A Distributed and Scalable Machine Learning Approach for Big Data

Hongliang Guo, and **Jie Zhang**
School of Computer Science and Engineering
Nanyang Technological University, Singapore
guohl@ntu.edu.sg zhangj@ntu.edu.sg

Abstract

With the rapid development of data sensing and collection technologies, we can easily obtain large volumes of data (big data). However, big data poses huge challenges to many popular machine learning techniques which take all the data at the same time for processing. To address the big data related challenges, we first partition the data along its feature space, and apply the parallel block coordinate descent algorithm for distributed computation; then, we continue to partition the data along the sample space, and propose a novel matrix decomposition and combination approach for distributed processing. The final results from all the entities are guaranteed to be the same as the centralized solution. Extensive experiments performed on Hadoop confirm that our proposed approach is superior in terms of both testing errors and convergence rate (computation time) over the canonical distributed machine learning techniques that deal with big data.

1 Introduction

Machine Learning (ML) has been a hot research area in the past few decades. Traditionally, a bottleneck that prohibits the development of more intelligent systems is the limited amount of data. Without enough data, computers can hardly make intelligent decisions as human beings do. However, this situation has changed with the rapid growth of data sensing and collection technologies in recent years. This has opened a wide range of new opportunities, both because the best algorithm for a given problem may change dramatically as more data becomes available [Mayer-Schnberger, 2013], and because such a wealth of data promises solutions to problems that could not be previously approached.

Besides opportunities, challenges are also coming along. Big data not only changes the tools that can be used for intelligent decision making, but also changes our entire way of thinking about knowledge extraction and interpretation. Many of the traditional machine learning algorithms require the loading of the entire data set into one computer; this step becomes impossible when data sets are incredibly large. Therefore, big data is driving the need for scalable, parallel and online intelligent algorithms.

In this paper, we propose a novel matrix decomposition and combination approach combined with the parallel block coordinate descent (PBCD) algorithm to make the computation effort distributed for several of the most popular machine learning algorithms, e.g., support vector machine (SVM) [Burges, 1998], and logistic regression [Lee *et al.*, 2006]. After applying our proposed approach to these machine learning algorithms, they can be readily used to solve problems involving big data. To be more specific, the key challenge in big data processing is that both feature space and sample space are very large. In our proposed approach, we first decompose the feature space of the data and use the PBCD algorithm to iteratively compute the optimal solution. However, since the data sample space is also large, we still cannot process the whole data at the same time. In this case, we continue to separate the data along the sample space, and apply our novel matrix decomposition and combination approach to generate the final results.

Experimental results on large data sets verify that our approach achieves better convergence rate than four other canonical distributed or big data machine learning algorithms, on both regression and classification tasks.

2 Related Work

There are two popular groups of distributed machine learning algorithms. The first group decompose/distribute the first-order derivative information, i.e., gradient, so as to make the canonical machine learning algorithms distributed [Zinkevich *et al.*, 2010; Mateos *et al.*, 2010; Gopal and Yang, 2013; Forero *et al.*, 2010; Zhu *et al.*, 2008]. When faced with various coupled constraints, they usually apply distributed optimization techniques like dual decomposition [Sontag *et al.*, 2010], alternating direction method of multipliers (ADMM) [Boyd *et al.*, 2011] to unify the constraints into objectives and perform decomposition thereafter. For example, Mateos *et al.* [2010] propose a distributed algorithm for sparse linear regression. They first formulate the LASSO [Tibshirani, 1994] linear regression as a convex optimization problem, and then apply ADMM to make the computation process distributed.

The second group investigate the special (usually sparse) structure of the problem, and propose efficient distributed algorithms accordingly [Defazio *et al.*, 2014; El Ghaoui *et al.*, 2011; Wang *et al.*, 2013; Mairal, 2013; Yin *et al.*, 2013;

Mairal, 2014]. For example, Defazio *et al.* [2014] propose a permutable incremental method for big data processing. They target at a specific problem which is to minimize the summation of squares. The specific problem nature (finite summation of squares) allows to take advantage of its strong convexity. EL Ghaoui *et al.* [2011] make use of the sparse connection information within high-dimensional data, and revise the principal component analysis (PCA) and graphical computation methods to their respective sparse version. The algorithm has been applied to understand text information.

In summary, making use of the first order information, i.e., the gradient information, can be general in terms of application domains. However, since the higher-order information of a problem is completely ignored, this kind of algorithms usually do not guarantee a high convergence rate. In the worst case, some of the algorithms (when using dual decomposition) only guarantee convergence in infinity. On the other hand, making use of the specific structure of the problem (i.e., sparsity) usually has very good convergence performance. However, since they are exploiting the specific nature of the problem, the generality of the algorithm is limited. Our proposed approach is trying to bridge the gap between the two groups. On one hand, our approach makes use of both the first order information and the second order information which is the inverse of the Hessian matrix, thus performs faster than the algorithms in the first group, as will be demonstrated in Section 5. On the other hand, our approach does not exploit the specific structure (i.e., sparsity) of specific problems, hence has a wider application compared to the algorithms in the second group.

As will be shown in the next section, we are, in essence, proposing an efficient matrix manipulation approach for large scale matrix inversion. Hence, we provide the brief literature review on large scale matrix inversion. The bottleneck of many large scale problems attributes to the efficient computation of the inverse of large scale matrices [Bai *et al.*, 1996]. Csanky [1976] summarizes the canonical parallel computation methods for the large scale matrix inversion problem and gives the theoretical lower bounds of the computation complexities of such methods. Our proposed matrix inversion approach actually reaches that lower bounds.

