MP^{2}SDA: Multi-Party Parallelized Sparse Discriminant Learning

JIANG BIAN, University of Central Florida
HAOYI XIONG, Baidu Inc.
YANJIE FU, Missouri University of Science and Technology
JUN HUAN, Baidu Inc.
ZHISHAN GUO, University of Central Florida

Sparse Discriminant Analysis (SDA) has been widely used to improve the performance of classical Fisher's Linear Discriminant Analysis in supervised metric learning, feature selection, and classification. With the increasing needs of distributed data collection, storage, and processing, enabling the Sparse Discriminant Learning to embrace the multi-party distributed computing environments becomes an emerging research topic. This article proposes a novel multi-party SDA algorithm, which can learn SDA models effectively without sharing any raw data and basic statistics among machines. The proposed algorithm (1) leverages the direct estimation of SDA to derive a distributed loss function for the discriminant learning, (2) parameterizes the distributed loss function with local/global estimates through bootstrapping, and (3) approximates a global estimation of linear discriminant projection vector by optimizing the "distributed bootstrapping loss function" with gossip-based stochastic gradient descent. Experimental results on both synthetic and real-world benchmark datasets show that our algorithm can compete with the aggregated SDA with similar performance, and significantly outperforms the most recent distributed SDA in terms of accuracy and F1-score.

CCS Concepts: • **Information systems** → **Spatial-temporal systems**; • **Computing methodologies** → **Distributed algorithms**;

Additional Key Words and Phrases: Sparse discriminant analysis, multi-party, distributed, parallelized

ACM Reference format:
Jiang Bian, Haoyi Xiong, Yanjie Fu, Jun Huan, and Zhishan Guo. 2020. MP^{2}SDA: Multi-Party Parallelized Sparse Discriminant Learning. *ACM Trans. Knowl. Discov. Data* 14, 3, Article 26 (March 2020), 22 pages.
https://doi.org/10.1145/3374919

1 INTRODUCTION

The Fisher's Linear Discriminant Analysis (LDA) [15] is a method to find the linear combination of features that separates two or multiple classes, where it can be used in supervised learning

The first and the second author contributed equally to the article.
Work supported by NSF RAISE CA-FW-HTF 1937833 and NSF CRII CSR 1755965.
Authors' addresses: J. Bian and Z. Guo, University of Central Florida, 4328 Scorpius St, Orlando, Florida, 32816; emails: bjbj11111@knights.ucf.edu, zsguo@ucf.edu; H. Xiong and J. Huan, Big Data Lab, Baidu Inc., 10 Shangdi 10th St, Haidian District, Beijing, 100094, China; email: haoyi.xiong.fr@ieee.org, jhuan@ittc.ku.edu; Y. Fu, Missouri University of Science and Technology, 500W 15th St, Rolla, Missouri, 65401; email: yanjie.fu@ucf.edu.

1556-4681/2020/03-ART26 $15.00
https://doi.org/10.1145/3374919

and feature selection. Considering a set of observations (training data), LDA can project the high-dimensional data points to low-dimensional space, and achieve optimal classification performances by minimizing the overlaps between different classes in the low-dimensional space. Further, when the number of measurements of each sample exceeds the number of samples in each class, where it is so-called the High-Dimensional and Low Sample Size (HDLSS) settings, to improve the performances of LDA, Sparse Discriminant Analysis (SDA) [8] has been proposed with sparsity pursuit. While a wide variety of methods [8, 12, 32, 35, 36, 45] have been proposed, Cai et al. [8] studied a direct estimator that can estimate SDA straightforwardly from labeled data with a provable guarantee in asymptotic property and classification accuracy.

As far as we know, multi-party computing [10, 17] becomes one of popular computing paradigm due to the increasing needs of distributed data collection, storage and processing, where it also benefits the privacy-preserved manner in different kinds of applications. In most multi-party computing platform, "no raw data sharing" is an important pre-condition, where a machine-learning model should be trained using all data stored in distributed machines (i.e., parties) without any cross-machine raw data sharing. Specifically, such multi-party distributed machine-learning algorithms can be accelerated by parallel computing and typically be divided into two types—*data-centric* and *model-centric* methods [6, 14, 26, 37, 42, 46, 47, 54]. On each machine, the data-centric algorithm first estimates the same set of parameters (of the model) using the local data, then aggregates the estimated parameters via model-averaging for global estimation. The model with aggregated parameters is considered as the trained model based on the overall data (from multiple parties) and before aggregated these parameters can be estimated through parallel computing structure in an easy way. Meanwhile, model-centric algorithms require multiple machines to share the same loss function with "updatable parameters," and allow each machine to update the parameters in the loss function using the local data so as to minimize the loss. Based on this characteristic, model-centric algorithm commonly updates the parameters sequentially so that the additional time consumption in updating is sometimes a tough nut for specific applications. Even so, compared with the data-centric, the model-centric methods usually can achieve better performances, as it minimizes the risk of the model [26, 39, 47]. To advance the distributed performance of classical SDA, recently, Tian and Gu et al. [40] proposed a data-centric SDA algorithm, which leverages the advantage of parallel computing. Although it is intuitive that the model-centric counterpart for SDA could receive better performance, few work has been carried out due to the challenge in terms of efficiency (i.e., the time consumption in sequential updating) through parallel computing.

To fill the gap, we are motivated to design a novel **model-centric** SDA learning algorithm for multi-party parallelized discriminant learning. In this article, we propose multi-party parallelized SDA (namely MP^2SDA) that enables the direct estimation of SDA [8] to embrace the multi-party parallel computing environment for sparse discriminant learning. Not only MP^2SDA can achieve a better performance provided by the model-centric algorithm, it also promotes the efficiency of the algorithm through parallel computing mechanism. Specifically, MP^2SDA first establishes multiple threads (sets of machines) for parallel computing. In each thread, MP^2SDA allocates the mean and covariance matrix estimation tasks to each machine and allows each machine to estimate its local mean vectors and covariance matrices based on the local data. Then, MP^2SDA estimates the global mean over all the data using the local means via the gossip-based stochastic gradient descent (SGD). Further, MP^2SDA proposes a distributed bootstrapping loss function and model the loss function using the global mean and local covariance matrices. Finally, a gossip-based parallel SGD algorithm is employed to minimize the distributed bootstrapping loss function and estimate the global discriminant projection vector. Compared with the approach in [8], which aggregates all data on a single machine to learn the model, MP^2SDA can effectively approximate to the optimal solution without sharing any raw data. Compared with [40], which aggregates the locally learned models through model-averaging and hard-thresholding, *MP^{2}SDA* models and minimizes a distributed loss function based on SDA, parameterized with global/local estimates, straightforwardly. Moreover, compared to normal single thread model-centric algorithm [48], *MP^{2}SDA* additionally processing parallel computing (multiple threads) when estimating the model parameters to improve the performance with fast convergence rate.

Contributions. We summarize the contribution of the proposed *MP^{2}SDA* algorithm in three aspects. **First**, we study and formulate the problem of sparse discriminant learning on the top of multi-party parallel computing platform, while assuming the raw data distributed on machines (parties) are not sharable and accelerating the training procedure through parallel computing. To the best of our knowledge, this is the first study on SDA, by addressing (1) *multi-party computing platform without sharing raw data*, (2) *model-centric*[1] *learning with a shared loss function*, and (3) *distributed optimization issues with parallel computing*. Note that *multi-party* parallel computing systems [7] usually leverage the secured communication and computation to keep the local data, on each party, private, while our work assumes the local raw data and basic statistics (on each machine) are not accessible by others. Thus we do not make the further assumption on cryptography issue. **Second**, to achieve the above goals, we design the *MP^{2}SDA* algorithm which leverages the direct estimation of SDA to derive a distributed loss function of the discriminant learning, parameterizes the distributed loss function with local/global estimates through bootstrapping, and approximates a global estimation of linear discriminant projection vector by optimizing the "distributed bootstrapping loss function" and further improving the estimation through parallel computing. **Finally**, we evaluate *MP^{2}SDA* on both pseudo-random simulation and real-world benchmark datasets for binary classification. The results show that *MP^{2}SDA* can compete with the centralized SDA with similar performances, outperform the single thread version and significantly outperform the state-of-the-art distributed SDA [40].

The rest of the article is organized as follows. In Section 2, we introduce the related works. In Section 3, we review the LDA model and the direct estimation of SDA by Cai and Liu, then we introduce the problem formulation of our work. In Section 4, we propose the framework of *MP^{2}SDA* and present the details of *MP^{2}SDA* algorithm. In Section 5, we evaluate the proposed algorithms on synthetic datasets and benchmark datasets. In addition, we compare *MP^{2}SDA* with baseline centralized algorithms. In Section 6, we present a discussion and raise up some open issues. Finally, we conclude the article in Section 7. We summarize the notations used in the following sections in Table 1.

2 RELATED WORKS

In this section, we first introduce the related works in the SDA, then present the most relevant work in distributed learning algorithms. Finally, we compare our work with existing studies, show the intuition of the proposed method.

Sparse Discriminant Analysis. SDA is frequently used to improve classical LDA through leveraging shrunken estimators [23, 43, 45, 52], such as Graphical Lasso [18]. Specifically, using a sparse regularized method to replace the precision matrix used in LDA [4, 49–51], the original ill-posed problem led by HDLSS settings can be well settled. With faster convergence rate, the sparse estimators often perform better than the sample estimators [9] when approximate the inverse of the population covariance matrix. Typically, the sparse LDA with the asymptotic properties can be closer to the optimal LDA than the other traditional sample estimators.

Distributed Learning Algorithm. With increasing the volume of "big data," mining/training such tremendous data models with a single machine can be very slow [54]. Not only that,

[1]Transfer from traditional data-centric paradigm to model centric paradigm.

