac{(n-m)\gamma}{m}\right)(y_i^{(t)} - y_i^\star)^2 \geq & \left(\frac{n}{2m\sigma} + \frac{n\gamma}{m}\right)\mathbb{E}[(y_i^{(t+1)} - y_i^\star)^2 | \mathcal{F}_t] + \frac{n\mathbb{E}[(y_i^{(t+1)} - y_i^{(t)})^2 | \mathcal{F}_t]}{2m\sigma} \\ & + \langle a_i, x^\star - \overline{x}^{(t)} \rangle \Big(y_i^{(t)} - y_i^\star + \frac{n}{m}\mathbb{E}[y_i^{(t+1)} - y_i^{(t)} | \mathcal{F}_t] \Big). \end{aligned} \tag{33}$$

Then summing over all indices $i = 1, 2, \ldots, n$ and dividing both sides of the inequality by n, we have

$$\left(\frac{1}{2m\sigma} + \frac{(n-m)\gamma}{mn}\right)\|y^{(t)} - y^\star\|_2^2 \geq \left(\frac{1}{2m\sigma} + \frac{\gamma}{m}\right)\mathbb{E}[\|y^{(t+1)} - y^\star\|_2^2|\mathcal{F}_t] + \frac{\mathbb{E}[\|y^{(t+1)} - y^{(t)}\|_2^2|\mathcal{F}_t]}{2m\sigma} + \mathbb{E}\Big[\Big\langle u^{(t)} - u^\star + \frac{1}{m}\sum_{k\in K}(y_k^{(t+1)} - y_k^{(t)})a_k,\, x^\star - \overline{x}^{(t)}\Big\rangle\Big|\mathcal{F}_t\Big], \tag{34}$$

where $u^\star = \frac{1}{n}\sum_{i=1}^n y_i^\star a_i$ is a shorthand notation, and $u^{(t)} = \frac{1}{n}\sum_{i=1}^n y_i^{(t)} a_i$ is defined in Algorithm 2. We used the fact that $\sum_{i=1}^n (y_i^{(t+1)} - y_i^{(t)})a_i = \sum_{k\in K}(y_k^{(t+1)} - y_k^{(t)})a_k$, since only the coordinates in K are updated.

We still need an inequality characterizing the relation between $x^{(t+1)}$ and $x^{(t)}$. Following the same steps for deriving inequality (32), and using the λ-strong convexity of function g, it is not difficult to show that

$$\frac{\|x^{(t)} - x^\star\|_2^2}{2\tau} \geq \left(\frac{1}{2\tau} + \lambda\right)\|x^{(t+1)} - x^\star\|_2^2 + \frac{\|x^{(t+1)} - x^{(t)}\|_2^2}{2\tau} + \Big\langle u^{(t)} - u^\star + \frac{1}{m}\sum_{k\in K}(y_k^{(t+1)} - y_k^{(t)})a_k, x^{(t+1)} - x^\star\Big\rangle. \tag{35}$$

Taking expectation over both side of inequality (35), then adding it to inequality (34), we have

$$\begin{aligned}\frac{\|x^{(t)} - x^\star\|_2^2}{2\tau} + \left(\frac{1}{2\sigma} + \frac{(n-m)\gamma}{n}\right)\frac{\|y^{(t)} - y^\star\|_2^2}{m} \;\geq\; & \left(\frac{1}{2\tau} + \lambda\right)\mathbb{E}[\|x^{(t+1)} - x^\star\|_2^2|\mathcal{F}_t] \\ & + \left(\frac{1}{2\sigma} + \gamma\right)\frac{\mathbb{E}[\|y^{(t+1)} - y^\star\|_2^2|\mathcal{F}_t]}{m} + \frac{\mathbb{E}[\|x^{(t+1)} - x^{(t)}\|_2^2|\mathcal{F}_t]}{2\tau} + \frac{\mathbb{E}[\|y^{(t+1)} - y^{(t)}\|_2^2|\mathcal{F}_t]}{2m\sigma} \\ & + \mathbb{E}\left[\left(\frac{y^{(t)} - y^\star}{n} + \frac{y^{(t+1)} - y^{(t)}}{m}\right)^T A(x^{(t+1)} - x^{(t)} - \theta(x^{(t)} - x^{(t-1)}))\Big|\mathcal{F}_t\right].\end{aligned} \tag{36}$$

For the last term of inequality (36), we have plugged in the definitions of $u^{(t+1)}$, $u^\star$ and $\overline{x}^{(t)}$, and used the relation that $(y^{(t+1)} - y^{(t)})^T A = \sum_{k\in K}(y_k^{(t+1)} - y_k^{(t)})a_k^T$. The matrix A is a n-by-d matrix, whose i-th row is equal to the vector a_i^T.