3 Our Distributed ML Approach

In this section, we propose a distributed and efficient approach to solve the following mathematical problem:

$$\begin{aligned} \underset{\boldsymbol{\beta}}{\text{minimize}} \quad & \sum_{i=1}^{N} {\boldsymbol{\xi_i}}^2 \\ \text{subject to} \quad & \mathbf{X}\boldsymbol{\beta} - \mathbf{Y} = \boldsymbol{\xi} \end{aligned} \tag{1}$$

where $\mathbf{X} \in \mathbb{R}^{m \times n}$ and $\mathbf{Y} \in \mathbb{R}^m$. The most challenging part of this problem is that both m and n are large (i.e., in the scale of millions), which cannot be solved by default centralized solutions. In fact, for most supervised machine learning algorithms, when the data size (m) and data dimension (n) are very large, the key challenging computation step is also how to calculate $\boldsymbol{\beta}$ out of the problem formulated in Eq. 1. For example, in SVM, the key step is to solve $\mathbf{K}\boldsymbol{\beta} = \mathbf{Y}$, where $\mathbf{K}$ is the kernel matrix; in linear discriminative analysis (LDA) and logistic regression, $\mathbf{H}\boldsymbol{\beta} = \mathbf{Y}$, where $\mathbf{H}$ is the Hessian of the respective objective functions; and for linear regression, $\mathbf{X}\boldsymbol{\beta} = \mathbf{Y}$, where $\mathbf{X} \in \mathbb{R}^{m \times n}$ is the data matrix.

Since the challenging problem in the aforementioned machine learning algorithms share the similar nature, we only illustrate how to solve the general problem in Eq. 1 in a distributed and efficient manner.

We first clarify the mainly used symbols here: 1) m: the size of data (the number of data samples); 2) n: the number of features of the raw data; 3) N_1: the number of computation entities for column (feature space) separation; 4) N_2: the number of computation entities for row (data size) separation; 5) m_i: the number of data samples in computer i; 6) n_j: the number of features in computer j; 7) $\mathbf{X} \in \mathbb{R}^{m \times n}$: the (transformed) data matrix with m data samples and n features; 8) $\boldsymbol{Y} \in \mathbb{R}^m$: The label information of the data; 9) $\boldsymbol{\beta} \in \mathbb{R}^n$: the parameters to be learned by the machine learning algorithm.

3.1 Divide and Conquer Strategy

We consider the case where both the data size (m) and feature space (n) are very large. In this case, we perform the divide-and-conquer strategy as demonstrated in the parallel block coordinate descent (PBCD) algorithm [Bradley *et al.*, 2011]. More specifically, we first partition the data matrix $\mathbf{X}$ along its column, and represent it as $\mathbf{X} = (\mathbf{X}_1, \mathbf{X}_2, \ldots, \mathbf{X}_{N_1})$ where $\mathbf{X}_i \in \mathbb{R}^{m \times n_i}$; we then partition $\boldsymbol{\beta}$ along its row accordingly, and have $\boldsymbol{\beta} = \left({\boldsymbol{\beta_1}}^\top, {\boldsymbol{\beta_2}}^\top, \ldots, {\boldsymbol{\beta_{N_1}}}^\top\right)^\top$ where $\boldsymbol{\beta_i} \in \mathbb{R}^{n_i}$.

The idea of the parallel block coordinate descent method is that at each iteration, we update the block parameter ($\boldsymbol{\beta_i}$) by a little bit assuming that all the other block parameters are not changing. The update rule is as follows:

$$\boldsymbol{\beta}_i^{k+1} = (1-\alpha)\boldsymbol{\beta}_i^k + \alpha({\mathbf{X}_i}^\top \mathbf{X}_i)^{-1}(\mathbf{Y} - \sum_{j=1, j\neq i}^{N_1} \mathbf{X}_j \boldsymbol{\beta}_j^k) \tag{2}$$

where we superscript with k to denote the value of the scripted quantity at iteration k.

The convergence proof and convergence rate of PBCD have been provided in [Tseng, 2001]. The key extensive computation part of Eq. 2 is $({\mathbf{X}_i}^\top \mathbf{X}_i)^{-1}$. Since $\mathbf{X}_i$ is only large in one dimension (m), we will use our proposed matrix decomposition and combination approach to solve it.

3.2 Matrix Decomposition and Combination

Here we introduce a novel approach to compute the inverse of ${\mathbf{X}_i}^\top \mathbf{X}_i$ in an efficient and distributed manner. Note that $\mathbf{X}_i \in \mathbb{R}^{m \times n_i}$, and m is very large, therefore it is impossible to store the matrix $\mathbf{X}_i$ in one computer's memory. In this case, $({\mathbf{X}_i}^\top \mathbf{X}_i)^{-1}$ is only a conceptual representation. However, note that n_i is within a reasonable range, because we already divide the data along its feature space in Section 3.1.

Suppose we have N_2 computers, each of which stores a subset of the data samples. Then, we partition $\mathbf{X}_i$ into N_2 parts, and each part contains m_i samples. For each of the m_i samples, the computer contains its whole feature space of $\mathbf{X}_i$.

Now, in essence, we have partitioned the data into N_2 parts, so the matrix $\mathbf{X}_i$ can be represented as:

$$\mathbf{X}_i = \left(\mathbf{X}_i^1, \mathbf{X}_i^2, \ldots, \ \mathbf{X}_i^{N_2}\right)^\top. \tag{3}$$

$(\mathbf{X}_i{}^\top \mathbf{X}_i)^{-1}$ can then be represented as:

$$(\mathbf{X}_i{}^\top \mathbf{X}_i)^{-1} = (\sum\nolimits_{j=1}^{N_2} (\mathbf{X}_i^j)^\top \mathbf{X}_i^j)^{-1} \tag{4}$$

So, the key computation step is to obtain the result of $(\sum_{j=1}^{N_2} (\mathbf{X}_i^j)^\top \mathbf{X}_i^j)^{-1}$ in a distributed and efficient manner, where $\mathbf{X}_i^j$ is an $m_j \times n_i$ matrix. In order to refer to this term easily, we define it as $\mathbf{M}$. Therefore, we have:

$$\mathbf{M} = \sum\nolimits_{j=1}^{N_2} (\mathbf{X}_i^j)^\top \mathbf{X}_i^j. \tag{5}$$

Since n_i is not very large, each computer is able to perform the singular value decomposition (SVD) [Zhang, 2004] over its data matrix ($\mathbf{X}_i^j$), which can then be represented as:

$$\mathbf{X}_i^j = \mathbf{U}_i^j \mathbf{\Omega}_i^j (\mathbf{V}_i^j)^\top \tag{6}$$

where $\mathbf{U}_i^j$ is an $m_j \times m_j$ orthogonal matrix, $\boldsymbol{\Omega_i^j}$ is an $m_j \times n_i$ rectangle diagonal matrix with non-negative real numbers on the main diagonal, and $(\mathbf{V}_i^j)^\top$ is an $n_j \times n_j$ orthogonal matrix. Here, an orthogonal matrix is a square matrix with real entries whose columns and rows are orthogonal unit vectors.