Table 1. Summary of Notations

Symbol	Definition	Symbol	Definition
N	the total number of training samples	ε	the tuning parameter of Direct SDA
n	the number of training samples on each machine	$(\cdot)_k$	the k^{th} element of the input vector
m	the number of machines	t	the index of iteration
S	the size of Leader set	$\bar{\mu}^j$, $\bar{\mu}_+^j$, $\bar{\mu}_-^j$	the sample means on the j^{th} machine
p	the number of dimensions of data sample	$\widehat{\Sigma}^j$	the covariance matrix estimated on the j^{th} machine
Z	the data for classification	$\widehat{\mu}$, $\widehat{\mu}_+$, $\widehat{\mu}_-$	the global estimation of means
$\psi_F(\cdot)$	the linear discriminant rule of Fisher's LDA	$\widehat{\mu}^*$, $\widehat{\mu}_+^*$, $\widehat{\mu}_-^*$	the averaged global estimation of means
μ, μ_+, μ_-	the population means	T_+^j, T_-^j	sets of positive/negative samples on the j^{th} machine
Σ, Θ	the population covariance matrix and its inverse	$\widehat{\beta}^*$	the global estimation of Direct SDA
$\bar{\mu}$, $\bar{\mu}_+$, $\bar{\mu}_-$	the sample estimation of means	$\bar{\beta}^*$	the averaged global estimation of Direct SDA
$\bar{\Sigma}$, $\bar{\Theta}$	the sample covariance matrix and its inverse	$\widehat{\beta}_T^*$	the truncated global estimation of Direct SDA
β, β^*	linear discriminant projection vector (optimal)	$\widehat{D}^j$	de-sparsified precision matrix on the j^{th} machine

large-scale data problem is not just the size of the data to be mined but also its location and homogeneity. Data may be distributed across a set of locations or machines for several reasons. For example, several datasets concerning medical (personal) information (e.g., allergic history) might be owned by separate hospitals that have reasons for keeping the data private. To handle the above issues, various distributed/parallelized machine-learning algorithms were proposed, e.g., distributed decision tree [3], parallel support vector machine [53], and parallel rule induction [30, 31].

Comparison to the existing work. Since the proposed *MP*^{2}*SDA* algorithm is mainly compared with the centralized SDA algorithm and distributed SDA algorithm in this article, it is necessary to briefly introduce these two baseline algorithms. For the centralized SDA proposed by Cai et al. [8], this algorithm leverages a sparse LDA classifier by directly estimating the linear discriminant projection vector (aka, β). In contrast to the methods which are based on separate estimation of the precision matrix [34] and the difference of the mean vectors [38], the centralized SDA exploits the approximate sparsity of β and as a consequence allows cases where it can still perform well even when precision matrix and/or the difference of the mean vectors cannot be estimated consistently [16]. However, in the centralized SDA algorithm, all samples are located in an only single machine, where the bottleneck of the computation will turn up when the size of data grows and the algorithm is not easy to implement when the data is not sharable for some specific reasons. For the distributed SDA proposed by Tian et al. [40], this algorithm is an extended distributed version based on the centralized SDA algorithm. Instead of estimating a single SDA estimator, the distributed SDA algorithm considers to estimate the debiased local estimators [22] and aggregate to a sparsified one through averaging and hard-thresholding. With an approximate convergence rate as the centralized SDA, the distributed SDA algorithm is communication-efficient and can better adapt to the "big data" environment. Nonetheless, the distributed SDA still cannot address the "raw data sharing problem" when aggregates the local estimators. Combining the distributed techniques and the concept of non-sharable raw data, *MP*^{2}*SDA* is proposed in this article. Note that the proposed *MP*^{2}*SDA* can be categorized as a new designed model-centric distributed SDA algorithm, differd from the existing centralized SDA and data-centric distributed SDA. We will introduce the detailed designs and demonstrate the superiority of our algorithm.

3 BACKGROUNDS AND PROBLEM FORMULATION

In this section, we first present the model of Fisher's LDA. Then, we introduce the direct estimation of SDA proposed by Cai et al. [9]. Then we address the robust estimator under uncertain parameters using the "bootstrapping loss function" minimization. Finally, we formulate the research problem of this article.

3.1 Sparse Linear Discriminant Analysis

Fisher's LDA Model: LDA, which leverages a linear combination of features that characterize or separate two or more classes of objects or events. LDA has been shown to perform well and enjoy certain optimality as the sample size tends to infinity while the dimension is fixed [8]. Given the LDA classifier $\psi_F(Z)$ based on the given p-dimensional data vector Z that is drawn from one of two distributions $\mathcal{N}(\mu_+, \Sigma)$ and $\mathcal{N}(\mu_-, \Sigma)$ with equal prior probabilities, the binary classification problem can be solved by

$$\psi_F(Z) = sign\left\{(Z-\mu)^T\Theta(\mu_+ - \mu_-)\right\}, \tag{1}$$

where $\mu = (\mu_+ + \mu_-)/2$; $\Theta = \Sigma^{-1}$ is the inverse covariance matrix; μ_+ and μ_- are the mean vectors of the positive samples and negative samples, respectively; and $\psi_F(Z)$ classifies Z into positive class if and only if $\psi_F(Z) = 1$. In practice, μ_+, μ_- and Θ are unknown, we need to estimate μ_+, μ_- and Θ from observations. Specifically, we assume the data Z is randomly drawn from $\mathcal{N}(\mu_+, \Sigma)$ and $\mathcal{N}(\mu_-, \Sigma)$ with equal priors.

A simple way to estimate μ_+, μ_- and Θ is to use their sample estimator: $\bar{\mu}_+, \bar{\mu}_-, \bar{\Theta} = \bar{\Sigma}^{-1}$, where $\bar{\Sigma}$ is pooled sample covariance matrix estimation [28] with respect to the two classes. Note that, under the HDLSS settings, $\bar{\Sigma}$ is often singular [27] and $\bar{\Sigma}^{-1}$ usually does not exist [34]. Thus, to train LDA, researchers [8, 35] proposed to estimate the linear discriminate projection vector $\beta = \Theta(\mu_+ - \mu_-)$, instead of estimating Θ and $\mu_+ - \mu_-$ separately.

Loss Function of Direct SDA (Sparse β) Estimation: Based on the Equation (1), Cai and Liu (2011) [8] proposed a direct estimation method for sparse LDA by estimating β through a constrained ℓ_1 minimization method:

$$\underset{\beta\in\mathfrak{R}^p}{\text{argmin}}\left\{|\beta|_1 \quad s.t.\ |\bar{\Sigma}\beta - (\bar{\mu}_+ - \bar{\mu}_-)|_\infty \le \varepsilon\right\}, \tag{2}$$

where ε is a tuning parameter.

3.2 Bootstrapping Loss Function Minimization

In this section, we introduce a robust estimator that can minimize the loss function with uncertain parameters. Given a loss function $\mathcal{L}(\omega|\theta)$, where θ is an unknown parameter following a known probabilistic distribution with density function $\mathcal{P}(\theta)$. To approximate the optimal ω^* that minimizes the loss under the uncertainty of θ, we need a solution to minimize the expectation of loss over θ

$$\omega^* = \underset{\omega}{\text{argmin}}\mathbb{E}_{\theta\sim\mathcal{P}}\left(\mathcal{L}(\omega|\theta)\right). \tag{3}$$

To simplify the computation, a bootstrapping solution is frequently used, where the algorithm first randomly draws $\theta_1, \theta_2, \ldots, \theta_m$ from the distribution with the density function $\mathcal{P}(\theta)$, and then approximates ω^* by minimizing the bootstrapping loss function

$$\widehat{\omega}_m = \underset{\omega}{\text{argmin}}\sum_{i=1}^{m}\left(\mathcal{L}(\omega|\theta_i)\right)/m. \tag{4}$$

As $\theta_1, \theta_2, \ldots, \theta_m$ are drawn from the distribution randomly, the sum of loss functions can approximate the expectation of loss function under Central-Limit Theorem [20] with large m, where $\lim_{m\to\infty}\widehat{\omega}_m = \omega^*$. Recent studies [24, 25] show that the bootstrapping loss function minimization can obtain a robust estimation of ω under the uncertainty of θ.

3.3 Stochastic Gradient Descent

In order to solve the optimization problem in Equation (4), a lot of optimization algorithms have been proposed. Among them, Gradient Descent (GD) is an iterative optimization algorithm, where, with an initial setting of ω, the algorithm updates ω using the gradient information of ω. The SGD algorithm keeps updating ω iteratively, until the total number of iterations exceeds the maximum allowed value or the updated error converges. Specifically, in each (the $t+1^{th}$) iteration, the GD algorithm updates ω_t and obtains ω_{t+1} using the following scheme:

$$\omega_{t+1} \leftarrow \omega_t - \eta \cdot \sum_{i=1}^{m} \triangledown \mathcal{L}(\omega_t|\theta_i)/m, \tag{5}$$

where η refers to the step size and $\sum_{i=1}^{m} \triangledown \mathcal{L}(\omega|\theta_i)$ is the sum of gradients.

However, sometimes, the sum of gradient functions are not available. For example, in distributed computing environments, θ_i's are distributed in multiple machines and are not sharable. In this case, SGD algorithm has been proposed to solve the optimization problem in Equation (4) in an ad-hoc manner. In each iteration, compared to GD, the SGD randomly picks up one θ_i from $\theta_1 \ldots \theta_m$, and obtains ω_{t+1} using the gradient of a single loss function $\mathcal{L}(\theta_t|\theta_i)$. Specifically, in the iteration, SGD randomly selects an integer $i \in [1, m]$, then it updates ω using

$$\omega_{t+1} \leftarrow \omega_t - \eta \cdot \triangledown \mathcal{L}(\omega_t|\theta_i). \tag{6}$$

Note that, in distributed optimization problems, where θ_i's are distributed on different machines, the aforementioned algorithm can be implemented as a gossip-based SGD through exchanging the (updated) ω between machines to approximate the optimal solution.

3.4 Parallelized Stochastic Gradient Descent

To further accelerating the optimization process, we leverage the parallelized SGD framework to solve the optimization problem in Equation (4). Suppose the SGD algorithm can be regarded as a single thread with the index k, we reclaim the Equation (6) as

$$\omega_{t+1}^{k} \leftarrow \omega_t^{k} - \eta \cdot \triangledown \mathcal{L}(\omega_t^{k}|\theta_i), \tag{7}$$

where $k \in \{1 \ldots S\}$, S is the size of multiple threads (Leaders). Note that each k^{th} thread runs an independent SGD algorithm and the k^{th} optimal result $\widehat{\omega}^k$ can be obtained when the SGD algorithm converged. Once we have all the converged $\widehat{\omega}^k$ from S threads, the overall optimal result can be averaged by

$$\bar{\omega} \leftarrow \frac{1}{S} \sum_{k=1}^{S} \omega_k. \tag{8}$$

Actually, the multi-thread process run the SGD algorithm in parallel and does not affect other threads when passing the message among the selected machines. To demonstrate the speedup of the parallelized SGD algorithm, we briefly introduce the convergence analysis of the algorithm. Specifically, according to the concentration for distribution [55], the parallelized SGD algorithm is converging to a stationary distribution exponentially faster than the traditional SGD. Also, the guarantees for stationary distribution achieving have been proved [55].

3.5 Problem Formulation

Given m machines, where each (the j^{th}) machine stores n labeled samples with sample estimation of means and covariance matrix $\bar{\mu}^j$, $\bar{\mu}_+^j$, $\bar{\mu}_-^j$ and $\bar{\Sigma}^j$, our work intends to estimate the linear discriminant projection vector β using the estimator listed in Equation (2), while ensuring that the raw data, $\bar{\mu}^j$, $\bar{\mu}_+^j$, $\bar{\mu}_-^j$ and $\bar{\Sigma}^j$ on each machine are not shared with other machines.