For the rest of the proof, we lower bound the last term on the right-hand-side of inequality (36). In particular, we have

$$\begin{aligned}\left(\frac{y^{(t)} - y^\star}{n} + \frac{y^{(t+1)} - y^{(t)}}{m}\right)^T A(x^{(t+1)} - x^{(t)} - \theta(x^{(t)} - x^{(t-1)})) &= \frac{(y^{(t+1)} - y^\star)^T A(x^{(t+1)} - x^{(t)})}{n} \\ & - \frac{\theta(y^{(t)} - y^\star)^T A(x^{(t)} - x^{(t-1)})}{n} + \frac{n-m}{mn}(y^{(t+1)} - y^{(t)})^T A(x^{(t+1)} - x^{(t)}) \\ & - \frac{\theta}{m}(y^{(t+1)} - y^{(t)})^T A(x^{(t)} - x^{(t-1)}).\end{aligned} \tag{37}$$

Recall that $\|a_k\|_2 \le R$ and $1/\tau = 4\sigma R^2$ according to (10). We have

$$\begin{aligned}
|(y^{(t+1)} - y^{(t)})^T A(x^{(t+1)} - x^{(t)})| &\le \frac{\|x^{(t+1)} - x^{(t)}\|_2^2}{4\tau/m} + \frac{\|(y^{(t+1)} - y^{(t)})^T A\|_2^2}{m/\tau} \\
&= \frac{\|x^{(t+1)} - x^{(t)}\|_2^2}{4\tau/m} + \frac{(\sum_{k\in K} |y_k^{(t+1)} - y_k^{(t)}| \cdot \|a_k\|_2)^2}{4m\sigma R^2} \\
&\le \frac{m\|x^{(t+1)} - x^{(t)}\|_2^2}{4\tau} + \frac{\|y^{(t+1)} - y^{(t)}\|_2^2}{4\sigma},
\end{aligned}$$

Similarly, we have

$$|(y^{(t+1)} - y^{(t)})^T A(x^{(t)} - x^{(t-1)})| \le \frac{m\|x^{(t)} - x^{(t-1)}\|_2^2}{4\tau} + \frac{\|y^{(t+1)} - y^{(t)}\|_2^2}{4\sigma}.$$

The above upper bounds on the absolute values imply

$$\begin{aligned}
(y^{(t+1)} - y^{(t)})^T A(x^{(t+1)} - x^{(t)}) &\ge -\frac{m\|x^{(t+1)} - x^{(t)}\|_2^2}{4\tau} - \frac{\|y^{(t+1)} - y^{(t)}\|_2^2}{4\sigma}, \\
(y^{(t+1)} - y^{(t)})^T A(x^{(t)} - x^{(t-1)}) &\ge -\frac{m\|x^{(t)} - x^{(t-1)}\|_2^2}{4\tau} - \frac{\|y^{(t+1)} - y^{(t)}\|_2^2}{4\sigma}.
\end{aligned}$$

Combining the above two inequalities with lower bounds (36) and (37), we obtain

$$\begin{aligned}
\frac{\|x^{(t)} - x^\star\|_2^2}{2\tau} + \left(\frac{1}{2\sigma} + \frac{(n-m)\gamma}{n}\right)\frac{\|y^{(t)} - y^\star\|_2^2}{m} &\ge \left(\frac{1}{2\tau} + \lambda\right)\mathbb{E}[\|x^{(t+1)} - x^\star\|_2^2|\mathcal{F}_t] \\
&+ \left(\frac{1}{2\sigma} + \gamma\right)\frac{\mathbb{E}[\|y^{(t+1)} - y^\star\|_2^2|\mathcal{F}_t]}{m} + \frac{\mathbb{E}[\|x^{(t+1)} - x^{(t)}\|_2^2|\mathcal{F}_t] - \theta\|x^{(t)} - x^{(t-1)}\|_2^2}{4\tau} \\
&+ \frac{\mathbb{E}[(y^{(t+1)} - y^\star)^T A(x^{(t+1)} - x^{(t)})|\mathcal{F}_t] - \theta(y^{(t)} - y^\star)^T A(x^{(t)} - x^{(t-1)})}{n}. \qquad (38)
\end{aligned}$$

Recall that the parameters τ, σ, and θ are chosen to be

$$\tau = \frac{1}{2R}\sqrt{\frac{m\gamma}{n\lambda}}, \quad \sigma = \frac{1}{2R}\sqrt{\frac{n\lambda}{m\gamma}}, \quad \text{and} \quad \theta = 1 - \frac{1}{(n/m) + R\sqrt{(n/m)/(\lambda\gamma)}}.$$

Plugging in these assignments, we find that

$$\frac{1/(2\tau)}{1/(2\tau) + \lambda} = 1 - \frac{1}{1 + 1/(2\tau\lambda)} \le \theta \quad \text{and} \quad \frac{1/(2\sigma) + (n-m)\gamma/n}{1/(2\sigma) + \gamma} = 1 - \frac{1}{n/m + n/(2m\sigma\gamma)} = \theta$$

