2020

Statistical and Stochastic Learning Algorithms for Distributed and Intelligent Systems

Jiang Bian
University of Central Florida

STATISTICAL AND STOCHASTIC LEARNING ALGORITHMS FOR DISTRIBUTED AND INTELLIGENT SYSTEMS

by

JIANG BIAN
B.Eng. Logistics System Engineering, Huazhong University of Science and Technology, 2014
M.Sc., Industrial Systems and Engineering, University of Florida, 2016

A dissertation submitted in partial fulfilment of the requirements
for the degree of Doctor of Philosophy
in the Department of Electrical and Computer Engineering
in the College of Engineering and Computer Science
at the University of Central Florida
Orlando, Florida

Fall Term
2020

Major Professor: Zhishan Guo

ABSTRACT

In the big data era, statistical and stochastic learning for distributed and intelligent systems focuses on enhancing and improving the robustness of learning models that have become pervasive and are being deployed for decision-making in real-life applications including general classification, prediction, and sparse sensing. The growing prospect of statistical learning approaches such as Linear Discriminant Analysis and distributed Learning being used (e.g., community sensing) has raised concerns around the robustness of algorithm design. Recent work on anomalies detection has shown that such Learning models can also succumb to the so-called 'edge-cases' where the real-life operational situation presents data that are not well-represented in the training data set. Such cases have been the primary reason for quite a few mis-classification bottleneck problems recently. Although initial research has begun to address scenarios with specific Learning models, there remains a significant knowledge gap regarding the detection and adaptation of learning models to 'edge-cases' and extreme ill-posed settings in the context of distributed and intelligent systems. With this motivation, this dissertation explores the complex in several typical applications and associated algorithms to detect and mitigate the uncertainty which will substantially reduce the risk in using statistical and stochastic learning algorithms for distributed and intelligent systems.

A special feeling of gratitude to my loving parents whose words of encouragement and push for tenacity ring in my ears. I also dedicate this dissertation to my many friends who have supported me throughout the process. I will always appreciate and love you all.

ACKNOWLEDGMENTS

I wish to thank my committee members who were more than generous with their expertise and precious time. A special thanks to Dr. Zhishan Guo, my committee chairman for his countless hours of reflecting, reading, encouraging, and most of all patience throughout the entire process. Thank you Dr. Jun Wang, Dr. Yaser Pourmohammadi Fallah, Dr. Arthur Huang, and Dr. Haoyi Xiong for agreeing to serve on my committee. I would like to acknowledge and thank my school division for allowing me to conduct my research and providing any assistance requested. Special thanks goes to the members of staff development and human resources department for their continued support. Finally I would like to thank the colleagues in our Lab that assisted me with this project. Their excitement and willingness to provide feedback made the completion of this research an enjoyable experience.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1: INTRODUCTION

Statistical learning is regarded as one of the most beautifully developed branches of artificial intelligence. It provides the theoretical basis for many of today's machine learning algorithms. The learning theory helps to explore what permits to draw valid conclusions from empirical data. The statistical learning begins with a class of hypotheses and uses empirical data to select one hypothesis from the class. If the data generating mechanism is benign, then it is observed that the difference between the training error and test error of a hypothesis from the class is small. The statistical learning generally avoids metaphysical statements about aspects of the true underlying dependency, and thus is precise by referring to the difference between training and test error.

The goals of statistical learning are understanding and prediction. Learning falls into many categories, including supervised learning, unsupervised learning, online learning, and reinforcement learning. From the perspective of statistical learning theory [1], supervised learning is best understood. Supervised learning involves learning from a training set of data. Every point in the training is an input-output pair, where the input maps to an output. The learning problem consists of inferring the function that maps between the input and the output, such that the learned function can be used to predict the output from future input.

Depending on the type of output, supervised learning problems are either problems of regression or problems of classification. If the output takes a continuous range of values, it is a regression problem. In facial recognition, for instance, a picture of a person's face would be the input, and the output label would be that person's name. The input would be represented by a large multidimensional vector whose elements represent pixels in the picture. After learning a function based on the training set data, that function is validated on a test set of data, data that did not appear in the training set.

1.1 Motivation

Many statistical learning algorithms and models are described in terms of being stochastic. This is because many optimization and learning algorithms both must operate in stochastic domains and because some algorithms make use of randomness or probabilistic decisions.

Stochastic domains are those that involve uncertainty. This uncertainty can come from a target or objective function that is subjected to statistical noise or random errors. It can also come from the fact that the data used to fit a model is an incomplete sample from a broader population. Finally, the models chosen are rarely able to capture all of the aspects of the domain, and instead must generalize to unseen circumstances and lose some fidelity.

Most statistical learning algorithms are stochastic because they make use of randomness during learning. Using randomness is a feature, not a bug. It allows the algorithms to avoid getting stuck and achieve results that deterministic (non-stochastic) algorithms cannot achieve. For example, some machine learning algorithms even include "stochastic" in their name such as: Stochastic Gradient Descent [2]. Stochastic gradient descent optimizes the parameters of a model, such as an artificial neural network, that involves randomly shuffling the training dataset before each iteration that causes different orders of updates to the model parameters. In addition, model weights in a neural network are often initialized to a random starting point.

1.1.1 Stochasticity and Randomness

Due to that fact that many machine learning algorithms make use of randomness, their nature (e.g. behavior and performance) is also stochastic. The stochastic nature of machine learning algorithms is most commonly seen on complex and nonlinear methods used for classification and regression predictive modeling problems. These algorithms make use of randomness during the

process of constructing a model from the training data which has the effect of fitting a different model each time same algorithm is run on the same data. In turn, the slightly different models have different performance when evaluated on a hold out test dataset. This stochastic behavior of nonlinear machine learning algorithms is challenging for researchers who assume that learning algorithms will be deterministic, e.g. fit the same model when the algorithm is run on the same data. This stochastic behavior also requires that the performance of the model must be summarized using summary statistics that describe the mean or expected performance of the model, rather than the performance of the model from any single training run. In summary, stochastic is one of the most important characteristics of statistical learning and I will discuss additional advantages of leveraging the stochastic optimization to address some practical issues in distributed intelligent systems in next section.

1.1.2 Limitations in Modern Distributed Systems

With increasing the volume of "big data", mining/training such tremendous data models with a single machine can be very slow [3]. Not only that, 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 crossed a set of locations or machines for several reasons. For example, several data sets concerning medical (personal) information (e.g. allergic history) might be owned by separate hospitals that have reasons for keeping the data private. The traditional statistical learning algorithm is no longer fitting the big data scenario, where the famous "the curse of dimensionality" [4] will degrade significantly the performance of them. To handle the above issues, various distributed/parallelized machine learning algorithms were proposed, e.g., distributed decision tree [5], parallel support vector machine [6] and parallel rule induction [7, 8].

In addition to distributed learning, Multi-Party computing [9, 10] becomes one of popular com-

puting 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 [3, 11–17]. 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 [11, 15, 18]. To advance the distributed performance of classical statistical learning algorithms, Tian and Gu et al. [19] proposed a data-centric algorithm, which leverages the advantage of parallel computing. Although it is intuitive that the model-centric counterpart for statistical learning algorithm 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.

1.2 Contributions and Organization

To address the issue of "the curse of dimensionality" in statistical and stochastic learning algorithm in bid data era, we improve the performance of the classical linear discriminant analysis, one of the most common used statistical learning algorithm, by proposing a causality/covariance regularization and a de-biased estimation. These two key designs will be discussed in Chapter 2.

To fill the gap between the centralized statistical learning and popular distributed statistical learning, we are motivated to design a novel **model-centric** learning framework. In Chapter 3,4, and 5, we mainly discuss the contribution we made for building such framework to improve the performance of statistical learning algorithm in distributed intelligent systems. Not only our proposal can achieve a better performance provided by the model-centric algorithm, it also promotes the efficiency of the algorithm through parallel computing mechanism. Specifically, the gossip-based stochastic gradient descent plays an important role in the optimization and federated learning, which demonstrates that stochastic as one of the key characteristics can naturally benefit the distributed learning patterns. Compared with the approach in [20], which aggregates all data on a single machine to learn the model, our proposal can effectively approximate to the optimal solution without sharing any raw data. Compared with [19], which aggregates the locally learned models through model-averaging and hard-thresholding, our models and minimizes a distributed loss function based on specific statistical learning model, parameterized with global/local estimates, straightforwardly. Moreover, compared to normal single thread model-centric algorithm [21], our design additionally processing parallel computing when estimating the model parameters to improve the performance with fast convergence rate.

In the following chapters, I will separately introduce the proposed research topic on top of different applications to illustrate how am I pursuing the solution of adapting statistical learning via stochastic optimization to embracing the distributed learning context in modern intelligent systems. The

flow will start with a introduction of classical statistical learning algorithm with its application. Then, the innovative modification to fit the distributed scenarios will be discussed. Finally, I will make a conclusion and cast a future direction on this topic.

CHAPTER 2: DISCRIMINANT ANALYSIS FOR INTELLIGENT MEDICAL SYSTEMS

Fisher's Linear Discriminant Analysis (FLD) [22] is a well-known technique for feature extraction and dimension reduction [23]. It has been widely used in many applications, such as face recognition [24], image retrieval, etc. An intrinsic limitation of classical FLD is that its objective function relies on the *well-estimated* and *non-singular* covariance matrices.

For many applications, such as the micro-array data analysis, all scatter matrices can be *singular* or *ill-posed* since the data is often with high dimension but low sample size (HDLSS) [25].

The classical FLD classifier relies on two key parameters – the mean vector of each type and the precision matrix. Under the HDLSS settings, the sample precision matrix (a.k.a., the inverse of sample covariance matrix) used in FLD is usually ill-estimated and quite different from the inverse of population/true covariance matrix [25]. For example, the largest eigenvalue of the sample covariance matrix is not a consistent estimate of the largest eigenvalue of the population covariance matrix, and the eigenvectors of the sample covariance matrix can be nearly orthogonal to the truth when the number of dimensions is greater than the number of samples [26]. Such inconsistency between the true and the estimated precision matrices degrades the accuracy of FLD classifiers under the HDLSS settings [27].

A plethora of excellent work has been conducted to address such HDLSS data classification problem for FLD. For example, Krzanowski et al. [28] suggested to use pseudo-inverse to approximate the inverse covariance matrix, when the sample covariance matrix is singular. However, the precision of pseudo-inverse FLD is usually low and not well guaranteed. Other techniques include the two-stage algorithm PCA+FLD [29], FLD based on Kernels [30] and/or other nonparametric

statistics [31]. To overcome the singularity of the sample covariance matrices, instead of estimating inverse covariance matrix and mean vectors separately, [20] proposed to estimate the projection vector for discrimination directly. More popularly, regularized FLD approaches [28, 32] are proposed to solve the problem. These methods can improve the performance of FLD either empirically or theoretically [33,34], while few of them can directly address the ill-estimated inverse covariance matrix estimation issue.

One representative regularization approach is *Covariance-Regularized FLD* [32] that replaces the precision matrix used in FLD with a shrunken estimator, such as Graphical Lasso [35], so as to achieve a *"superior prediction"*. Intuitively, through replacing precision matrix used in FLD with a sparse regularized estimation, the ill-posed problem caused by the HDLSS settings can be well addressed. The sparse estimators usually converge to the inverse of true/population covariance matrix faster than the sample estimators [25]. With the asymptotic properties, the sparse FLD should be close to the optimal FLD. However, the way that the sparsity and the convergence rate of the precision matrix estimator would affect the classification accuracy is not well studied in literature.

Further, with induced sparsity, the inverse covariance estimator becomes biased [36]. The performance of sparse FLD is frequently bottlenecked due to the bias of the sparse estimators. Recently, researchers tried to de-bias the Lasso estimator [36], through adjusting the ℓ_1-penalty for the regularized estimation, so as to achieve a better regression performance. Inspired by this line of research, we propose to improve sparse FLD with different purposes in this chapter. In the following subsections, we will illustrate varieties of learning approaches to overcome the above-mentioned ill-posed problems in three common-seen scenarios (case studies).

2.1 Preliminaries

In this section, we first briefly introduce the binary classifier using FLD, then present the practice of CRLD based on Graphical Lasso.

2.1.1 FLD for Binary Classification

To use the Fisher's Linear Discriminant Analysis (FLD), given the $i.i.d.$ labeled data pairs $(x_1, \ell_1) \ldots (x_m, \ell_m)$, we first estimate the sample covariance matrix $\bar{\Sigma}$ using the pooled sample covariance matrix estimator with respect to the two classes [22], then estimate the sample precision matrix as $\bar{\Theta} = \bar{\Sigma}^{-1}$. Further, $\bar{\mu}_+$ and $\bar{\mu}_-$ are estimated as the mean vectors of the positive samples and the negative samples in the m training samples, respectively.

Given all estimated parameters $\bar{\Sigma}$ (and $\bar{\Theta} = \bar{\Sigma}^{-1}$), $\bar{\mu}_+$ and $\bar{\mu}_-$, the FLD model classifies a new data vector x as the result of:

$$\begin{aligned} \bar{f}(x) &= \underset{\ell \in \{-,+\}}{\operatorname{argmax}}\ \delta(x, \bar{\Theta}, \bar{\mu}_\ell, \pi_\ell), \ \textbf{where} \\ \delta(x, \bar{\Theta}, \bar{\mu}_\ell, \pi_\ell) &= x^\top \bar{\Theta} \bar{\mu}_\ell - \frac{1}{2} \bar{\mu}_\ell^\top \bar{\Theta} \bar{\mu}_\ell + \log \pi_\ell, \end{aligned} \tag{2.1}$$

where π_+ and π_- refer to the (foreknown) frequencies of positive samples and negative samples in the whole population, respectively.

2.1.2 Covariance-Regularized FLD via Graphical Lasso

This algorithm, referred to as the Covariance-Regularized FLD (CRLD) via Graphical Lasso, was derived from the *Scout family* of FLD introduced by *Witten et al.* in [32]. Compared to the clas-

sical FLD, this baseline algorithm leverages Graphical Lasso estimator to replace the precision matrix estimated using sample covariance matrix. The proposed algorithm is implemented using the discriminant function defined in Eq. 2.1, as:

$$\widehat{f}(x) = \underset{\ell \in \{-,+\}}{\operatorname{argmax}}\ \delta(x, \widehat{\Theta}, \bar{\mu}_\ell, \pi_\ell), \tag{2.2}$$

where $\widehat{\Theta}$ refers to the Graphical Lasso estimator based on the sample covariance matrix $\bar{\Sigma}$:

$$\widehat{\Theta} = \underset{\Theta > 0}{\operatorname{argmin}} \left(\operatorname{tr}(\bar{\Sigma}\Theta) - \log\det(\Theta) + \lambda \sum_{j \neq k} |\Theta_{jk}| \right). \tag{2.3}$$

Note that, as a linear classifier, the CRLD decision rule introduced in Eq. 2.2 can be re-formulated in a linear model, such as:

$$\begin{aligned} \widehat{f}(x) &= \operatorname{sign}\left(\delta(x, \widehat{\Theta}, \bar{\mu}_+, \pi_+) - \delta(x, \widehat{\Theta}, \bar{\mu}_-, \pi_-) \right) \\ &= \operatorname{sign}\left(x^\top \widehat{\beta}^G + c^g \right), \end{aligned} \tag{2.4}$$

where $\operatorname{sign}(\cdot)$ function returns $+1$ if the input is non-negative, and -1 when the input is negative. The vector $\widehat{\beta}^G = \widehat{\Theta}(\bar{\mu}_+ - \bar{\mu}_-)$ and the scalar $c^g = -\frac{1}{2} \cdot (\bar{\mu}_+ + \bar{\mu}_-)^\top \widehat{\beta}^G + log(\pi_+/\pi_-)$. Obviously, $\widehat{\beta}^G$ is the vector of projection coefficients for linear classification.

2.2 *CRLEDD*: Regularized Causalities Learning for Early Detection of Diseases

2.2.1 Backgrounds

The early disease detection is one of the most prevalent tasks in statistical learning and machine learning, and it plays an important role in modern medical diagnosis and pre-treatment systems.

From the aspect of feature extraction, image is the mainstream data type for discovering the latent correlation among the factor of diseases and thereby helps us recognize or classify them. For example, [37, 38] propose to use SAR [39] image data to process the object recognition and the target segmentation, where the statistical-based texture features such as KWE [40] and KCE [41] are well-studied [42] as the basis to support the classification. From the aspect of learning model, [43] propose a hierarchical learning architecture which integrates the well-known CNN [44] and MLP [45] to recognize the target image object. However, most of theses preliminary work are based on the image data, where sometimes it is difficult to collect such highly related image data in disease detection task due to the privacy and technical issue (e.g., for some disease, we do not even know the source of the lesion). Fortunately, for general diagnosis, we still have the common electronic health records associated with each patient, which has been wide-used in the medical systems.

Electronic Health Records (EHR) [46] play a critical role in modern health information management and service innovations. A patient's EHR contains his/her medical visit history, medication, diagnoses, treatment plans, allergies and so on. One significant feature is the interchangeability of EHR, as a standard protocol for medical/health data generation, storage and communication. The health information is built and managed by authorized institutions in a unified digital format (e.g., ICD-9/10, CPT-9/10 used in EHR standards) such that researchers and scientists can share and analyze the EHR data to enable innovative health services, such as providing computer-assisted diagnosis and offering medication advice. Among these services, early detection of diseases, using their past longitudinal health information of the EHR system, has recently attracted significant attention from the research community. There has been a series of works [46–51], which attempt to predict future disease of patients, through data mining techniques using EHR data. Prior literature usually first selected important features, such as diagnosis-frequencies [46], pairwise diagnosis transitions [49], and graphs of diagnosis sequences [51], to represent the EHR data of the patients.