Specifically, we assume the n data samples on each machine are randomly drawn from the probability distributions $\mathcal{N}(\mu_+, \Sigma)$ and $\mathcal{N}(\mu_-, \Sigma)$ with equal priors. Given $\bar{\mu}^j$, $\bar{\mu}^j_+$, $\bar{\mu}^j_-$ and $\bar{\Sigma}^j$ estimated using the local data stored on each (the j^{th}) machine, with asymptotic properties that

$$\lim_{m\to\infty} \frac{1}{m}\sum_{j=1}^{m} \bar{\mu}^j_+ = \mu_+, \quad \lim_{m\to\infty} \frac{1}{m}\sum_{j=1}^{m} \bar{\mu}^j_- = \mu_-, \quad \lim_{m\to\infty} \frac{1}{m}\sum_{j=1}^{m} \bar{\Sigma}^j = \Sigma,$$

our work intends to estimate/approximate $\widehat{\beta}^*$ that

$$\widehat{\beta}^* = \underset{\beta\in\Re^p}{\mathrm{argmin}}\left\{ |\beta|_1 \quad s.t.\ |\Sigma\beta - (\mu_+ - \mu_-)|_\infty \le \varepsilon \right\}. \tag{9}$$

Note that all computation tasks are allocated to run on each machine, while each machine can only access the local raw data and local estimations i.e., $\bar{\mu}^j$, $\bar{\mu}^j_+$, $\bar{\mu}^j_-$ and $\bar{\Sigma}^j$. Raw data sharing or local estimation (means and covariance matrix) sharing are not allowed.

4 MODELS AND PROPOSED ALGORITHMS

In this section, we present the algorithm design of *MP^{2}SDA*, where we first introduce the overall framework, then we cover the key algorithms used in *MP^{2}SDA*.

4.1 Overall Framework Design

In this section, we present the framework design of *MP^{2}SDA* algorithm which consists of the following two phases:

— *Training Phase*– Given the n labeled data pairs for training on each (the j^{th}) machine, *MP^{2}SDA* sorts the n data into two sets – T^j_+ and T^j_- for the positive training samples and negative training samples, respectively. A three-stage learning algorithm is employed to (in **Stage I**) first estimate the local mean vectors $\bar{\mu}^j$, $\bar{\mu}^j_+$ and $\bar{\mu}^j_-$ using T^j_+ and T^j_- for (each) j^{th} machine, and approximate the averaged global means $\widehat{\mu}^*$, $\widehat{\mu}^*_+$, and $\widehat{\mu}^*_-$ using the gossip-based SGD over all m machines. Then, with the global mean vectors, *MP^{2}SDA* (in **Stage II**) estimates the local covariance matrix $\widehat{\Sigma}^j$ on each (the j^{th}) machine using the local data but the global means. The algorithm further (in **Stage III**) estimates the truncated linear discriminant projection vector $\widehat{\beta}^*_T$ using $\widehat{\mu}^*_+$, $\widehat{\mu}^*_-$, and $\widehat{\Sigma}^j$ $(1 \le j \le m)$ with the same gossip-based optimization paradigm. Finally, the training phase of *MP^{2}SDA* outputs $\widehat{\beta}^*_T$ and $\widehat{\mu}^*$ as the model of SDA.

— **Testing Phase** – Suppose a new data vector Z arrives at a random machine. With the SDA model $\widehat{\beta}^*_T$ and $\widehat{\mu}^*$ learned in training phase, *MP^{2}SDA* outputs the classification result (i.e., ±1) as the computing result of $sign((Z - \widehat{\mu}^*)^T \widehat{\beta}^*_T)$.

In the following sections, we present the detailed design of the three-stage algorithm for the *MP^{2}SDA* training.

4.2 Multi-Party Message Passing Mechanism

As shown in Figure 1, the Multi-Party Random Message Passing Mechanism is proposed and adopted in Stage I and Stage III. The whole process consists of three parts which are *Initialization*, *Multi-round of Message Passing* and *Averaging and Truncation*. Specifically, in *Initialization* part, through the leader selection, each leader can start initializing the required parameters and independently possess a thread of machines for message passing. Each of the orange block stands for the machine participated in the multi-party community and one or some of them are selected

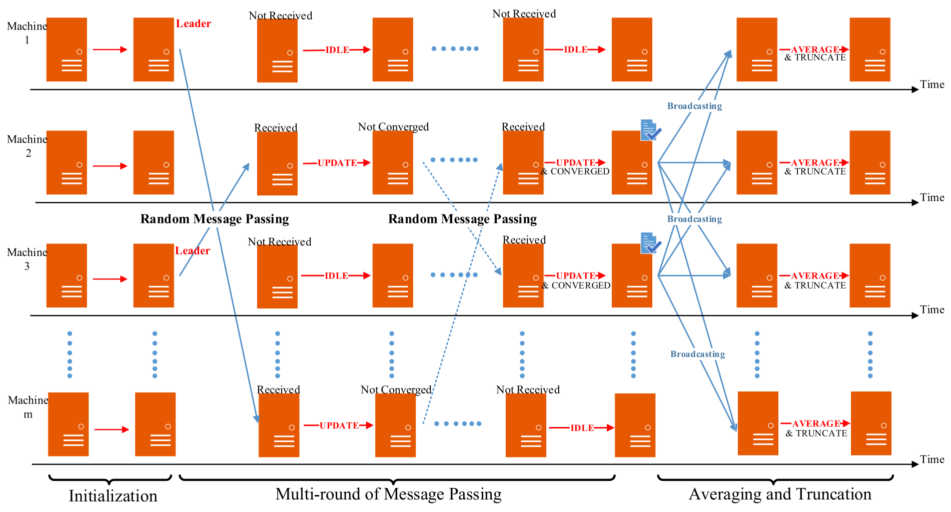


Fig. 1. Multi-party random message passing mechanism.

to be the leaders for the following message passing job (e.g., red leader superscripts have been marked on the machine 1 and machine 3). Then, in *Multi-round Random of Message Passing*, the randomly selected machine (leader) in its thread updates the target value based on the receiving message and passes to the next machine for another round until converged. The solid blue lines represent one time of message passing from one machine to another machine and the machine which receive (marked with received on top on machine block) the message will update the target value, while the machine which do not receive the message will stay idle for this round of message passing. Note that the blue dotted lines differentiate from the solid one due to the fact that it will run more than one round of massage passing until converged. Finally, in *Averaging and Truncation*, the converged target value from all threads are aggregated and truncated to obtain the optimal target value, where every machine can receive the optimal target value by broadcasting in the end. The machine block marked by the checked superscript represents the target value passing through that machine has been converged and will be broadcasted to all the machines (solid blue line). Then each machine will process the last step to average and truncate the received value.

4.3 Stage I: Global Mean Estimation

Due to the parallelism of multi-party computing, MP^2SDA needs specific "Leaders" which are considered a group of starting machines, where these machines can initialize the parameters there to be used and start independent threads among each other. As shown in Algorithm 1, among m machines, MP^2SDA first randomly pick up a set of machines (denote as the set $\mathcal{L}_S$ with size $S \leq m$) through function $LeaderSetSelection()$, where each machine in $\mathcal{L}_S$ initialize a group of key factors $(\widehat{\mu}, \widehat{\mu}_+, \widehat{\mu}_-, t)$ as $(\mathbf{0}, \mathbf{0}, \mathbf{0}, 1)$, where $\mathbf{0}$ refers to a p-dimensional vector with all zero elements and 1 refers to the first update of the algorithm. Then, the initialized key factors will be sent to the next selected machine for Algorithm 2.

Given the local training samples T_+^j and T_-^j on each machine j, MP^2SDA first estimates the local mean vectors $\bar{\mu}^j$, $\bar{\mu}_+^j$, and $\bar{\mu}_-^j$. Algorithm 2 is a gossip-based stochastic gradient decent algorithm that intends to approximate the global means using the estimators listed in Equation (10).

$$\widehat{\mu} = \operatorname*{argmin}_{\mu \in \mathfrak{R}^{1\times p}} \frac{1}{m} \sum_{j=1}^{m} |\mu - \mu^j|_\infty, \; \widehat{\mu}_+ = \operatorname*{argmin}_{\mu \in \mathfrak{R}^{1\times p}} \frac{1}{m} \sum_{j=1}^{m} |\mu - \mu_+^j|_\infty, \; \widehat{\mu}_- = \operatorname*{argmin}_{\mu \in \mathfrak{R}^{1\times p}} \frac{1}{m} \sum_{j=1}^{m} |\mu - \mu_-^j|_\infty, \quad (10)$$

ALGORITHM 1: Leader Selection on the j^{th} Machine

```
begin
    $\mathcal{L}_\mathcal{S} \leftarrow LearderSetElection(S)$;
    if $j \in \mathcal{L}_\mathcal{S}$ then
        INITIALIZE $(\mathbf{0}, \mathbf{0}, \mathbf{0}, 1)$ to $(\widehat{\mu}, \widehat{\mu}_+, \widehat{\mu}_-, t)$;
        Draw $j_{next} \in \{1 \ldots m\}$ uniformly at random;
        SEND $(\widehat{\mu}, \widehat{\mu}_+, \widehat{\mu}_-, t)$ to the $j_{next}$ machine for Algorithm 2;
    end
end
```

ALGORITHM 2: Global Mean Vectors Estimation Algorithm on j^{th} Machine

```
Data:
$\bar{\mu}^j$, $\bar{\mu}^j_+$, and $\bar{\mu}^j_-$ — the local mean vectors based on training samples on the $j^{th}$ Machine
Parameter:
$\eta$ — step size
$\Delta_{max}$ — maximumly allowed perturbation
$t_{max}$ — maximum number of allowed updates
begin
    /* On receiving the message from the previous machine */
    RECEIVE $(\widehat{\mu}, \widehat{\mu}_+, \widehat{\mu}_-, t)$
    /* Updating mean vectors on the $j^{th}$ machine */
    $\widehat{\mu} \leftarrow \widehat{\mu} - \eta \cdot \nabla|\widehat{\mu} - \bar{\mu}^j|_\infty$
    $\widehat{\mu}_+ \leftarrow \widehat{\mu}_+ - \eta \cdot \nabla|\widehat{\mu}_+ - \bar{\mu}^j_+|_\infty$
    $\widehat{\mu}_- \leftarrow \widehat{\mu}_- - \eta \cdot \nabla|\widehat{\mu}_- - \bar{\mu}^j_-|_\infty$
    $t \leftarrow t + 1$
    /* Checking convergence conditions */
    $\Delta = max\left\{\left|\widehat{\mu} - \bar{\mu}^j\right|_\infty, \left|\widehat{\mu}_+ - \bar{\mu}^j_+\right|_\infty, \left|\widehat{\mu}_- - \bar{\mu}^j_-\right|_\infty\right\}$
    if $\Delta \geq \Delta_{max}$ AND $t \leq t_{max}$ then
        /* Not converged, continuing the algorithm */
        Draw $j_{next} \in \{1 \ldots m\}$ uniformly at random;
        SEND $(\widehat{\mu}, \widehat{\mu}_+, \widehat{\mu}_-, t)$ to the $j^{th}_{next}$ machine;
    else
        /* Converged, sharing the estimates to all machines */
        BROADCAST $(\widehat{\mu}, \widehat{\mu}_+, \widehat{\mu}_-)$ to All machines;
    end
end
```