Therefore, if we define a sequence $\Delta^{(t)}$ such that

$$\begin{aligned}
\Delta^{(t)} = \left(\frac{1}{2\tau} + \lambda\right)\mathbb{E}[\|x^{(t)} - x^\star\|_2^2] + \left(\frac{1}{2\sigma} + \gamma\right)\frac{\mathbb{E}[\|y^{(t)} - y^\star\|_2^2]}{m} \\
+ \frac{\mathbb{E}[\|x^{(t)} - x^{(t-1)}\|_2^2]}{4\tau} + \frac{\mathbb{E}[(y^{(t)} - y^\star)^T A(x^{(t)} - x^{(t-1)})]}{n},
\end{aligned}$$

then inequality (38) implies the recursive relation $\Delta^{(t+1)} \leq \theta \cdot \Delta^{(t)}$, which implies

$$\left(\frac{1}{2\tau}+\lambda\right)\mathbb{E}[\|x^{(t)}-x^{\star}\|_2^2]+\left(\frac{1}{2\sigma}+\gamma\right)\frac{\mathbb{E}[\|y^{(t)}-y^{\star}\|_2^2]}{m} + \frac{\mathbb{E}[\|x^{(t)}-x^{(t-1)}\|_2^2]}{4\tau}+\frac{\mathbb{E}[(y^{(t)}-y^{\star})^T A(x^{(t)}-x^{(t-1)})]}{n} \;\leq\; \theta^t\cdot\Delta^{(0)}, \tag{39}$$

where

$$\Delta^{(0)}=\left(\frac{1}{2\tau}+\lambda\right)\|x^{(0)}-x^{\star}\|_2^2+\left(\frac{1}{2\sigma}+\gamma\right)\frac{\|y^{(0)}-y^{\star}\|_2^2}{m}.$$

To eliminate the last two terms on the left-hand side of inequality (39), we notice that

$$\begin{aligned}\frac{\left|(y^{(t)}-y^{\star})^T A(x^{(t)}-x^{(t-1)})\right|}{n} &\leq \frac{\|x^{(t)}-x^{(t-1)}\|_2^2}{4\tau}+\frac{\|A\|_2^2\,\|y^{(t)}-y^{\star}\|_2^2}{n^2/\tau}\\ &\leq \frac{\|x^{(t)}-x^{(t-1)}\|_2^2}{4\tau}+\frac{nR^2\|y^{(t)}-y^{\star}\|_2^2}{n^2/\tau}\\ &= \frac{\|x^{(t)}-x^{(t-1)}\|_2^2}{4\tau}+\frac{\|y^{(t)}-y^{\star}\|_2^2}{4n\sigma}\\ &\leq \frac{\|x^{(t)}-x^{(t-1)}\|_2^2}{4\tau}+\frac{\|y^{(t)}-y^{\star}\|_2^2}{4m\sigma},\end{aligned}$$

where in the second inequality we used $\|A\|_2^2 \leq \|A\|_F^2 \leq nR^2$, in the equality we used $\tau\sigma = 1/(4R^2)$, and in the last inequality we used $m \leq n$. The above upper bound on absolute value implies

$$\frac{(y^{(t)}-y^{\star})^T A(x^{(t)}-x^{(t-1)})}{n} \;\geq\; -\frac{\|x^{(t)}-x^{(t-1)}\|_2^2}{4\tau}-\frac{\|y^{(t)}-y^{\star}\|_2^2}{4m\sigma}.$$

The theorem is established by combining the above inequality with inequality (39).

B Proof of Theorem 2

The proof of Theorem 2 mimics the steps for proving Theorem 1. We start by establishing relation between $(y^{(t)}, y^{(t+1)})$ and between $(x^{(t)}, x^{(t+1)})$. Suppose that the quantity $\widetilde{y}_i$ minimizes the function $\phi_i^*(y) - y\langle a_i, \overline{x}^{(t)}\rangle + \frac{p_i n}{2\sigma}(y - y_i^{(t)})^2$. Then, following the same argument for establishing inequality (32), we obtain

$$\frac{p_i n}{2\sigma}(y_i^{(t)}-y_i^{\star})^2 \geq \left(\frac{p_i n}{2\sigma}+\gamma\right)(\widetilde{y}_i-y_i^{\star})^2+\frac{p_i n(\widetilde{y}_i-y_i^{(t)})^2}{2\sigma}+\langle a_i, x^{\star}-\overline{x}^{(t)}\rangle(\widetilde{y}_i-y_i^{\star}). \tag{40}$$

Note that $i = k$ with probability p_i. Therefore, we have

$$\begin{aligned}(\widetilde{y}_i-y_i^{\star})^2 &= \frac{1}{p_i}\mathbb{E}[(y_i^{(t+1)}-y_i^{\star})^2|\mathcal{F}_t]-\frac{1-p_i}{p_i}(y_i^{(t)}-y_i^{\star})^2,\\ (\widetilde{y}_i-y_i^{(t)})^2 &= \frac{1}{p_i}\mathbb{E}[(y_i^{(t+1)}-y_i^{(t)})^2|\mathcal{F}_t],\\ \widetilde{y}_i &= \frac{1}{p_i}\mathbb{E}[y_i^{(t+1)}|\mathcal{F}_t]-\frac{1-p_i}{p_i}y_i^{(t)},\end{aligned}$$

where $\mathcal{F}_t$ represents the sigma field generated by all random variables defined before iteration t. Substituting the above equations into inequality (40), and averaging over $i = 1, 2, \ldots, n$, we have