Then, a wide range of supervised learning algorithms were adopted to build predictive models for early disease detection, on top of well-represented EHR data.

Specifically, supervised learning tools such as Linear Classification, Logistic Regression, Linear Discriminant Analysis (LDA), Decision Tree (DT), Random Forest (RF), and Bayesian Network [46, 49] have been adopted to train various predictive models, where a critical step is to learn model parameters from training dataset. However, from the viewpoint of "inverse problem" [37, 52, 53], learning parameters from training data is frequently ill-posed [54]. It is difficult to recover the patterns of causalities between variables (e.g., evidence of diagnosis in EHR data), when the number of training samples is limited but the dimension of EHR data (e.g., types of evidence used in prediction) is large. Such causalities consist of discriminative information and thus are the keys to build predictive models. For example, to train a linear classifier for discriminant projection, we need to first learn an optimal *Slope Vector*. Literature [55] has shown that when the size of training data is less than the dimension of the data (aka EHR data), the estimated slope vector would be "ill-posed" with weak capacity of discrimination, when using traditional Ordinary Least Squares (OLS) or Maximum Likelihood Estimation (MLE) estimator [56, 57]. In this case, the performance of such linear classifiers with ill-posed estimation of parameters will be degraded significantly [58]. Thus, we consider the key challenge of training predictive models for EHR-based early detection of diseases as a type of ill-posed inverse problem.

To understand the ill-posed inverse problem in machine learning, Vapnik and Chervonekis proposed Structural Risk Minimization (SRM) theory [59]. The SRM theory decomposes the error of predictive model into two parts: training error and generalization error. According to the SRM theory [60], the training of traditional models mainly focuses on minimizing the training error over the training set, without appropriately controlling the generalization error. To understand the generalizability of the model, they further proposed VC dimension [61] (Vapnik-Chervonenkis dimension) as a measure of potential generalization error, leveraging the complexity of the model.

More recently, they proposed the regularization method to balance training error and generalization error, with respect to the VC dimension of the trained model, to tackle the ill-posed inverse problem in parameter learning. Usually, these regularization methods intend to approximate the *sparse(st)* parameter estimation, while *lowering* the training error [62].

For example, to regularize linear classification, Support Vector Machine (SVM) [63] has been proposed to leverage the sparse estimation of the slope vector for discriminative linear projection, where a *Lasso* [64] estimator is used to balance the training error and ℓ_1-norm of the slope vector [65] (which is closely related to the VC dimension of linear classification model). Another example, to improve the performance of Logistic Regression [66], ℓ_1-norm regularization has been applied to balance the trade-off between training error and generalization error. Further, even for more complicated classification tools such as neural network [67], the regularization is frequently used to avoid over-fitting (control the generalization error) of the model.

2.2.2 Framework of CRLEDD

In this section, we introduce the *CRLEDD* framework. *CRLEDD* consists of three phases as shown in Figure 2.1. First, we use diagnosis-frequency vectors to represent the EHR data. Then, we estimate the covariance matrices used in LDA with respect to our problem formulation and estimate sparse covariance matrix via Graphical Lasso. After that, we adopt LDA with newly estimated parameters to predict whether the new patient will develop the targeted disease.

Phase I: EHR Data Representation — There are many existing approaches to represent EHR data including the use of diagnosis-frequencies [46, 47], pairwise diagnosis transition [49], and graph representations of diagnosis sequences [51]. Among these approaches, the diagnosis-frequency is a common way to represent EHR data.

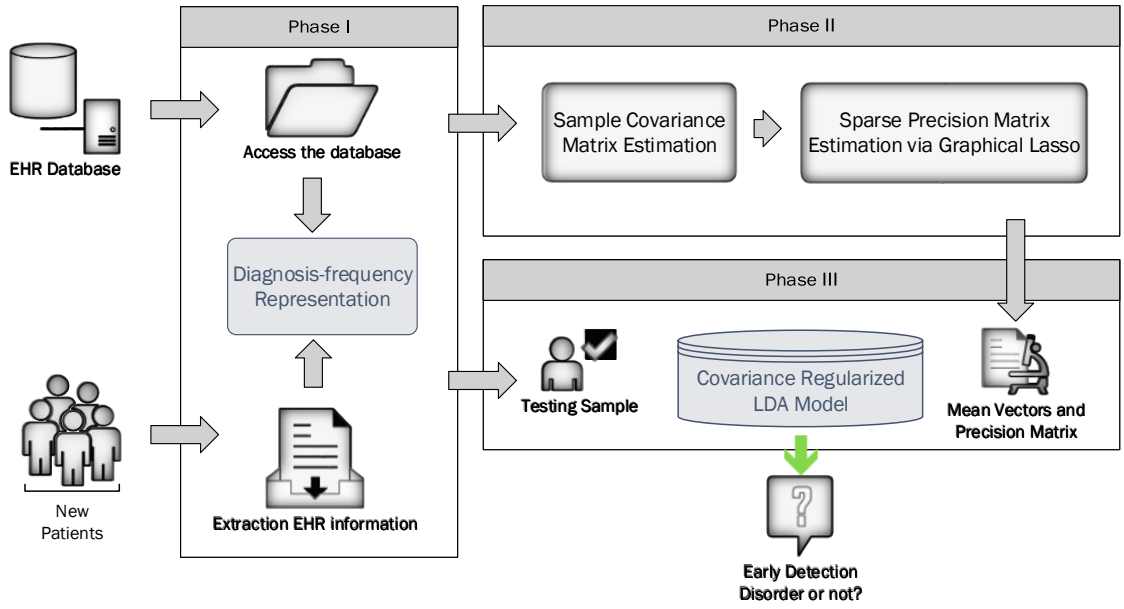


Figure 2.1: Overview of the three-phase framework: *CRLEDD* – Regularized Causalities Learning for Early Detection of Diseases using Electronic Health Record (EHR) Data. Depending on the functionality, the framework are divided into three phases which are **Data Representation**, **Correlation Analysis**, **Supervised Learning and Prediction**.

Given each patient's EHR data, the proposed method first retrieves the diagnosis codes [68] recorded during each visit. Next, the frequency of each diagnosis in all past visits is counted, followed by further transforming the frequency of each diagnosis into a vector of frequencies. For example, $\langle 1, 0, \ldots, 3 \rangle$, where 0 means that the second disease has not been diagnosed in any of the past visits. In this paper, we denote the dimension of diagnosis-frequency vectors as p. Note that the dimension $p \geq 15,000$ when using ICD-9 codes, $p \geq 250$ even when using clustered ICD-9 codes [69], while the number of samples for training m is significantly less than p.

Phase II: Correlation Analysis — Given the patients' EHR data as a training set, this phase estimates the sparse precision matrices for each type of the disease for two classes of patients (diagnosed with target disease or not) with following two steps:

1. **Sample Covariance Matrix Estimation with Extracted Diagnosis-frequency Vector —** *CRLEDD* combines diagnosis-frequency vector for each patient with his/her label (indicating whether the patient has been diagnosed with the targeted disease). Then we estimate the sample covariance matrices using maximized likelihood estimator.

2. **Sparse Precision Matrix Estimation Using Graphical Lasso —** Given sample covariance matrices $\bar{\Sigma}$, *CRLEDD* estimates the sparse precision matrix using Graphical Lasso estimator.

Note that the covariance matrices for the two classes of patients are estimated in this phase through a unified process.

Phase III: Supervised Learning and Prediction — Given the estimated matrices $\bar{\Sigma}$ as well as the training samples, this phase first trains the optimal model for LDA prediction. Then, it uses the LDA model for new patient prediction.

Given all parameters $\bar{\Sigma}$, $\bar{\mu}_{+1}$ (the mean vector of the sample consisting of the patients with target disease), and $\bar{\mu}_{-1}$ (the mean vector of sample consisting of the patients without target disease), the LDA model classifies a new patient's data x as the result of:

$$\underset{l \in \{+1,-1\}}{argmax} \left(x^T \bar{\Sigma}^{-1} \bar{\mu}_l - \frac{1}{2} \bar{\mu}_l^T \bar{\Sigma}^{-1} \bar{\mu}_l + \log \alpha_l \right), \tag{2.5}$$

where l is the label needs to be identified to predict if a certain patient is diagnosed with the target disease or not. l can be either positive one or negative one. Positive one means the patient will be predicted to have the target disease, while negative one means the patient will not be predicted to have the target disease. α_{+1} and α_{-1} refer to the empirical frequencies of positive samples (i.e., patients with the target disease) and negative samples (i.e., patients without the target disease) in the whole population.

2.2.3 Implementation of the Log-Divergence Minimization Algorithm via Graphical Lasso

Suppose we have m samples with dimension p and sample covariance matrix $\bar{\Sigma}$. In order to solve the optimization problem in Eq. 2.3 to obtain the $\hat{\Theta}$, the Graphical Lasso algorithm [35] is used to estimate $\hat{\Theta}^{-1}$ and recover $\hat{\Theta}$ after convergence. The details of this algorithm are listed as follow. Let $\mathbf{W} = \Theta^{-1}$ and $\mathbf{S} = \bar{\Sigma}$, then partitioning $\mathbf{W}$ and $\mathbf{S}$

$$\mathbf{W} = \begin{pmatrix} \mathbf{W}_{11} & w_{12} \\ w_{12}^T & w_{22} \end{pmatrix}, \mathbf{S} = \begin{pmatrix} \mathbf{S}_{11} & s_{12} \\ s_{12}^T & s_{22} \end{pmatrix} \tag{2.6}$$

The solution for w_{12} satisfies

$$w_{12} = \arg\min_{y} \left\{ y^T \mathbf{W}_{11}^{-1} y : |y - s_{12}|_\infty \leq \lambda \right\} \tag{2.7}$$

This is a box-constrained quadratic program that was once solved by Banerjee et al. [70] using an interior point procedure. It has been illustrated that the iterates in this procedure remain positive definite and invertible, even if $P > N$ when the procedure is initialized with a positive definite matrix. Thus, here the **SPD** of $\mathbf{W}$ can be ensured.

Using convex duality, Banerjee et al. [70] showed that solving Eq. 2.7 is equivalent to solving the dual problem

$$\min_{\beta} \left\{ \frac{1}{2} \left| \mathbf{W}_{11}^{\frac{1}{2}} \beta - b \right|^2 + \lambda \left| \beta \right|_1 \right\}, \tag{2.8}$$

where $b = \mathbf{W}_{11}^{\frac{1}{2}} s_{12}$. If β solves Eq. 2.8, then $w_{12} = \mathbf{W}_{11}\beta$ solves Eq. 2.7. Expression of Eq. 2.8 resembles a Lasso form, and is the basis for the approach of Graphical Lasso.

To verify the equivalence of the solutions between Eq. 2.3 and Eq. 2.8 directly, the relation $\mathbf{W}\Theta = \mathbf{I}$ can be expanded as below:

$$\begin{pmatrix} \mathbf{W}_{11} & w_{12} \\ w_{12}^T & w_{22} \end{pmatrix} \begin{pmatrix} \Theta_{11} & \theta_{12} \\ \theta_{12}^T & \theta_{22} \end{pmatrix} = \begin{pmatrix} \mathbf{I} & 0 \\ 0^T & 1 \end{pmatrix}. \tag{2.9}$$

Now the sub-gradient equation [71] for the maximization of the log-likelihood of Eq. 2.3 is

$$\mathbf{W} - \mathbf{S} - \lambda Sign(\Theta) = 0, \tag{2.10}$$

using the fact that the derivative of $\log\det(\Theta)$ equals $\Theta^{-1} = \mathbf{W}$.

The upper right block of the gradient equation from Eq. 2.10 is

$$w_{12} - s_{12} - \lambda Sign(\theta_{12}) = 0. \tag{2.11}$$

On the other hand, the sub-gradient equation from Eq. 2.8 works out to be

$$\mathbf{W}_{11}\beta - s_{12} + \lambda Sign(\beta) = 0, \tag{2.12}$$

where $w_{12} = -\mathbf{W}_{11}\theta_{12}/\theta_{22} = \mathbf{W}_{11}\beta$. The equivalence of the first two terms is obvious. For the sign terms, since $\mathbf{W}_{11}\theta_{12} + w_{12}\theta_{22} = 0$ from Eq. 2.10, we have that $\theta_{12} = -\theta_{22}\mathbf{W}_{11}^{-1}w_{12}$. Since $\theta_{22} > 0$, it follows that $Sign(\theta_{12}) = -Sign(\mathbf{W}_{11}^{-1}w_{12}) = -Sign(\beta)$. This proves the equivalence. Thus, we can solve the Lasso problem Eq. 2.8 instead of solving the original Eq. 2.3.

In terms of inner products, the lasso estimates for the pth variable on the others take $\mathbf{S}_{11}$ and s_{12} as the input data , where p is the dimension of the samples. To solve Eq. 2.8, we instead use $\mathbf{W}_{11}$

and s_{12}, where $\mathbf{W}_{11}$ is our current estimate of the upper block of $\mathbf{W}$. We then update w and cycle through all of the variables until convergence. The main steps of this estimation process are shown in the following Algorithm.

Algorithm: *The ℓ_1-penalized Log-divergence Minimization via Graphical Lasso*

1, **Initialize** $\mathbf{W} = \mathbf{S} + \lambda \mathbf{I}$. The diagonal of $\mathbf{W}$ remains unchanged in what follows.

2, **Repeat** for $j = 1, 2, ...p, 1, 2, ...p, ...$ **until** convergence:

(a) Partition the matrix $\mathbf{W}$ into two parts.
Part 1: all but the jth row and column.
Part 2: the jth row and column.

(b) Solve the estimating equation

$$\mathbf{W}_{11}\beta - s_{12} + \lambda Sign(\beta) = 0$$

using the cyclical coordinate-descent algorithm for the modified *Lasso*.

(c) Update $w_{12} = \mathbf{W}_{11}\hat{\beta}$.

3, In the final cycle (for each j) solve for $\hat{\theta}_{12} = -\hat{\beta} \cdot \hat{\theta}_{22}$, with $1/\hat{\theta}_{22} = w_{22} - w_{12}^T \hat{\beta}$.

Note that the *Lasso* [64] problem in step (b) above can be efficiently solved by cyclical coordinate-descent algorithm [72]. Here are the details. Let $V = \mathbf{W}_{11}$, then the update has the form

$$\hat{\beta}_i \leftarrow S((s_{12})_j - \sum_{k \neq j} V_{kj}\hat{\beta}_k, \lambda)/V_{jj} \tag{2.13}$$

for $j = 1, 2, ...p, 1, 2, ...p, ...$, where S is the soft-threshold operator:

$$S(x,t) = sign(x)(|x| - t)_+. \tag{2.14}$$

It cycles through the predictors until convergence.

Although step 2 has estimated $\hat{\Theta}^{-1} = \mathbf{W}$, it can recover $\hat{\Theta} = \mathbf{W}^{-1}$ relatively cheaply. Note that from the partitioning in Eq. 2.10, we have

$$\begin{aligned} \mathbf{W}_{11}\theta_{12} + w_{12}\theta_{22} &= 0 \\ w_{12}^T\theta_{12} + w_{22}\theta_{22} &= 1, \end{aligned} \tag{2.15}$$

from which we derive the standard partitioned inverse expressions

$$\begin{aligned} \theta_{12} &= -\mathbf{W}_{11}^{-1} w_{12}\theta_{22} \\ \theta_{22} &= 1/(w_{22} - w_{12}^T \mathbf{W}_{11}^{-1} w_{12}). \end{aligned} \tag{2.16}$$

According to Eq. 2.16, $\hat{\theta}_{22}$ and $\hat{\theta}_{12}$ can be easily computed in step 3. The Graphical Lasso algorithm stores all the coefficients β for each of the p problems in a $p \times p$ matrix, and compute $\hat{\theta}$ after convergence. As was discussed in [70], the estimator $\hat{\theta}$ should be Symmetric Positive-Definite (SPD) and Sparse. Furthermore, the recent work [73] leverages the similar method to estimate covariance matrix and proves its superiority under HDLSS settings.

2.2.4 Evaluation

In this section, we first introduce the data preprocessing based on the raw EHR data. After that, the existing algorithms that will be used as the baseline settings when comparing with *CRLEDD* are given. Then, the experimental results are demonstrated and discussed.

2.2.4.1 Data Preparation

To evaluate *CRLEDD*, we select the de-identified EHR data of 10 participating schools from the entire dataset including 31 student health centers across the U.S. with totally over 1 million patients and 6 million visits records provided by the College Health Surveillance Network (CHSN) [74]. The available information includes ICD-9 diagnostic codes, CPT procedural codes, and limited demographic information. There are over 200,000 enrolled students in those 10 schools representing all geographic regions of the U.S. The demography of enrolled students (sex, race/ethnicity, age, undergraduate/graduate status) in the selected dataset closely matches the demography of the students in the universities throughout the U.S.

We select the most common mental health disorders, anxiety and mood disorders from primary care data, as the target disease for early detection. Thousands of ICD-9 codes are clustered into 283 categories according to the AHRQ Clinical Classification Software and expert opinions [69]. We use his/her diagnosis-frequency vector based on the clustered code set to represent each patient, where four clustered codes (i.e., 651, 657, 658, 662) represent anxiety and mood disorders.

Note that in our research, we do not predict these four types of mental disorders separately, as these four disorders are often co-occurring in clinical practices [75]. Further, patients with less than two visits were excluded from the analysis.

Notably, the visit data and corresponding diagnosis information within one-month of the first diagnosis of anxiety/depression in the target group is excluded for the aim of early detection at least one to three-month prior to diagnosis. The diagnosis-frequency vectors are used as predictors in our experiment and only include the diagnosis frequency of non-mental health diagnoses with all mental health related information removed. In this case, our experiment is equivalent to predicting whether a patient is likely to have or develop a mental health disorder based on their diagnosis

history.