Substituting Eq. (6) into Eq. (5), we can get:

$$\begin{aligned} \mathbf{M} &= \sum\nolimits_{j=1}^{N_2} (\mathbf{X}_i^j)^\top \mathbf{X}_i^j \\ &= \sum\nolimits_{j=1}^{N_2} (\mathbf{U}_i^j \mathbf{\Omega}_i^j (\mathbf{V}_i^j)^\top)^\top \mathbf{U}_i^j \mathbf{\Omega}_i^j (\mathbf{V}_i^j)^\top \\ &= \sum\nolimits_{j=1}^{N_2} \mathbf{V}_i^j (\mathbf{\Omega}_i^j)^\top (\mathbf{U}_i^j)^\top \mathbf{U}_i^j \mathbf{\Omega}_i^j (\mathbf{V}_i^j)^\top. \end{aligned} \tag{7}$$

Since $\mathbf{U}_i^j$ is an orthogonal matrix, and we define $\mathbf{\Lambda}_i^j = (\mathbf{\Omega}_i^j)^\top \mathbf{\Omega}_i^j$, then, we can continue from Eq. (7) and get:

$$\mathbf{M} = \sum\nolimits_{j=1}^{N_2} \mathbf{V}_i^j \boldsymbol{\Lambda_i^j} (\mathbf{V}_i^j)^\top \tag{8}$$

where $\mathbf{V}_i^j$ is an $n_j \times n_j$ orthogonal matrix, and $\boldsymbol{\Lambda_i^j}$ is an $n_j \times n_j$ diagonal matrix with non-negative diagonal elements.

Now, the inverse of each to-be-summed element in Eq. (8) is very easy to compute. Considering the fact that $\mathbf{V}_i^j$ is an orthogonal matrix, we have

$$\left(\mathbf{V}_i^j \mathbf{\Lambda}_i^j (\mathbf{V}_i^j)^\top\right)^{-1} = \mathbf{V}_i^j (\mathbf{\Lambda}_i^j)^{-1} (\mathbf{V}_i^j)^\top. \tag{9}$$

The computation process of $(\mathbf{\Lambda}_i^j)^{-1}$ is just to compute the inverse of each diagonal element of matrix $\mathbf{\Lambda}_i^j$. However, computing the inverse of $\mathbf{M}$ which is the summation of those terms, is not a trivial task. It will be addressed next.

Matrix Merging Process

Ideally, if we can represent $\mathbf{M}$ in such a form:

$$\mathbf{M} = \mathbf{V} \mathbf{\Lambda} \mathbf{V}^\top \tag{10}$$

where $\mathbf{V}$ is an orthogonal matrix, and $\mathbf{\Lambda}$ is a diagonal matrix with positive diagonal values, then, it is easy to compute the inverse of $\mathbf{M}$. Before transforming Eq. (8) into the form of Eq. (10) directly, we start with an easier step by merging two matrices. The problem is to represent $\mathbf{V}_i^{j_1} \mathbf{\Lambda}_i^{j_1} (\mathbf{V}_i^{j_1})^\top + \mathbf{V}_i^{j_2} \mathbf{\Lambda}_i^{j_2} (\mathbf{V}_i^{j_2})^\top$ into the form of Eq. (10). Let us define:

$$\mathbf{M}_{ij} = \mathbf{V}_i^{j_1} \mathbf{\Lambda}_i^{j_1} (\mathbf{V}_i^{j_1})^\top + \mathbf{V}_i^{j_2} \mathbf{\Lambda}_i^{j_2} (\mathbf{V}_i^{j_2})^\top. \tag{11}$$

Since both of the two terms in Eq. (11) are symmetric positive definite matrices, according to Theorem 1 below, there exists an invertible matrix $\mathbf{P}$, such that $\mathbf{P}^\top \mathbf{V}_i^{j_1} \mathbf{\Lambda}_i^{j_1} (\mathbf{V}_i^{j_1})^\top \mathbf{P} = \mathbf{I}$, and $\mathbf{P}^\top \mathbf{V}_i^{j_2} \mathbf{\Lambda}_i^{j_2} (\mathbf{V}_i^{j_2})^\top \mathbf{P} = \text{diag}(\lambda_1, \lambda_2, \ldots, \lambda_{n_i})$. This conclusion means that $\mathbf{P}$ satisfies the following equation:

$$\mathbf{P}^\top \mathbf{M}_{ij} \mathbf{P} = \mathbf{I} + \text{diag}(\lambda_1, \lambda_2, \ldots, \lambda_{n_i}). \tag{12}$$

After simple deduction, we can get that:

$$\begin{aligned} \mathbf{M}_{ij} &= (\mathbf{P}^{-1})^\top (\mathbf{I} + \text{diag}(\lambda_1, \lambda_2, \ldots, \lambda_{n_i})) \mathbf{P}^{-1} \\ &= (\mathbf{P}^{-1})^\top \text{diag}(\lambda_1 + 1, \lambda_2 + 1, \cdots, \lambda_{n_i} + 1) \mathbf{P}^{-1}. \end{aligned} \tag{13}$$

Continuing to apply the singular value decomposition to the second half of Eq. (13), which is $(\mathbf{P}^{-1})^\top \text{diag}(\lambda_1 + 1, \lambda_2 + 1, \cdots, \lambda_{n_i} + 1) \mathbf{P}^{-1}$, we can represent $\mathbf{M}_{ij}$ as: $\mathbf{M}_{ij} = \mathbf{V}_{ij} \mathbf{\Lambda}_{ij} \mathbf{V}_{ij}^\top$.

Next, we first propose and prove a theorem for the existence of such a $\boldsymbol{P}$ matrix, and then deliver the method for computing $\boldsymbol{P}^{-1}$.