$$\sum_{i=1}^{n}\left(\frac{1}{2\sigma}+\frac{(1-p_i)\gamma}{p_i n}\right)(y_i^{(t)}-y_i^\star)^2 \geq \sum_{i=1}^{n}\left(\frac{1}{2\sigma}+\frac{\gamma}{p_i n}\right)\mathbb{E}[(y_i^{(t+1)}-y_i^\star)^2|\mathcal{F}_t] + \frac{\mathbb{E}[(y_k^{(t+1)}-y_k^{(t)})^2|\mathcal{F}_t]}{2\sigma} + \mathbb{E}[\langle (u^{(t)}-u^\star)+(y_k^{(t+1)}-y_k^{(t)})a_k/(p_k n), x^\star - \overline{x}^{(t)}\rangle|\mathcal{F}_t], \tag{41}$$

where $u^\star = \frac{1}{n}\sum_{i=1}^n y_i^\star a_i$ and $u^{(t)} = \frac{1}{n}\sum_{i=1}^n y_i^{(t)} a_i$ have the same definition as in the proof of Theorem 1. For the relation between $x^{(t)}$ and $x^{(t+1)}$, we follow the steps in the proof of Theorem 1 to obtain

$$\frac{\|x^{(t)}-x^\star\|_2^2}{2\tau} \geq \left(\frac{1}{2\tau}+\lambda\right)\|x^{(t+1)}-x^\star\|_2^2 + \frac{\|x^{(t+1)}-x^{(t)}\|_2^2}{2\tau} + \langle (u^{(t)}-u^\star)+(y_k^{(t+1)}-y_k^{(t)})a_k/(p_k n), x^{(t+1)}-x^\star\rangle. \tag{42}$$

Taking expectation over both sides of inequality (42) and adding it to inequality (41) yields

$$\begin{aligned}\frac{\|x^{(t)}-x^\star\|_2^2}{2\tau} + \sum_{i=1}^{n}\left(\frac{1}{2\sigma}+\frac{(1-p_i)\gamma}{p_i n}\right)(y_i^{(t)}-y_i^\star)^2 &\geq \left(\frac{1}{2\tau}+\lambda\right)\mathbb{E}[\|x^{(t+1)}-x^\star\|_2^2|\mathcal{F}_t] \\ &+ \sum_{i=1}^{n}\left(\frac{1}{2\sigma}+\frac{\gamma}{p_i n}\right)\mathbb{E}[(y_i^{(t+1)}-y_i^\star)^2|\mathcal{F}_t] + \frac{\|x^{(t+1)}-x^{(t)}\|_2^2}{2\tau} + \frac{\mathbb{E}[(y_k^{(t+1)}-y_k^{(t)})^2|\mathcal{F}_t]}{2\sigma} \\ &+ \mathbb{E}\Big[\underbrace{\Big(\frac{(y^{(t)}-y^\star)^T A}{n}+\frac{(y_k^{(t+1)}-y_k^{(t)})a_k^T}{p_k n}\Big)((x^{(t+1)}-x^{(t)})-\theta(x^{(t)}-x^{(t-1)}))}_{v}\Big|\mathcal{F}_t\Big],\end{aligned} \tag{43}$$

where the matrix A is a n-by-d matrix, whose i-th row is equal to the vector a_i^T.

Next, we lower bound the last term on the right-hand side of inequality (43). Indeed, it can be expanded as

$$\begin{aligned}v = &\frac{(y^{(t+1)}-y^\star)^T A(x^{(t+1)}-x^{(t)})}{n} - \frac{\theta(y^{(t)}-y^\star)^T A(x^{(t)}-x^{(t-1)})}{n} \\ &+ \frac{1-p_k}{p_k n}(y_k^{(t+1)}-y_k^{(t)})a_k^T(x^{(t+1)}-x^{(t)}) - \frac{\theta}{p_k n}(y_k^{(t+1)}-y_k^{(t)})a_k^T(x^{(t)}-x^{(t-1)}).\end{aligned} \tag{44}$$

Note that the probability p_k given in (17) satisfies

$$p_k \geq \frac{\|a_k\|_2}{2\sum_{i=1}^n \|a_i\|_2} = \frac{\|a_k\|_2}{2n\bar{R}}, \qquad k = 1, \ldots, n.$$

Since the parameters τ and σ satisfies $\sigma\tau\bar{R}^2 = 1/16$, we have $p_k^2 n^2/\tau \geq 4\sigma\|a_k\|_2^2$ and consequently

$$\begin{aligned}\frac{|(y_k^{(t+1)}-y_k^{(t)})a_k^T(x^{(t+1)}-x^{(t)})|}{p_k n} &\leq \frac{\|x^{(t+1)}-x^{(t)}\|_2^2}{4\tau} + \frac{\|(y_k^{(t+1)}-y_k^{(t)})a_k\|_2^2}{p_k^2 n^2/\tau} \\ &\leq \frac{\|x^{(t+1)}-x^{(t)}\|_2^2}{4\tau} + \frac{(y_k^{(t+1)}-y_k^{(t)})^2}{4\sigma}.\end{aligned}$$

Similarly, we have

$$\frac{|(y_k^{(t+1)} - y_k^{(t)}) a_k^T (x^{(t)} - x^{(t-1)})|}{p_k n} \le \frac{\|x^{(t)} - x^{(t-1)}\|_2^2}{4\tau} + \frac{(y_k^{(t+1)} - y_k^{(t)})^2}{4\sigma}.$$