2.2.4.2 *Baseline Algorithms and Comparison Settings*

To understand the performance impact of *CRLEDD* beyond classic LDA, we first propose two kinds of baseline approaches to compare against *CRLEDD*, then two types of discriminative learning models are prepared for the comparison:

Regularized LDA Classifiers (three algorithms) – First, we use the typical *LDA* classifier, which employs the sample covariance estimation. Then, we consider the *Shrinkage* LDA [76] using shrinkage covariance estimator with the sparsity parameter β. Finally, we propose to use *DIAG*–a special *Shrinkage* with $\beta = 0.0$.

Downstream Classifiers (four algorithms) – We start with *Support Vector Machine* (SVM, with regularization parameter $C = 1.0$) [46], and then use *Logistic Regression* (Log. Reg.) [77]. Finally, we adopt two Adaboost classifiers ensembling 10 and 50 logistic regression classifiers (AdaBoost(10) and AdaBoost(50)).

With the seven baseline algorithms, we perform experiments with training samples and testing samples. We randomly select 50, 100, 150, 200, and 250 patients with mental health disorders as the positive **training samples**, and randomly select the same number of patients without a mental health diagnosis as negative training samples to maintain the balance. In terms of **testing samples**, we randomly select 500, 1000, 1500, and 2000 patients from each of the two patient classes (positive/negative) to build the testing set.

Then, we reveal the initial settings of some key parameters in proposed *CRLEDD* algorithm. The L1 regularization parameter λ is set to be 1, 10, 100 for comparison. The tolerance to declare convergence for graphical lasso is set to be 10^{-4}, and the number of maximum iteration for its

optimization is set to be 100. For each setting, we execute the seven algorithms and repeat 30 times. Then, we compare the accuracy and F1-Score of different algorithms.

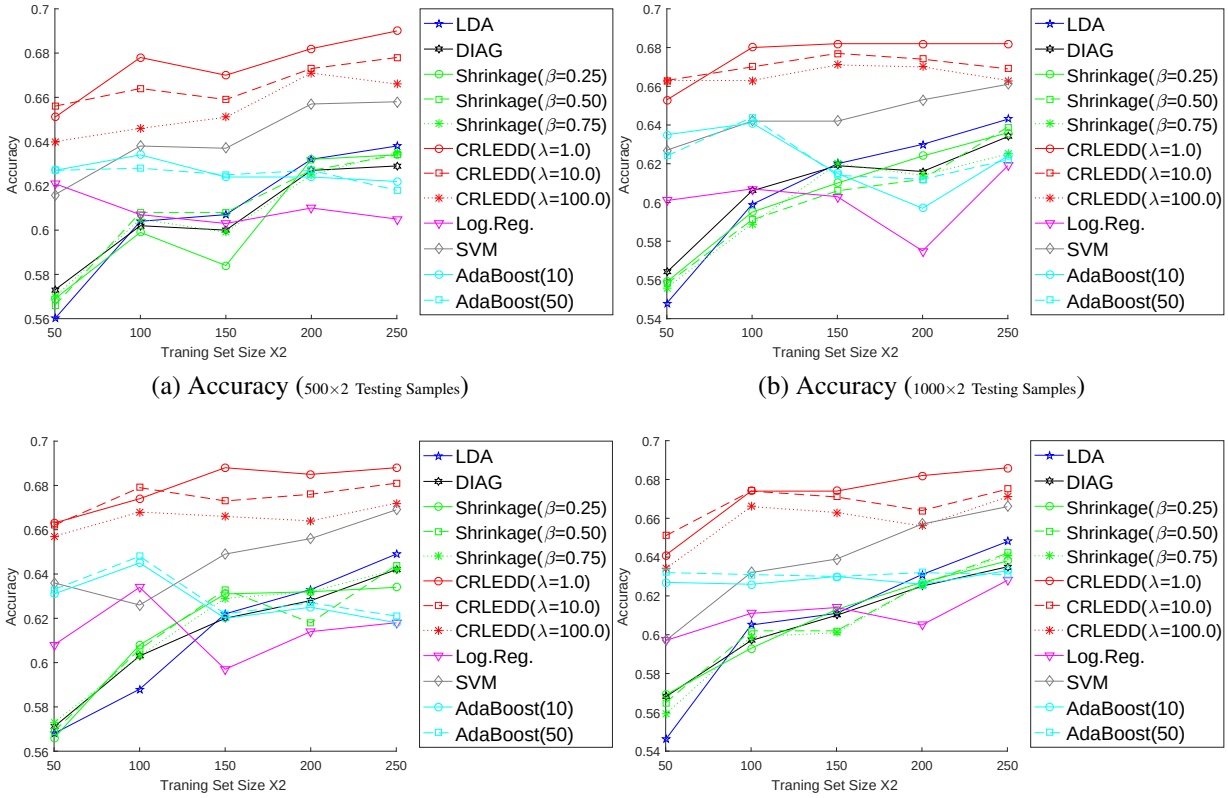
Also we perform an experiment to compare ℓ_1-norm error of estimator between LDA and *CRLEDD* with different sample sizes. Specifically, we randomly select 100 and 200 patients from each of the two patient classes (positive/negative) to build the **testing samples**.

2.2.4.3 Experiment Results

In this experiment, two types of comparison results are demonstrated:

1) ***Accuracy and F1-Score Comparison***: Figures 2.2 and 2.3 present the performance in terms of accuracy and F1-score of our method and baselines with various sizes of testing samples given different training sample sizes (more results are attached in the appendix). As can be seen from the experiment results, *CRLEDD* clearly outperforms the baseline algorithms in terms of overall accuracy, and F1-score, in all settings. Specifically, *CRLEDD* achieves 3.1%-20.9% higher accuracy and 11.7%-31.9% higher F1-score, compared to the typical LDA; *CRLEDD* achieves 7.5%-15.7% higher accuracy and 13.8%-41.9% higher F1-score, compared to the DIAG; *CRLEDD* achieves 6.7%-19.2% higher accuracy and 12.3%-71.6% higher F1-score, compared to the Shrinkage. Compared to those robust classifiers such as SVM, Logistic Regression, and AdaBoost, *CRLEDD* still clearly outperforms these baseline algorithms. Thus we can conclude that *CRLEDD* overall outperforms the baseline algorithms in all experimental settings.

2) ***Sensitivity and Specificity Comparison***: Table 2.1 additionally presents the performance with regards to sensitivity and specificity. The sensitivity is the percentage of patients who are correctly diagnosed as having the corresponding disease. As can be seen in the table, when training sample is 100 and testing sample is 1000, the sensitivity of *CRLEDD* is 0.842 in average, obviously higher

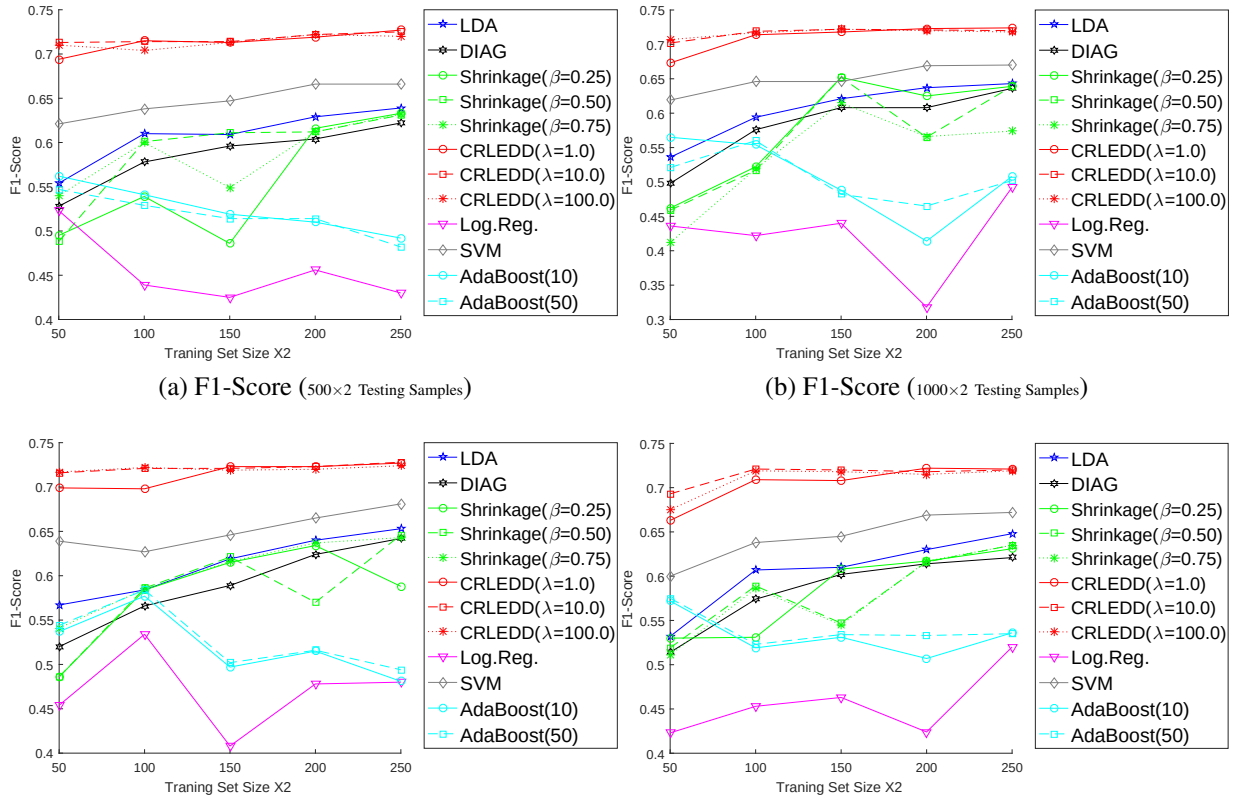


(a) Accuracy (500×2 Testing Samples) (b) Accuracy (1000×2 Testing Samples)

(c) Accuracy (1500×2 Testing Samples) (d) Accuracy (2000×2 Testing Samples)

Figure 2.2: Accuracy Performance Comparison between *CRLEDD* and Baselines with Small Training Datasets (Testing Sample Size $=500\times2$, 1000×2, 1500×2, 2000×2 from left top to right bottom, 90 days in advance).

than the sensitivity of SVM that have the highest value 0.633 among other baseline algorithms, which can explain that the *CRLEDD* has greater ability to correctly detect patients than the other baseline algorithms. The specificity which measures the proportion of people who are correctly identified as not having the disease, provided by the *CRLEDD* is lower than the other baseline algorithms. According to the table, *CRLEDD* achieves the highest value of the specificity as 0.510 when $\lambda = 1$, which is still lower than the LDA that have the lowest value 0.571 among other baseline algorithm. Similarly, this also occurs when the training sample is 500 and the testing sample is 4000. While, the *CRLEDD* is not better than the baseline algorithms in regards to specificity, it



(a) F1-Score (500×2 Testing Samples) (b) F1-Score (1000×2 Testing Samples)

(c) F1-Score (1500×2 Testing Samples) (d) F1-Score (2000×2 Testing Samples)

Figure 2.3: F1-Score Performance Comparison between *CRLEDD* and Baselines with Small Training Datasets (Testing Sample Size =500×2, 1000×2, 1500×2, 2000×2 from left top to right bottom, 90 days in advance).

performs better with regards to correctly identifying those individuals with the disease. Further, we expect a high number of false positives because mental health disorders are often unrecognized in primary care settings such as the student health centers. This oversight leads to adverse outcomes and higher costs when patients with anxiety/depression cannot receive proper treatment on time.

Trade-off. Moreover, we can observe that the *CRLEDD* sacrifices some specificity to achieve high sensitivity to some degree (33% gain in sensitivity VS 17% loss in Specificity when comparing with LDA). However, we see the utility of *CRLEDD* as an opportunity to perform psychological screening (e.g.; PHQ-9 [78]) in a primary care setting which could further identify the student's

Table 2.1: Sensitivity and Specificity Comparison

	Accuracy	F1-Score	**Sensitivity**	**Specificity**
Training Set:50× 2, Testing Set: 500×2				
AdaBoost (×10)	0.627 ± 0.045	0.562 ± 0.091	0.498 ± 0.135	0.756 ± 0.073
AdaBoost (×50)	0.627 ± 0.036	0.547 ± 0.091	0.471 ± 0.137	0.783 ± 0.072
CRLEDD ($\lambda = 1.0$)	$\mathbf{0.651} \pm 0.026$	$\mathbf{0.694} \pm 0.026$	$\mathbf{0.793} \pm 0.057$	$\mathbf{0.510} \pm 0.060$
CRLEDD ($\lambda = 10.0$)	$\mathbf{0.656} \pm 0.017$	$\mathbf{0.713} \pm 0.008$	$\mathbf{0.855} \pm 0.027$	$\mathbf{0.456} \pm 0.055$
CRLEDD ($\lambda = 100.0$)	$\mathbf{0.640} \pm 0.029$	$\mathbf{0.710} \pm 0.010$	$\mathbf{0.878} \pm 0.034$	$\mathbf{0.402} \pm 0.088$
LDA	0.560 ± 0.020	0.554 ± 0.032	0.548 ± 0.054	0.571 ± 0.046
Logistic Regression	0.621 ± 0.037	0.523 ± 0.100	0.440 ± 0.148	0.801 ± 0.081
SVM	0.616 ± 0.017	0.621 ± 0.029	0.633 ± 0.065	0.599 ± 0.064
DIAG	0.573 ± 0.023	0.528 ± 0.050	0.484 ± 0.076	0.662 ± 0.060
Shrinkage ($\beta = 0.25$)	0.569 ± 0.028	0.495 ± 0.169	0.469 ± 0.169	0.670 ± 0.122
Shrinkage ($\beta = 0.5$)	0.566 ± 0.025	0.488 ± 0.166	0.459 ± 0.164	0.672 ± 0.118
Shrinkage ($\beta = 0.75$)	0.570 ± 0.016	0.540 ± 0.039	0.509 ± 0.063	0.630 ± 0.045
Training Set:250× 2, Testing Set: 2000×2				
AdaBoost (×10)	0.633 ± 0.027	0.536 ± 0.089	0.447 ± 0.140	0.818 ± 0.086
AdaBoost (×50)	0.631 ± 0.026	0.535 ± 0.087	0.445 ± 0.137	0.818 ± 0.085
CRLEDD ($\lambda = 1.0$)	$\mathbf{0.686} \pm 0.006$	$\mathbf{0.721} \pm 0.009$	$\mathbf{0.813} \pm 0.029$	$\mathbf{0.558} \pm 0.026$
CRLEDD ($\lambda = 10.0$)	$\mathbf{0.675} \pm 0.007$	$\mathbf{0.720} \pm 0.006$	$\mathbf{0.838} \pm 0.021$	$\mathbf{0.512} \pm 0.028$
CRLEDD ($\lambda = 100.0$)	$\mathbf{0.671} \pm 0.009$	$\mathbf{0.719} \pm 0.004$	$\mathbf{0.844} \pm 0.028$	$\mathbf{0.497} \pm 0.043$
LDA	0.648 ± 0.009	0.648 ± 0.018	0.651 ± 0.037	0.644 ± 0.025
Logistic Regression	0.628 ± 0.028	0.520 ± 0.095	0.427 ± 0.146	0.828 ± 0.090
SVM	0.666 ± 0.009	0.672 ± 0.014	0.687 ± 0.030	0.644 ± 0.023
DIAG	0.635 ± 0.015	0.621 ± 0.030	0.601 ± 0.053	0.668 ± 0.032
Shrinkage ($\beta = 0.25$)	0.638 ± 0.012	0.631 ± 0.027	0.621 ± 0.051	0.656 ± 0.032
Shrinkage ($\beta = 0.5$)	0.642 ± 0.011	0.635 ± 0.026	0.626 ± 0.050	0.657 ± 0.032
Shrinkage ($\beta = 0.75$)	0.641 ± 0.010	0.635 ± 0.024	0.628 ± 0.046	0.655 ± 0.030

risk of a mental health disorder. Because of this, we focus more on correctly diagnosing those patients with the target disease.