Theorem 1. *$\forall\ \mathbf{A} \in \mathbf{S}_{++}^n$, $\mathbf{B} \in \mathbf{S}^n$, $\exists \mathbf{P}$, s.t.: (1) $\exists \mathbf{P}^{-1}$; (2) $\mathbf{P}^\top \mathbf{A} \mathbf{P} = \mathbf{I}$; (3) $\mathbf{P}^\top \mathbf{B} \mathbf{P} = \text{diag}(\lambda_1, \lambda_2, \ldots, \lambda_n)$.*

Here, $\mathbf{S}_{++}^n$ refers to symmetric positive definite $n \times n$ matrices and $\mathbf{S}^n$ refers to symmetric $n \times n$ matrices. Suppose that $\mathbf{A}$ and $\mathbf{B}$ are both $n \times n$ matrices, $\mathbf{A}$ is positive definite, and $\mathbf{B}$ is symmetric. Then, there exists an invertible matrix $\mathbf{P}$, which makes $\mathbf{A}$ congruent to the identity matrix [Zhang, 2004] and in the meantime, makes $\mathbf{B}$ diagonal. Here, the symbol $\mathbf{I}$ denotes the identity matrix.

Proof. Since $\mathbf{A}$ is positive definite, then $\mathbf{A}$ is congruent to the identity matrix. $\mathbf{A} \in \mathbf{S}_{++}^n \Rightarrow \exists \mathbf{P}_1$, s.t. $\mathbf{P}_1{}^\top \mathbf{A} \mathbf{P}_1 = \mathbf{I}$. Since $\mathbf{P}_1{}^\top \mathbf{B} \mathbf{P}_1$ is symmetric, then we can orthogonally diagonalize it to a diagonal matrix. $\mathbf{B} \in \mathbf{S} \Rightarrow \mathbf{P}_1{}^\top \mathbf{B} \mathbf{P}_1 \in \mathbf{S} \Rightarrow \exists \mathbf{P}_2$ s.t. (1) $\mathbf{P}_2{}^\top \mathbf{P}_2 = \mathbf{P}_2 \mathbf{P}_2{}^\top = \mathbf{I}$; (2) $\mathbf{P}_2{}^\top \mathbf{P}_1{}^\top \mathbf{B} \mathbf{P}_1 P_2 = \text{diag}(\lambda_1, \lambda_2, \ldots, \lambda_n)$. Defining $\mathbf{P} = \mathbf{P}_1 P_2$, we then have $\mathbf{P}^\top \mathbf{A} \mathbf{P} = \mathbf{I}$, in the meanwhile, we have $\mathbf{P}^\top \mathbf{B} \mathbf{P} = \text{diag}(\lambda_1, \lambda_2, \ldots, \lambda_n)$. □

Now, we have finished the merging process of the matrices. The merged matrix ($\mathbf{M}_{ij}$) is also represented in the same form as that before the merge. We can continue with the merging process when $\mathbf{M}_{ij}$ 'meets' another matrix. Thus, the key computation step now is to find matrix $\mathbf{P}^{-1}$.

The Computation Process of $\mathbf{P}^{-1}$

In Eq. (11), $\mathbf{\Lambda}_i^{j_1}$ is a diagonal matrix with all positive diagonal elements. So, suppose $\mathbf{\Lambda}_i^{j_1} = \text{diag}(\lambda_1, \lambda_2, \ldots, \lambda_L)$ and define $\left(\mathbf{\Lambda}_i^{j_1}\right)^{-\frac{1}{2}} = \text{diag}(\lambda_1^{-\frac{1}{2}}, \lambda_2^{-\frac{1}{2}}, \ldots, \lambda_L^{-\frac{1}{2}})$, $\mathbf{P}_1 = \left(\mathbf{\Lambda}_i^{j_1}\right)^{-\frac{1}{2}} \mathbf{V}_i^\top$, then:

$$\mathbf{P}_1^{-1} = \mathbf{V}_i \left(\mathbf{\Lambda}_i^{j_1}\right)^{\frac{1}{2}}. \tag{14}$$

Multiplying Eq. (11) by $\mathbf{P}_1$ from the left and by $\mathbf{P}_1^\top$ from the right, then we can obtain:

$$\begin{aligned}\mathbf{P}_1\mathbf{M}_{ij}\mathbf{P}_1^\top &= \mathbf{P}_1\mathbf{V}_i\mathbf{\Lambda}_i^{j_1}\mathbf{V}_i^\top\mathbf{P}_1^\top + \mathbf{P}_1\mathbf{V}_j\mathbf{\Lambda}_i^{j_2}\mathbf{V}_j^\top\mathbf{P}_1^\top \\ &= \mathbf{I} + \left(\mathbf{\Lambda}_i^{j_1}\right)^{-\frac{1}{2}}\mathbf{V}_i^\top\mathbf{V}_j\mathbf{\Lambda}_i^{j_2}\mathbf{V}_j^\top\mathbf{V}_i\left(\left(\mathbf{\Lambda}_i^{j_1}\right)^{-\frac{1}{2}}\right)^\top. \quad (15)\end{aligned}$$

Here, we define $\left(\mathbf{\Lambda}_i^{j_2}\right)^{\frac{1}{2}} = \text{diag}(\lambda_1^{\frac{1}{2}}, \lambda_2^{\frac{1}{2}}, \ldots, \lambda_L^{\frac{1}{2}})$, and then Eq. (15) can be further transformed to:

$$\begin{aligned}\mathbf{P}_1\mathbf{M}_{ij}\mathbf{P}_1^\top =& \mathbf{I} + \left(\mathbf{\Lambda}_i^{j_1}\right)^{-\frac{1}{2}}\mathbf{V}_i^\top\mathbf{V}_j\left(\mathbf{\Lambda}_i^{j_2}\right)^{\frac{1}{2}} \\ &\left(\left((\mathbf{\Lambda}_i^{j_2})^{\frac{1}{2}}\right)^\top\mathbf{V}_j^\top\mathbf{V}_i\left(\left(\mathbf{\Lambda}_i^{j_1}\right)^{-\frac{1}{2}}\right)^\top. \quad (16)\end{aligned}$$