Combining the above two inequalities with lower bounds (43) and (44), we obtain

$$\begin{aligned}
\frac{\|x^{(t)} - x^\star\|_2^2}{2\tau} + \sum_{i=1}^n \left(\frac{1}{2\sigma} + \frac{(1-p_i)\gamma}{p_i n}\right)(y_i^{(t)} - y_i^\star)^2 &\ge \left(\frac{1}{2\tau} + \lambda\right)\mathbb{E}[\|x^{(t+1)} - x^\star\|_2^2 | \mathcal{F}_t] \\
+ \sum_{i=1}^n \left(\frac{1}{2\sigma} + \frac{\gamma}{p_i n}\right)\mathbb{E}[(y_i^{(t+1)} - y_i^\star)^2 | \mathcal{F}_t] &+ \frac{\mathbb{E}[\|x^{(t+1)} - x^{(t)}\|_2^2 | \mathcal{F}_t] - \theta\|x^{(t)} - x^{(t-1)}\|_2^2}{4\tau} \\
+ \frac{\mathbb{E}[(y^{(t+1)} - y^\star)^T A(x^{(t+1)} - x^{(t)}) | \mathcal{F}_t] - \theta(y^{(t)} - y^\star)A(x^{(t)} - x^{(t-1)})}{n}&. \qquad (45)
\end{aligned}$$

Recall that the parameters τ, σ, and θ are chosen to be

$$\tau = \frac{1}{4\bar{R}}\sqrt{\frac{\gamma}{n\lambda}}, \quad \sigma = \frac{1}{4\bar{R}}\sqrt{\frac{n\lambda}{\gamma}}, \quad \text{and} \quad \theta = 1 - \frac{1}{2n + 2\bar{R}\sqrt{n/(\lambda\gamma)}}.$$

Plugging in these assignments and using the fact that $p_i \ge 1/(2n)$, we find that

$$\frac{1/(2\tau)}{1/(2\tau) + \lambda} \le \theta \quad \text{and} \quad \frac{1/(2\sigma) + (1-p_i)\gamma/(p_i n)}{1/(2\sigma) + \gamma/(p_i n)} \le \theta \quad \text{for } i = 1, 2, \ldots, n.$$

Therefore, if we define a sequence $\Delta^{(t)}$ such that

$$\begin{aligned}
\Delta^{(t)} = \left(\frac{1}{2\tau} + \lambda\right)\mathbb{E}[\|x^{(t)} - x^\star\|_2^2] + \sum_{i=1}^n \left(\frac{1}{2\sigma} + \frac{\gamma}{p_i n}\right)\mathbb{E}[(y_i^{(t)} - y_i^\star)^2] \\
+ \frac{\mathbb{E}[\|x^{(t)} - x^{(t-1)}\|_2^2]}{4\tau} + \frac{\mathbb{E}[(y^{(t)} - y^\star)^T A(x^{(t)} - x^{(t-1)})]}{n},
\end{aligned}$$

then inequality (45) implies the recursive relation $\Delta^{(t+1)} \le \theta \cdot \Delta^{(t)}$, which implies

$$\begin{aligned}
\left(\frac{1}{2\tau} + \lambda\right)\mathbb{E}[\|x^{(t)} - x^\star\|_2^2] + \left(\frac{1}{2\sigma} + \frac{2\gamma}{n}\right)\mathbb{E}[\|y^{(t)} - y^\star\|_2^2] \\
+ \frac{\mathbb{E}[\|x^{(t)} - x^{(t-1)}\|_2^2]}{4\tau} + \frac{\mathbb{E}[(y^{(t)} - y^\star)^T A(x^{(t)} - x^{(t-1)})]}{n} \;\le\; \theta^T \Delta^{(0)}, \qquad (46)
\end{aligned}$$

where

$$\begin{aligned}
\Delta^{(0)} &= \left(\frac{1}{2\tau} + \lambda\right)\|x^{(0)} - x^\star\|_2^2 + \sum_{i=1}^n \left(\frac{1}{2\sigma} + \frac{\gamma}{p_i n}\right)(y_i^{(0)} - y_i^\star)^2 \\
&\le \left(\frac{1}{2\tau} + \lambda\right)\|x^{(0)} - x^\star\|_2^2 + \left(\frac{1}{2\sigma} + 2\gamma\right)\|y^{(0)} - y^\star\|_2^2.
\end{aligned}$$