3) ***Estimator Error Comparison***: We assume *CRLEDD* improves LDA because that the sparse precision matrix used in *CRLEDD* is more "precise" than the sample precision matrix used in simple LDA models when the training sample size is limited. Thus, we compare the ℓ_1-norm error of these two estimators and the results show that *CRLEDD* can always outperform with less error in different sample sizes. Figure 2.4 presents the average error between precision matrices in ℓ_1-norm. The results show that, compared to LDA (Σ_S^{-1}), the precision matrix estimated in *CRLEDD*

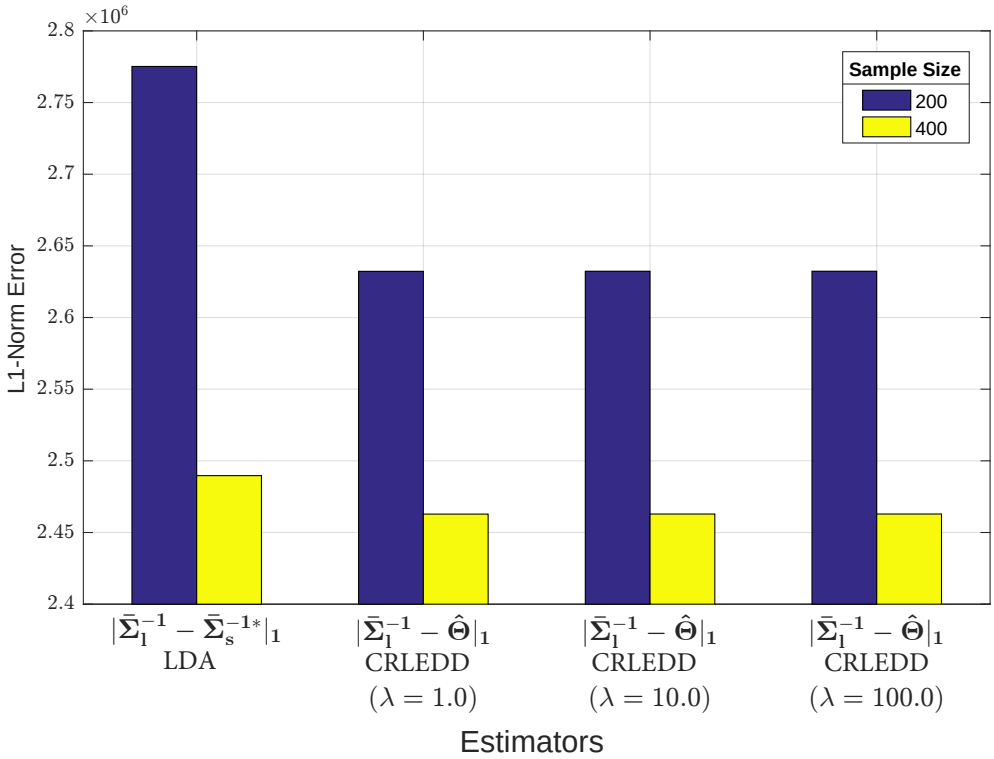


Figure 2.4: ℓ_1-norm Error Comparisons of Estimators on Different Sample Sizes

($\hat{\Theta}$) using small samples is **closer** to the precision matrix estimated using large samples. Note that we repeat the comparison in each setting for 30 times to estimate the average errors.

4) ***Causality Graph Visualization***: To validate the key algorithm of *CRLEDD*, we draw a causality graph based on the precision matrix in Eq. 2.3. Specifically, we randomly select a training set with 4000 balanced samples and threshold [79] the *Graphical Lasso* ($\lambda = 0.1$) at level

$$\Phi^{-1}(1 - \frac{\alpha}{p(p-1)})\hat{\sigma}_{ij}/\sqrt{n} \tag{2.17}$$

where $\alpha = 0.05$ and $\hat{\sigma}_{ij}^2 = \hat{\Theta}_{ii}\hat{\Theta}_{jj} + \hat{\Theta}_{ij}^2$. We leverage this threshold to pick up the strong causalities node pairs at the 95% significance level. As shown in Figure 2.5, each node in the graph represents a category of disorder and the thickness of the edge shows the intensity of the causal-

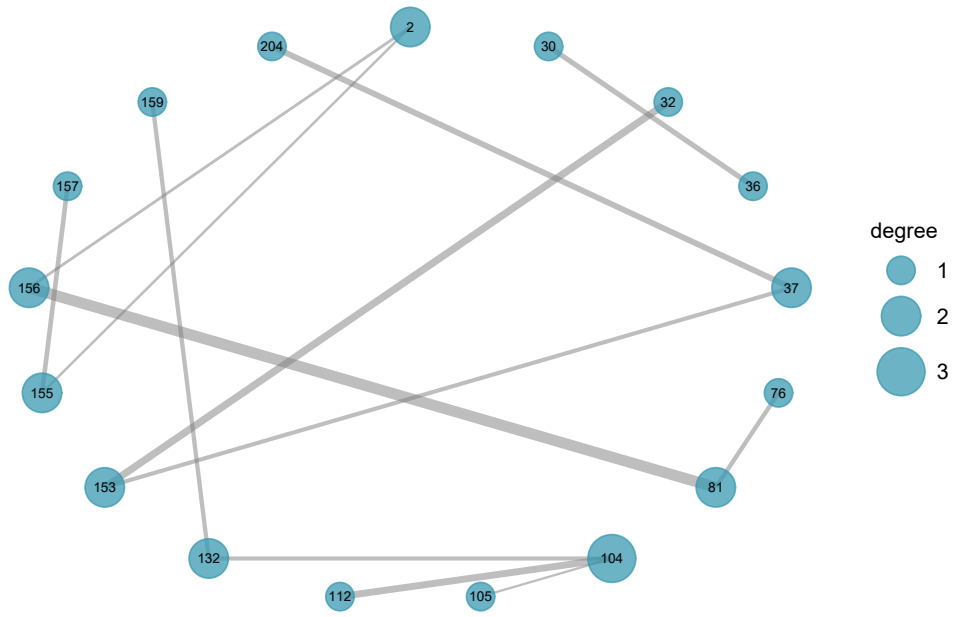


Figure 2.5: Causality Graph

ity. Further, we present the undirected disorder pairs by ranking their causality in the Figure 2.6. According to our results, we speculate that the disorders can be grouped into those that are related to anxiety and mood disorders such as other upper respiratory infections, other connective tissue diseases, and administrative/social admission. Other diagnoses are the ones that are unrelated to anxiety/depression such as immunizations and screening for infectious disease, and contraceptive and procreative management. We hypothesize that in the highest risk level that their are pairs of which both or only one of diagnoses are related to anxiety/depression in the higher risk group. For example, prior epidemiological studies suggest that upper respiratory infections affect mood and cognition, and psychological stress which is a significant risk factor for upper respiratory infections [80, 81]. Further clinical investigation is needed to fully understand these disorder pairs, but in general, these findings are informative for the early detection of anxiety/depression.

Level	Code One	Code Two
Highest	(156)Medical examination/evaluation	(81)Contraceptive and procreative management
Higher	(112)Other connective tissue disease	(104)Other non-traumatic joint disorders
Higher	(153)Allergic reaction	(32)Asthma
Higher	(204)Nutritional deficiencies	(37)Other upper respiratory disease
Higher	(30)Other upper respiratory infections	(36)Other lower respiratory disease
Middle	(153)Allergic reaction	(37)Other upper respiratory disease
Middle	(132)Sprains and Strains	(104)Other non-traumatic joint disorders
Middle	(132)Sprains and Strains	(159)Residual codes; unclassified
Middle	(76)Menstrual disorders	(81)Contraceptive and procreative management
Middle	(157)Other aftercare	(155)Administrative/Social admission
Lower	(156)Medical examination/evaluation	(2)Immunization and Screening for infectious disease
Lower	(155)Administrative/Social admission	(2)Immunization and Screening for infectious disease
Lowest	(105)Spondylosis; Intervertebral disc disorders; other back prob	(104)Other non-traumatic joint disorders

Figure 2.6: Causality Ranking of Disorders Pairs (Undirected).

2.2.4.4 Conclusion on Experiment Results

In the experiments, we evaluate *CRLEDD* using the empirical EHR datasets, and compare the algorithm with other classifiers under the same balanced dataset settings. The overall evaluation result shows that our algorithm significantly outperforms the existing linear discriminant analysis classifiers and other downstream classifiers, with both higher accuracy and F1-score. The case studies based on the estimated precision matrix show that the Graphical Lasso estimator used

in *CRLEDD* can reduce the ℓ_1-norm estimation error and improve the accuracy of classification, on top of the classical LDA classifiers. Further, we visualize the graph of casualties discovered from the data, which makes sense in the medical contexts [80, 81]. It is reasonable to conclude that, through lowering the error of precision matrix estimation, *CRLEDD* efficiently recovers the casualties between diagnoses related to the social anxiety/depression population from the data, then it improves the classification accuracy/F1-score by incorporating the well-recovered casualties. Note that our algorithm, along with all other baseline algorithms, is evaluated under balanced settings.

Efficiency Comparison. Also, we compare the time consumption of *CRLEDD* algorithm with most competitive algorithm SVM (500 patients for training and 2000 patients for testing). On average, *CRLEDD* takes 334.75 seconds for training and testing which is slightly more than the SVM algorithm (295.21 seconds) but achieve 15% better accuracy. (The experiment platform is Windows OS with 2.8GHz CPU).

2.3 *DBLD*: The De-Biased Estimation for Covariance-Regularized FLD

In this section, we introduce our proposed algorithm *DBLD*— De-Biased Fisher's Linear Discriminant Analysis (via Graphical Lasso), then present the theoretical analysis on the theoretical properties of the proposed algorithms.

Given the $i.i.d.$ labeled data pairs $(x_1, \ell_1) \dots (x_m, \ell_m)$ drawn from the two classes with certain priors, as shown in Algorithm . The algorithm first **(i)** estimates the sample estimation of covariance matrices and the mean vectors, then **(ii)** leverages CRLD to estimate the shrunken projection vector $\widehat{\beta}^G$. Further, *DBLD* **(iii)** proposes a de-biased estimator (denoted as DeBias function) to de-bias $\widehat{\beta}^G$ and obtain the projection vector $\widehat{\beta}^D$. Finally, we introduce a decision rule that enables

classification using the estimated $\widehat{\beta}^D$.

Algorithm: *DBLD* Estimation Algorithm (Algorithm 1)

procedure *DBLD*$((x_1,\ell_1)\ldots\ (x_m,\ell_m))$
/*(i) Sample Estimators for Mean and Covariance */
    $\mathbb{X}_+ \leftarrow$ PositiveSampleSet$((x_1,\ell_1)..(x_m,\ell_m))$;
    $\mathbb{X}_- \leftarrow$ NegativeSampleSet$((x_1,\ell_1)..(x_m,\ell_m))$;
    $\bar{\mu}_+ \leftarrow \frac{1}{|\mathbb{X}_+|}\cdot\sum_{x\in\mathbb{X}_+} x$, $\bar{\mu}_- \leftarrow \frac{1}{|\mathbb{X}_-|}\cdot\sum_{x\in\mathbb{X}_-} x$;
    $\bar{\Sigma}_+ \leftarrow \frac{1}{|\mathbb{X}_+|}\cdot\sum_{x\in\mathbb{X}_+}(x-\bar{\mu}_+)(x-\bar{\mu}_+)^\top$;
    $\bar{\Sigma}_- \leftarrow \frac{1}{|\mathbb{X}_-|}\cdot\sum_{x\in\mathbb{X}_-}(x-\bar{\mu}_-)(x-\bar{\mu}_-)^\top$;
    $\bar{\mu} \leftarrow \frac{|\mathbb{X}_+|\cdot\bar{\mu}_+ + |\mathbb{X}_-|\cdot\bar{\mu}_-}{|\mathbb{X}_+|+|\mathbb{X}_-|}$, $\bar{\Sigma} \leftarrow \frac{|\mathbb{X}_+|\cdot\bar{\Sigma}_+ + |\mathbb{X}_-|\cdot\bar{\Sigma}_-}{|\mathbb{X}_+|+|\mathbb{X}_-|}$;
/*(ii) CRLD Estimator (to obtain $\widehat{\beta}^G$) */
    $\widehat{\Theta} \leftarrow$ GraphicalLasso$(\bar{\Sigma},\lambda)$;
    $\widehat{\beta}^G \leftarrow \widehat{\Theta}(\bar{\mu}_+ - \bar{\mu}_-)$;
/*(iii) *DBLD* Estimator (to obtain $\widehat{\beta}^D$) */
    $\mathbf{X} \leftarrow [x_1, x_2, \ldots x_m]$; /*$p\times m$ matrix */
    $\mathbf{L} \leftarrow [\ell_1,\ell_2,\ldots\ell_m]^\top$; /*$m\times 1$ matrix */
    $\mathbf{U} \leftarrow [\bar{\mu},\bar{\mu},\ldots\bar{\mu}]$;
    /*$\mathbf{U}$ is an $m\times p$ matrix, every column is $\bar{\mu}$*/
    $c \leftarrow -\bar{\mu}^\top\widehat{\beta}^G$;
    $\mathbf{C} \leftarrow [c,c,\ldots,c]^\top$;
    /*$\mathbf{C}$ is a $m\times 1$ matrix, every row is c*/
    $\widehat{\beta}^D \leftarrow \widehat{\beta}^G + \frac{1}{m}\cdot\widehat{\Theta}(\mathbf{X}-\mathbf{U})\left(2\cdot\mathbf{L}-\mathbf{X}^\top\widehat{\beta}^G-\mathbf{C}\right)$
    return $\widehat{\beta}^D$;
end procedure

In the following section, we present the design of the De-Biased Estimator (denoted as DeBiasing function in Algorithm) to obtain $\widehat{\beta}^D$, then introduce the decision rule for optimal classification. Later we analyze the theoretical properties of $\widehat{\beta}^D$.

2.3.1 The De-Biased Estimator

Inspired by the De-biased Lasso [82], we propose to improve the performance of CRLD through de-biasing β^G. Given m labeled training data $(x_1,\ell_1),(x_2,\ell_2),\ldots(x_m,\ell_m)$ with balanced labels,

the Graphical Lasso estimator $\widehat{\Theta}$ on the data and the CRLD model (i.e., $\hat{\beta}^G$), we propose a novel de-biased estimator of $\hat{\beta}^D$ that takes the form as

$$\widehat{\beta}^D \leftarrow \widehat{\beta}^G + \frac{1}{m} \cdot \widehat{\Theta}(\mathbf{X} - \mathbf{U}) \left(2 \cdot \mathbf{L} - \mathbf{X}^\top \widehat{\beta}^G - \mathbf{C}\right), \tag{2.18}$$

where we denote $\mathbf{X}$ as an $p \times m$ matrix where $1 \leq i \leq m$ and the ith column is x_i; $\mathbf{L}$ as an $m \times 1$ matrix (i.e., vector) whose i^{th} row is $\ell_i \in \{\pm 1\}$; $\mathbf{U}$ is a $p \times m$ matrix where each column is $\bar{\mu}$ (as line 7 in Algorithm); and $\mathbf{C}$ is an $m \times 1$ matrix where each row is c (as line 16 in Algorithm).

2.3.1.1 The DBLD Classifier

Given the de-biased estimator $\widehat{\beta}^D$, the *DBLD* classifies the input vector x using the following rule:

$$\widehat{f}^D(x) = \text{sign}\left(\left(x^\top - \frac{\bar{\mu}_+ + \bar{\mu}_-}{2}\right)^\top \widehat{\beta}^D + \log(\pi_+/\pi_-)\right). \tag{2.19}$$

In the following section, we present the analytical results of *DBLD*, including the computational complexity of de-biasing and statistical rate of convergence.

2.3.1.2 Complexity Analysis of DBLD

In this section, we analyze the computational complexity for the three steps of **Algorithm** 1. The step (i) estimates the sample covariance matrices and mean vectors, which consumes at most $\mathscr{O}(p^2 \cdot m)$ operations. The step (ii) performs Graphical Lasso and matrix multiplication, where the complexity based on standard implementation [35] is upper-bounded by $\mathscr{O}(p^3)$. The step (iii) de-biasing is implemented as an exact formula with $\mathscr{O}(p^2)$ complexity.

Remark 1. All three steps of **Algorithm** 1 are scalable on both the number of dimensions (p) and

the number of training samples (m). The overall complexity of the three steps is $\mathscr{O}(p^3 + p^2 \cdot m)$. Under the HDLSS setting $p > m$, the computational complexity of *DBLD* is upper-bounded by $\mathscr{O}(p^3)$. On the other hand, with large sample setting where $m \geq p$, the worst case computational complexity of *DBLD* is bounded by $\mathscr{O}(p^2 \cdot m)$. Obviously, the proposed de-biasing estimator (i.e., step (iii)) with complexity $\mathscr{O}(p^2)$ would not bound the performance, when compared to the first two steps.

2.3.1.3 Convergence Analysis of DBLD

In order to analyze the performance of *DBLD*, we first define the linear projection vector of the optimal FLD as β^*. Given m samples $(x_1, \ell_1), \ldots (x_m, \ell_m)$ drawn i.i.d. from $\mathscr{N}(\mu_+^*, \Sigma^*)$ and $\mathscr{N}(\mu_-^*, \Sigma^*)$ with the equal priors for training, the optimal projection vector should be $\beta^* = \Theta^*(\mu_+^* - \mu_-^*)$ and $\Theta^* = \Sigma^{*-1}$. We intend to understand how close $\widehat{\beta}^G$ and $\widehat{\beta}^D$ approximate to the optimal estimation β^*.

Assumption 1. *We follow the assumptions made in [83] that a positive constant K having*

$$1/\mathscr{K} \leq \lambda_{min}(\Sigma^*) \leq \lambda_{max}(\Sigma^*) \leq \mathscr{K}$$

exists. The operators $\lambda_{min}(\cdot)$ and $\lambda_{max}(\cdot)$ denote the smallest and largest eigenvalues respectively. In this way, there exists $\|\Sigma^*\|_2 \leq \mathscr{K}$ and $\|\Theta^*\|_2 \leq \mathscr{K}$.

Assumption 2. *We further follow the assumption that, the data vectors for training are all realized from a random vector X and there exists an constant $\mathscr{B}$ having $|X|_2 \leq \mathscr{B}$. Thus there has $|\bar{\mu}_+|_2 \leq \mathscr{B}$ and $|\bar{\mu}_-|_2 \leq \mathscr{B}$.*

Theorem 1. *With appropriate setting of tuning parameter $\lambda \asymp \sqrt{\log p/m}$ (in Eq 2.3), the ℓ_2-vector-*

norm convergence rate of CRLD $\widehat{\beta}^G$ approximating to the optimal estimation β^* is:

$$|\widehat{\beta}^G - \beta^*|_2 = \mathscr{O}_p\left(\sqrt{\frac{(p+d)\log p}{m}}\right), \tag{2.20}$$

where $d = max_{1\leq i\leq p}|\{j : \Sigma^{*^{-1}}_{i,j} \neq 0\}|$ refers to the maximal degree of the graph (i.e., population inverse covariance matrix).