Define $\mathbf{Q} = \left(\mathbf{\Lambda}_i^{j_1}\right)^{-\frac{1}{2}}\mathbf{V}_i^\top\mathbf{V}_j\left(\mathbf{\Lambda}_i^{j_2}\right)^{\frac{1}{2}}$, then Eq. (16) is simplified as

$$\mathbf{P}_1\mathbf{M}_{ij}\mathbf{P}_1^\top = \mathbf{I} + \mathbf{Q}\mathbf{Q}^\top. \quad (17)$$

Applying singular value decomposition on $\mathbf{Q}$, we can get:

$$\mathbf{Q} = \mathbf{U}_q\mathbf{\Lambda}_q\mathbf{V}_q^\top \quad (18)$$

where $\mathbf{U}_q$ and $\mathbf{V}_q$ are orthogonal matrices, and $\mathbf{\Lambda}_q$ is a diagonal matrix. Replacing $\mathbf{Q}$ in Eq. (17) by Eq. (18), we get:

$$\begin{aligned}\mathbf{P}_1\mathbf{M}_{ij}\mathbf{P}_1^\top &= \mathbf{I} + \mathbf{U}_q\mathbf{\Lambda}_q\mathbf{V}_q^\top(\mathbf{U}_q\mathbf{\Lambda}_q\mathbf{V}_q^\top)^\top \quad (19)\\ &= \mathbf{I} + \mathbf{U}_q\mathbf{\Lambda}_q\mathbf{V}_q^\top\mathbf{V}_q(\mathbf{\Lambda}_q)^\top\mathbf{U}_q^\top = \mathbf{I} + \mathbf{U}_q\mathbf{\Lambda}_q(\mathbf{\Lambda}_q)^\top\mathbf{U}_q^\top \\ &= \mathbf{U}_q\mathbf{U}_q^\top + \mathbf{U}_q\mathbf{\Lambda}_q(\mathbf{\Lambda}_q)^\top\mathbf{U}_q^\top = \mathbf{U}_q(\mathbf{I} + \mathbf{\Lambda}_q\mathbf{\Lambda}_q^\top)\mathbf{U}_q^\top\end{aligned}$$

Solving Eq. (19) for $\mathbf{M}_{ij}$ and replacing $\mathbf{P}_1^{-1}$ by Eq. (14),

$$\begin{aligned}\mathbf{M}_{ij} &= \mathbf{P}_1^{-1}\mathbf{U}_q(\mathbf{I} + \mathbf{\Lambda}_q\mathbf{\Lambda}_q^\top)\mathbf{U}_q^\top(\mathbf{P}_1^{-1})^\top \\ &= \mathbf{V}_i\left(\mathbf{\Lambda}_i^{j_1}\right)^{\frac{1}{2}}\mathbf{U}_q(\mathbf{I} + \mathbf{\Lambda}_q\mathbf{\Lambda}_q^\top)\mathbf{U}_q^\top\left(\mathbf{V}_i\left(\mathbf{\Lambda}_i^{j_1}\right)^{\frac{1}{2}}\right)^\top \quad (20)\end{aligned}$$

So, the inverse of the matrix $\mathbf{P}$ as described in Eq. (12) can be directly calculated out as:

$$\mathbf{P}^{-1} = \left(\mathbf{V}_i\left(\mathbf{\Lambda}_i^{j_1}\right)^{\frac{1}{2}}\mathbf{U}_q\right)^\top. \quad (21)$$

Transforming Merged Matrix into Standard Form

Now, we know $\mathbf{P}^{-1}$, and then the merged matrix can be represented as:

$$\mathbf{M}_{ij} = (\mathbf{P}^{-1})^\top(\mathbf{I} + \mathbf{\Lambda}_q\mathbf{\Lambda}_q^\top)\mathbf{P}^{-1}. \quad (22)$$

$\mathbf{M}_{ij}$ is an $n \times n$ matrix. Applying singular value decomposition over $\mathbf{M}_{ij}$, we can represent it as:

$$\mathbf{M}_{ij} = \mathbf{V}_{ij}\mathbf{\Lambda}_{ij}\mathbf{V}_{ij}^\top \quad (23)$$

where $\mathbf{V}_{ij}$ is an orthogonal matrix and $\mathbf{\Lambda}_{ij}$ is diagonal matrix with all positive diagonal elements.

So far, we are able to merge two matrices, each of which is of the form as represented in Eq. (10), and continue to represent the merged matrix in the form of Eq. (10). As the process goes on, we will in the end reach the final representation also in the form of Eq. (10).

4 Analysis of Computation Complexity

In this section, we only analyze in detail the computation complexity of our proposed matrix manipulation (decomposition and combination) approach, as for the PBCD part, it has been well analyzed in [Tseng, 2001]. The number of iterations needed for PBCD to converge is $O(N_1)$, where N_1 is the number of blocks. Within each iteration, the computation complexity mainly relies on how to compute the inverse of $\mathbf{X}_i^\top\mathbf{X}_i$. Thus, if the computation complexity of $\mathbf{X}_i^\top\mathbf{X}_i$ is C, then the overall computation complexity is $C \times O(N_1)$.

4.1 Analysis of Matrix Manipulation Approach

Before analyzing the matrix manipulation approach's computation complexity, we first lay down the evaluation metric.

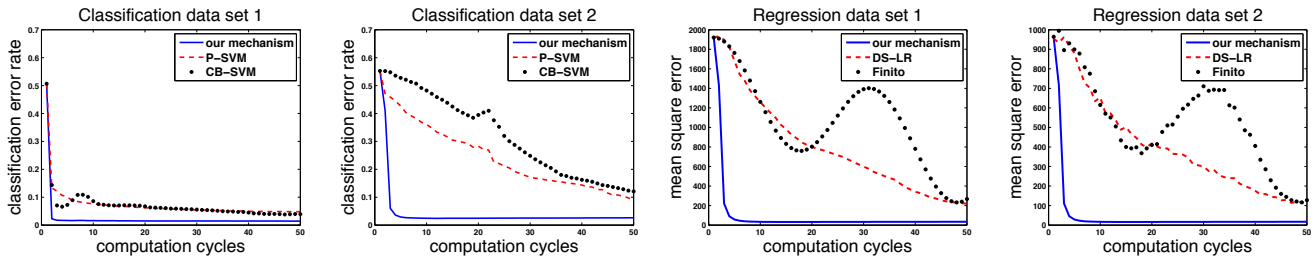
Flops and Computation Complexity on Basic Operations