Specifically, Algorithm 1 first receives the input mean vectors (initialed as **0** in the first run), then it updates the input mean vectors using the local means, and randomly picks up the next machine and sends the updated mean vectors for further updating. Algorithm 1 keeps picking up the next machine for the updating, until (1) the total number of updates t exceeds the maximal number of updates, or (2) the updating process converges (i.e., $max\{|\widehat{\mu} - \bar{\mu}^j|_\infty, |\widehat{\mu}_+ - \bar{\mu}^j_+|_\infty, |\widehat{\mu}_- - \bar{\mu}^j_-|_\infty\} \leq \Delta_{max}$). Once the updating process completes, Algorithm 1 broadcasts all m machines with the final global mean estimations $\widehat{\mu}$, $\widehat{\mu}_+$, and $\widehat{\mu}_-$ for Algorithm 2 computation. Note that the notation $\nabla|\widehat{\mu} - \bar{\mu}^j|_\infty$ refers to the gradient of function $|\widehat{\mu} - \bar{\mu}^j|_\infty$ over $\widehat{\mu}$ and can be implemented as:

$$\left(\nabla|\widehat{\mu} - \bar{\mu}^j|_\infty\right)_k = \begin{cases} sign((\widehat{\mu} - \bar{\mu}^j)_k), & \text{if } |(\widehat{\mu} - \bar{\mu}^j)_k| \text{ is the maximal for } 1 \leq k \leq p \\ 0, & else \end{cases} \tag{11}$$

where $(\cdot)_k$ refers to the kth element in the input vector.

ALGORITHM 3: Local Covariance Matrix Estimation (with Global Mean) on the j^{th} Machine

```
Data:
$T^j$ — training sample on $j = 1, 2, \ldots, m$ machine
Parameter:
$\lambda$ — Graphical Lasso regularization parameter
begin
    RECEIVE and AVERAGE $(\widehat{\mu}, \widehat{\mu}_+, \widehat{\mu}_-)_i$ from all machines;
    /* $i \in \{1, 2 \ldots S\}$ (start from $S$ leaders)                          */
    $(\widehat{\mu}^*, \widehat{\mu}^*_+, \widehat{\mu}^*_-) \leftarrow \frac{1}{S} \sum_{i=1}^{S} (\widehat{\mu}, \widehat{\mu}_+, \widehat{\mu}_-)_i$;
    /* Sample covariance matrix estimation                                        */
    $\bar{\Sigma}^j_+ = (T^j_+ - \widehat{\mu}^*_+)(T^j_+ - \widehat{\mu}^*_+)^T$
    $\bar{\Sigma}^j_- = (T^j_- - \widehat{\mu}^*_-)(T^j_- - \widehat{\mu}^*_-)^T$
    $\bar{\Sigma}^j = \frac{1}{2}(\bar{\Sigma}^j_+ + \bar{\Sigma}^j_-)$
    /* Precision matrix estimation through Graphical Lasso [44]                  */
    $\widehat{\Theta}^j \leftarrow glasso(\Sigma^j, \lambda)$
    /* De-sparsify precision matrix                                               */
    $\widehat{D}^j \leftarrow \left(2\widehat{\Theta}^j - \widehat{\Theta}^j \bar{\Sigma}^j \widehat{\Theta}^j\right)$
    /* Obtain the de-sparsified covariance matrix                                 */
    $\widehat{\Sigma}^j \leftarrow (\widehat{D}^j)^{-1}$
    /* Continuing on next machine                                                 */
    $\mathcal{L}_S \leftarrow LearderSetElection(S)$;
    if $j \in \mathcal{L}_S$ then
        INITIALIZE $(\mathbf{0}, 1)$ to $(\widehat{\beta}^*, t)$;
        Draw $j_{next} \in \{1 \ldots m\}$ uniformly at random;
        SEND $(\widehat{\beta}^*, t)$ to the $j_{next}$ machine for Algorithm 4;
    end
end
```

4.4 Stage II: Local Covariance Matrix Estimation

At the beginning of the Algorithm 3, all machines receive the same group of global mean vectors and average them to obtain the averaged global mean vectors. Based on the averaged global mean vectors $\widehat{\mu}^*_+$ and $\widehat{\mu}^*_-$, *MP*^{2}*SDA* runs Algorithm 3 in parallel on each machine without any inter-machine communication requirement. Specifically, this stage first estimates the sample covariance matrix $\bar{\Sigma}^j$ using the averaged global mean vectors. Then, to handle the HDLSS settings, the algorithm leverages the de-sparsified Graphical Lasso estimator [21] ($\widehat{D}^j$) to improve the estimation of the inverse covariance matrix. Finally, matrix inverse is used to estimate the covariance matrix $\widehat{\Sigma}^j$ on the j^{th} machine.

Moreover, Algorithm 3 also executes another *LeaderSetElection()* function to reselect "Leaders" to run Algorithm 4 in the next stage. Specifically, *MP*^{2}*SDA* randomly picks up a group of machines and initializes $(\mathbf{0}, 1)$ to $(\widehat{\beta}^*, t)$ on these machines, where $\mathbf{0}$ refers to a p-dimensional vector with all zero elements and 1 refers to the first update of the algorithm. Then, these initialized $(\widehat{\beta}^*, t)$ pairs are sent to the next selected machine for Algorithm 4.

4.5 Stage III: Sparse Discriminant Projection Vector Estimation

Given the local covariance matrix $\widehat{\Sigma}$ on each machine j and the averaged global mean vectors $\widehat{\mu}^*_+$ and $\widehat{\mu}^*_-$, this stage intends to approximate the global estimation of $\widehat{\beta}^*$ via gossip-based stochastic gradient decent. Indeed, Algorithm 4 minimizes the following loss function over the m machines

ALGORITHM 4: $\widehat{\beta}^*$ Estimation on the j^{th} Machine

```
Data:
$\widehat{\Sigma}^j$ — the local covariance matrix on the $j^{th}$ machine
Parameter:
$\eta$ — step size
$\Delta_{min}$ — minimum allowed perturbation
$t_{max}$ — maximum number of allowed updates
$\lambda$ — regularization parameter
begin
    /* On receiving the message from the previous machine */
    RECEIVE $(\widehat{\beta}^*, t)$
    /* Selecting the $k^{th}$ row of vector $(\widehat{\Sigma}^j \widehat{\beta}^* - (\widehat{\mu}_+ - \widehat{\mu}_-))$ with the maximal absolute value */
    $k \leftarrow \underset{1 \le k' \le p}{argmax} \left| \left( \widehat{\Sigma}^j \widehat{\beta}^* - (\widehat{\mu}_+ - \widehat{\mu}_-) \right)_{k'} \right|$
    /* Updating each row of $\widehat{\beta}^*$ on the $j^{th}$ machine */
    $\beta' \leftarrow \langle 0, 0, \dots, 0 \rangle^T$
    /* initializing $\beta$ with a $p$-dimensional 0 vector */
    foreach $1 \le l \le p$ do
        /* Note: $\widehat{\Sigma}^{j,k,l}$ is the scaler on the $k^{th}$ row and the $l^{th}$ column of the matrix $\widehat{\Sigma}^j$ */
        $g_l \leftarrow sign(\widehat{\beta}^*_l) \cdot \lambda + sign(\widehat{\Sigma}^{j,k} \widehat{\beta}^* - (\widehat{\mu}_+ - \widehat{\mu}_-)) \cdot \widehat{\Sigma}^{j,k,l}$
        /* Update each row of local $\beta$ based on $\widehat{\beta}^*$ */
        $\beta'_l \leftarrow \widehat{\beta}^*_l - \eta \cdot g_l$
    end
    $t \leftarrow t + 1$
    /* Checking convergence conditions */
    $\Delta = \left| \widehat{\beta}^* - \beta' \right|_1$
    /* Update $\widehat{\beta}^*$ after calculating the $\Delta$ */
    $\widehat{\beta}^* \leftarrow \beta'$
    if $\Delta \ge \Delta_{max}$ AND $t \le t_{max}$ then
        /* Not converged, continuing the algorithm */
        Draw $j_{next} \in \{1 \dots m\}$ uniformly at random;
        SEND $(\widehat{\beta}^*, t)$ to the $j^{th}_{next}$ machine;
    else
        /* Converged, sharing the estimates to all machines */
        BROADCAST $\widehat{\beta}^*$ to All machines;
    end
end
```

through gossip-based stochastic gradient decent:

$$\widehat{\beta}^* \leftarrow \underset{\beta \in \mathbb{R}^p}{argmin}\ \lambda \cdot |\beta|_1 + \frac{1}{m} \sum_{j=1}^{m} \left| \widehat{\Sigma}^j \beta - (\widehat{\mu}_+ - \widehat{\mu}_-) \right|_\infty , \tag{12}$$

where λ is a regularization parameter. Specifically, Algorithm 4 first receives the input $\widehat{\beta}^*$ for updating (initialed as **0** in the first run), then it updates the inputed $\widehat{\beta}^*$ vector using $\widehat{\Sigma}^j$ and $\widehat{\mu}_+/\widehat{\mu}_-$, and randomly picks up the next machine and sends the updated $\widehat{\beta}^*$ for further updating. Algorithm 4 keeps picking up the next machine for the updating, until (1) the times of updates t exceeds the maximal number of updates or (2) the updating process converges. Once the updating process completes, Algorithm 4 broadcasts all m machines with the final global estimation of $\widehat{\beta}^*$. To this end, each machine receives the same group of $\widehat{\beta}^*$ (start from S "Leaders"), which is shown in

ALGORITHM 5: Averaging and Truncating on the j^{th} Machine

```
begin
    RECEIVE and AVERAGE $\widehat{\beta}_i^*$ from all machines;
    /* $i \in \{1, 2 \dots S\}$ (start from $S$ leaders)    */
    $\bar{\beta}^* \leftarrow \frac{1}{S} \sum_{i=1}^{S} \widehat{\beta}_i^*$;
    $\widehat{\beta}_T^* \leftarrow Truncate(\bar{\beta}^*)$;
end
```

Algorithm 5. The same as the Stage I, *MP*^{2}*SDA* averages these received β^* and run the *Truncate(x)* function, where this function can set all elements in vector x with relatively small value ($|x| \leq 10^{-4}$) to zero, to obtain the final β_T^*. Finally, each machine has the well estimated β_T^* and $\widehat{\mu}^*$ as the trained SDA model.