Proof. Here, we first prove the upper bound of $|\widehat{\beta}^G - \beta^*|_\infty$. As was defined $\widehat{\beta}^G = \widehat{\Theta}(\bar{\mu}_+ - \bar{\mu}_-)$, then we have:

$$|\widehat{\beta}^G - \beta^*|_2 = |\widehat{\Theta}(\bar{\mu}_+ - \bar{\mu}_-) - \Theta^*(\mu^*_+ - \mu^*_-)|_2. \tag{2.21}$$

Considering the inequities $|x+y|_2 \leq |x|_2 + |y|_2$ and $|Ax|_2 \leq ||A||_2 \cdot |x|_2$, we have

$$\begin{aligned} |\widehat{\beta}^G - \beta^*|_2 \leq & ||(\widehat{\Theta} - \Theta^*)||_2 \cdot |\bar{\mu}_+ - \bar{\mu}_-|_2 \\ & + \|\Theta^*\|_2 \left(|\bar{\mu}_+ - \mu^*_+|_2 + |\bar{\mu}_- - \mu^*_-|_2\right). \end{aligned} \tag{2.22}$$

According to [83], when $\lambda \asymp \sqrt{\log p/m}$, we consider the spectral-norm convergence rate $\|\widehat{\Theta} - \Theta^*\|_2 \leq \|\widehat{\Theta} - \Theta^*\|_F = \mathscr{O}_p(\sqrt{(p+d)\cdot \log\ p/m})$, the asymptotic rate of sample mean vector [84] is $|\bar{\mu}_+ - \mu^*_+|_2 = \mathscr{O}_p(\sqrt{p/m})$ and $|\bar{\mu}_- - \mu^*_-|_2 = \mathscr{O}_p(\sqrt{p/m})$, with the increasing number of dimensions p and number of samples m.

Further, there has $\|\Theta^*\|_2 \leq \mathscr{K}$ (**Assumption 1**) and ℓ_2-norms of all mean vectors are bounded by $\mathscr{B}$. In this way, there must exist positive constants C_1 and C_2 having:

$$|\widehat{\beta}^G - \beta^*|_2 \leq C_1 \cdot 2\mathscr{B}\sqrt{\frac{(p+d)\log\ p}{m}} + C_2\mathscr{K}\sqrt{\frac{p}{m}}. \tag{2.23}$$

Thus, according to the definition of asymptotic rate, we conclude the convergence rate as:

$$|\widehat{\beta}^G - \beta^*|_2 = \mathscr{O}_p\left(\sqrt{\frac{(p+d)\log\ p}{m}}\right). \tag{2.24}$$

□

Theorem 2. *With appropriate setting of tuning parameter λ (in Eq 2.3), the ℓ_2-vector-norm convergence rate of DBLD $\widehat{\beta}^G$ approximating to the optimal estimation β^* is:*

$$|\widehat{\beta}^D - \beta^*|_2 = \mathscr{O}_p\left(\sqrt{\frac{p\log\ p}{m}}\right). \tag{2.25}$$

Proof. Here, we prove the upper bound of $|\widehat{\beta}^D - \beta^*|_\infty$. Consider the definition of the de-biased FLD estimator $\widehat{\beta}^D$ introduced in Eq. 2.18, we have

$$\widehat{\beta}^D = \widehat{\beta}^G + \frac{2}{m}\cdot\widehat{\Theta}\mathbf{XL} - \frac{2}{m}\cdot\widehat{\Theta}\mathbf{UL} - \frac{1}{m}\cdot\widehat{\Theta}(\mathbf{X}-\mathbf{U})(\mathbf{X}-\mathbf{U})^\top\widehat{\beta}^G. \tag{2.26}$$

With the assumption of equal priors ($\pi_+ = \pi_- = 0.5$), $\mathbf{L}$ is a $m\times 1$ label matrix that half of its elements are $+1$ while the rest are all -1. $\mathbf{X}$ refers to a $p\times m$ matrix, where each column is a sample of data e.g., $x_1, x_2, \ldots, x_m$. As was defined $\widehat{\beta}^G = \widehat{\Theta}(\bar{\mu}_+ - \bar{\mu}_-) = \frac{2}{m}\cdot\widehat{\Theta}\mathbf{XL}$. As $\mathbf{U}$ is a matrix in which each column is a constant vector $(\bar{\mu}_+ + \bar{\mu}_-)/2$ and $\mathbf{L}$ is a vector with half elements as 1 and half elements as -1, thus $\frac{2}{m}\cdot\widehat{\Theta}\mathbf{UL} = \frac{2}{m}\cdot\widehat{\Theta}(\mathbf{UL}) = \mathbf{0}$. As each column of $\mathbf{X}$ refers to a sample drawn from the original data distribution, thus $\frac{1}{m}(\mathbf{X}-\mathbf{U})(\mathbf{X}-\mathbf{U})^\top = \bar{\Sigma}_s$ is the sample covariance matrix estimator. With all above in mind, we have

$$\widehat{\beta}^D = \widehat{\beta}^G + \left(\mathbf{I} - \widehat{\Theta}\bar{\Sigma}_s\right)\widehat{\beta}^G, \tag{2.27}$$

where $\mathbf{I}$ refers to a $p\times p$ identity matrix. Note that $\left(\mathbf{I} - \widehat{\Theta}\bar{\Sigma}\right)\widehat{\beta}^G$ can be considered as the de-

sparsification term that de-biases $\widehat{\beta}^G$.

Thus, considering the asymptotic rate of sample mean vector [84] is $|\bar{\mu}_+ - \mu_+^*|_2 = \mathscr{O}_p(\sqrt{p/m})$ and $|\bar{\mu}_- - \mu_-^*|_2 = \mathscr{O}_p(\sqrt{p/m})$, we have

$$\begin{aligned} |\widehat{\beta}^D - \beta^*|_2 &\leq \left|\left|\left(2\cdot\mathbf{I} - \widehat{\Theta}\bar{\Sigma}_s\right)\widehat{\Theta} - \Theta^*\right|\right|_2 |\bar{\mu}_+ - \bar{\mu}_-|_2 + |\Theta^*(\bar{\mu}_+ - \mu_+^* - \bar{\mu}_- + \mu_-^*)|_2 \\ &\leq 2\mathscr{B}\left|\left|\left(2\cdot\mathbf{I} - \widehat{\Theta}\bar{\Sigma}_s\right)\widehat{\Theta} - \Theta^*\right|\right|_2 + C_2\mathscr{K}\sqrt{\frac{p}{m}}. \end{aligned} \tag{2.28}$$

According to [79], with appropriate setting of λ, the spectral-norm convergence rate of the de-sparisified estimator $\widehat{\Theta}^D = \left(2\cdot\widehat{\Theta} - \widehat{\Theta}\bar{\Sigma}_s\widehat{\Theta}\right)$ under mild conditions should be $\|\widehat{\Theta}^D - \Theta^*\|_\infty = \mathscr{O}_p(\sqrt{\log\ p/m})$, then there exists $\|\widehat{\Theta}^D - \Theta^*\|_2 = \mathscr{O}_p(\sqrt{p\log\ p/m})$, with the varying number of dimensions p and number of samples m. In this way, with high probability, we conclude the convergence rate:

$$|\widehat{\beta}^D - \beta^*|_2 = \mathscr{O}_p\left(\sqrt{\frac{p\log\ p}{m}}\right). \tag{2.29}$$

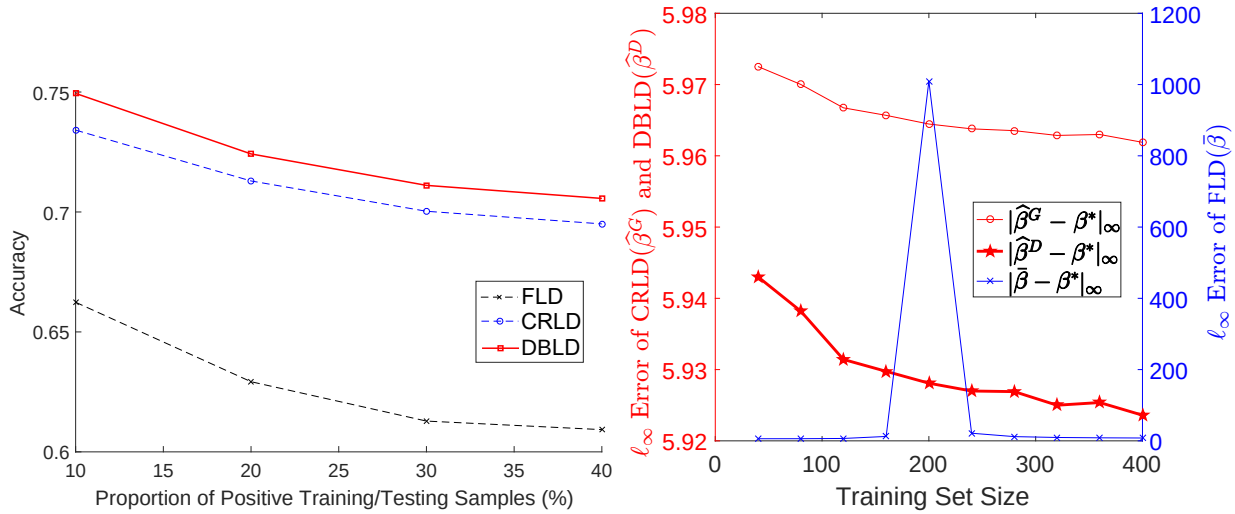
□

Remark 2. Compared to CRLD's projection vector $\widehat{\beta}^G$, our method *DBLD* recovers the linear projection vector $\widehat{\beta}^D$ with a faster asymptotic rate, i.e., $\sqrt{p\log\ p/m}$ v.s. $\sqrt{(p+d)\log\ p/m}$ in a mild condition. Thus, it would benefit to some applications, such as dimensionality reduction and feature selection. Our later experimental results show that *DBLD* outperforms CRLD with higher classification accuracy, due to the faster statistical rate of convergence.

Remark 3. The proposed algorithm provides a *sub-optimal* solution, when compared to [20]. Our work intend to propose an estimator of β^* through approximating Σ^*, μ_+^* and μ_-^* separately, while [20] approximates $\widehat{\beta}^*$ straightforwardly via so-called "direct estimation".

2.3.2 Evaluation

To validate our algorithms, we evaluate our algorithms on a synthesized dataset (imported from [20]), 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 each element $\Sigma_{i,j}^* = 0.8^{|i-j|}$, $1 \leq i \leq p$ and $1 \leq j \leq p$. 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 $\mu_-^* = \mathbf{0}$. In our experiment, we set $p = 200$. To simulate the HDLSS settings, we train CRLD and *DBLD*, with 20 to 200 samples randomly drawn from the distributions with equal priors, and test the two algorithms using 500 randomly generated samples. For each settings, we repeat the experiments for 100 times and report the averaged results, in a cross-validation manner.



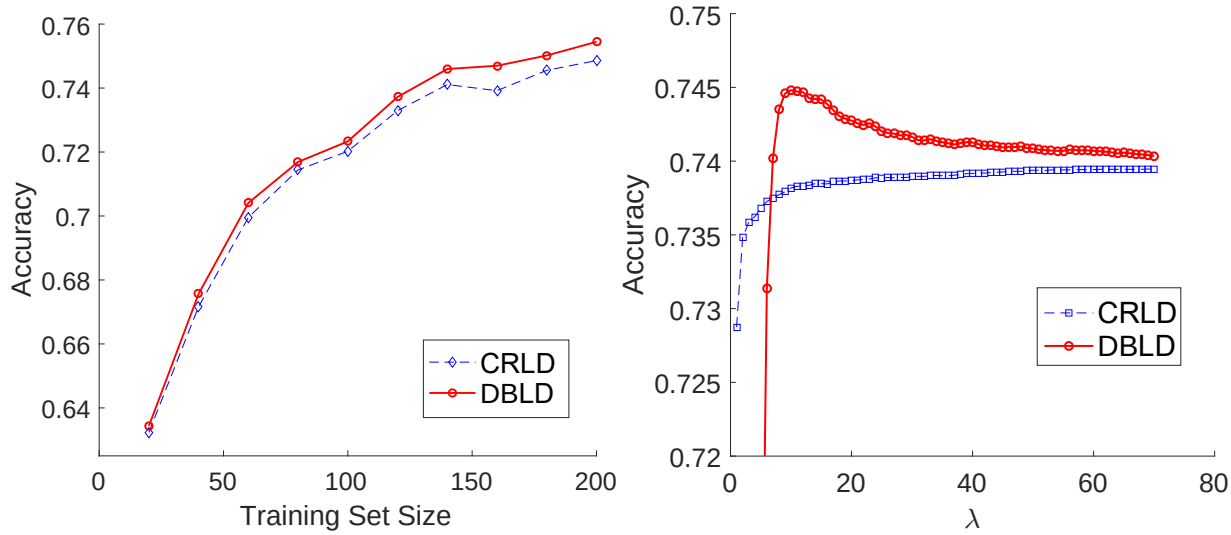
(a) Accuracy on Unbalanced Datasets ($m = 160$)

(b) Asymptoticity

Figure 2.7: More Performance Comparison based on Pseudo-Random Synthesized Data

In this experiment, we compare *DBLD*, CRLD and FLD (with pseudo inverse). The results of FLD is not included here, as it performs extremely worse than both CRLD and *DBLD* under the HDLSS

settings. Figure. 2.8(a) presents the comparison between *DBLD* and CRLD, in terms of accuracy, where each algorithm is fine tuned with the best parameter λ. A detailed example of parameter tuning is reported in Figure. 2.8(b), where we run both algorithms, with training set size as 160, when varying λ from 1 to 70. From Figure. 2.8(a), it is obvious that *DBLD* outperforms CRLD marginally. The λ tuning comparison addressed in Figure. 2.8(b) shows that, given a small λ, both CRLD and *DBLD* cannot perform well, as the sparse approximation of $\widehat{\beta}^G$ and $\widehat{\beta}^D$ cannot be well recovered in such case [32]. When $\lambda \geq 6$, *DBLD* starts outperforming CRLD, while the advantage of *DBLD* to CRLD decreases when increasing λ. However, even with an extremely large λ, *DBLD* still outperforms CRLD. In Figure 2.7(a), we present the evaluation results based on unbalanced datasets, where the accuracy of algorithms using $m = 160$ training samples drawn with varying priors is illustrated. The proportion of positive training samples is varying from 10% to 40%. It is obvious that all algorithms achieve their best performance when the proportion of positive training sample is 10% (the most unbalanced case).

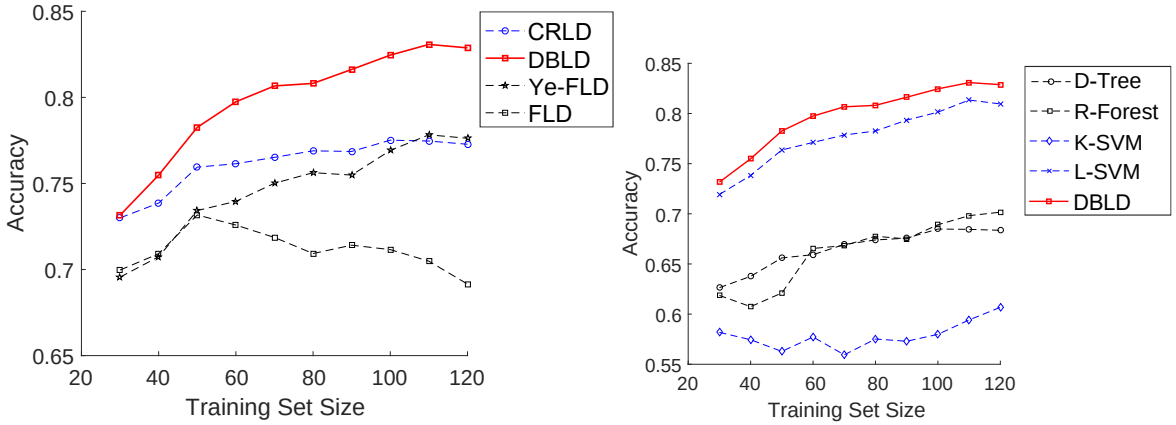


(a) *DBLD* vs. CRLD

(b) λ Tuning

Figure 2.8: Classification Accuracy of *DBLD* vs. CRLD on Pseudo-Random Synthesized Data

To further verify our algorithms, we propose the optimal FLD classifier $\beta^* = \Theta^*(\mu_+^* - \mu_-^*)$, which is all based on the population parameters. We compare the $\widehat{\beta}^D$, $\widehat{\beta}^G$ and $\bar{\beta}$ estimated by *DBLD*, CRLD and FLD (with pseudo-inverse) to β^*. Figure. 2.7(b) presents the comparison among $|\widehat{\beta}^D - \beta^*|_\infty$, $|\widehat{\beta}^G - \beta^*|_\infty$ and $|\bar{\beta} - \beta^*|_\infty$. It is obvious that $\widehat{\beta}^D$ is more close to β^* than $\widehat{\beta}^G$ and $\bar{\beta}$. This observation further verifies the **Theorem 1** and **2**. We also compare the accuracy of β^* to CRLD, *DBLD* and FLD. β^* outperforms these algorithms and the accuracy of β^* is around 84.4% It is reasonable to conclude that *DBLD* outperforms CRLD, because $\widehat{\beta}^D$ is more close to β^*.



(a) *DBLD* vs. FLDs (b) *DBLD* vs. Downstream

Figure 2.9: Performance Comparison on Benchmark Datasets ($p = 300$ and $p \gg m$, D-Tree: Decision Tree, R-Forest: Random Forest, K-SVM: Kernel SVM, and L-SVM:Linear SVM)

2.3.2.1 Benchmark Evaluation Results

In Figure. 2.9(a), we compare *DBLD* and other FLD algorithms, including FLD with pseudo-inverse, Sparse FLD via Graphical Lasso (CRLD) and Ye-FLD derived from [29], on the Web datasets [85]. To simulate the HDLSS settings ($p \gg m$), we vary the training sample sizes from 30 to 120 while using 400 samples for testing. The numbers of dimensions p is 300. For each algorithm, reported result is averaged over 100 randomly selected subsets of the training/testing data

Table 2.2: Early Detection of Diseases Accuracy Comparison between *DBLD* and Baselines.