The computation cost of an operation can often be expressed through the number of floating-point operations (flops). A flop is defined as an addition, subtraction, multiplication or division of two floating-point numbers [Boyd and Vandenberghe, 2004]. To evaluate the complexity of an approach, we count the total number of flops, express it as a function (usually a polynomial) of the dimensions of the matrices and vectors involved, and simplify the expression by ignoring all terms except the leading terms. So, we can express the computation complexity of an SVD operation over a matrix $\mathbf{A} \in \mathbf{R}^{m\times n}$ as $O(m^2n + n^3)$. The computation complexity of a matrix-matrix product $\boldsymbol{M} = \mathbf{AB}$, where $\mathbf{A} \in \mathbf{R}^{m\times n}$ and $\mathbf{B} \in \mathbf{R}^{n\times p}$, is $O(mnp)$.

Worst Case Complexity over a Single Computer

Since we are developing a distributed approach, computing the overall computation complexity is not meaningful. So, we here analyze the computation complexity over one single computer. Since the data size is not necessarily the same, and the computation process for one computer may end before the approach exits, we consider the computation complexity of the 'worst' computer. Here, the term 'worst computer' refers to the computer performing the most complex computation over the whole process.

Examining the approach flow process, we can see that we are required to perform SVD over matrices $\mathbf{X}_i$, which needs $O(m_i^2n_j + n_j^3)$ flops. Since we are considering the 'worst computation unit', the computation complexity can be expressed as $O(\max_i(m_i^2)\max_j n_j + \max_j(n_j^3))$. Then, the approach requires the computation of $\mathbf{U}_q$ and $\mathbf{P}^{-1}$. The computation of $\mathbf{U}_q$ requires three steps of $\max_j n_j \times \max_j n_j$ matrix-matrix multiplication, and one step of SVD over a $\max_j n_j \times \max_j n_j$ matrix, and the computation complexity of the matrix-matrix multiplication and SVD are both $O(\max_j(n_j^3))$. Thus, the computation complexity of calculating $\mathbf{U}_q$ is $O(\max_j(n_j^3))$. Likewise, the computation of $\mathbf{P}^{-1}$ requires two steps of $\max_j n_j \times \max_j n_j$ matrix-matrix multiplication, and thus the computation complexity is $O(\max_j(n_j^3))$. Multiplying the matrices together involves two steps of $\max_j n_j \times \max_j n_j$ matrix-matrix multiplication, and hence the computation complexity is also $O(\max_j(n_j^3))$. Performing SVD over the final $\max_j n_j \times \max_j n_j$ matrix requires $O(\max_j(n_j^3))$. Summing together,



(a) (b) (c) (d)

Figure 1: Error versus computation cycles: (a) classification error comparison for classification data set 1; (b) classification error comparison for classification data set 2; (c) mean square error comparison for regression data set 1; (d) mean square error comparison for regression data set 2.

and dropping the constant terms, we can conclude that the computation complexity is $O(\max_j (n_j^3))$. The above computation complexity is for one matrix merge operation. Now we need to count how many operations of matrix merge we actually need. We have N_2 computation units. After one round of two-computer merge, we are left with $\lceil \frac{N_2}{2} \rceil$ computers, where $\lceil x \rceil$ is the ceiling operator returning the smallest integer which is larger than x. So, after at most $\lceil \log_2 N_2 \rceil$ operations of matrix merge process, the approach ends.

In summary, the computation complexity for the worst computer is $O(\max_i (m_i^2) \max_j n_j + \max_j (n_j^3) + \lceil \log_2 N_2 \rceil \max_j (n_j^3))$. Dropping constant terms, we can reach $O(\max_i (m_i^2) \max_j (n_j) + \lceil \log_2 N_2 \rceil \max_j (n_j{}^3))$.

4.2 The Overall Computation Complexity

To sum up, the overall computation complexity of the proposed matrix manipulation approach together with PBCD, is $O(N_1(\max_i (m_i^2) \max_j (n_j) + \lceil \log_2 N_2 \rceil \max_j (n_j^3)))$. If we assume a uniform division among the data feature space among N_1, and also a uniform division among data samples over N_2 (which is suggested), the worst case computer's computation complexity would be $O(N_1(\lceil \frac{m}{N_2} \rceil^2 \lceil \frac{n}{N_1} \rceil + \lceil \log_2 N_2 \rceil \lceil \frac{n}{N_1} \rceil^3))$.

5 Experimental Evaluation

We evaluate the proposed approach through answering the following common questions: 1) How accurate is our distributed approach? Are we computing the same results as the centralized algorithm? 2) How fast is our approach? Is it (much) faster than canonical distributed machine learning methods that deal with big data?

5.1 Experimental Settings

In order to test the efficiency of the inherently distributed algorithm, we implement it in a distributed computation framework (Hadoop). Experimental settings are as follows: amazon instance category–r3 large; number of instances–5; operating system–Red Hat Enterprise Linux7.1(HVM)-64bit; RAM–15GB; Number of virtual CPUs for each instance–2

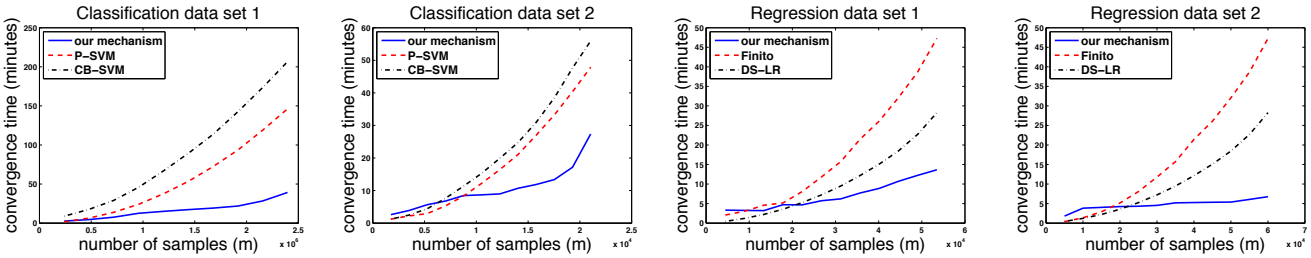
We test the performance of our algorithm and two benchmark algorithms on both regression and classification tasks. For classification, we select two data sets. One is the URL Reputation data set [Ma *et al.*, 2009] from the UCI machine learning repository. The total number of instances is 2,396,130 and the total number of attributes is 3,231,961. We name this data set as **classification data set 1**. The other one is the UJIIndoorLoc data set [Joaquin Torres-Sospedra, 2014]. The number of instances is 21,048, and the number of features is 529. We name the data set as **classification data set 2**. For regression, the first data set is the Relative location of CT slices on axial axis data set [Graf *et al.*, 2011] from the UCI machine learning repository. The total number of instances is 53,500 and the total number of attributes is 386. We name the data set as **regression data set 1**. The second data set we choose is the BlogFeedback data set [Buza, 2014]. The number of data samples is 60,021, and number of features is 281. We name it as **regression data set 2**.

