Communication Efficient Distributed Learning with Feature Partitioned Data

Bingwen Zhang[1] and Jun Geng[2] and Weiyu Xu[3] and Lifeng Lai[4]
[1] Dept. of Elec. and Comp. Engr., Worcester Poly. Inst., Worcester, MA, bzhang@wpi.edu
[2] Sch. of Elec. and Info. Engr., Harbin Inst. of Tech., Harbin, China, jgeng@hit.edu.cn
[3] Dept. of Elec. and Comp. Engr., U. of Iowa, Iowa City, IA, weiyu-xu@uiowa.edu
[4] Dept. of Elec. and Comp. Engr., U. of California, Davis, CA, lflai@ucdavis.edu

***Abstract*—One major bottleneck in the design of large scale distributed machine learning algorithms is the communication cost. In this paper, we propose and analyze a distributed learning scheme for reducing the amount of communication in distributed learning problems under the feature partition scenario. The motivating observation of our scheme is that, in the existing schemes for the feature partition scenario, large amount of data exchange is needed for calculating gradients. In our proposed scheme, instead of calculating the exact gradient at each iteration, we only calculate the exact gradient sporadically. We provide precise conditions to determine when to perform the exact update, and characterize the convergence rate and bounds for total iterations and communication iterations. We further test our algorithm on real data sets and show that the proposed scheme can substantially reduce the amount of data transferred between distributed nodes.**

***Index Terms*—Distributed learning, Feature partitioned data, Communication efficiency, Inexact update**

I. Introduction

The design of distributed optimization algorithms for machine learning tasks has recently attracted significant research interests [1], [2], [3], [4], [5], [6], [7], [8], [9], [10], [11], [12], [13]. In the distributed optimization problems, the whole dataset is divided into small parts with each part stored in one machine. Depending on how the dataset is partitioned, there are two basic scenarios (and mixtures of these two basic scenarios): sample partition [14], in which each machine has data samples related to all features, and feature partition, in which each machine has data related to only a subset of features [4], [15].

There have been a large number of recent interesting works on the sample partition scenario[1], [2], [3], [4], [16]. Compared with the sample partition scenario, the feature partition scenario is relatively less well understood. The feature partition scenario arises naturally in applications where different features are collected at different machines. Furthermore, for high dimensional data, each machine only deals with a small subset of features and hence this approach reduces the memory requirements and per-iteration computational cost. However, one major challenge associated with the feature partition scenario is that, unlike in the sample partition scenario where each machine can compute important quantities such as gradient (or an approximate of it) using only local data, each machine in the feature partition scenario may not be able to compute these quantities using only local data anymore. Among limited number of works on the feature partition scenario, in [14], [17], the authors propose to use randomized (block) coordinate descent to solve distributed learning problems for the feature partition scenario. As pointed out in [14] (will also be discussed in detail in the sequel), the communication cost associated with computing gradients, which are needed to calculate the next update, is very high.

In this paper, we aim to design communication efficient distributed learning algorithms for the feature partition scenario by addressing the high communication cost issue associated with this scenario. Our key observation is that, in most of the existing works on the feature partition scenario, the main communication cost comes from the high volume of data exchange needed for the computation of gradients. Based on this observation, our main idea is that, instead of computing the gradient at each iteration, we only calculate the exact gradient sporadically (precise conditions as when to calculate the exact update will be given in the sequel). In the iterations when exact gradients are not computed, we will use the most recently calculated gradient for updating. As a result, instead of using the exact gradients at each iteration, we will use an approximation of the gradient as a proxy in computing the next update. The main design challenge is how to select the time instants at which exact gradients are computed so that the total amount of communication is reduced, while at the same time the approximation error caused by the sporadicly calculated gradients are well controlled and the algorithm can still converge. Based on this idea, we design a general communication efficient distributed learning algorithm for the feature partition scenario. Under mild technical assumptions, we further analyze the iteration complexity until convergence and provide an upper-bound on the number of exact updates (i.e., the number of iterations where we have high communication costs) required. We further conduct tests of our algorithms on both synthesized data and real data. From these tests, we observe that the proposed scheme can substantially reduce the communication overhead even when the technical assumptions used in the theoretical analysis are not met or difficult to verify (e.g., for the case with real data).

The remainder of this paper is organized as follows. In

Section II, we describe the problem setup, the challenges with existing approach and our proposed inexact update algorithm. In Section III, we provide several analytical results. In Section IV, we provide our numerical simulation results. In Section V, we offer concluding remarks.

II. Algorithm

Consider a dataset consisting of n samples $(\mathbf{x}_j, y_j), j = 1, \cdots, n$, in which $\mathbf{x}_j \in \mathbb{R}^d$ and $y_j \in \mathbb{R}$. Define $\mathbf{X} := [\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_n]^T \in \mathbb{R}^{n\times d}$ and $\mathbf{y} := (y_1, y_2, \cdots, y_n)^T$. From this dataset, one would like to infer the value of parameters $\boldsymbol{\beta} \in \mathbb{R}^d$ that specify the relationship between $\mathbf{X}$ and $\mathbf{y}$. In the widely used empirical risk minimization (ERM) framework, one infers the value of $\boldsymbol{\beta}$ via solving the following optimization problem:

$$\min_{\boldsymbol{\beta}\in\mathbb{R}^d} \varphi(\boldsymbol{\beta}) := \mathcal{L}(\boldsymbol{\beta}) + \mathcal{R}(\boldsymbol{\beta}), \tag{1}$$