	Training Set Size						
Algorithm	100	200	300	400	500	600	700
DBLD	**0.659±0.022**	**0.677±0.028**	**0.691±0.024**	**0.692±0.023**	**0.690±0.021**	**0.696±0.024**	**0.701±0.023**
FLD	0.543±0.034	0.586±0.033	0.616±0.022	0.642±0.029	0.642±0.022	0.657±0.025	0.658±0.026
Ye-FLD	0.627±0.050	0.620±0.077	0.652±0.063	0.620±0.067	0.655±0.062	0.637±0.064	0.670±0.045
Decision Tree	0.621±0.046	0.649±0.031	0.652±0.041	0.655±0.030	0.671±0.028	0.665±0.031	0.668±0.040
Linear SVM	0.615±0.026	0.628±0.030	0.647±0.023	0.666±0.029	0.666±0.021	0.670±0.030	0.675±0.029
Kernel SVM	0.635±0.032	0.669±0.027	0.674±0.039	0.678±0.021	0.668±0.038	0.688±0.024	0.682±0.029
AdaBoost	0.631±0.035	0.630±0.039	0.620±0.028	0.622±0.027	0.621±0.022	0.617±0.025	0.626±0.070
CRLD	0.658±0.023	0.676±0.024	0.682±0.028	0.686±0.022	0.683±0.021	0.692±0.025	0.695±0.018
Random Forest	0.590±0.035	0.602±0.035	0.653±0.031	0.602±0.040	0.674±0.024	0.666±0.026	0.658±0.032

Table 2.3: Early Detection of Diseases F1-Score Comparison between *DBLD* and other Baselines.

	Training Set						
Algorithm	100	200	300	400	500	600	700
DBLD	0.690±0.028	0.708±0.027	**0.722±0.024**	**0.729±0.018**	**0.727±0.0118**	**0.736±0.018**	**0.734±0.022**
FLD	0.539±0.048	0.580±0.044	0.611±0.030	0.646±0.027	0.644±0.025	0.662±0.028	0.663±0.032
Ye-FLD	0.644±0.100	0.657±0.124	0.688±0.071	0.678±0.057	0.698±0.035	0.698±0.035	0.712±0.027
Decision Tree	0.626±0.120	0.671±0.074	0.675±0.088	0.703±0.032	0.695±0.034	0.676±0.078	0.690±0.097
Linear SVM	0.616±0.031	0.627±0.041	0.651±0.026	0.675±0.031	0.675±0.026	0.680±0.035	0.690±0.031
Kernel SVM	**0.701±0.063**	**0.723±0.022**	0.702±0.115	0.726±0.016	0.681±0.115	0.734±0.019	0.715±0.071
AdaBoost	0.560±0.081	0.533±0.107	0.498±0.065	0.503±0.078	0.500±0.080	0.482±0.066	0.503±0.070
CRLD	0.696±0.021	0.716±0.021	0.719±0.024	0.725±0.018	0.721±0.015	0.733±0.021	0.734±0.016
Random Forest	0.419±0.126	0.509±0.102	0.613±0.067	0.509±0.110	0.661±0.036	0.640±0.058	0.603±0.063

with equal priors. CRLD and *DBLD* are fine-tuned with the best λ. The experimental settings show that *DBLD* consistently outperforms other competitors in different settings. The non-monotonic trend of FLD with the increasing training set size is partially due to the poor performance of pseudo inverse used in FLD.

In addition to FLD classifiers, we also compared *DBLD* with other downstream algorithms including *Decision Tree*, *Random Forest*, *Linear Support Vector Machine (SVM)* and *Kernel SVM with Gaussian Kernel*. The comparison results are listed in Figure. 2.9(b). All algorithms are fine-tuned with the best parameters under our experiment settings.

2.3.2.2 Early Detection of Diseases on EHR Datasets

To demonstrate the effectiveness of *DBLD* in handling the real problems, we evaluate *DBLD* on the real-world Electronic Health Records (EHR) data for early detection of diseases [49]. In this application, each patient's EHR data is represented by a $p = 295$ dimensional vector, referring to the outpatient record on the physical disorders diagnosed. Patients are labeled with either "positive" or "negative", indicating whether he/she was diagnosed with depression & anxiety disorders. Through supervised learning on the datasets, the trained binary classifier is expected to predict whether a (new) patient is at-risk or would develop to the depression & anxiety disorders from their historical outpatient records (physical disorder records) [49].

We evaluate *DBLD* and other competitors, including Linear Support Vector Machine, Nonlinear SVM with Gaussian Kernel, Decision Tree, AdaBoost, Random Forest and other FLD baselines, with varying training dataset size m from 100 to 700. Table 2.2 presents the comparison results. To simplify the comparison, we only present the results of the algorithm with fine-tuned parameter, which is selected through 10-fold cross-validation. It is obvious that *DBLD* and CRLD outperform other baseline algorithms significantly, while *DBLD* performs better than CRLD. The advantage of *DBLD* over other algorithms, such as SVM, is extremely obvious when the size of training dataset m is small. With the increasing sample size, though the margins of *DBLD* over the rest of algorithms decrease, *DBLD* still outperforms other algorithms. We also measured the F1-score of all algorithms, *DBLD* still outperforms other competitors in the most cases. Please refer to

Table 2.4: Description of Datasets for Classification

Datasets	# Features	# Samples
Leukemia	7,128	72 (47 / 25)
Colon	2,000	62 (40 / 22)

Table 2.3 for details.

2.3.2.3 Leukemia and Colon Cancer Datasets

We evaluate *DBLD*, CRLD and other baseline algorithms, including Decision Tree, Random Forest and SVM, using leukemia and colon cancer datasets (derived from [85,86]) under HDLSS settings (i.e., $p = 7,128$ and $2,000$ vs. $m = 20$).

Table 2.4 presents the description of two datasets [85, 86] that we used to evaluate the proposed and baseline algorithms. "Leukemia" refers to the leukemia cancer dataset [86] that includes 7,128 features and totally 72 samples (for training and testing). In this datasets, 47 samples are labeled as "ALL" class while 25 samples are identified as "AML". On the other hand, "Colon" refers to the colon cancer datasets [85] that are with 2,000 features and totally 62 samples, where 40 samples are negative and 22 samples are identified as positive. Both datasets are with a ultra-large number of dimensions but with extremely low sample sizes (i.e., $p \gg m$).

To accurately estimate the performance of algorithms using these datasets under HDLSS settings, we use cross-validation to limit the potential over-fitting. In each round of cross-validation, we first randomly drawn 20 samples with equal prior from the datasets as the training set, and randomly drawn 20 samples with equal prior from the disjoint set of training set as the testing set. For each round of cross validation, there are no common samples shared by the two sets. We use

Table 2.5: Accuracy and F1-Score Comparison between *DBLD* and other Baselines Based on Colonar and Leuk Cancer Datasets.

	Colon		**Leukemia**	
Algorithm	Accuracy	F1 Score	Accuracy	F1 Score
DBLD	**0.803**	**0.802**	**0.964**	**0.964**
CRLD	0.633	0.630	0.690	0.690
Decision Tree	0.669	0.658	0.804	0.800
Random Forest	0.801	0.798	0.957	0.956
SVM	0.797	0.812	0.906	0.914

the training set to train each classifier (i.e., $p = 7,128$ or $2,000$ and $m = 20$), so as to simulate the extremely HDLSS settings, then test the trained classifiers using the testing set. For each experiment, we repeat the cross-validation for 100 rounds. All algorithms (including baselines and *DBLD*) are tuned to have the best accuracy. The experiment results are shown in Table 2.5. All results show that *DBLD* significantly improves CRLD, and it outperforms all baseline algorithms with the highest accuracy and F1-score. Please note that though we trained classifiers using less training data, baselines in our experiments perform comparably with the test errors reported in [86].

2.3.2.4 Summary of Experiment Results

We evaluate *DBLD* with a limited number of samples for training i.e., $p > m$ or $m \not\gg p$, to understand its performance under HDLSS setting. For large sample scenario, i.e., when $m \gg p$, the sample-based estimators may provide a robust estimation of LDA. In this case, singularity issues might not exist, then regularization and further the de-biasing procedures are not mandatory.

CHAPTER 3: MULTI-PARTY SPARSE DISCRIMINANT ANALYSIS FOR DISTRIBUTED INTELLIGENT MEDICAL SYSTEMS

The Fisher's Linear Discriminant Analysis (LDA) [22] is a method to find the linear combination of features that separates two or multiple classes, where it can be used in supervised learning 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) [20] has been proposed with sparsity pursuit. While a wide variety of methods [20, 32, 87–90] have been proposed, Cai et al. [20] 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 [9,10] 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 [3, 11–17]. 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 [11, 15, 18]. To advance the distributed performance of classical SDA, recently, Tian and Gu et al. [19] 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 paper, we propose Multi-Party Parallelized SDA (namely MP^2SDA) that enables the direct estimation of SDA [20] 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. 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 stochastic gradient

descent algorithm is employed to minimize the distributed bootstrapping loss function and estimate the global discriminant projection vector. Compared with the approach in [20], 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 [19], which aggregates the locally learned models through model-averaging and hard-thresholding, MP^2SDA 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 [21], MP^2SDA additionally processing parallel computing (multiple threads) when estimating the model parameters to improve the performance with fast convergence rate.

3.1 Backgrounds

In this section, we first present the model of Fisher's Linear Discriminant Analysis (LDA). Then, we introduce the Direct Estimation of sparse linear discriminant analysis (SDA) proposed by Cai et al [25]. Then we address the robust estimator under uncertain parameters using the "bootstrapping loss function" minimization. Finally, we formulate the research problem of this paper.

3.1.1 Sparse Linear Discriminant Analysis

Fisher's LDA Model: Linear Discriminant Analysis (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 [20]. 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

$$\psi_F(Z) = sign\Big\{(Z-\mu)^T\Theta(\mu_+ - \mu_-)\Big\}, \tag{3.1}$$

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; $\psi_F(Z)$ 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 [91] with respect to the two classes. Note that, under the High Dimensional Low Sample Size (HDLSS) settings, $\bar{\Sigma}$ is often singular [54] and $\bar{\Sigma}^{-1}$ usually does not exist [92]. Thus, to train LDA, researchers [20,88] 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 eq. 3.1, Cai and Liu (2011) [20] proposed a direct estimation method for sparse linear discriminant analysis by estimating β through a constrained ℓ_1 minimization method:

$$\underset{\beta \in \Re^p}{\operatorname{argmin}}\Bigg\{|\beta|_1 \quad s.t. |\bar{\Sigma}\beta - (\bar{\mu}_+ - \bar{\mu}_-)|_\infty \leq \varepsilon\Bigg\}, \tag{3.2}$$

where ε is a tuning parameter.

3.1.1.1 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 $\mathscr{L}(\omega|\theta)$, where θ is an unknown parameter following a known probabilistic distribution with density function $\mathscr{P}(\theta)$. 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

$$\omega^* = \underset{\omega}{\operatorname{argmin}} \mathbb{E}_{\theta \sim \mathscr{P}} \left(\mathscr{L}(\omega|\theta) \right). \tag{3.3}$$

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

$$\widehat{\omega}_m = \underset{\omega}{\operatorname{argmin}} \sum_{i=1}^{m} \left(\mathscr{L}(\omega|\theta_i) \right) / m. \tag{3.4}$$

As $\theta_1, \theta_2, ..., \theta_m$ are drawn from the distribution randomly, the sum of loss functions can approximate the expectation of loss function under Central-Limit Theorem [93] with large m, where $\lim_{m \to \infty} \widehat{\omega}_m = \omega^*$. Recent studies [94, 95] show that the bootstrapping loss function minimization can obtain a robust estimation of ω under the uncertainty of θ.

3.1.1.2 Stochastic Gradient Descent

In order to solve the optimization problem in Eq. 3.4, 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:

$$\omega_{t+1} \leftarrow \omega_t - \eta \cdot \sum_{i=1}^{m} \bigtriangledown \mathscr{L}(\omega_t|\theta_i)/m, \tag{3.5}$$

where η refers to the step size and $\sum_{i=1}^{m} \bigtriangledown \mathscr{L}(\omega|\theta_i)$ 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, Stochastic Gradient Descent (SGD) algorithm has been proposed to solve the optimization problem in Eq. 3.4 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 $\mathscr{L}(\theta_t|\theta_i)$. Specifically, in the iteration, SGD randomly selects an integer $i \in [1,m]$, then it updates ω using

$$\omega_{t+1} \leftarrow \omega_t - \eta \cdot \bigtriangledown \mathscr{L}(\omega_t|\theta_i). \tag{3.6}$$

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 stochastic gradient descent through exchanging the (updated) ω between machines to approximate the optimal solution.

3.1.1.3 Parallelized Stochastic Gradient Descent

To further accelerating the optimization process, we leverage the Parallelized SGD framework to solve the optimization problem in Eq. 3.4. Suppose the SGD algorithm can be regarded as a single thread with the index k, we reclaim the Eq. 6 as

$$\omega_{t+1}^k \leftarrow \omega_t^k - \eta \cdot \bigtriangledown \mathscr{L}(\omega_t^k|\theta_i), \tag{3.7}$$

where $k \in \{1...S\}$, S is the size of multiple threads (Leaders). Note that each kth thread runs an independent SGD algorithm and the kth 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

$$\bar{\omega} \leftarrow \frac{1}{S}\sum_{k=1}^{S} \omega_k. \tag{3.8}$$

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 [96], the Parallelized SGD algorithm is converging to a stationary distribution exponentially faster than the traditional stochastic gradient descent. Also, the guarantees for stationary distribution achieving have been proved [96].

3.1.1.4 Problem Formulation

Given m machines, where each (the jth) 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 Eq. 3.2, 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 jth) 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

$$\widehat{\beta}^* = \underset{\beta \in \Re^p}{\operatorname{argmin}} \left\{ |\beta|_1 \quad s.t. |\Sigma\beta - (\mu_+ - \mu_-)|_\infty \leq \varepsilon \right\}. \tag{3.9}$$

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.

3.2 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 jth) 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 stochastic gradient descent 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 jth) 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 $\widehat{\Sigma}^j$ ($1 \leq j \leq m$) 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 $sign\left((Z-\widehat{\mu}^*)^T\widehat{\beta}_T^*\right)$.

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

3.2.1 Multi-Party Message Passing Mechanism

As shown in Fig. 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 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 received (marked with received on top on machine block)the message will update the target value, while the machine not received 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.

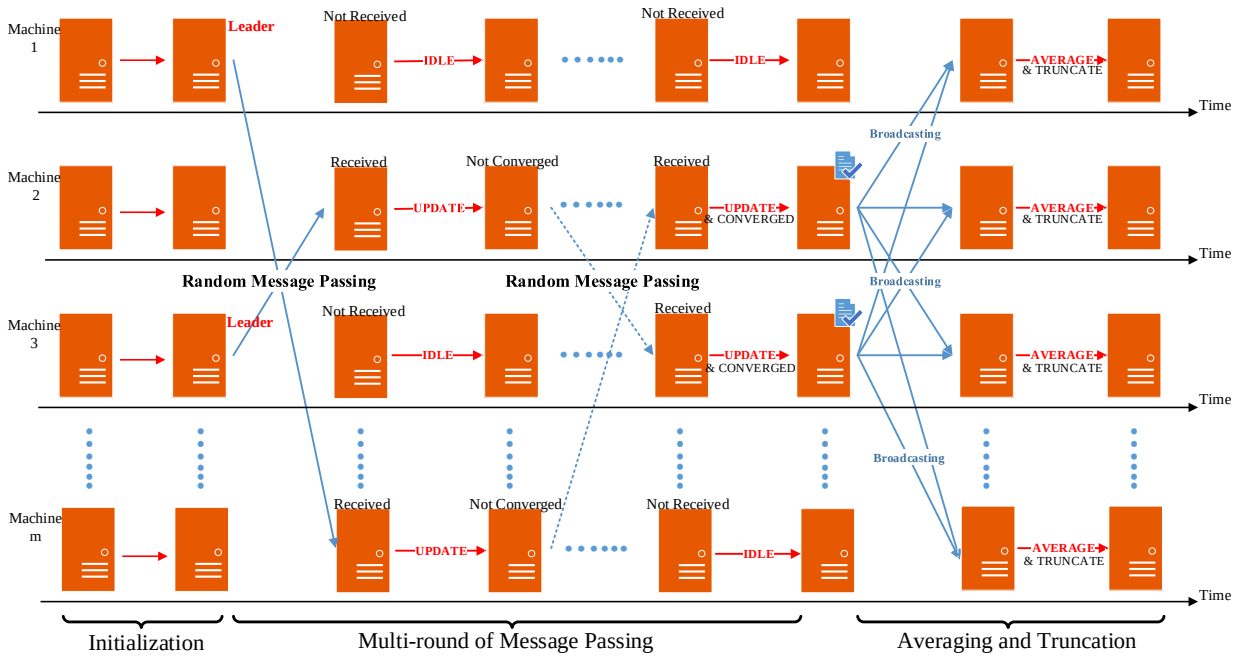


Figure 3.1: Multi-Party Random Message Passing Mechanism

3.2.1.1 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 $\mathscr{L}_{\mathscr{S}}$ with size $S \leq m$) through function $LeaderSetSelection()$, where each machine in $\mathscr{L}_{\mathscr{S}}$ 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.