For the classification task, we apply our approach on support vector machines, and compare it with two other state-of-the-art algorithms, namely Consensus-Based Distributed Support Vector Machines (CB-DSVM) [Forero *et al.*, 2010] and Parallelizing Support Vector Machines on Distributed Computers (P-SVM) [Zhu *et al.*, 2008]. For the regression problem, we apply our proposed distributed machine learning approach on linear regression, and compare the performance with two canonical distributed linear regression algorithms, namely distributed sparse linear regression (DS-LR) [Mateos *et al.*, 2010], and Finito [Defazio *et al.*, 2014]. It is worth noting here that Finito was not proposed specifically for making linear regression algorithms distributed. However, since Finito targets at distributing the optimization problem of minimizing summation of squares, it is straightforward to apply it to linear regression algorithms.

Since almost all canonical distributed machine learning algorithms, including our proposed approach, guarantee to converge to the optimum, i.e., the same result as the corresponding centralized machine learning algorithm, the key performance index is then the convergence rate, i.e., how fast these approaches can converge.

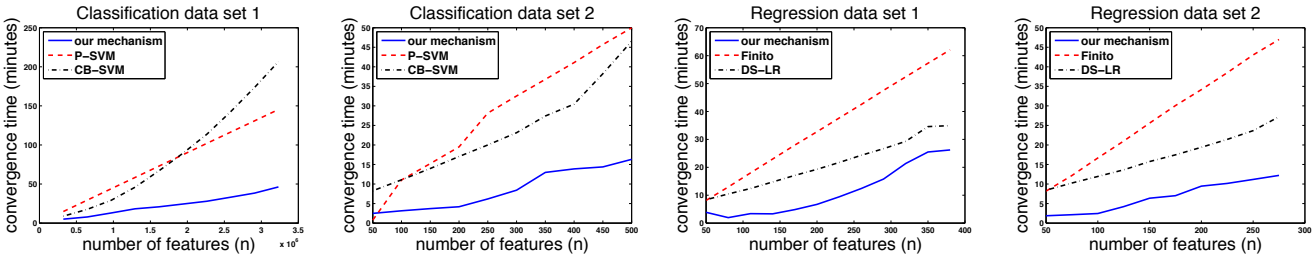
5.2 Experimental Results

We implement all the competing distributed machine learning algorithms in the same environmental context (i.e., the same number of machines, the same configuration of Hadoop and EC2, the same training and testing data set, the same cross



(a) (b) (c) (d)

Figure 2: Convergence time versus data sample (m): (a) classification convergence time comparison over classification data set 1; (b) classification convergence time comparison over classification data set 2; (c) regression convergence time comparison over regression data set 1; (d) regression convergence time comparison over regression data set 2.



(a) (b) (c) (d)

Figure 3: Convergence time versus data feature space (n): (a) classification convergence time comparison over classification data set 1; (b) classification convergence time comparison over classification data set 2; (c) regression convergence time comparison over regression data set 1; (d) regression convergence time comparison over regression data set 2.

validation scheme (10-fold cross validation)), and compare the testing error decreasing curves in Fig. 1a to Fig. 1d. In these four figures, we bring about a concept called cycle. We define a cycle as the time point, in which all the computers have finished one round of computation. The average computation time needed for one cycle for our approach as well as the other four algorithms (two for classification and two for regression) is given in Table 1. Note that for both classification and regression data sets, it is impossible to perform the centralized machine learning algorithm, thus there is no performance measurement being plotted in the figures.

Table 1: System execution time per cycle for the our approach, P-SVM, CB-SVM, Finito and DS-LR (in minutes)

Classification	Ours	P-SVM	CB-SVM
Data Set 1	6.46	18.56	25.85
Data Set 2	2.10	4.89	5.70
Regression	Ours	Finito	DS-LR
Data Set 1	0.53	4.55	2.43
Data Set 2	0.48	4.42	2.58

As shown in all the four figures in Fig. 1, our algorithm achieves better testing error performance in every cycle. It means that, given time constraint, if we have to stop the algorithm at an arbitrary time point, our approach would yield better performance. Here, we wish to point out (details are explained in [Defazio *et al.*, 2014]) that the regression error might increase as the number of iterations (cycles in our figure) increases, because the underlying random sampling approach might temporarily drive the parameters further away from the optimum. This phenomenon appears in Fig. 1c and Fig. 1d in our experimentation.

Now, we further evaluate how the computation time of our approach changes when the size of the data (m) and the number of features (n) vary. We argue that this evaluation process is quite important in that it verifies the scalability of our approach with respect to data size and feature space.

Fig. 2a to Fig. 2d show how the system convergence time changes as we change the number (m) of data samples for different data sets and different tasks, i.e., classification and regression. Likewise, Fig. 3a to Fig. 3d show how the system convergence time changes as we change the number (m) of data samples for different data sets and different tasks. Here, the system converges if the computed error at the current cycle is the same as the error at the previous cycle. The total time consumed (for the 'worst' computer) till the end of the current cycle is defined as system convergence time. We can see that our approach has better scalability performance when the number of data samples (m) and the number of features (n) increase.