in which $\mathcal{L}(\boldsymbol{\beta})$ is the loss function that measures how well the parameters $\boldsymbol{\beta}$ fit the data, and $\mathcal{R}(\boldsymbol{\beta})$ is the penalty function that measures model complexity. Different forms of $\mathcal{L}$ and $\mathcal{R}$ lead to different popular machine learning algorithms. For example, if $\mathcal{L}(\boldsymbol{\beta}) = \frac{1}{2}\|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|^2$, namely the residual sum of squares (RSS), then setting $\mathcal{R}(\boldsymbol{\beta}) = \lambda\|\boldsymbol{\beta}\|_1$ leads to Lasso [18] and $\mathcal{R}(\boldsymbol{\beta}) = \lambda\|\boldsymbol{\beta}\|^2$ leads to ridge regression [19]. As another example, if $\mathcal{L}(\boldsymbol{\beta}) = \sum_{j=1}^n \log\left(1 + e^{-y_j \mathbf{x}_j^T \boldsymbol{\beta}}\right)$ and $y_j \in \{-1, 1\}$ with penalty functions $\mathcal{R}(\boldsymbol{\beta}) = \lambda\|\boldsymbol{\beta}\|_1$ or $\mathcal{R}(\boldsymbol{\beta}) = \lambda\|\boldsymbol{\beta}\|^2$, we have logistic regression with penalty [20], [21]. If $\mathcal{L}(\boldsymbol{\beta}) = \frac{1}{2}\max\left\{0, 1 - y_j \mathbf{x}_j^T \boldsymbol{\beta}\right\}$ and $\mathcal{R}(\boldsymbol{\beta}) = \lambda\|\boldsymbol{\beta}\|_1$, we have l_1-SVM [22].

The optimization problem (1) has been extensively investigated in the centralized setting where all data is stored in one machine, e.g. in [23], [24] and references therein. In this paper, we consider a distributed setup, in which this dataset is stored in m machines, each of which stores only part of the dataset. We focus on the challenging feature partition scenario where the whole dataset is partitioned by features. Let $[\mathbf{X}^{[1]}, \mathbf{X}^{[2]}, \cdots, \mathbf{X}^{[m]}]$ be a column-wise partition of the dataset. Machine i stores $\mathbf{X}^{[i]} \in \mathbb{R}^{n\times d_i}$ and hence $\sum_{i=1}^m d_i = d$. Let $[{\boldsymbol{\beta}^{[1]}}^T, {\boldsymbol{\beta}^{[2]}}^T, \cdots, {\boldsymbol{\beta}^{[m]}}^T]^T \in \mathbb{R}^d$ be the parameter vector of the corresponding partition with $\boldsymbol{\beta}^{[i]} \in \mathbb{R}^{d_i}$. Let $\mathbf{x}^{[j,i]} \in \mathbb{R}^{d_i}$ and $\mathbf{X}^{[i]} := [\mathbf{x}^{[i,1]}, \mathbf{x}^{[i,2]}, \cdots, \mathbf{x}^{[i,n]}]^T$, where $1 \le j \le n$ and $1 \le i \le m$. Figure 1 illustrates the scenario considered along with notation mentioned above.

To facilitate the analysis, we make the following assumptions that are typically made in existing literatures, for example see [14], [25], [26].

Assumption 1. *$\mathcal{L}$ is strongly convex and $\mathcal{R}$ is convex.*

Remark 1. *Assumption 1 implies that*

$$\mathcal{L}(\mathbf{u}) \ge \mathcal{L}(\mathbf{v}) + \langle \nabla\mathcal{L}(\mathbf{v}), \mathbf{u} - \mathbf{v}\rangle + \frac{\mu_{\mathcal{L}}}{2}\|\mathbf{u} - \mathbf{v}\|^2, \tag{2}$$

$$\mathcal{R}(\mathbf{u}) \ge \mathcal{R}(\mathbf{v}) + \langle \mathbf{s}(\mathbf{v}), \mathbf{u} - \mathbf{v}\rangle, \tag{3}$$

where $\mathbf{s}(\mathbf{v}) \in \partial\mathcal{R}(\mathbf{v})$ and $\mu_{\mathcal{L}} > 0$ with $\partial\mathcal{R}(\mathbf{v})$ being the subgradient of $\mathcal{R}$ at $\mathbf{v}$.

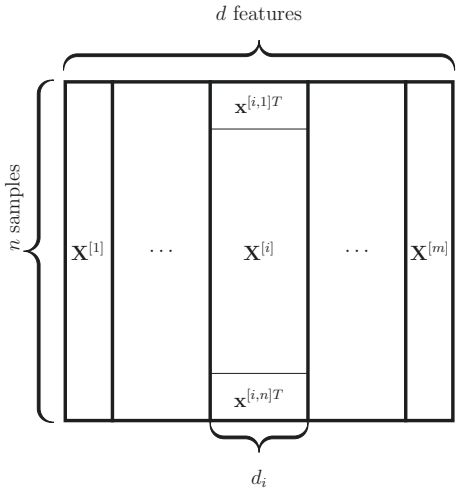

Fig. 1. Feature partitioned data matrix $\mathbf{X}$ for m machines with $\sum_{i=1}^m d_i = d$.