4.6 Remark on the Algorithm

In this section, we first analyze the optimality of the algorithm in a Bayesian estimator point of view, then we brief the algorithm in a multi-party computing viewpoint.

Convergence of $\widehat{\beta}_T^*$. Suppose the size of training set on each machine n is sufficiently large and all these samples are drawn i.i.d. from Gaussian distributions $\mathcal{N}(\mu_+, \Sigma)$ and $\mathcal{N}(\mu_-, \Sigma)$. We can assume that the local sample covariance matrix $\bar{\Sigma}^j$ estimated from local raw data on each (the j^{th}) machine should follow an inverse wishart distribution $\mathcal{W}^{-1}(\Sigma, v(n))$, where $v(n)$ is a function on n for the degree of freedom. With infinite number of machines $m \rightarrow \infty$ and infinite number of gossip message passing (i.e., $t \rightarrow \infty$), the algorithm can converge to the minimum of $\widehat{R}(\beta)$ (as the loss function $\widehat{R}$ is convex [8]), where

$$\widehat{R}(\beta) = \mathbb{E}_{\Sigma \sim \mathcal{W}^{-1}(\Sigma, v(n))} \left(\lambda \cdot |\beta|_1 + |\Sigma\beta - (\widehat{\mu}_+ - \widehat{\mu}_-)|_\infty\right). \tag{13}$$

According to the definition of Bayes estimator [2], this loss function can be viewed as a Bayes estimator based on the posterior expectation on risk. We first denote the optimal solution of original sparse LDA listed Equation (2), based on the population parameter Σ and μ_+/μ_-, as β_{SDA}^*. Regarding to the asymptotic efficiency of the Bayes estimator, we conclude:

$$\sqrt{m \times n} \cdot (\widehat{\beta}^* - \beta_{SDA}^*) \rightarrow \mathcal{N}\left(0, I(\beta_{SDA}^*)^{-1}\right),$$

where $I(\beta_{SDA}^*)$ refers to the fisher information of β_{SDA}^*.

Communication Complexity of *MP*^{2}*SDA* Algorithm. Due to the property of parallelized SGD adopted in our work, we mainly discuss the communication complexity of the proposed *MP*^{2}*SDA* algorithm. Suppose the total training sample size is N, the number of dimensions of the data sample is p, the number of the machine is m and the total number of iteration is T, then the communication complexity of *MP*^{2}*SDA* is $O(S \cdot p \cdot T)$.

Multi-Party Computing Properties. Apparently, the proposed algorithm works efficiently, without sharing raw data directly between each machine. Thanks to ℓ_∞-norm loss function used for global mean estimation, the local means on each machine are not shared with others directly. Further, the local covariance matrices are not shared due to the same reason. Note that, according to the above asymptotic analysis, the performance of *MP*^{2}*SDA* is comparable to those centralized methods that raw data sharing is required. Our subsequent experimental analysis based on real-world data will further verify this point—in most cases, *MP*^{2}*SDA* achieves comparable performance to the centralized method derived from [8] using all aggregated data, with similar accuracy and F1-Score.

5 EXPERIMENT

In this section, we use both synthetic data and real-world data to evaluate the performance of *MP^{2}SDA* algorithm. Specifically, we compare our algorithm with distributed SDA algorithm and centralized SDA algorithm. For centralized SDA, all samples are collected on one machine based on the algorithm proposed by [8]. For distributed SDA, we adopt the algorithm proposed by [40] which estimate the global estimator by aggregating local unbiased estimators through averaging with a hard threshold. Note that we fix the size of the leader set as 10% of the total number of machines in each setting as follow to observe the performance of the parallel computing mechanism. Please also refer to the source code for further detailed implementations.[2]

5.1 Synthetic Data Experiments

Experiment Setup. To validate our algorithm, we evaluate our algorithm on a synthesized dataset, which is obtained through a pseudo-random simulation. The synthetic data are generated by two predefined Gaussian distributions $\mathcal{N}(\mu_+^*, \Sigma^*)$ and $\mathcal{N}(\mu_-^*, \Sigma^*)$ with equal priors. The settings of μ_+^*, μ_-^* and Σ^* are as follows: Σ^* is a $p \times p$ symmetric and positive-definite matrix, where $p = 200$, each element $\Sigma_{i,j}^* = 0.8^{|i-j|}$, $1 \le i \le p$, and $1 \le j \le p$. μ_+^* and μ_-^* are both p-dimensional vectors, where $\mu_+^* = \langle 1, 1, \ldots, 1, 0, 0, \ldots, 0 \rangle^T$ (the first 10 elements are all 1's, while the rest $p - 10$ elements are 0's) and $\mu_-^* = \mathbf{0}$. While noting that the number of samples from two Gaussian distributions are equal on each machine. (Settings of the two Gaussian distributions first appear in [40].) In order to evaluate the performance of algorithms for comparison, we obtain the accuracy, F1-score, receiver operating characteristic (ROC) curve and area under the ROC curve (AUC) from the classification results. Specifically, accuracy and F1-score are calculated by maximizing the accuracy/F1-score across all possible cutoffs in ROC curve and AUC stands for the area under the ROC curve. Usually, a higher AUC means the model has a better fit on the datasets.

Parameters Tuning: For the centralized SDA algorithm, there is only one regularization parameter λ_{Glasso} in Algorithm 2. By the theoretical result in [8], we can tune a proper λ_{Glasso} in the order of $O\sqrt{\frac{log(p)}{N}}$. Therefore, we set $\lambda_{Glasso} = C\sqrt{\frac{log(p)}{N}}$ and tune C by grid search. For the proposed algorithm *MP^{2}SDA*, other than λ_{Glasso}, there is one more parameter to be tuned – λ in Algorithm 3. We process a similar grid search directly on this λ. For the distributed SDA algorithm, we follow the same procedure to tune key parameters described in the experiment section of [40] by Tian and Gu (2016). We report the best results based on fine-tuned parameters for all methods. Also, we fix the testing samples at 400 for all the following experiments.

For better comparing the proposed *MP^{2}SDA* with centralized SDA and distributed SDA, we artificially set up two experimental settings. On the one hand, for distributed computing, the number of workload is the critical factor which may affect the performance of the algorithm. In this case, we keep the total number of sample fixed to all the algorithms to check whether varying number of machines can bring some differences, which means the number of samples distributed on each machine is decreasing with growth of the number of machines. Since the number of samples on each machine represent the workload for each machine, this setting intend to measure the performance trading-off between the parallelism and the computing power of the machines. The detailed settings are illustrated in Setting 1. On the other hand, if we fix the number of samples on each machine instead of fixing the total number of samples, the workload of each machine will be same so as to guarantee the same computing power. In such a setting, the primary goal is to explore how parallelism can benefit the party of machines without the limit of the total number of samples. The detailed settings are presented in Setting 2. Note that the Setting 2 is more suitable

[2]https://github.com/Fred1991/2017Jiang_Bian_ICDM_MultipartySDA.

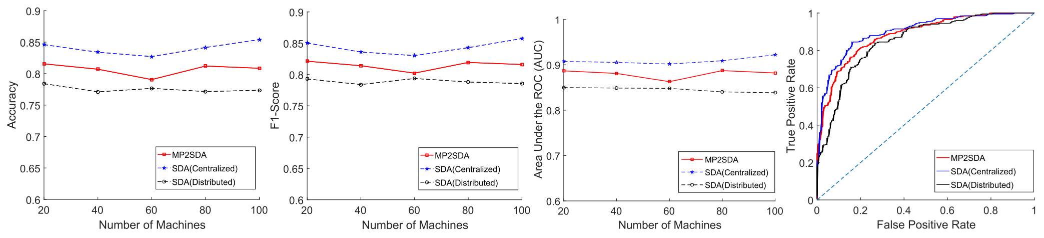


Fig. 2. Performance Comparison among *MP^{2}SDA*, SDA (centralized) and SDA (distributed) on synthetic datasets. We compare the accuracy, F1-score, AUC, and ROC curve of each algorithm **when the total training sample size is fixed as 20,000**. (Note that the ROC curve is drawn when the number of machines is 100)

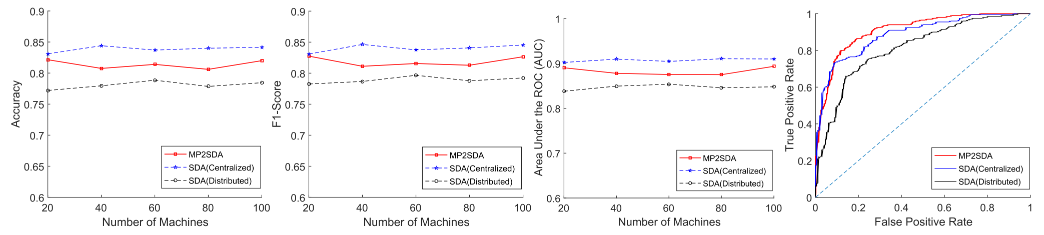


Fig. 3. Performance comparison among *MP^{2}SDA*, SDA (centralized) and SDA (distributed) on synthetic datasets. We compare the accuracy, F1-score, AUC, and ROC curve of each algorithm **when the training sample size on each machine is set as 400**. (Note that the ROC curve is drawn when the number of machines is 100)

to reveal the effect of parallelism, while Setting 1 is more reasonable in practice since most of the time the number total samples (data) are limited.

Setting 1 – Fix the total training sample size and vary the number of machines: To investigate the effect of the number of machines m, we fix the total training sample size $N = 20{,}000$ and vary the number of machines. Figure 2 shows how the accuracy, F1-score and AUC of *MP^{2}SDA* (we use MP^{2}SDA in all the figures), centralized SDA and distributed SDA change as the number of machines grows. For each m, we repeat each algorithm for 10 times and report the average value.

From Figure 2, we can find that *MP^{2}SDA* algorithm outperforms distributed SDA algorithm on accuracy, F1-score and AUC. It is unsurprising that centralized SDA outperforms both *MP^{2}SDA* and distributed SDA on accuracy, F1-score and AUC.

Setting 2 – Fix the training sample size on each machine and vary the number of machines: We alter the settings to evaluate the effect of averaging. We increase the number of machines m linearly as the total training sample size N, that is, the sample size on each machine n is fixed. More specifically, we choose $n = 400$. Figure 3 displays the accuracy, F1-score and AUC of the three algorithms.