Algorithm: Leader Selection on the jth Machine (Algorithm 1)

begin

$\mathscr{L}_{\mathscr{S}} \leftarrow LearderSetElection(S)$;

if $j \in \mathscr{L}_{\mathscr{S}}$ **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

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 Eq. 3.10.

$$\widehat{\mu} = \underset{\mu \in \Re^{1\times p}}{\operatorname{argmin}} \frac{1}{m} \sum_{j=1}^{m} |\mu - \mu^j|_\infty,\ \widehat{\mu}_+ = \underset{\mu \in \Re^{1\times p}}{\operatorname{argmin}} \frac{1}{m} \sum_{j=1}^{m} |\mu - \mu_+^j|_\infty,\ \widehat{\mu}_- = \underset{\mu \in \Re^{1\times p}}{\operatorname{argmin}} \frac{1}{m} \sum_{j=1}^{m} |\mu - \mu_-^j|_\infty, \tag{3.10}$$

Specifically, the Algorithm 1 first receives the input mean vectors (initialed as $\mathbf{0}$ 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., $max\left\{\left|\widehat{\mu} - \bar{\mu}^j\right|_\infty, \left|\widehat{\mu}_+ - \bar{\mu}_+^j\right|_\infty, \left|\widehat{\mu}_- - \bar{\mu}_-^j\right|_\infty\right\} \leq \Delta_{max}$). 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:

$$\left(\nabla|\widehat{\mu}-\bar{\mu}^j|_\infty\right)_k = \begin{cases} sign((\widehat{\mu}-\bar{\mu}^j)_k), & \text{if } |(\widehat{\mu}-\bar{\mu}^j)_k| \text{ is the maximal for } 1 \leq k \leq p \\ 0, & else \end{cases} \tag{3.11}$$

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

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

Data:
$\bar{\mu}^j$, $\bar{\mu}^j_+$, and $\bar{\mu}^j_-$ — the local mean vectors based on training samples on the jth Machine
Parameter:
η — step size
Δ_{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

3.2.1.2 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 High-Dimensional Low Sample Size settings, the algorithm leverages the de-sparsified Graphical Lasso estimator [79] ($\widehat{D}^j$) to improve the estimation of the inverse covariance matrix. Finally, matrix inverse is used to estimate the covariance matrix $\widehat{\Sigma}^j$ on the jth 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.

3.2.1.3 Stage III: Sparse Discriminant Projection Vector Estimation

Given the local covariance matrix $\widehat{\Sigma}$ on each machine j and the averaged global mean vectors $\widehat{\mu}_+^*$, $\widehat{\mu}_-^*$, 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 through gossip-based stochastic gradient decent:

$$\widehat{\beta}^* \leftarrow \underset{\beta \in \mathbb{R}^p}{argmin}\ \lambda \cdot |\beta|_1 + \frac{1}{m} \sum_{j=1}^{m} \left| \widehat{\Sigma}^j \beta - (\widehat{\mu}_+ - \widehat{\mu}_-) \right|_\infty, \tag{3.12}$$

Algorithm: Local Covariance Matrix Estimation (with Global Mean) on the jth Machine (Algorithm 3)

Data:
T^j — training sample on $j = 1, 2, ..., m$ machine
Parameter:
λ — Graphical Lasso regularization parameter
begin
  RECEIVE and **AVERAGE** $(\widehat{\mu}, \widehat{\mu}_+, \widehat{\mu}_-)_i$ from all machines;
  `/* i ∈ {1,2...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 [97] */`
  $\widehat{\Theta}^j \leftarrow glasso(\Sigma^j, \lambda)$
  `/* 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 */`
  $\mathscr{L}_{\mathscr{S}} \leftarrow LearderSetElection(S)$;
  if $j \in \mathscr{L}_{\mathscr{S}}$ **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

where λ is a regularization parameter. Specifically, Algorithm 4 first receives the input $\widehat{\beta}^*$ for updating (initialed as $\mathbf{0}$ 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. 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.

Algorithm: Averaging and Truncating on the jth Machine (Algorithm 5)

begin

RECEIVE and **AVERAGE** $\widehat{\beta}_i^*$ from all machines;

`/* i ∈ {1,2...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

3.2.2 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 jth) machine should follow an inverse wishart distribution $\mathscr{W}^{-1}(\Sigma, v(n))$, where $v(n)$ is a function on n for the degree of freedom. With infinite number of machines $m \to \infty$ and infinite number of gossip message passing (i.e., $t \to \infty$), the algorithm can converge to the minimum of $\widehat{R}(\beta)$ (as the loss function $\widehat{R}$ is convex [20]), where

$$\widehat{R}(\beta) = \mathbb{E}_{\Sigma \sim \mathscr{W}^{-1}(\Sigma, v(n))}\left(\lambda \cdot |\beta|_1 + |\Sigma\beta - (\widehat{\mu}_+ - \widehat{\mu}_-)|_\infty\right). \tag{3.13}$$

According to the definition of Bayes estimator [98], this loss function can be viewed as a Bayes Es-

timator based on the posterior expectation on risk. We first denote the optimal solution of original sparse LDA listed Eq. 3.2, based on the population parameter Σ and μ_+/μ_-, as β^*_{SDA}. Regarding to the asymptotic efficiency of the Bayes estimator, we conclude:

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

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 stochastic gradient descent 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 $\mathscr{O}(S \cdot p \cdot T)$.

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 [20] using all aggregated data, with similar Accuracy and F1-Score.

3.3 Evaluation

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 [20]. For distributed SDA, we adopt the algorithm proposed by [19] 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.

3.3.1 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 $\Sigma_{i,j}^* = 0.8^{|i-j|}$, $1 \leq i \leq p$ and $1 \leq j \leq p$. 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 [19].) In order to evaluate the performance of algorithms for comparison, we obtain the accuracy, F1-score, ROC curve and 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 [20], 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^2SDA, 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 [19] 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^2SDA 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 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 $N = 20000$ and vary

the number of machines. Figure 2 shows how the accuracy, F1-score and AUC of MP^2SDA (we use MP2SDA 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^2SDA algorithm outperforms distributed SDA algorithm on accuracy, F1-score and AUC. It is unsurprising that centralized SDA outperforms both MP^2SDA and distributed SDA on accuracy, F1-score and AUC.

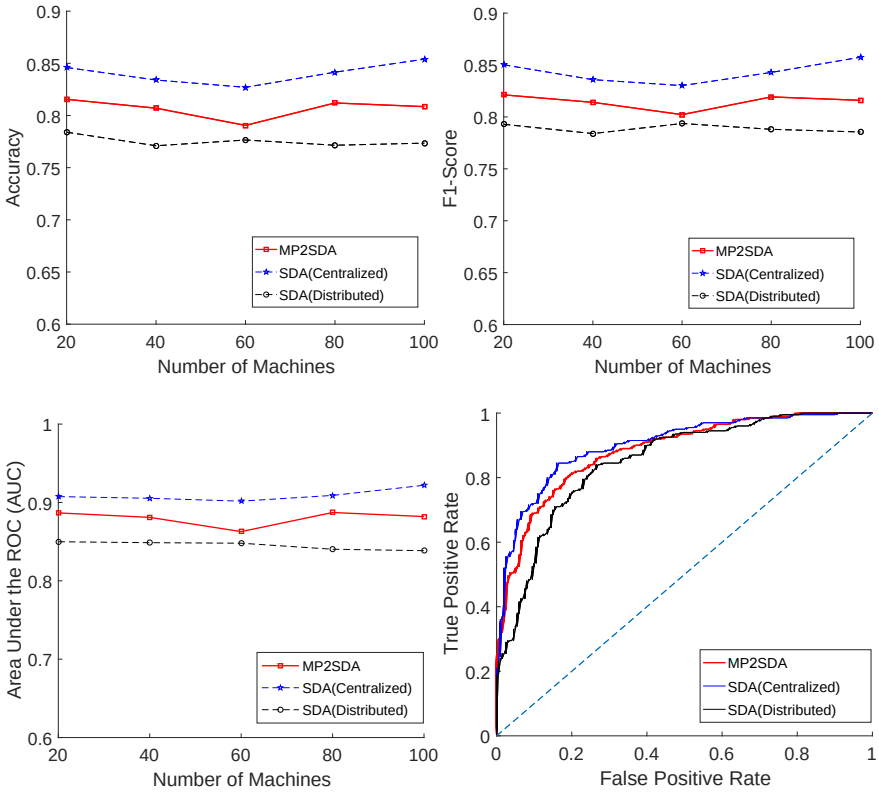


Figure 3.2: Performance Comparison among MP^2SDA, SDA(centralized) and SDA(Distributed) on synthetic datasets. We compare the Accuracy, F1-Score, AUC and ROC curve of each algorithm **when the total training sample size is fixed as** 20000. (Note that the ROC curve is drawn when the number of machines is 100)

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^2SDA still outperforms distributed SDA algorithm on accuracy, F1-score and AUC. Similarly, centralized SDA outperforms both MP^2SDA and distributed SDA algorithm. We notice that the performance of MP^2SDA 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.

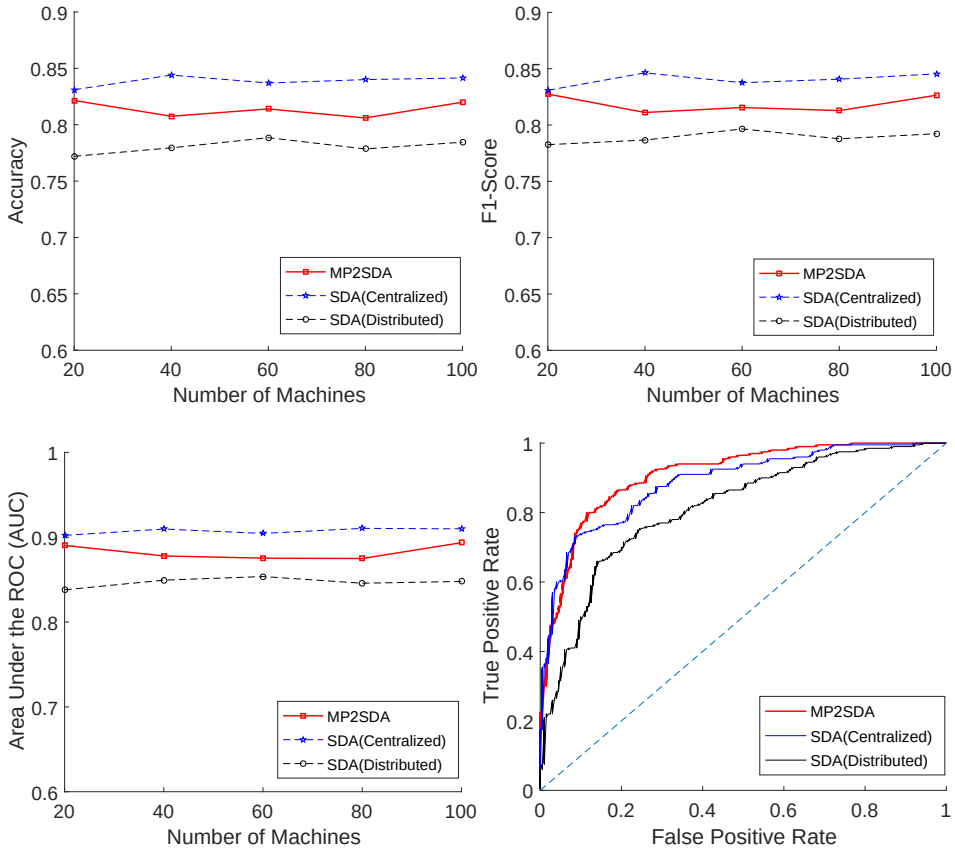


Figure 3.3: Performance Comparison among MP^2SDA, SDA(centralized) and SDA(Distributed) on synthetic datasets. We compare the Accuracy, F1-Score, AUC and ROC curve of each algorithm **when the training sample size on each machine is set as** 400. (Note that the ROC curve is drawn when the number of machines is 100)

Supplement – Receiver Operating Characteristic (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^2SDA 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^2SDA overlaps with or even covers the ROC curve of centralized SDA in Figure 2, which shows that the performance of MP^2SDA 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^2SDA 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^2SDA 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^2SDA outperforms distributed SDA significantly.

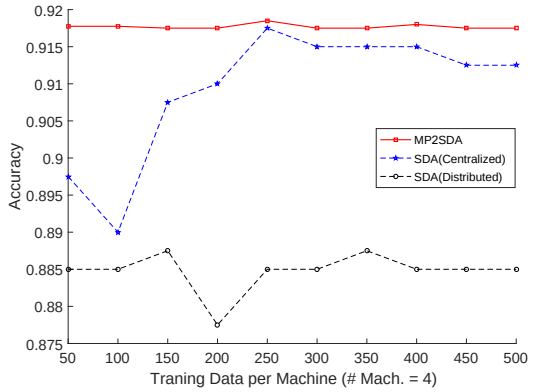
Moreover, according to the stable trends of each of the indicators (accuracy, F1-Score and AUC), we can conclude that the parallelism or the distributed assignment does not harm the overall performance and reach the saturation interval for our specific settings. Then, the stable performance provides 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.

3.3.2 Benchmark Data Experiments

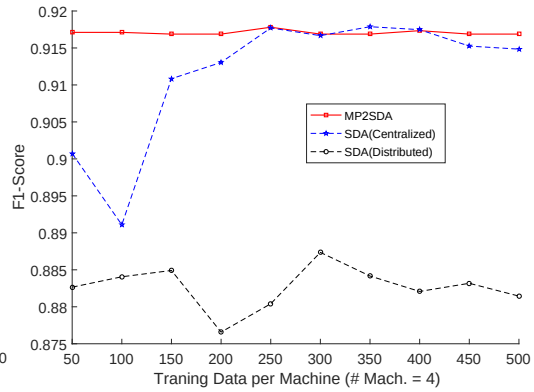
Experiment Setup: To verify the effectiveness of MP^2SDA algorithm on real datasets, we use Phishing, Splice and Mushrooms datasets [99] to conduct the comparison. Specifically, we set the size of total training samples varied from 200 to 2000 with 400 testing samples, while the numbers of dimensions 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^2SDA, $\lambda = 15$ 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^2SDA with distributed SDA and centralized SDA on each benchmark datasets. Figure. 4(a)(b) presents the performance of each algorithm on Phishing datasets. We can observe that MP^2SDA 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^2SDA is still comparable to centralized SDA and obviously superior to distributed SDA. Figure. 4(c)(d) shows that MP^2SDA outperforms distributed SDA and centralized SDA on Mushrooms dataset. The performance gap between MP^2SDA and the other two alternatives tends to be stable when the training sample size grows. In Figure. 4(e)(f), the performances of these three algorithms are close to each other on Splice dataset. In most cases, MP^2SDA slightly outperforms distributed SDA and centralized SDA.

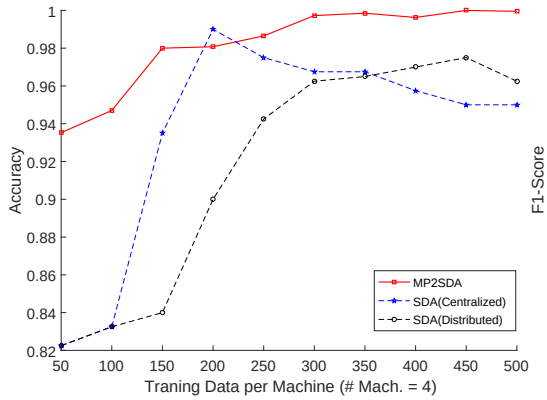
Further, we compare MP^2SDA algorithm with other centralized baseline algorithms in the same setting. For comparison, we categorize MP^2SDA and the baseline algorithms into groups of distributed algorithms and centralized algorithms. The distributed algorithms include MP^2SDA, MPSDA and distributed SDA. The centralized algorithms include centralized SDA, centralized two-stage LDA (Ye-LDA), centralized Linear SVM, centralized Kernel SVM, centralized Random



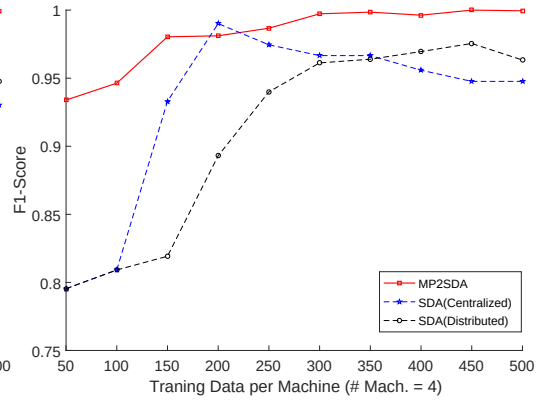
(a) Accuracy on Phishing Datasets



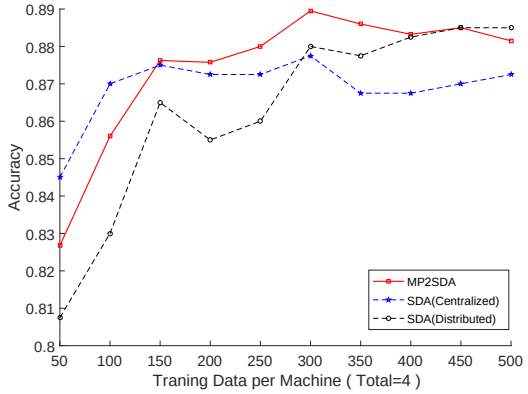
(b) F1-Score on Phishing Datasets



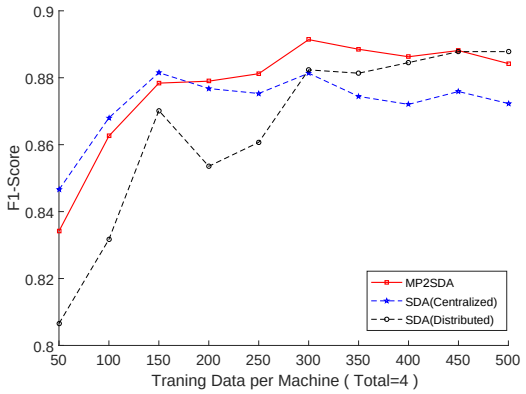
(c) Accuracy on Mushrooms Datasets



(d) F1-Score on Mushrooms Datasets



(e) Accuracy on Splice Datasets



(f) F1-Score on Splice Datasets

Figure 3.4: Performance Comparison among MP^2SDA, SDA(centralized) and SDA(Distributed) with Different Benchmark Datasets (Testing Sample Size = 400 and Machine Number = 4).