Assumption 2. *The loss function $\mathcal{L}$ is differentiable and there exists a positive semidefinite matrix $\mathbf{M}$ such that*

$$\mathcal{L}(\boldsymbol{\beta} + \mathbf{h}) \le \mathcal{L}(\boldsymbol{\beta}) + \langle \nabla\mathcal{L}(\boldsymbol{\beta}), \mathbf{h}\rangle + \frac{1}{2}\mathbf{h}^T\mathbf{M}\mathbf{h}. \tag{4}$$

Remark 2. *Let the largest eigenvalue of $\mathbf{M}$ be upper bounded by L. Then* (4) *implies*

$$\mathcal{L}(\boldsymbol{\beta} + \mathbf{h}) \le \mathcal{L}(\boldsymbol{\beta}) + \langle \nabla\mathcal{L}(\boldsymbol{\beta}), \mathbf{h}\rangle + \frac{L}{2}\|\mathbf{h}\|^2. \tag{5}$$

(4) *and* (5) *are equivalent since* (5) *can be written into form of* (4) *by taking $\mathbf{M} = L\mathbf{I}$. This assumption coupled with Assumption 1 implies that the derivative of the loss function is Lipschitz continuous [27, Theorem 2.1.5]:*

$$\|\nabla\mathcal{L}(\boldsymbol{\beta} + \mathbf{h}) - \nabla\mathcal{L}(\boldsymbol{\beta})\| \le L\|\mathbf{h}\|. \tag{6}$$

Remark 3. *Combining* (2) *and* (4)*, we have $\mu_{\mathcal{L}} \le L$.*

Assumption 3. *The penalty function is separable for each machine*

$$\mathcal{R}(\boldsymbol{\beta}) = \sum_{i=1}^m \mathcal{R}_i(\boldsymbol{\beta}^{[i]}), \tag{7}$$

where $\mathcal{R}_i$ is a $\mathbb{R}^{d_i} \to \mathbb{R}$ function.

Under these assumptions, in [14], the authors propose a distributed coordinate descent algorithm to solve problem (1). For reader's convinence, we list the algorithm proposed in [14] as Algorithm 1 below with modified notations. In this algorithm, we use $\boldsymbol{\beta}_k$ to denote the parameter at kth iteration, β_k^l to denote the lth element in vector $\boldsymbol{\beta}_k$, $\nabla\mathcal{L}(\boldsymbol{\beta}_k)_l$ to denote the lth element of the gradient and $\mathbf{M}_{ll}$ to denote the lth element of the diagonal of $\mathbf{M}$.

This algorithm depends on the lth element of the gradient $\nabla\mathcal{L}(\boldsymbol{\beta}_k)$. However, it should be noted that the gradient at each iteration cannot be computed locally by each machine. For example, if the loss function is the residual sum of squares,

then the gradient at each machine actually involves data at all other machines. In particular, in this case, the loss function is

$$\mathcal{L}(\boldsymbol{\beta}) = \frac{1}{2}\left\|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\right\|^2 = \frac{1}{2}\left\|\mathbf{y} - \sum_{i=1}^{m}\mathbf{X}^{[i]}\boldsymbol{\beta}^{[i]}\right\|^2. \quad (8)$$

For machine i^*, its local gradient vector is

$$\nabla\mathcal{L}(\boldsymbol{\beta}^{[i^*]}) = -\mathbf{X}^{[i^*]T}\left(\mathbf{y} - \sum_{i=1}^{m}\mathbf{X}^{[i]}\boldsymbol{\beta}^{[i]}\right). \quad (9)$$

In (9), the gradient vector for machine i^* is related to not only its local dataset $\mathbf{X}^{[i^*]}$, but also datasets at all other machines. If at each iteration, we update $\boldsymbol{\beta}_k$, then the communication cost is huge since we need to transfer almost the whole dataset at each iteration to compute $\nabla\mathcal{L}(\boldsymbol{\beta}^{[i^*]})$. In particular, each machine needs to send $\mathbf{X}^{[i]}$ and $\boldsymbol{\beta}^{[i]}$ so that each machine can compute $\nabla\mathcal{L}(\boldsymbol{\beta}^{[i]})$. Another alternative is to update $\sum_{i=1}^{m}\mathbf{X}^{[i]}\boldsymbol{\beta}^{[i]}$ as in [14]. At each iteration, machine i updates $\mathbf{X}^{[i]}\boldsymbol{\beta}^{[i]}$ instead of $\boldsymbol{\beta}^{[i]}$. Even if we update $\sum_{i=1}^{m}\mathbf{X}^{[i]}\boldsymbol{\beta}^{[i]}$, the amount of data transmitted at each iteration by each machine is n since $\mathbf{X}^{[i]}\boldsymbol{\beta}^{[i]}$ is a vector of length n. Hence, *the communication cost is too high if we calculate the exact gradient vector at each iteration*. Motivated by this observation, we focus on designing distributed coordinate gradient descent algorithms with low communication overhead.

Algorithm 1 Distributed Coordinate Descent of [14]

1: Input: Step-size parameter $\gamma > 0$, and τ that specifies the number of coordinates to optimize at each iteration
2: $k = 0$
3: **while** a stopping condition is not satisfied **do**
4: **for** each machine i in parallel **do**
5: Pick a random set of coordinates $\hat{S}_i$ of $\mathbf{X}^{[i]}$ with cardinality $|\hat{S}_i| = \tau$
6: **for** each feature index l in $\hat{S}_i$ **do**
7: $\beta_{k+1}^l = \beta_k^l + \text{argmin}_\delta[\nabla\mathcal{L}(\boldsymbol{\beta}_k)_l\delta + \frac{\mathbf{M}_{ll}\gamma}{2}\delta^2 + \mathcal{R}_l(\boldsymbol{\beta}_k^l + \delta)]$
8: **end for**
9: **end for**
10: $k = k + 1$
11: **end while**