The result shows that the performance of *MP^{2}SDA* still outperforms distributed SDA algorithm on accuracy, F1-score, and AUC. Similarly, centralized SDA outperforms both *MP^{2}SDA* and distributed SDA algorithm. We notice that the performance of *MP^{2}SDA* is close to the performance (accuracy, F1-score, and AUC) of centralized SDA when the number of machines is equal to or less than 20. The same situation occurs when the number of machines is equal to or greater than 100.

Supplement – ROC curves: Additionally, we present the ROC curves of three algorithms as an auxiliary indicator to analyze the performances. The setting is picked up among the above experiments. Specifically, we run the simulation at the setting of 100 machines and choose the data from the last repeat to draw the ROC curve. When the total training sample size is fixed, the ROC curve in Figure 1 shows that *MP^{2}SDA* algorithm outperforms distributed SDA, although it does not surpass the performance of centralized SDA. While when the training sample size on each machine is fixed, the ROC curve of *MP^{2}SDA* overlaps with or even covers the ROC curve of centralized SDA in Figure 2, which shows that the performance of *MP^{2}SDA* algorithm is comparable to the performance of centralized SDA. This result is consistent with the variation tendency of the result on accuracy, F1-score, and AUC in Setting 2.

Summary: In synthetic data experiments, we compare the performance of *MP^{2}SDA* with distributed SDA and centralized SDA in two settings. At most circumstance, centralized SDA has the best performance compared to the other two algorithms. Typically, the performance of *MP^{2}SDA* can approach the performance of centralized SDA in Setting 2 with the sample size on each machine increased ($\geqslant 100$) or stayed relatively low ($\leqslant 20$). Note that, in both settings, *MP^{2}SDA* outperforms distributed SDA significantly.

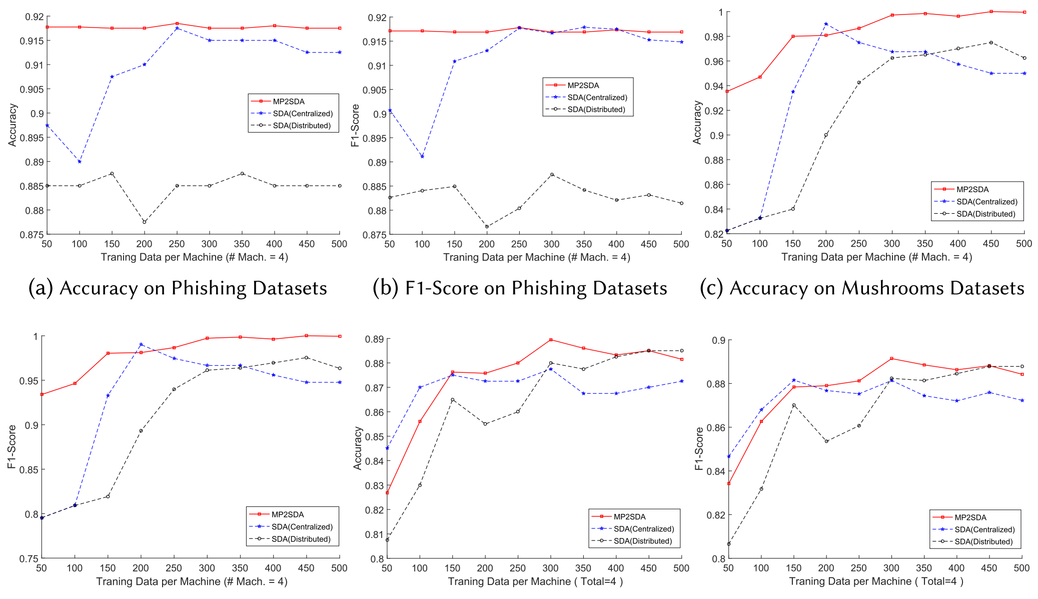
Moreover, according to the stable trends of each of the indicators (accuracy, F1-score, and AUC), we can conclude that the parallelism or the distributed assignment does not harm the overall performance and reach the saturation interval for our specific settings. Then, the stable performance provides us an excellent computing environment that we can fully leverage the advantages of the multi-party computing, where we will show the high efficiency it can achieve in the next section.

5.2 Benchmark Data Experiments

Experiment Setup: To verify the effectiveness of *MP^{2}SDA* algorithm on real datasets, we use Phishing, Splice, and Mushrooms datasets [29] to conduct the comparison. Specifically, we set the size of total training samples varied from 200 to 2000 with 400 testing samples, while the numbers of dimensions p are $p = 54$ (Phishing), $p = 35$ (Splice), and $p = 60$ (Mushrooms), respectively. The number of machines is fixed at 4. We repeat each algorithm for 10 times and report the average value. The adopted well-tuned parameters for the regularization terms are as follow: For *MP^{2}SDA*, $\lambda = 15$ and $\lambda_{glasso} = 1$; For MPSDA, $\lambda = 10$ and $\lambda_{glasso} = 1$; For centralized SDA, $\lambda_{glasso} = 0.01$; For distributed SDA, $\lambda_{glasso} = 0.1$.

In this experiment, we compare the classification accuracy and F1-score of *MP^{2}SDA* with distributed SDA and centralized SDA on each benchmark datasets. Figure 4(a) and (b) presents the performance of each algorithm on Phishing datasets. We can observe that *MP^{2}SDA* obviously outperforms distributed SDA and centralized SDA when the training sample size is smaller than 250, even when the training sample size is greater than 250, *MP^{2}SDA* is still comparable to centralized SDA and obviously superior to distributed SDA. Figure 4(c) and (d) shows that *MP^{2}SDA* outperforms distributed SDA and centralized SDA on Mushrooms dataset. The performance gap between *MP^{2}SDA* and the other two alternatives tends to be stable when the training sample size grows. In Figure 4(e) and (f), the performances of these three algorithms are close to each other on Splice dataset. In most cases, *MP^{2}SDA* slightly outperforms distributed SDA and centralized SDA.

Further, we compare *MP^{2}SDA* algorithm with other centralized baseline algorithms in the same setting. For comparison, we categorize *MP^{2}SDA* and the baseline algorithms into groups of distributed algorithms and centralized algorithms. The distributed algorithms include *MP^{2}SDA*, MPSDA, and distributed SDA. The centralized algorithms include centralized SDA, centralized two-stage LDA (Ye-LDA), centralized Linear SVM, centralized Kernel SVM, centralized Random Forest and centralized Decision Tree. All the algorithms are fine-tuned. Tables 2–4 presents the accuracy with the standard deviation of each algorithm in varying total training sample size. We notice that for two groups, the centralized algorithms have overall better performance compared to distributed algorithms. For comparison in the distributed group, *MP^{2}SDA* significantly outperforms distributed SDA on Mushrooms and Phishing datasets. On Splice dataset, *MP^{2}SDA* slightly outperforms distributed SDA in most cases.



(a) Accuracy on Phishing Datasets (b) F1-Score on Phishing Datasets (c) Accuracy on Mushrooms Datasets

(d) F1-Score on Mushrooms Datasets (e) Accuracy on Splice Datasets (f) F1-Score on Splice Datasets

Fig. 4. Performance Comparison among *MP^{2}SDA*, SDA (centralized) and SDA (distributed) with different benchmark datasets (testing sample size = 400 and machine number = 4)

Efficiency Comparison. Also, we compare the time consumption of *MP^{2}SDA* algorithm (0.61×10^3 seconds, 2 leader machines) and MPSDA (1.13×10^3 seconds) with centralized SDA algorithm (3.97 seconds) on Mushrooms datasets (4 machines with 2,000 total training samples). Note that the communication time between each machine account for a large proportion in the total time consumption of *MP^{2}SDA*. Actually, on each machine, *MP^{2}SDA* and MPSDA only take 0.93 seconds which is much less than the centralized SDA algorithm. (The experiment platform is Windows OS with 2.8GHz CPU).

Summary: In benchmark data experiments, we first compare the performance of *MP^{2}SDA* with distributed SDA and centralized SDA on real-world benchmark datasets. In most instances, *MP^{2}SDA* can compete with centralized SDA, even outperform centralized SDA on Mushrooms and Phishing datasets. Like the results on synthetic datasets, *MP^{2}SDA* overall outperforms distributed SDA on three benchmark datasets. Then, we additionally compare *MP^{2}SDA* with other centralized baseline algorithms. The result shows that these well-tuned centralized baseline algorithms dominantly outperform *MP^{2}SDA* and distributed SDA. While in the distributed algorithm group, *MP^{2}SDA* still outperforms distributed SDA. The additional efficiency comparison among *MP^{2}SDA*, MPSDA, and centralized SDA shows that *MP^{2}SDA* is more efficient than MPSDA (also centralized SDA on each machine) due to its fast convergence rate which is benefited by the parallel computing mechanism.

6 DISCUSSION

In this section, we compare *MP^{2}SDA* with our previous studies and discuss the open issues.

Compare to the previous version. The proposed *MP^{2}SDA* is an extension of the previous work *MPSDA* [5]. Specifically, we mainly target on the following four novel parts to improve the previous version *MPSDA*: (1) To further accelerate the optimization process of the previous version

Table 2. Accuracy Comparison among *MP^{2}SDA*, SDA (Centralized), and SDA (Distributed) on **Phishing Datasets**

Algorithm	Total Training Set Size									
	200	400	600	800	1000	1200	1400	1600	1800	2000
Distributed Algorithm (*number of machines*, $m = 4$)										
MP^{2}SDA	**0.918±0.001**	**0.918±0.001**	**0.918±0.000**	**0.918±0.000**	**0.919±0.002**	**0.918±0.000**	**0.918±0.000**	**0.918±0.002**	**0.918±0.000**	**0.918±0.000**
MPSDA	0.914±0.001	0.911±0.005	0.909±0.001	0.911±0.001	0.914±0.001	0.914±0.001	0.914±0.000	0.916±0.001	0.914±0.000	0.914±0.000
SDA (Distributed)	0.885±0.000	0.885±0.000	0.888±0.000	0.878±0.000	0.885±0.000	0.885±0.000	0.888±0.000	0.885±0.000	0.885±0.000	0.885±0.000
Centralized Algorithm										
SDA (Centralized)	0.898±0.000	0.890±0.000	0.908±0.000	0.910±0.000	0.918±0.000	0.915±0.000	0.915±0.000	0.915±0.000	0.913±0.000	0.913±0.000
Ye-LDA	0.932±0.024	0.949±0.017	0.947±0.020	0.954±0.016	0.954±0.018	0.948±0.019	0.951±0.015	0.945±0.020	0.953±0.016	0.950±0.017
Linear SVM	0.984±0.010	0.998±0.002	0.998±0.002	0.999±0.001	0.999±0.001	0.999±0.001	0.999±0.001	0.999±0.001	0.999±0.001	0.999±0.001
Kernel SVM	0.969±0.025	0.995±0.004	0.996±0.004	0.998±0.002	0.999±0.001	0.999±0.001	0.999±0.001	0.999±0.001	0.999±0.001	0.999±0.001
Random Forest	0.947±0.027	0.962±0.017	0.984±0.012	0.962±0.020	0.991±0.007	0.987±0.008	0.985±0.007	0.960±0.018	0.993±0.005	0.995±0.004
Decision Tree	0.981±0.016	0.994±0.006	0.998±0.002	0.997±0.003	0.997±0.003	0.999±0.001	0.998±0.002	0.998±0.002	0.998±0.002	0.999±0.001