To eliminate the last two terms on the left-hand side of inequality (46), we notice that

$$\begin{aligned}\frac{|(y^{(t)}-y^\star)^T A(x^{(t)}-x^{(t-1)})|}{n} &\le \frac{\|x^{(t)}-x^{(t-1)}\|_2^2}{4\tau}+\frac{\|y^{(t)}-y^\star\|_2^2\,\|A\|_2^2}{n^2/\tau}\\ &\le \frac{\|x^{(t)}-x^{(t-1)}\|_2^2}{4\tau}+\frac{\|y^{(t)}-y^\star\|_2^2\,\|A\|_F^2}{n^2/\tau}\\ &= \frac{\|x^{(t)}-x^{(t-1)}\|_2^2}{4\tau}+\frac{\|y^{(t)}-y^\star\|_2^2\,\sum_{i=1}^n\|a_i\|_2^2}{16\sigma(\sum_{i=1}^n\|a_i\|_2)^2}\\ &\le \frac{\|x^{(t)}-x^{(t-1)}\|_2^2}{4\tau}+\frac{\|y^{(t)}-y^\star\|_2^2}{16\sigma},\end{aligned}$$

where in the equality we used $n^2/\tau = n^2\cdot 16\sigma\bar{R}^2 = 16\sigma\left(\sum_{i=1}^n\|a_i\|_2\right)^2$. This implies

$$\frac{|(y^{(t)}-y^\star)^T A(x^{(t)}-x^{(t-1)})|}{n} \;\ge\; -\frac{\|x^{(t)}-x^{(t-1)}\|_2^2}{4\tau}-\frac{\|y^{(t)}-y^\star\|_2^2}{16\sigma}.$$

Substituting the above inequality into inequality (46) completes the proof.

C Efficient update for $(\ell_1+\ell_2)$-norm penalty

From Section 5.2, we have the following recursive formula for $t\in[t_0+1,t_1]$,

$$x_j^{(t+1)} = \begin{cases} \frac{1}{1+\lambda_2\tau}(x_j^{(t)}-\tau u^{(t_0+1)}-\tau\lambda_1) & \text{if } x_j^{(t)}-\tau u_j^{(t_0+1)}>\tau\lambda_1,\\ \frac{1}{1+\lambda_2\tau}(x_j^{(t)}-\tau u^{(t_0+1)}+\tau\lambda_1) & \text{if } x_j^{(t)}-\tau u_j^{(t_0+1)}<-\tau\lambda_1,\\ 0 & \text{otherwise.}\end{cases} \tag{47}$$

Given $x_j^{(t_0+1)}$ at iteration t_0, we present an efficient algorithm for calculating $x_j^{(t_1)}$. We begin by examining the sign of $x_j^{(t_0+1)}$.

Case I ($x_j^{(t_0+1)}=0$): If $-u_j^{(t_0+1)}>\lambda_1$, then equation (47) implies $x_j^{(t)}>0$ for all $t>t_0+1$. Consequently, we have a closed-form formula for $x_j^{(t_1)}$:

$$x_j^{(t_1)} = \frac{1}{(1+\lambda_2\tau)^{t_1-t_0-1}}\Big(x_j^{(t_0+1)}+\frac{u_j^{(t_0+1)}+\lambda_1}{\lambda_2}\Big)-\frac{u_j^{(t_0+1)}+\lambda_1}{\lambda_2}. \tag{48}$$

If $-u_j^{(t_0+1)}<-\lambda_1$, then equation (47) implies $x_j^{(t)}<0$ for all $t>t_0+1$. Therefore, we have the closed-form formula:

$$x_j^{(t_1)} = \frac{1}{(1+\lambda_2\tau)^{t_1-t_0-1}}\Big(x_j^{(t_0+1)}+\frac{u_j^{(t_0+1)}-\lambda_1}{\lambda_2}\Big)-\frac{u_j^{(t_0+1)}-\lambda_1}{\lambda_2}. \tag{49}$$

Finally, if $-u_j^{(t_0+1)}\in[-\lambda_1,\lambda_1]$, then equation (47) implies $x_j^{(t_1)}=0$.

Case II ($x_j^{(t_0+1)} > 0$): If $-u_j^{(t_0+1)} \geq \lambda_1$, then it is easy to verify that $x_j^{(t_1)}$ is obtained by equation (48). Otherwise, We use the recursive formula (47) to derive the latest time $t^+ \in [t_0+1, t_1]$ such that $x_j^{t^+} > 0$ is true. Indeed, since $x_j^{(t)} > 0$ for all $t \in [t_0+1, t^+]$, we have a closed-form formula for $x_j^{t^+}$:

$$x_j^{t^+} = \frac{1}{(1+\lambda_2\tau)^{t^+-t_0-1}}\Big(x_j^{(t_0+1)} + \frac{u_j^{(t_0+1)}+\lambda_1}{\lambda_2}\Big) - \frac{u_j^{(t_0+1)}+\lambda_1}{\lambda_2}. \tag{50}$$

We look for the largest t^+ such that the right-hand side of equation (50) is positive, which is equivalent of

$$t^+ - t_0 - 1 < \log\Big(1 + \frac{\lambda_2 x_j^{(t_0+1)}}{u_j^{(t_0+1)}+\lambda_1}\Big)/\log(1+\lambda_2\tau). \tag{51}$$

Thus, t^+ is the largest integer in $[t_0+1, t_1]$ such that inequality (51) holds. If $t^+ = t_1$, then $x_j^{(t_1)}$ is obtained by (50). Otherwise, we can calculate $x_j^{t^++1}$ by formula (47), then resort to Case I or Case III, treating t^+ as t_0.