To reduce the communication costs associated with the distributed coordinate descent algorithms, we design Algorithm 2. In Algorithm 2, we try to reduce the communication cost by only calculating the exact gradient sporadicly (we will discuss when to calculate the exact update in the sequel). In the iterations when exact gradients are not computed, we will use the most recently calculated gradient to compute the next update. As the result, instead of using the exact gradients $\nabla\mathcal{L}(\boldsymbol{\beta}_k)$ at each iteration, we will use an approximation of the gradient $\nabla\mathcal{L}(\boldsymbol{\beta}_k) + \mathbf{e}_k$, where $\mathbf{e}_k$ is the approximation error vector, to compute the next update. Our main idea is to carefully select the time instants at which exact gradient are computed so that the approximation error vectors caused by the sporadicly calculated gradients are well controlled and the algorithm still converges.

Algorithm 2 Distributed Coordinate Descent Algorithm with Inexact Update

1: input an initial point $\boldsymbol{\beta}_0$ and a nonnegative sequence $\{\epsilon_k\}$.
2: $k = 0$.
3: **while** a stopping condition is not satisfied **do**
4: **if** $k == 0$ or $\|\boldsymbol{\beta}_k - \boldsymbol{\beta}_{k-1}\| > \epsilon_k$ **then** ▷ need communication, amount of m
5: compute exact value of $\nabla\mathcal{L}(\boldsymbol{\beta}_k)$ ▷ need communication
6: **for** each machine i in parallel **do**
7: $\mathbf{h}^{[i]*} = \text{argmin}_{\mathbf{h}^{[i]}\in\mathbb{R}^{d_i}}\langle\nabla\mathcal{L}(\boldsymbol{\beta}_k^{[i]}), \mathbf{h}^{[i]}\rangle + \frac{L}{2}\|\mathbf{h}^{[i]}\|^2 + \mathcal{R}_i(\boldsymbol{\beta}_k^{[i]} + \mathbf{h}^{[i]})$. ▷ update using exact gradient
8: $\boldsymbol{\beta}_{k+1}^{[i]} = \boldsymbol{\beta}_k^{[i]} + \mathbf{h}^{[i]*}$
9: **end for**
10: $k = k + 1$
11: **else**
12: $\boldsymbol{\beta}_{fixed} = \boldsymbol{\beta}_{k-1}$ ▷ $\nabla\mathcal{L}(\boldsymbol{\beta}_{k-1})$ is known, no communication
13: **while** stopping condition is not satisfied and $\|\boldsymbol{\beta}_k - \boldsymbol{\beta}_{fixed}\| \le \epsilon_k$ **do**
14: **for** each machine i in parallel **do**
15: $\mathbf{h}^{[i]*} = \text{argmin}_{\mathbf{h}^{[i]}\in\mathbb{R}^{d_i}}\langle\nabla\mathcal{L}(\boldsymbol{\beta}_{fixed}^{[i]}), \mathbf{h}^{[i]}\rangle + \frac{L}{2}\|\mathbf{h}^{[i]}\|^2 + \mathcal{R}_i(\boldsymbol{\beta}_k^{[i]} + \mathbf{h}^{[i]})$. ▷ update using inexact gradient
16: $\boldsymbol{\beta}_{k+1}^{[i]} = \boldsymbol{\beta}_k^{[i]} + \mathbf{h}^{[i]*}$
17: **end for**
18: $k = k + 1$
19: **end while**
20: **end if**
21: **end while**

Now, we provide more details about Algorithm 2. Lines 3-21 are the main body of the algorithm and can be split into two parts. Lines 4-10 form the exact update part and Lines 11-20 form the inexact update part. The inputs of the algorithm are the initial starting point $\boldsymbol{\beta}_0$ and a nonnegative sequence $\{\epsilon_k\}$ that will be used as thresholds to determine whether we should perform exact update or inexact update at iteration k.

Line 4 specifies the conditions when we will enter the exact update part. In particular, for the first iteration, i.e. $k = 0$, we perform exact update. For iterations $k \ge 1$, we perform exact updates if the distance between the current parameter vector and the previous parameter vector is large (exceeds the threshold ϵ_k). The main intuition is that a large distance between the current parameter vector and the previous parameter vector implies that the objective function is changing fast in the neighborhood of the current parameter vector, and hence we should calculate the exact gradient. Notice in Line 4 of Algorithm 2, we need to compute $\|\boldsymbol{\beta}_k - \boldsymbol{\beta}_{k-1}\|$ to judge whether it exceeds the threshold ϵ_k at iteration k. To compute this, each machine i can compute and transmit a scalar $\|\boldsymbol{\beta}_k^{[i]} - \boldsymbol{\beta}_{k-1}^{[i]}\|$ and we have $\|\boldsymbol{\beta}_k - \boldsymbol{\beta}_{k-1}\|^2 = \sum_{i=1}^{m}\|\boldsymbol{\beta}_k^{[i]} - \boldsymbol{\beta}_{k-1}^{[i]}\|^2$. The

amount of communication for each machine is 1 for the step. Furthermore, we should note that the algorithm requires to memorize the $\boldsymbol{\beta}_{k-1} \in \mathbb{R}^d$ or $\boldsymbol{\beta}_{k-1}^{[i]} \in \mathbb{R}^{d_i}$ for each machine i to accomplish this step. Line 5 updates the exact gradient vector which requires communication whose amount of data transferred is n from the example in (8) and (9). Line 6-10 perform the classic exact update. For an exact iteration