Table 3. Accuracy Comparison among *MP^{2}SDA*, SDA (Centralized), and SDA (Distributed) on **Mushrooms Datasets**

Algorithm	Total Training Set Size									
	200	400	600	800	1000	1200	1400	1600	1800	2000
Distributed Algorithm (*number of machines*, $m = 4$)										
MP^{2}SDA	**0.935±0.001**	**0.947±0.016**	**0.980±0.000**	**0.981±0.002**	**0.987±0.006**	**0.997±0.004**	**0.999±0.003**	**0.996±0.004**	**0.999±0.000**	**0.999±0.001**
MPSDA	0.933±0.005	0.939±0.005	0.957±0.009	0.966±0.003	0.971±0.010	0.977±0.006	0.988±0.007	0.987±0.004	0.987±0.002	0.989±0.002
SDA (Distributed)	0.823±0.000	0.833±0.000	0.840±0.000	0.900±0.000	0.943±0.000	0.963±0.000	0.965±0.000	0.970±0.000	0.975±0.000	0.963±0.000
Centralized Algorithm										
SDA (Centralized)	0.823±0.000	0.833±0.000	0.935±0.000	0.990±0.000	0.975±0.000	0.968±0.000	0.968±0.000	0.958±0.000	0.950±0.000	0.950±0.000
Ye-LDA	0.932±0.024	0.949±0.017	0.947±0.020	0.954±0.016	0.954±0.018	0.948±0.020	0.951±0.015	0.945±0.020	0.953±0.016	0.950±0.017
Linear SVM	0.984±0.010	0.998±0.002	0.998±0.002	0.999±0.001	0.999±0.001	0.999±0.001	0.999±0.001	0.999±0.001	0.999±0.0017	0.999±0.001
Kernel SVM	0.969±0.025	0.994±0.004	0.996±0.004	0.998±0.002	0.999±0.001	0.999±0.001	0.999±0.001	0.999±0.001	0.999±0.001	0.999±0.001
Random Forest	0.947±0.027	0.962±0.017	0.984±0.013	0.962±0.018	0.991±0.007	0.987±0.010	0.985±0.007	0.961±0.018	0.993±0.005	0.995±0.003
Decision Tree	0.981±0.016	0.994±0.006	0.998±0.002	0.997±0.003	0.997±0.003	0.999±0.001	0.998±0.002	0.999±0.001	0.998±0.002	0.999±0.001

Table 4. Accuracy Comparison among *MP^{2}SDA*, SDA (Centralized), and SDA (Distributed) on **Splice Datasets**

Algorithm	Total Training Set Size									
	200	400	600	800	1000	1200	1400	1600	1800	2000
Distributed Algorithm (*number of machines*, $m = 4$)										
MP^{2}SDA	**0.827±0.004**	**0.855±0.002**	**0.877±0.003**	**0.876±0.003**	**0.880±0.003**	**0.890±0.001**	**0.887±0.003**	**0.881±0.002**	**0.885±0.002**	**0.880±0.003**
MPSDA	0.817±0.001	0.839±0.003	0.857±0.006	0.861±0.007	0.865±0.010	0.876±0.006	0.879±0.004	0.878±0.009	0.880±0.005	0.879±0.005
SDA (Distributed)	0.808±0.000	0.830±0.000	0.865±0.000	0.855±0.000	0.860±0.000	0.880±0.000	0.878±0.000	0.883±0.000	0.885±0.000	0.885±0.000
Centralized Algorithm										
SDA (Centralized)	0.845±0.000	0.870±0.000	0.875±0.000	0.873±0.000	0.873±0.000	0.878±0.000	0.868±0.000	0.868±0.000	0.870±0.000	0.873±0.000
Ye-LDA	0.781±0.020	0.802±0.016	0.817±0.020	0.832±0.016	0.829±0.019	0.827±0.018	0.827±0.019	0.824±0.018	0.836±0.018	0.837±0.019
Linear SVM	0.745±0.030	0.803±0.015	0.819±0.016	0.837±0.018	0.835±0.017	0.838±0.017	0.838±0.019	0.829±0.016	0.845±0.020	0.849±0.020
Kernel SVM	0.809±0.009	0.832±0.016	0.844±0.025	0.865±0.054	0.867±0.028	0.864±0.042	0.868±0.051	0.866±0.063	0.876±0.031	0.88±0.044
Random Forest	0.882±0.034	0.869±0.029	0.934±0.013	0.874±0.033	0.953±0.012	0.939±0.011	0.947±0.012	0.87±0.029	0.961±0.009	0.946±0.014
Decision Tree	0.857±0.030	0.897±0.022	0.902±0.028	0.913±0.022	0.917±0.021	0.919±0.023	0.92±0.020	0.916±0.020	0.917±0.024	0.92±0.022

MPSDA, we adopt a parallelized framework, where the multiple threads take the place of single thread, and add two more algorithms with the revised original algorithms to make it happen. All the details are presented in the new Section 3. (2) We also analyze the new convergence of $\widehat{\beta}_T^*$, where we not only cover the asymptotic rate, but also combine the convergence rate of parallelized SGD and the convergence rate of SDA estimator. The detailed problems are addressed in Section 3-F. (3) According to the aforementioned changes in algorithms and theoretical properties, we conduct more complementary experiments which include threads impact experiments, new performance comparison experiments among the baseline algorithm on synthetic and benchmark datasets. The extended results are shown in the new Section 4. (4) Further, we make a discussion on the time complexity of *MP^{2}SDA* compared to the baseline algorithms and the previous version *MPSDA* in Section 4-Summary.

Open Issues. We divide open issues into four respects as follows:

- **Data and Metrics.** In the experiment section, we train the estimator using the balanced sample data, wherein the future, we can directly work on unbalanced datasets [1] which are more common in real-world datasets. Further, we can evaluate the classification model of *MP^{2}SDA* from other perspectives (e.g., precision and recall [41]).
- **Optimization Algorithms.** In this research, we adopted gossip-based SGD to solve the distributed optimization problem for SDA training. However, there are some other advanced optimization methods that can obtain better performance. For example, De and Goldstein have proposed an efficient distributed SGD algorithm with variance reduction [13]. We can also extend the proposed algorithm *MP^{2}SDA* on the Non-convex optimization function [11, 33].
- **Models and Inference.** In this article, we mainly focus on the linear discriminant model for classification. In future work, we plan to further improve the performance of distributed classifier training by leveraging more complex models, such as deep neural network [14, 19]. On the other hand, we also plan to leverage our *MP^{2}SDA* model for distributed statistical inference. For example, the non-zero elements of $\widehat{\beta}^*$ should be variables that significantly correlate to the response of statistics.
- **Multi-Party Computing.** In this article, we assume the local raw data and basic statistics (on each party/machine) are not accessible by others, and make no assumption on cryptography issues that frequently appear in multi-party computing research [7, 10]. Indeed, the security and privacy of *MP^{2}SDA* can be enhanced by addressing these issues. In our future work, we plan to study the secured communication and computation schemes to further improve multi-party SDA learning.

In this section, we summarized key open issues of our research. In future work, we will try to address these issues.

7 CONCLUSION

In this article, we proposed *MP^{2}SDA*—a set of novel multi-party SDA algorithms. Specifically, *MP^{2}SDA* is designed to enable sparse discriminant learning effectively without sharing any raw data and basic statistics (means and covariance matrix estimated using the data on specific machine) between machines. Based on our original conference paper [5], we design a multi-party random message passing mechanism to satisfy the need of parallel computing, and a revised stable three-stage training procedure for SDA estimation is proposed on top of multi-party parallel computing platform, where gossip-based SGD algorithms are used to minimize a bootstrapping loss function derived from Cai et al. [8]. We also extend the analysis of parallel stochastic gradient decent method to support our design. Note that, during the optimization procedures, only the gradients of loss function are exchanged between machines in a gossip manner and no raw data or basic statistics are shared directly. The experimental results on synthetic datasets and real-world benchmark datasets for classification show that that *MP^2SDA* is comparable to the aggregated SDA with similar performance. Furthermore, *MP^2SDA* significantly outperforms state-of-the-art distributed SDA algorithm based on model average in most cases.

REFERENCES

[1] Gustavo EAPA Batista, Andre CPLF Carvalho, and Maria Carolina Monard. 2000. Applying one-sided selection to unbalanced datasets. In *Mexican International Conference on Artificial Intelligence*. Springer, 315–325.

[2] James O. Berger. 2013. *Statistical Decision Theory and Bayesian Analysis*. Springer Science & Business Media.

[3] Kanishka Bhaduri, Ran Wolff, Chris Giannella, and Hillol Kargupta. 2008. Distributed decision-tree induction in peer-to-peer systems. *Statistical Analysis and Data Mining* 1, 2 (2008), 85–103.

[4] Jiang Bian, Laura E. Barnes, Guanling Chen, and Haoyi Xiong. 2017. Early detection of diseases using electronic health records data and covariance-regularized linear discriminant analysis. In *2017 IEEE EMBS International Conference on Biomedical & Health Informatics (BHI'17)*. IEEE, 457–460.

[5] Jiang Bian, Haoyi Xiong, Wei Cheng, Wenqing Hu, Zhishan Guo, and Yanjie Fu. 2017. Multi-party sparse discriminant learning. In *2017 IEEE International Conference on Data Mining (ICDM'17)*. IEEE, 745–750.

[6] Jiang Bian, Haoyi Xiong, Yanjie Fu, and Sajal K. Das. 2018. CSWA: Aggregation-free spatial-temporal community sensing. In *32nd AAAI Conference on Artificial Intelligence*.

[7] Dan Bogdanov, Margus Niitsoo, Tomas Toft, and Jan Willemson. 2012. High-performance secure multi-party computation for data mining applications. *International Journal of Information Security* 11, 6 (2012), 403–418.

[8] Tony Cai and Weidong Liu. 2011. A direct estimation approach to sparse linear discriminant analysis. *Journal of the American Statistical Association* 106, 496 (2011), 1566–1577.