Case III ($x_j^{(t_0+1)} < 0$): If $-u_j^{(t_0+1)} \leq -\lambda_1$, then $x_j^{(t_1)}$ is obtained by equation (49). Otherwise, we calculate the largest integer $t^- \in [t_0+1, t_1]$ such that $x_j^{t^-} < 0$ is true. Using the same argument as for Case II, we have the closed-form expression

$$x_j^{t^-} = \frac{1}{(1+\lambda_2\tau)^{t^--t_0-1}}\Big(x_j^{(t_0+1)} + \frac{u_j^{(t_0+1)}-\lambda_1}{\lambda_2}\Big) - \frac{u_j^{(t_0+1)}-\lambda_1}{\lambda_2}. \tag{52}$$

where t^- is the largest integer in $[t_0+1, t_1]$ such that the following inequality holds:

$$t^- - t_0 - 1 < \log\Big(1 + \frac{\lambda_2 x_j^{(t_0+1)}}{u_j^{(t_0+1)}-\lambda_1}\Big)/\log(1+\lambda_2\tau). \tag{53}$$

If $t^- = t_1$, then $x_j^{(t_1)}$ is obtained by (52). Otherwise, we can calculate $x_j^{t^-+1}$ by formula (47), then resort to Case I or Case II, treating t^- as t_0.

Finally, we note that formula (47) implies the monotonicity of $x_j^{(t)}$ ($t = t_0+1, t_0+2, \dots$). As a consequence, the procedure of either Case I, Case II or Case III is executed for at most once. Hence, the algorithm for calculating $x_j^{(t_1)}$ has $\mathcal{O}(1)$ time complexity.

References

[1] A. Beck and M. Teboulle. A fast iterative shrinkage-threshold algorithm for linear inverse problems. *SIAM Journal on Imaging Sciences*, 2(1):183–202, 2009.

[2] D. P. Bertsekas. Incremental proximal methods for large scale convex optimization. *Mathematical Programming, Ser. B*, 129:163–195, 2011.

[3] D. P. Bertsekas. Incremental gradient, subgradient, and proximal methods for convex optimization: a survey. In S. Sra, S. Nowozin, and S. J. Wright, editors, *Optimization for Machine Learning*, chapter 4. The MIT Press, 2012.

[4] D. Blatt, A. O. Hero, and H. Gauchman. A convergent incremental gradient method with a constant step size. *SIAM Journal on Optimization*, 18(1):29–51, 2007.

[5] L. Bottou. Large-scale machine learning with stochastic gradient descent. In Y. Lechevallier and G. Saporta, editors, *Proceedings of the 19th International Conference on Computational Statistics (COMPSTAT'2010)*, pages 177–187, Paris, France, August 2010. Springer.

[6] O. Bousquet and A. Elisseeff. Stability and generalization. *Journal of Machine Learning Research*, 2:499–526, 2002.

[7] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends in Machine Learning*, 3(1):1–122, 2010.

[8] A. Chambolle and T. Pock. A first-order primal-dual algorithm for convex problems with applications to imaging. *Journal of Mathematical Imaging and Vision*, 40(1):120–145, 2011.

[9] K.-W. Chang, C.-J. Hsieh, and C.-J. Lin. Coordinate descent method for large-scale l_2-loss linear support vector machines. *Journal of Machine Learning Research*, 9:1369–1398, 2008.

[10] J. Dean and S. Ghemawat. MapReduce: Simplfied data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[11] J. Duchi and Y. Singer. Efficient online and batch learning using forward backward splitting. *Journal of Machine Learning Research*, 10:2873–2898, 2009.

[12] R.-E. Fan and C.-J. Lin. LIBSVM data: Classification, regression and multi-label. URL: http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets, 2011.

[13] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, New York, 2nd edition, 2009.

[14] J.-B. Hiriart-Urruty and C. Lemaréchal. *Fundamentals of Convex Analysis*. Springer, 2001.

[15] C.-J. Hsieh, K.-W. Chang, C.-J. Lin, S. Keerthi, and S. Sundararajan. A dual coordinate descent method for large-scale linear svm. In *Proceedings of the 25th International Conference on Machine Learning (ICML)*, pages 408–415, 2008.

[16] R. Johnson and T. Zhang. Accelerating stochastic gradient descent using predictive variance reduction. In *Advances in Neural Information Processing Systems 26*, pages 315–323. 2013.

[17] J. Langford, L. Li, and T. Zhang. Sparse online learning via truncated gradient. *Journal of Machine Learning Research*, 10:777–801, 2009.

[18] Q. Lin, Z. Lu, and L. Xiao. An accelerated proximal coordinate gradient method and its application to regularized empirical risk minimization. Technical Report MSR-TR-2014-94, Microsoft Research, 2014. arXiv:1407.1296.

[19] P. L. Lions and B. Mercier. Splitting algorithms for the sum of two nonlinear operators. *SIAM Journal on Numerical Analysis*, 16(6):964–979, December 1979.

[20] MPI Forum. MPI: a message-passing interface standard, Version 3.0. Document available at `http://www.mpi-forum.org`, 2012.

[21] A. Nedić and D. P. Bertsekas. Incremental subgradient methods for nondifferentiable optimization. *SIAM Journal on Optimization*, 12(1):109–138, 2001.