$$\begin{aligned}
& \boldsymbol{\beta}_k \\
\overset{(a)}{=} \; & \operatorname*{argmin}_{\boldsymbol{\beta}\in\mathbb{R}^d} \langle \nabla\mathcal{L}(\boldsymbol{\beta}), \boldsymbol{\beta} - \boldsymbol{\beta}_{k-1}\rangle + \frac{L}{2}\|\boldsymbol{\beta} - \boldsymbol{\beta}_{k-1}\|^2 + \mathcal{R}(\boldsymbol{\beta}) \\
= \; & \operatorname*{argmin}_{\boldsymbol{\beta}\in\mathbb{R}^d} \frac{1}{L}\mathcal{R}(\boldsymbol{\beta}) + \frac{1}{2}\left\|\boldsymbol{\beta} - \left(\boldsymbol{\beta}_{k-1} - \frac{1}{L}\nabla\mathcal{L}(\boldsymbol{\beta}_{k-1})\right)\right\|^2 \\
\overset{(b)}{=} \; & prox_{\frac{1}{L}\mathcal{R}}\left(\boldsymbol{\beta}_{k-1} - \frac{1}{L}\nabla\mathcal{L}(\boldsymbol{\beta}_{k-1})\right),
\end{aligned}$$

where (a) is based on $\sum_{i=1}^m \langle \nabla\mathcal{L}(\boldsymbol{\beta}^{[i]}), \boldsymbol{\beta}^{[i]} - \boldsymbol{\beta}_{k-1}^{[i]}\rangle = \langle \nabla\mathcal{L}(\boldsymbol{\beta}), \boldsymbol{\beta} - \boldsymbol{\beta}_{k-1}\rangle$ and Assumption 3, and in (b) we use the definition of proximity operator

$$prox_f(\mathbf{v}) = \operatorname*{argmin}_{\mathbf{w}} f(\mathbf{w}) + \frac{1}{2}\|\mathbf{w} - \mathbf{v}\|^2.$$

As illustrated in Line 11, for $k \geq 1$, if $\|\boldsymbol{\beta}_k - \boldsymbol{\beta}_{k-1}\| \leq \epsilon_k$, then we enter the inexact update part. The main idea is that a small distance between the current parameter vector and the previous parameter vector implies that the value of the objective function does not change dramatically in the neighborhood of the current parameter vector, and hence we can use previously calculated gradient to compute the next update. In Line 12, we take the $\boldsymbol{\beta}_{fixed} = \boldsymbol{\beta}_{k-1}$ to utilize the condition in Line 11 that $\boldsymbol{\beta}_k$ is very close to $\boldsymbol{\beta}_{k-1}$. Combining the condition for the while loop in Line 13 and the fact that this is the first step to enter the while loop, we know that the $(k-1)$th iteration is an exact iteration, which means $\nabla\mathcal{L}(\boldsymbol{\beta}_{fixed})$ is already computed in the exact update part. It should be noticed that $\nabla\mathcal{L}(\boldsymbol{\beta}_{fixed})$ should be stored in machines. To accomplish this, each machine i can store $\nabla\mathcal{L}(\boldsymbol{\beta}_{k-1}^{[i]}) \in \mathbb{R}^{d_i}$. In Lines 13-18, we continuously use $\nabla\mathcal{L}(\boldsymbol{\beta}_{fixed})$ as the approximation gradient vector instead of computing the exact one until it can no longer be used. Lines 14-17 perform update as in Lines 6-10. The only difference here is that the approximated gradient vector $\nabla\mathcal{L}(\boldsymbol{\beta}_{fixed})$ is used. In these inexact iterations, we have

$$\begin{aligned}
\boldsymbol{\beta}_k &= prox_{\frac{1}{L}\mathcal{R}}\left(\boldsymbol{\beta}_{k-1} - \frac{1}{L}\nabla\mathcal{L}(\boldsymbol{\beta}_{fixed})\right) \\
&= prox_{\frac{1}{L}\mathcal{R}}\left(\boldsymbol{\beta}_{k-1} - \frac{1}{L}\nabla\mathcal{L}(\boldsymbol{\beta}_{k-1}) + \mathbf{e}_{k-1}\right),
\end{aligned}$$

in which $\mathbf{e}_{k-1} = \frac{1}{L}\left(\nabla\mathcal{L}(\boldsymbol{\beta}_{k-1}) - \nabla\mathcal{L}(\boldsymbol{\beta}_{fixed})\right)$. So we actually perform a proximal gradient method at each step, the only difference is that we introduce error in inexact iterations.

We will discuss in detail how to select the sequence $\{\epsilon_k\}$ in Section III. Informally, to guarantee the convergence of the algorithm, we choose $\{\epsilon_k\}$ to be summable. Furthermore, we should select the sequence $\{\|\epsilon_k\|\}$ to be diminishing as well. This selection ensures that, as we get closer to the optimal solution, the error introduced by the inexact update also gets smaller. Thus, this selection will prevent injecting a large error into the gradient vector when we get to the close neighborhood of the optimal solution, as a large gradient error will lead to a large deviation in the result that would make it difficult for the algorithm to converge.

III. Performance Analysis

In this section, we analyze the convergence rate and the communication cost of Algorithm 2. First, we give an explicit value sequence of $\{\epsilon_k\}$ and provide two propositions about the convergence rate and an upper bound on the number of iterations at which exact update is carried out. In the following, we analyze the basic version of our algorithm, namely at each step we do not additionally use conditions in Proposition 1 to check whether the current inexact update is good enough or not.