6 Conclusion and Future Work

This paper proposes a distributed approach for large scale machine learning. We show the performance comparison of our approach when applied to SVM and linear regression with four other canonical distributed or big data machine learning algorithms. Experimental results verify that our approach

achieved better convergence rate. In the future, we will further investigate the PBCD part of our approach, since it still requires a lot of computation efforts.

Acknowledgments

This work is supported by the MOE AcRF Tier 1 funding (M4011261.020) awarded to Dr. Jie Zhang.

References

[Bai *et al.*, 1996] Zhaojun Bai, Gark Fahey, and Gene Golub. Some large-scale matrix computation problems. *Journal of Computational and Applied Mathematics*, 74(12):71 – 89, 1996.

[Boyd and Vandenberghe, 2004] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

[Boyd *et al.*, 2011] Stephen Boyd, Neal Parikh, Eric Chu, Borja Peleato, and Jonathan Eckstein. *Distributed Optimization and Statistical Learning via the Alternating Direction Method of Multipliers*. Now Publishers Inc., 2011.

[Bradley *et al.*, 2011] Joseph K. Bradley, Aapo Kyrola, Danny Bickson, and Carlos Guestrin. Parallel coordinate descent for l1-regularized loss minimization. In *Proc. of ICML*, 2011.

[Burges, 1998] Christopher J. C. Burges. A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, 2(2):121–167, 1998.

[Buza, 2014] Krisztian Buza. Feedback prediction for blogs. In Myra Spiliopoulou, Lars Schmidt-Thieme, and Ruth Janning, editors, *Data Analysis, Machine Learning and Knowledge Discovery*. Springer, 2014.

[Csanky, 1976] L. Csanky. Fast parallel matrix inversion algorithms. *SIAM Journal on Computing*, 5(4):618–623, 1976.

[Defazio *et al.*, 2014] Aaron J. Defazio, Tibrio S. Caetano, and Justin Domke. Finito: A faster, permutable incremental gradient method for big data problems. In *Proc. of ICML*, 2014.

[El Ghaoui *et al.*, 2011] L. El Ghaoui, G.-C. Li, V.-A. Duong, V. Pham, A. Srivastava, and K. Bhaduri. Sparse machine learning methods for understanding large text corpora. In *Proc. of the Conference on Intelligent Data Understanding*, 2011.

[Forero *et al.*, 2010] Pedro A. Forero, Alfonso Cano, and Georgios B. Giannakis. Consensus-based distributed support vector machines. *JMLR*, 11:1663–1707, 2010.

[Gopal and Yang, 2013] Siddharth Gopal and Yiming Yang. Distributed training of large-scale logistic models. In *Proc. of ICML*, 2013.

[Graf *et al.*, 2011] Franz Graf, Hans-Peter Kriegel, Matthias Schubert, Sebastian Polsterl, and Alexander Cavallaro. 2D image registration in CT images using radial image descriptors. In *Proc. of Int. Conf. on Medical Image Computing and Computer-Assisted Intervention*, 2011.

[Joaquin Torres-Sospedra, 2014] Adolfo Martinez-Uso Tomar J. Arnau Joan P. Avariento Mauri Benedito-Bordonau Joaquin Huerta Joaquin Torres-Sospedra, Raul Montoliu. UJIIndoorLoc: A new multi-building and multi-floor database for WLAN fingerprint-based indoor localization problems. In *Proc. of Int. Conf. on Indoor Positioning and Indoor Navigation*, 2014.

[Lee *et al.*, 2006] Su-In Lee, Honglak Lee, Pieter Abbeel, and Andrew Y. Ng. Efficient L1 regularized logistic regression. In *Proc. of AAAI*, 2006.

[Ma *et al.*, 2009] Justin Ma, Lawrence K. Saul, Stefan Savage, and Geoffrey M. Voelker. Identifying suspicious urls: An application of large-scale online learning. In *Proc. of ICML*, 2009.

[Mairal, 2013] Julien Mairal. Stochastic majorization-minimization algorithms for large-scale optimization. In *Proc. of NIPS 26*, 2013.

[Mairal, 2014] Julien Mairal. Incremental majorization-minimization optimization with application to large-scale machine learning. In *Proc. of ICML*, 2014.

[Mateos *et al.*, 2010] Gonzalo Mateos, Juan Andrs Bazerque, and Georgios B. Giannakis. Distributed sparse linear regression. *IEEE Transactions on Signal Processing*, 58(10):5262–5276, 2010.

[Mayer-Schnberger, 2013] Viktor Mayer-Schnberger. *Big Data: A Revolution That Will Transform How We Live, Work and Think. Viktor Mayer-Schnberger and Kenneth Cukier*. John Murray Publishers, 2013.

[Sontag *et al.*, 2010] David Sontag, Amir Globerson, and Tommi Jaakkola. Introduction to Dual Decomposition for Inference. *Optimization for Machine Learning*, 45(1):977–1014, 2010.

[Tibshirani, 1994] Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society, Series B*, 58:267–288, 1994.

[Tseng, 2001] P. Tseng. Convergence of a block coordinate descent method for nondifferentiable minimization. *Journal of Optimization Theory and Applications*, 109(3):475–494, 2001.

[Wang *et al.*, 2013] Huahua Wang, Arindam Banerjee, Cho-Jui Hsieh, Pradeep K Ravikumar, and Inderjit S Dhillon. Large scale distributed sparse precision estimation. In *Proc. of NIPS 26*, 2013.

[Yin *et al.*, 2013] Junming Yin, Qirong Ho, and Eric Xing. A scalable approach to probabilistic latent space inference of large-scale networks. In *Proc. of NIPS*, 2013.

[Zhang, 2004] Xian-Da Zhang. *Matrix Analysis and Applications*. Tsinghua University Press, 2004.

[Zhu *et al.*, 2008] Kaihua Zhu, Hao Wang, Hongjie Bai, Jian Li, Zhihuan Qiu, Hang Cui, and Edward Y. Chang. Parallelizing support vector machines on distributed computers. In *Proc. of NIPS 20*, 2008.

[Zinkevich *et al.*, 2010] Martin A. Zinkevich, Alex Smola, Markus Weimer, and Lihong Li. Parallelized stochastic gradient descent. In *Proc. of NIPS 23*, 2010.