[22] D. Needell, N. Srebro, and R. Ward. Stochastic gradient descent, weighted sampling, and the randomized Kaczmarz algorithm. *arXiv preprint arXiv:1310.5715*, 2014.

[23] A. Nemirovski, A. Juditsky, G. Lan, and A. Shapiro. Robust stochastic approximation approach to stochastic programming. *SIAM Journal on Optimization*, 19(4):1574–1609, 2009.

[24] Y. Nesterov. *Introductory Lectures on Convex Optimization: A Basic Course*. Kluwer, Boston, 2004.

[25] Y. Nesterov. Efficiency of coordinate descent methods on huge-scale optimization problems. *SIAM Journal on Optimization*, 22(2):341–362, 2012.

[26] Y. Nesterov. Gradient methods for minimizing composite functions. *Mathematical Programming, Ser. B*, 140:125–161, 2013.

[27] J. Nocedal and S. J. Wright. *Numerical Optimization*. Springer, New York, 2nd edition, 2006.

[28] H. Ouyang, N. He, L. Tran, and A. Gray. Stochastic alternating direction method of multipliers. In *Proceedings of the 30th International Conference on Machine Learning (ICML)*, Atlanta, GA, USA, 2013.

[29] J. Platt. Fast training of support vector machine using sequential minimal optimization. In B. Schölkopf, C. Burges, and A. Smola, editors, *Advances in Kernel Methods — Support Vector Learning*, pages 185–208. MIT Press, Cambridge, MA, USA, 1999.

[30] B. T. Polyak and A. Juditsky. Acceleration of stochastic approximation by averaging. *SIAM Journal on Control and Optimization*, 30:838–855, 1992.

[31] P. Richtárik and M. Takáč. Iteration complexity of randomized block-coordinate descent methods for minimizing a composite function. *Mathematical Programming*, 144(1):1–38, 2014.

[32] N. L. Roux, M. Schmidt, and F. Bach. A stochastic gradient method with an exponential convergence rate for finite training sets. In *Advances in Neural Information Processing Systems 25*, pages 2672–2680. 2012.

[33] M. Schmidt, N. L. Roux, and F. Bach. Minimizing finite sums with the stochastic average gradient. Technical Report HAL 00860051, INRIA, Paris, France, 2013.

[34] S. Shalev-Shwartz and T. Zhang. Accelerated proximal stochastic dual coordinate ascent for regularized loss minimization. arXiv:1309.2375.

[35] S. Shalev-Shwartz and T. Zhang. Accelerated mini-batch stochastic dual coordinate ascent. In C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Weinberger, editors, *Advances in Neural Information Processing Systems 26*, pages 378–385. 2013.

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

[37] T. Suzuki. Dual averaging and proximal gradient descent for online alternating direction multiplier method. In *Proceedings of the 30th International Conference on Machine Learning (ICML)*, pages 392–400, Atlanta, GA, USA, 2013.

[38] T. Suzuki. Stochastic dual coordinate ascent with alternating direction method of multipliers. In *Proceedings of the 31st International Conference on Machine Learning (ICML)*, pages 736–744, Beijing, 2014.

[39] M. Takáč, A. Bijral, P. Richtárik, and N. Srebro. Mini-batch primal and dual methods for SVMs. In *Proceedings of the 30th International Conference on Machine Learning (ICML)*, 2013.

[40] P. Tseng. An incremental gradient(-projection) method with momentum term and adaptive stepsiz rule. *SIAM Journal on Optimization*, 8(2):506–531, 1998.

[41] P. Tseng. On accelerated proximal gradient methods for convex-concave optimization. Unpublished manuscript, 2008.

[42] H. Wang and A. Banerjee. Online alternating direction method. In *Proceedings of the 29th International Conference on Machine Learning (ICML)*, pages 1119–1126, Edinburgh, Scotland, UK, 2012.

[43] L. Xiao. Dual averaging methods for regularized stochastic learning and online optimization. *Journal of Machine Learning Research*, 11:2534–2596, 2010.

[44] L. Xiao and T. Zhang. A proximal stochastic gradient method with progressive variance reduction. Technical Report MSR-TR-2014-38, Microsoft Research, 2014. arXiv:1403.4699.

[45] T. Yang. Trading computation for communication: Distributed stochastic dual coordinate ascent. In *Advances in Neural Information Processing Systems 26*, pages 629–637. 2013.

[46] T. Zhang. Solving large scale linear prediction problems using stochastic gradient descent algorithms. In *Proceedings of the 21st International Conference on Machine Learning (ICML)*, pages 116–123, Banff, Alberta, Canada, 2004.

[47] X. Zhang, M. Burger, and S. Osher. A unifoed primal-dual algorithm framework based on Bregman iteration. *Journal of Scientific Computing*, 46(1):20–46, January 2011.

[48] P. Zhao and T. Zhang. Stochastic optimization with importance sampling. arXiv:1401.2753, 2014.

[49] L. W. Zhong and J. T. Kwok. Fast stochastic alternating direction method of multipliers. In *Proceedings of the 30th International Conference on Machine Learning (ICML)*, Atlanta, GA, USA, 2013.