Proposition 1. *Let D be an upper bound of $\|\boldsymbol{\beta}_0 - \boldsymbol{\beta}^*\|$. If we set $\epsilon_k = \frac{\mu_0}{L}\left(1 - \frac{\mu_\mathcal{L} - \mu_0}{L}\right)^k D$, where $0 < \mu_0 < \mu_\mathcal{L}$, then for Algorithm 2 we have*

$$\|\boldsymbol{\beta}_k - \boldsymbol{\beta}^*\| \leq \left(1 - \frac{\mu_\mathcal{L} - \mu_0}{L}\right)^k D. \tag{10}$$

Proposition 1 provides a sequence of explicit values for $\{\epsilon_k\}$ that can achieve an exponential convergence rate even with inexact gradient updates, although the convergence speed is slower than that of the algorithm with exact updates. The convergence rate also depends on the estimated upper bound D and we want D to be as close to $\|\boldsymbol{\beta}_0 - \boldsymbol{\beta}^*\|$ as possible.

Proposition 2. *Let N be the number of iterations until convergence for Algorithm 2. The number of exact update iterations (i.e., the iterations when large communication overhead is needed) is at most*

$$N\frac{\log\left(1 - \frac{\mu_\mathcal{L} - \mu_0}{L}\right)}{\log\left(1 - \frac{\mu_\mathcal{L}}{L}\right)}.$$

Proposition 2 provides an upperbound on the number of iterations where exact updates are carried out. If inexact updates are not introduced, the total number of iterations k to achieve $\|\boldsymbol{\beta}_k - \boldsymbol{\beta}^*\| \leq \epsilon$ is $\frac{\log\epsilon - \log\|\boldsymbol{\beta}_0 - \boldsymbol{\beta}^*\|}{\log\left(1 - \frac{\mu_\mathcal{L}}{L}\right)}$. If inexact updates are introduced, the corresponding number of iterations is $\frac{\log\epsilon - \log D}{\log\left(1 - \frac{\mu_\mathcal{L} - \mu_0}{L}\right)}$. Notice that if D is a tight bound, then Proposition 2 indicates that the communication iterations are approximately equal. However, Proposition 3 analyzes the worst case performance of the proposed algorithm. In practice, we find that by introducing inexact updates is more communication efficient on average, which is illustrated in the next section.

It is worth to discuss how to select D in practice when the upper bound of $\|\boldsymbol{\beta}_0 - \boldsymbol{\beta}^*\|$ is unknown. We notice that $\|\boldsymbol{\beta}_k - \boldsymbol{\beta}^*\|$ converges to 0 as $k \to \infty$ without inexactness. If we cannot find a upper bound D, then a lower bound of $\|\boldsymbol{\beta}_0 - \boldsymbol{\beta}^*\|$ is also fine with this method. Since we can use

exact iterations for first i rounds to make $\|\beta_i - \beta^*\| \leq D$, we can treat β_i as the initial point in Algorithm 2.

IV. Numerical Examples

In this section, we provide numerical examples to illustrate our results using synthesized data and real data for Lasso, where $\mathcal{L}(\beta) = \frac{1}{2}\|\mathbf{y} - \mathbf{X}\beta\|^2$ and $\mathcal{R}(\beta) = \lambda\|\beta\|_1$ with $\mathbf{y} \in \mathbb{R}^n$, $\mathbf{X} \in \mathbb{R}^{n\times d}$ and $\beta \in \mathbb{R}^d$.

We compare our algorithm with the case in which the exact update is calculated at every iteration, which is equivalent to set $\epsilon_k = 0$ for all k in Algorithm 2.

For synthesized data simulation, we use the error sequence $\{\epsilon_k\}$ stated in Proposition 1. The data matrix $\mathbf{X}$ is generated randomly with fixed known maximal and minimal eigenvalues of $\mathbf{X}^T\mathbf{X}$. The vector $\mathbf{y}$ is generated by linear regression. For Lasso, (4) holds for their loss functions with $\mathbf{M} = \mathbf{X}^T\mathbf{X}$, so $L = 2\lambda_{\max}(\mathbf{X}^T\mathbf{X})$ and $\mu_{\mathcal{L}} = 2\lambda_{\min}(\mathbf{X}^T\mathbf{X})$. We set $L = 20000$ and $\mu_{\mathcal{L}} = 2$, and we generate $\mathbf{X}$ with corresponding fixed $\lambda_{\max}(\mathbf{X}^T\mathbf{X})$ and $\lambda_{\min}(\mathbf{X}^T\mathbf{X})$. For synthesized data, we set the number of samples $n = 2000$ and the number of features d from 10 to 400 with increments being 10.

For simulations with real datasets, since the eigenvalues of $\mathbf{X}^T\mathbf{X}$ are unknown and hard to compute for real datasets, the error sequence in Proposition 1 can no longer be used. For practical concerns, we simply set $\epsilon_k = (1-\alpha)^\tau D$, where τ is the number of inexact iterations so far. Here we set L to be an easily computable value $\frac{L}{2} = \|\mathbf{X}\|_F^2 \geq \|\mathbf{X}^T\mathbf{X}\|_F \geq \lambda_{\max}(\mathbf{X}^T\mathbf{X})$, where $\|\cdot\|_F$ denotes the Frobenius norm of a matrix. We run simulations for different values of α to show the performance of this error sequence in practice.

We run simulations in pseudo-distributed environment. In our simulations, we care about the inexact communication iterations, which do not depend on the number of machines and the ways to partition the features of the dataset. Noticing that no matter how many machines we have, the inexact communication iterations are the same; therefore we run Algorithm 2 in one machine to simulate the case in a distributed cluster of machines.