[9] T. Tony Cai, Zhao Ren, Harrison H. Zhou, et al. 2016. Estimating structured high-dimensional covariance and precision matrices: Optimal rates and adaptive estimation. *Electronic Journal of Statistics* 10, 1 (2016), 1–59.

[10] Hao Chen and Ronald Cramer. 2006. Algebraic geometric secret sharing schemes and secure multi-party computations over small fields. In *Annual International Cryptology Conference*. Springer, 521–536.

[11] Yiu-ming Cheung and Jian Lou. 2015. Efficient generalized conditional gradient with gradient sliding for composite optimization. In *International Joint Conferences on Artificial Intelligence*. 3409–3415.

[12] Line Clemmensen, Trevor Hastie, Daniela Witten, and Bjarne Ersbøll. 2011. Sparse discriminant analysis. *Technometrics* 53, 4 (2011), 406–413.

[13] Soham De and Tom Goldstein. 2016. Efficient distributed SGD with variance reduction. In *2016 IEEE 16th International Conference on Data Mining (ICDM'16)*. IEEE, 111–120.

[14] Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Andrew Senior, Paul Tucker, Ke Yang, Quoc V. Le, et al. 2012. Large scale distributed deep networks. In *Advances in Neural Information Processing Systems*. 1223–1231.

[15] Richard O. Duda, Peter E. Hart, and David G. Stork. 2001. *Pattern Classification* (2nd Ed). Wiley.

[16] Heinz W. Engl and Charles W. Groetsch. 2014. *Inverse and Ill-posed Problems*. Vol. 4. Elsevier.

[17] Zhi Fengy, Haoyi Xiong, Chuanyuan Song, Sijia Yang, Baoxin Zhao, Licheng Wang, Zeyu Chen, Shengwen Yang, Liping Liu, and Jun Huan. 2019. SecureGBM: Secure multi-party gradient boosting. In *IEEE International Conference on Big Data (Big Data'19)*. IEEE.

[18] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. 2008. Sparse inverse covariance estimation with the graphical lasso. *Biostatistics* 9, 3 (2008), 432–441.

[19] Suyog Gupta, Wei Zhang, and Fei Wang. 2016. Model accuracy and runtime tradeoff in distributed deep learning: A systematic study. In *2016 IEEE 16th International Conference on Data Mining (ICDM'16)*. IEEE, 171–180.

[20] Wassily Hoeffding, Herbert Robbins, et al. 1948. The central limit theorem for dependent random variables. *Duke Mathematical Journal* 15, 3 (1948), 773–780.

[21] Jana Jankova, Sara van de Geer, et al. 2015. Confidence intervals for high-dimensional inverse covariance estimation. *Electronic Journal of Statistics* 9, 1 (2015), 1205–1229.

[22] Adel Javanmard and Andrea Montanari. 2014. Confidence intervals and hypothesis testing for high-dimensional regression. *Journal of Machine Learning Research* 15, 1 (2014), 2869–2909.

[23] W. J. Krzanowski, Philip Jonathan, W. V. McCarthy, and M. R. Thomas. 1995. Discriminant analysis with singular covariance matrices: Methods and applications to spectroscopic data. *Applied Statistics* 44, 1 (1995), 101–115.

[24] Christopher Z. Mooney, Robert D. Duval, and Robert Duvall. 1993. *Bootstrapping: A Nonparametric Approach to Statistical Inference*. Sage, 94–95.

[25] Per Aslak Mykland. 1992. Asymptotic expansions and bootstrapping distributions for dependent variables: A martingale approach. *The Annals of Statistics* 20, 2 (1992), 623–654.

[26] Róbert Ormándi, István Hegedűs, and Márk Jelasity. 2013. Gossip learning with linear models on fully distributed data. *Concurrency and Computation: Practice and Experience* 25, 4 (2013), 556–571.

[27] Finbarr O'Sullivan. 1986. A statistical perspective on ill-posed inverse problems. *Statistical Science* 1, 4 (1986), 502–518.

[28] M. Hashem Pesaran, Yongcheol Shin, and Ron P. Smith. 1999. Pooled mean group estimation of dynamic heterogeneous panels. *Journal of the American Statistical Association* 94, 446 (1999), 621–634.

[29] John C. Platt. 1999. Fast training of support vector machines using sequential minimal optimization. *Advances in Kernel Methods—Support Vector Learning*. MIT Press, Cambridge, MA, 185–208.

[30] Foster J. Provost and Daniel N. Hennessy. 1996. Scaling up: Distributed machine learning with cooperation. In *Annual Conference on Innovative Applications of Artificial Intelligence, Vol. 1.* 74–79.

[31] Guo-Jun Qi, Charu Aggarwal, Deepak Turaga, Daby Sow, and Phil Anno. 2015. State-driven dynamic sensor selection and prediction with state-stacked sparseness. In *21st ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 945–954.

[32] Guo-Jun Qi, Jinhui Tang, Zheng-Jun Zha, Tat-Seng Chua, and Hong-Jiang Zhang. 2009. An efficient sparse metric learning in high-dimensional space via l 1-penalized log-determinant regularization. In *26th Annual International Conference on Machine Learning*. ACM, 841–848.

[33] Hong Qian and Yang Yu. 2016. Scaling simultaneous optimistic optimization for high-dimensional non-convex functions with low effective dimensions. In *AAAI Conference on Artificial Intelligence.* 2000–2006.

[34] Sarunas Raudys and Robert P. W. Duin. 1998. Expected classification error of the Fisher linear classifier with pseudo-inverse covariance matrix. *Pattern Recognition Letters* 19, 5 (1998), 385–392.

[35] Jun Shao, Yazhen Wang, Xinwei Deng, Sijian Wang, et al. 2011. Sparse linear discriminant analysis by thresholding for high dimensional data. *The Annals of Statistics* 39, 2 (2011), 1241–1265.

[36] Xiangbo Shu, Jinhui Tang, Guo-Jun Qi, Zechao Li, Yu-Gang Jiang, and Shuicheng Yan. 2016. Image classification with tailored fine-grained dictionaries. *IEEE Transactions on Circuits and Systems for Video Technology* 28, 2 (2016), 454–467.

[37] Padhraic Smyth, Max Welling, and Arthur U. Asuncion. 2009. Asynchronous distributed learning of topic models. In *Advances in Neural Information Processing Systems.* 81–88.

[38] Charles M. Stein. 1981. Estimation of the mean of a multivariate normal distribution. *The Annals of Statistics* 9, 6 (1981), 1135–1151.

[39] Jinhui Tang, Richang Hong, Shuicheng Yan, Tat-Seng Chua, Guo-Jun Qi, and Ramesh Jain. 2011. Image annotation by k nn-sparse graph-based label propagation over noisily tagged web images. *ACM Transactions on Intelligent Systems and Technology* 2, 2 (2011), 14.

[40] Lu Tian and Quanquan Gu. 2016. Communication-efficient distributed sparse linear discriminant analysis. In *Proceeding of the 20th International Conference on Artificial Intelligence and Statistics (AISTATS'16).*

[41] Kai Ming Ting. 2011. Precision and recall. In *Encyclopedia of Machine Learning*. Springer, 781–781.

[42] Konstantinos I. Tsianos, Sean Lawlor, and Michael G. Rabbat. 2012. Consensus-based distributed optimization: Practical issues and applications in large-scale machine learning. In *50th Annual Allerton Conference on Communication, Control, and Computing (Allerton'12).* IEEE, 1543–1550.

[43] Jie Wen, Xiaozhao Fang, Jinrong Cui, Lunke Fei, Ke Yan, Yan Chen, and Yong Xu. 2018. Robust sparse linear discriminant analysis. *IEEE Transactions on Circuits and Systems for Video Technology* 29, 2 (2018), 390–403.

[44] Daniela M. Witten, Jerome H. Friedman, and Noah Simon. 2011. New insights and faster computations for the graphical lasso. *Journal of Computational and Graphical Statistics* 20, 4 (2011), 892–900.

[45] Daniela M. Witten and Robert Tibshirani. 2009. Covariance-regularized regression and classification for high dimensional problems. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 71, 3 (2009), 615–636.

[46] Zechao Li, Hanjiang Lai, Liyan Zhang, Shuicheng Yan, Xiangbo Shu, and Jinhui Tang. 2017. Personalized age progression with bi-level aging dictionary learning. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 40, 4 (2017), 905–917.

[47] Eric P. Xing, Qirong Ho, Wei Dai, Jin Kyu Kim, Jinliang Wei, Seunghak Lee, Xun Zheng, Pengtao Xie, Abhimanu Kumar, and Yaoliang Yu. 2015. Petuum: A new platform for distributed machine learning on big data. *IEEE Transactions on Big Data* 1, 2 (2015), 49–67.

[48] Eric P. Xing, Qirong Ho, Pengtao Xie, and Dai Wei. 2016. Strategies and principles of distributed machine learning on big data. *Engineering* 2, 2 (2016), 179–195.

[49] Haoyi Xiong, Wei Cheng, Jiang Bian, Wenqing Hu, Zeyi Sun, and Zhishan Guo. 2018. DBSDA: Lowering the bound of misclassification rate for sparse linear discriminant analysis via model debiasing. *IEEE Transactions on Neural Networks and Learning Systems* 30, 3 (2018), 707–717.

[50] Haoyi Xiong, Wei Cheng, Yanjie Fu, Wenqing Hu, Jiang Bian, and Zhishan Guo. 2018. De-biasing covariance-regularized discriminant analysis. In *27th International Joint Conference on Artificial Intelligence.* 2889–2897.

[51] Haoyi Xiong, Wei Cheng, Wenqing Hu, Jiang Bian, and Zhishan Guo. 2017. AWDA: An adaptive wishart discriminant analysis. In *2017 IEEE International Conference on Data Mining (ICDM'17)*. IEEE, 525–534.

[52] Shuangyan Yi, Zhenyu He, Yiu-Ming Cheung, and Wen-Sheng Chen. 2017. Unified sparse subspace learning via self-contained regression. *IEEE Transactions on Circuits and Systems for Video Technology* 28, 10 (2017), 2537–2550.

[53] Jian-Pei Zhang, Zhong-Wei Li, and Jing Yang. 2005. A parallel SVM training algorithm on large-scale classification problems. In *2005 International Conference on Machine Learning and Cybernetics*, Vol. 3. IEEE, 1637–1641.

[54] Yunhong Zhou, Dennis Wilkinson, Robert Schreiber, and Rong Pan. 2008. Large-scale parallel collaborative filtering for the Netflix prize. In *International Conference on Algorithmic Applications in Management*. Springer, 337–348.

[55] Martin Zinkevich, Markus Weimer, Lihong Li, and Alex J. Smola. 2010. Parallelized stochastic gradient descent. In *Advances in Neural Information Processing Systems*. 2595–2603.

Received March 2019; revised October 2019; accepted December 2019