A. Synthesized data

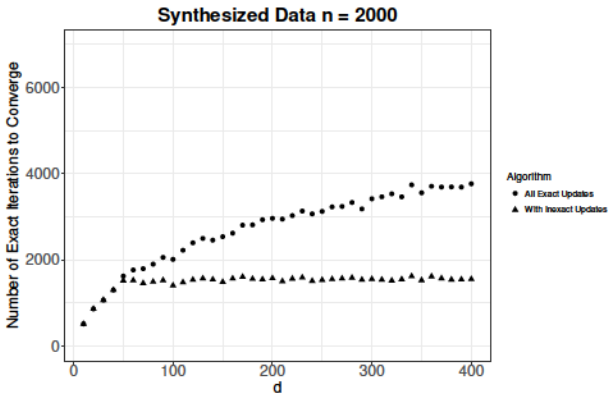

Fig. 2. Number of exact communication iterations for Lasso.

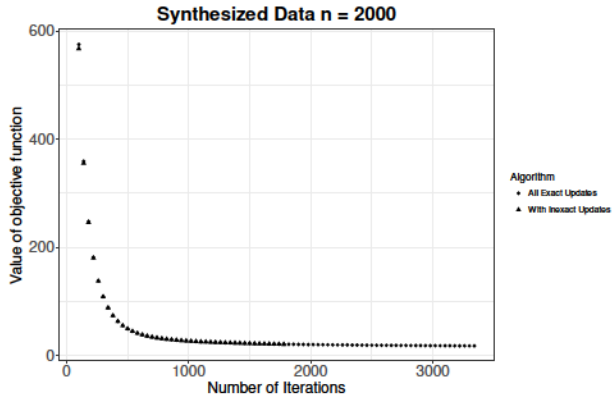

Fig. 3. Values of the objective function versus the number of iterations for Lasso $d = 400$.

In our simulation, we compare Algorithm 2 with that of Algorithm 2 without inexactness (Algorithm 2 without Lines 11 - 18). Comparing Figure 2 and Figure 3, we find that the values of the objective function after minimization by two methods are very close. However, the proposed scheme reduces nearly half of the total communications.

B. Real data

Next, we test our algorithm on real dataset: Communities and Crime Unnormalized Data Set [28]. The dataset contains statistics related to crime and social economics from 1990 US Census, 1990 US LEMAS survey and 1995 FBI UCR. The features contain statistics such as community population, per capita income, police operating budget and violent crime rate etc.

In this example, we study the murder rate (per 100K people) and try to build a sparse linear regression model between the murder rate and all other numeric variable in the dataset. Here we have $n = 2215$ and $d = 103$ (we omit the data features with missing data).

Let $f(\hat{\beta})$ denote the value of objective function at termination. The results computed by the algorithm without inexactness (with $\{\epsilon_k = 0\}$) is listed in Table I. The results with inexactness are summarized in Table II.

TABLE I
Basic algorithm for crime data

Iterations	$f(\hat{\beta})$
15300	681.4654

TABLE II
Algorithm with inexact iterations for crime data

D	α	Total iterations	Inexact iterations	$f(\hat{\beta})$
1	10^{-1}	15299	95	681.4653
1	10^{-2}	15287	992	681.4674
1	10^{-3}	13735	9965	681.6371

Table II shows that our scheme does not work well for $\alpha = 10^{-1}$ and works well for $\alpha = 10^{-3}$. Although this paper

does not provide theoretical results for the cases where $\mu_{\mathcal{L}}$ is unknown or $\mu_{\mathcal{L}} = 0$, Table II shows that our scheme works for these case in practice. It would be interesting to extend to these two cases in the theoretical analysis.

V. Conclusion

In this paper, we have proposed a general communication efficient scheme for the distributed learning problem of feature partitioned data. We have proposed an explicit algorithm using inexact updates. We have shown analytical results of the proposed algorithm which reveal its desirable properties under mild assumptions. We have also shown that the worst case performance of the proposed algorithm is comparable to that of the existing algorithm with exact updates. We have verified the efficiency of our algorithm using both synthesized data and real datasets.

VI. Acknowledgment

The work of J. Geng was supported by the National Natural Science Foundation of China under grant 61601144 and by the Fundamental Research Funds for the Central Universities under grant AUGA5710013915. The work of W. Xu was suported by National Institute of Health under grant 1R01EB020665 and by National Science Foundation under grant DMS-1418737. The work of L. Lai was supported by National Science Foundation under grants CNS-1660128 and CCF-1717943.

References

[1] Martin Jaggi, Virginia Smith, Martin Takáč, Jonathan Terhorst, Sanjay Krishnan, Thomas Hofmann, and Michael I. Jordan, "Communication-efficient distributed dual coordinate ascent," in *NIPS'14 Proceedings of the 27th International Conference on Neural Information Processing Systems*, Montreal, Canada, Dec. 2014, pp. 3068–3076.

[2] Chenxin Ma, Virginia Smith, , Martin Jaggi, Michael I. Jordan, Peter Richtárik, and Martin Takáč, "Adding vs. averaging in distributed primal-dual optimization," in *Proceedings of the 32th International Conference on Machine Learning*, Lille, France, Dec. 2015, vol. 37, pp. 3068–3076.

[3] Ohad Shamir, Nathan Srebro, and Tong Zhang, "Communication-efficient distributed optimization using an approximate newton-type method," in *Proceedings of the 31 st International Conference on Machine Learning*, Beijing, China, July 2014.

[4] Yuchen Zhang and Lin Xiao, "Communication-efficient distributed optimization of self-concordant empirical loss," *arXiv:1510.00263v1*, Jan. 2015.

[5] Rie Johnson and Tong Zhang, "Accelerating stochastic gradient descent using predictive variance reduction," in *NIPS'13 Proceedings of the 26th International Conference on Neural Information Processing Systems*, Stateline, NV, Dec. 2013.

[6] Keqin Liu and Qing Zhao, "Distributed learning in multi-armed bandit with multiple players," *IEEE Trans. Signal Processing*, vol. 58, pp. 5667 – 5681, Dec. 2010.

[7] Symeon Chouvardas, Konstantinos Slavakis, and Sergios Theodoridis, "Adaptive robust distributed learning in diffusion sensor networks," *IEEE Trans. Signal Processing*, vol. 59, pp. 4692 – 4707, July 2011.

[8] Paolo Di Lorenzo and Ali H. Sayed, "Sparse distributed learning based on diffusion adaptation," *IEEE Trans. Signal Processing*, vol. 61, pp. 1419–1433, Mar. 2013.

[9] Symeon Chouvardas, Konstantinos Slavakis, Yannis Kopsinis, and Sergios Theodoridis, "A sparsity promoting adaptive algorithm for distributed learning," *IEEE Trans. Signal Processing*, vol. 60, pp. 5412–5425, Oct. 2012.

[10] Jianshu Chen and Ali H. Sayed, "Diffusion adaptation strategies for distributed optimization and learning over networks," *IEEE Trans. Signal Processing*, vol. 60, pp. 4289–4305, Aug. 2012.

[11] Haipeng Zheng, Sanjeev R. Kulkarni, and H. Vincent Poor, "Attribute-distributed learning: Models, limits, and algorithmss," *IEEE Trans. Signal Processing*, vol. 59, pp. 386–398, Oct. 2011.

[12] Javier Matamoros, Sophie M. Fosson, Enrico Magli, and Carles Antón-Haro, "Distributed admm for in-network reconstruction of sparse signals with innovations," *IEEE Trans. on Signal and Information Processing over Networks*, vol. 1, pp. 225 – 234, Dec. 2015.

[13] Luca Canzian, Yu Zhang, and Mihaela van der Schaar, "Ensemble of distributed learners for online classification of dynamic data streams," *IEEE Trans. on Signal and Information Processing over Networks*, vol. 1, pp. 180 – 194, Sept. 2015.

[14] Peter Richtárik and Martin Takáč, "Distributed coordinate descent method for learning with big data," *Journal of Machine Learning Research*, Feb. 2016.

[15] Chenxin Ma and Martin Takáč, "Partitioning data on features or samples in communication-efficient distributed optimization?," *arXiv:1510.06688v1*, Oct. 2015.

[16] Feng Niu, Benjamin Recht, Christopher Ré, and Stephen J. Wright, "Hogwild!: A lock-free approach to parallelizing stochastic gradient descent," in *NIPS'11 Proceedings of the 24th International Conference on Neural Information Processing Systems*, Granada, Spain, Dec. 2011.

[17] Peter Richtárik and Martin Takáč, "Parallel coordinate descent methods for big data optimization," *Mathematical Programming*, vol. 156, pp. 433484, Dec. 2016.

[18] Robert Tibshirani, "Regression shrinkage and selection via the lasso," *Journal of the Royal Statistical Society*, vol. 7, no. 1, pp. 267–288, 1996.

[19] Arthur E. Hoerl and Robert W. Kennard, "Ridge regression: Biased estimation for nonorthogonal problems," *Technometrics*, vol. 12, no. 1, pp. 55–67, Feb. 1970.

[20] D. R. Cox, "The regression analysis of binary sequences," *Journal of the Royal Statistical Society*, vol. 20, no. 2, pp. 215–242, 1958.

[21] Su-In Lee, Honglak Lee, Pieter Abbeel, and Andrew Y. Ng, "Efficient l_1 regularized logistic regression," in *The Twenty-First National Conference on Artificial Intelligence and the Eighteenth Innovative Applications of Artificial Intelligence Conference*, Boston, MA, Jan. 2006.

[22] Ji Zhu, Saharon Rosset, Trevor Hastie, and Rob Tibshirani, "1-norm support vector machines," in *NIPS'03 Proceedings of the 16th International Conference on Neural Information Processing Systems*, 2003.

[23] Sahand N. Negahban, Pradeep Ravikumar, Martin J. Wainwright, and Bin Yu, "A unified framework for high-dimensional analysis of M-estimators with decomposable regularizers," *Statistical Science*, vol. 27, no. 4, pp. 538–557, Nov. 2012.

[24] Yurii Nesterov, "Gradient methods for minimizing composite objective function," *Center for Operations Research and Econometrics Discussion Paper*, Sep. 2007.

[25] Zhaosong Lu and Lin Xiao, "On the complexity analysis of randomized block-coordinate descent methods," *arXiv:1305.4723v1*, 2013.

[26] Peter Richtárik and Martin Takáč, "Iteration complexity of randomized block-coordinate descent methods for minimizing a composite function," *Mathematical Programming*, Dec. 2012.

[27] Yurii Nesterov, *Introductory lectures on convex optimization: a basic course*, Springer Science+Business Media, LLC, New York, 2004.

[28] "Communities and crime data set," https://archive.ics.uci.edu/ml/datasets/Communities+and+Crime+Unnormalized.