Forest and centralized Decision Tree. All the algorithms are fine-tuned. Table. 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^2SDA significantly outperforms distributed SDA on Mushrooms and Phishing datasets. On Splice dataset, MP^2SDA slightly outperforms distributed SDA in most cases.

Efficiency Comparison. Also, we compare the time consumption of MP^2SDA 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 2000 total training samples). Note that the communication time between each machine account for a large proportion in the total time consumption of MP^2SDA. Actually, on each machine, MP^2SDA 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)

Table 3.1: Accuracy Comparison among MP^2SDA, SDA(Centralized) and SDA(Distributed) on **Phishing Datasets**.

	Total Training Set Size									
Algorithm	200	400	600	800	1000	1200	1400	1600	1800	2000
Distributed Algorithm ($number\ of\ machines$, $m = 4$)										
MP^2SDA	**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 3.2: Accuracy Comparison among MP^2SDA, SDA(Centralized) and SDA(Distributed) on **Mushrooms Datasets**.

	Total Training Set Size									
Algorithm	200	400	600	800	1000	1200	1400	1600	1800	2000
Distributed Algorithm ($number\ of\ machines$, $m = 4$)										
MP^2SDA	**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 3.3: Accuracy Comparison among MP^2SDA, SDA(Centralized) and SDA(Distributed) on **Splice Datasets**.

	Total Training Set Size									
Algorithm	200	400	600	800	1000	1200	1400	1600	1800	2000
Distributed Algorithm ($number\ of\ machines$, $m = 4$)										
MP^2SDA	**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

Summary: In benchmark data experiments, we first compare the performance of MP^2SDA with distributed SDA and centralized SDA on real-world benchmark datasets. In most instances, MP^2SDA can compete with centralized SDA, even outperform centralized SDA on Mushrooms and Phishing datasets. Like the results on synthetic datasets, MP^2SDA overall outperforms distributed SDA on three benchmark datasets. Then, we additionally compare MP^2SDA with other centralized baseline algorithms. The result shows that these well-tuned centralized baseline algorithms dominantly outperform MP^2SDA and distributed SDA. While in the distributed algorithm group, MP^2SDA still outperforms distributed SDA. The additional efficiency comparison among MP^2SDA, MPSDA and centralized SDA shows that MP^2SDA 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.

Algorithm: $\widehat{\beta}^*$ Estimation on the jth Machine (Algorithm 4)

Data:
$\widehat{\Sigma}^j$ — the local covariance matrix on the jth machine
Parameter:
η — step size
Δ_{min} — minimum allowed perturbation
t_{max} — maximum number of allowed updates
λ — regularization parameter

```
begin
    /* On receiving the message from the previous machine */
    RECEIVE (β̂*, t)
    /* Selecting the k^th row of vector (Σ̂^j β̂* − (μ̂_+ − μ̂_−)) with the maximal absolute value */
    k ← argmax_{1≤k'≤p} | (Σ̂^j β̂* − (μ̂_+ − μ̂_−))_{k'} |
    /* Updating each row of β̂* on the j^th machine */
    β' ← ⟨0,0,...,0⟩^T
    /* initializing β with a p-dimensional 0 vector */
    foreach 1 ≤ l ≤ p do
        /* Note: Σ̂^{j,k,l} is the scaler on the k^th row and the l^th column of the matrix Σ̂^j */
        g_l ← sign(β̂*_l) · λ + sign(Σ̄^{j,k} β̂* − (μ̂_+ − μ̂_−)) · Σ̂^{j,k,l}
        /* Update each row of local β based on β̂* */
        β'_l ← β̂*_l − η · g_l
    end
    t ← t + 1
    /* Checking convergence conditions */
    Δ = | β̂* − β' |_1
    /* Update β̂* after calculating the Δ */
    β̂* ← β'
    if Δ ≥ Δ_max AND t ≤ t_max then
        /* Not converged, continuing the algorithm */
        Draw j_next ∈ {1...m} uniformly at random;
        SEND (β̂*, t) to the j^th_next machine;
    else
        /* Converged, sharing the estimates to all machines */
        BROADCAST β̂* to All machines;
    end
end
```

CHAPTER 4: AGGREGATION-FREE COMMUNITY SENSING FOR INTELLIGENT DISTRIBUTED ENVIRONMENT MONITORING SYSTEMS

Spatial-temporal community sensing is an efficient paradigm that leverages the mobile sensors of community members to monitor the spatial-temporal phenomena in the environment, such as air pollution or temperature. According to [100], there are two major roles in community sensing – the organizer and the participants – where the former is the individual or organization that creates the sensing task, recruits participants and collects the sensor data, while the latter (i.e., participants) involve in the sensing task and provide the sensing data. Frequently, the organizer pursues a high (or even full) spatial-temporal coverage of the collected sensor data. However incentives (e.g., monetary rewards) and the threats to privacy (e.g., exposing real-time locations) are two major concerns that may affect the willingness of the participants to join a community sensing task.

4.1 Motivations

In addition to the community sensing paradigm, a wide-spectrum of applications, ranging from vehicle traffic monitoring [101–105] to air quality sensing [106] and urban noise monitoring [107], have been proposed to efficiently monitor the environment of a large area through aggregating the real-time sensor and location data from the participants. Such applications use spatial-temporal coverage as the metric for overall task performance. Specifically, to characterize spatial-temporal coverage, the target area is split into subareas and the sensing duration is divided into a sequence of equal-length sensing cycles. In this way, the fraction of subareas covered by at least one sensor reading in each cycle is used to measure the spatial-temporal coverage.

For example, [108, 109] proposed to use the *full spatial-temporal coverage* as the criterion of the participants selection for community sensing, while [104, 110] studied the *partial spatial-temporal coverage* as the objectives of the optimization for budget-constrained participant selection. With the sensor data that partially covers the target area, [111–113] proposed *compressive community sensing*, which is capable of recovering the missing sensor data of the uncovered subareas from the data collected. Through the compressive community sensing, it is possible to accurately monitor the target area with even lower spatial-temporal coverage, thus resulting in reduced incentive consumption and fewer participants involved.

Though compressive community sensing can effectively reduce the required incentives and participants, it still aggregates the real-time location and sensor data from each participant, so as to first identify the covered subareas, fill with collected data, and then recover the missing data for the rest. To protect the location privacy of participants, the same of group of researchers [114, 115] proposed to leverage the *Differential Geo-Obfuscation* to replace each participants' real-time location with a "mock" location while insuring the recovery accuracy. With the Differential Geo-Obfuscation, the participants' locations are expected to be well obfuscated; however, it is still possible to attack the participant's location when certain prior knowledge is leaked. Thus, in our research, to further protect the real-time location privacy, we propose a novel *Aggregation-Free* Compressive Community Sensing framework, with following assumptions:

- **Assumption I:** *the organizer is **NOT** allowed to collect the real-time location or the sensor data from any participant*;
- **Assumption II:** *Each participant covers one or multiple subareas in each sensing cycles with his/her mobility, while the location and sensor data is locally stored on his/her mobile device without raw location/sensor data sharing*.

4.2 Preliminaries

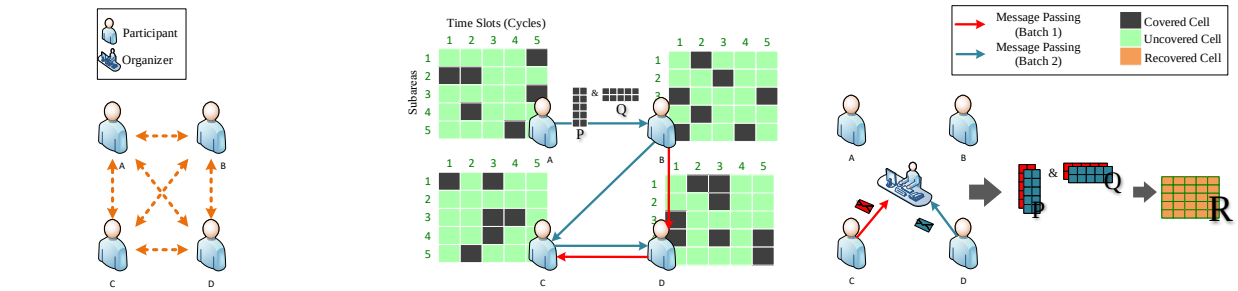
In this section, we first briefly introduce the previous work on the compressive community sensing. Then, we formulate the problem of our research.

4.2.1 Compressive Community Sensing

To derive the target full sensing matrix from partially collected sensing readings, the compressive community sensing wang2015ccs,wang2016sparse is commonly considered to be an effective approach, which consists of two parts: Aggregation and Inference.

4.2.1.1 Sensing Data Aggregation

Given the target region splitting into a set of subareas (denoted as S) and a set of m participants, in order to obtain the full picture of the target region for each sensing cycle (e.g., the tth cycle), the Compressive Community Sensing system first collects the sensing data from all participants. Specifically, the subareas covered by the jth participant in the tth sensing cycle ($t \in T$) is denoted as $S_j^t \subset S$. Thus, the overall coverage in the sensing cycle t can be denoted as $S^t = S_1^t \cup S_2^t \cup \ldots \cup S_m^t$. Due to the limited mobility of each participant and limited number of participants involved, the overall coverage can usually include a subset of subareas, i.e., $S^t \subseteq S$. Given the collected sensing data, the compressive community sensing system aggregates the data and assigns each covered subarea an unique sensor data value. For example, if multiple sensor data values are collected (from multiple participants) that covers the same subarea in a sensing cycle, the *averaged* value would be used as the value of such subarea in the sensing cycle. In this way, each subarea $s \in S^t$ has been covered with one sensor data value, through data aggregation, and the compressive community sensing system needs to infer the missing sensor data of the subareas in $S \backslash S^t$ to obtain the sensor



(a) **Phase I:** Secured P2P Network Establishment and Initialization

(b) **Phase II:** Distributed Compressive Community Sensing over Secured P2P

(c) **Phase III:** Spatial-Temporal Data Recovery

Figure 4.1: Overall Framework of *CSWA*

data of the whole target area.

4.2.1.2 Missing Data Inference

Given the aggregated sensor data of the covered subareas (S^t), there exists a wide-range of inferring techniques to infer the missing data of the uncovered subareas, such as expectation maximization [116] and singular spectrum analysis [117]. One of the powerful approach is the spatial-temporal compressive sensing [118, 119]. The essential idea of this approach is based on the nonnegative matrix factorization (NMF) [120, 121]. Given the aggregated sensor data of recent sensing cycles (the number of recent sensing cycles used for NMF is denoted as w), this approach first sorts the subareas using their indices from $1\ldots$ to $|S|$, then maps the data into a $|S| \times w$ matrix denote as R, where the element $R_{a,t}$ ($1 \leq a \leq |S|$ and $1 \leq t \leq w$) refers to the aggregated sensing value of the ath subarea and t^{th} sensing cycle (in the window). To recover the missing values in R, this approach obtains two non-negative ***matrix factors*** $P \in \mathbb{R}^{|S| \times l}$ and $Q \in \mathbb{R}^{l \times w}$ such that $R \approx PQ$, through NMF, where l stands for the *Size of Latent Space* of NMF.

Typically, there are four key factors affecting the performance of the compressive community

sensing: (1) *The Number of Subareas* that each participant covers in each sensing cycle; (2) *The Number of Participants (m)* which, together with the number of subareas per participant, can determine the coverage of collected sensor data; (3) *The Size of Windows (w)* that refers to the number of past sensing cycles used for matrix recovery (i.e., the width of the matrix for matrix completion); (4) *The Size of Latent Space (l)* that determines the rank of matrices for low-rank matrix recovery/completion.

4.2.2 Problem Formulation

Given a set of participants, where each participant's mobile device stores the raw sensor data locally (without raw data sharing), our proposed work intends to recover the sensing data of the target area while assuming that the organizer is not allowed to aggregate the sensor data from any participants. Specifically, we make following assumptions:

- For all the sensing cycles in T and subareas in S, there exists an unknown spatial-temporal sensor data matrix R^* ($R^* \in \mathbb{R}^{|S|\times T}$), where each element $R^*_{a',t'}$ ($1 \leq a' \leq |S|$ and $1 \leq t' \leq |T|$) refers to the real value of sensor data in the corresponding subarea a' and sensing cycle t'.
- In each sensing cycle (e.g., the tth cycle), each participant (e.g., the jth participant) covers a subset of subareas (i.e., $S^t_j \subseteq S$) in the target area. Thus, all the collected sensor data from the 1st to the tth sensing cycle of the jth participant can be represented as a matrix $R^j \in \mathbb{R}^{|S|\times t}$, where each element refers to the value of the sensor data collected in the corresponding subarea and cycle. Note that, to protect the location privacy, R^j is not known by the organizer.
- We denote the value of the sensor data collected by the jth participant in sensing cycle t at subarea a as $R^j_{a,t}$. Each sensor datum obtained is assumed to be the true value with (unknown) random noise, i.e., $R^j_{a,t} = R^*_{a,t} + \varepsilon^j_{a,t}$. For any two participants (i.e., the jth and k^{th}

participants), they might cover the same subarea (say, $a_j^t \cap S_k^t \neq \emptyset$ is possible), but are with *different* sensor data value obtained, due to the noise.

Our problem is that, in each sensing cycle t, with R^j $(1 \leq j \leq N)$ locally stored on each participant's device, there needs to estimate $\widehat{R}_{a,t}$ to

$$\textbf{minimize} \sum_{a=1}^{|S|} (\widehat{R}_{a,t} - R^*_{a,t})^2 \textbf{ for } 1 \leq t \leq T,$$

while ensuring that the organizer is prohibited to aggregate R^j from any participant and the raw sensor/location data sharing is not allowed between the participants.

4.3 *CSWA*: Aggregation-Free Spatial-Temporal Community Sensing

We propose a novel community sensing paradigm *CSWA*. Specifically, *CSWA* first establishes secured peer-to-peer network connections between each pairs participants. Then, *CSWA* proposes a decentralized non-negative matrix factorization algorithm based on *Parallelized Stochastic Gradient Descent* framework. Through learning the low-rank structure via distributed optimization, *CSWA* approximates the value of sensor data in each subarea (both covered and uncovered) for each sensing cycle using the sensor data that are locally stored in each participant's mobile device.

The characteristics of *CSWA* are as follows:

- We propose a novel community sensing framework *CSWA*, which is used to recover the environmental information in subareas, without aggregating sensor and location data from the community members who partially cover the target area. To the best of our knowledge, this is the first work that studies the problem of aggregation-free community sensing, by addressing the location privacy, distributed computing and optimization issues.

- To enable community sensing without location/sensor data aggregation, *CSWA* proposes a novel decentralized spatial-temporal compressive sensing framework that recovers the spatial-temporal information via decentralized Non-negative Matrix Factorization (NMF).The proposed solution operates on top of the *parallelized stochastic gradient descent*, which minimizes the loss function of NMF through secure Peer-to-Peer (P2P) message-passing over community members. The algorithm analysis shows that the proposed solution can efficiently approximates to the centralized NMF with the tolerable worst-case communication complexity.

- We evaluate *CSWA* using two large real-world datasets (i.e., temperature and air pollution). The experimental results demonstrate that *CSWA* tightly approximates to the state-of-the-art algorithms based on the data aggregation with centralized computation, and it outperforms the rest baselines with significant margin.

4.3.1 Framework Design

Before elaborating the proposed framework and algorithms, we make the following settings: (1) In order to simulate a secure peer-to-peer network over the community members, we define a set of participants, where these participants can receive or send messages (factor matrices) to each other trustfully and randomly; (2) When passing the message between two participants, the *receiver* can not send the updated matrix factors back to the *sender*, while the *sender* can easily recover the *receiver's* local sensing data by recalculating the return messages; (3) The organizer can only receive or access the related message when the updates (message passing) are finished. In this way, the private information such as real-time locations of the participants in each sensing cycle can be protected from the organizer.

The overall framework of *CSWA* consists of the following three phases (as illustrated in Figure.1):

4.3.1.1 Phase I: Secure P2P Network Establishment and Initialization

Prior to initializing the *batch* on the organizer, we first establish a secure peer-to-peer (P2P) network among m participants, while all the collected sensor data on the jth participant are mapped to a local data matrix R^j. Then, as shown in Algorithm 1, *CSWA* randomly picks a set of participants which is the *batch* (denoted as the set L with size N) from the secure network of m participants. Next, given the target data matrix $R \in \mathbb{R}^{|S| \times w}$, *CSWA* extracts the row and column number of R to construct the initial matrix factors $\hat{P}$ and $\hat{Q}$ on the organizer. Specifically, $\hat{P}_j$ is generated by a $|S| \times l$ Gaussian Random Matrix on the jth participant. Similarly, $\hat{Q}_j$ is generated by a $l \times w$ Gaussian Random Matrix on the same j^{th} participant. To avoid the aforementioned message transferring back between two participants, we initialize a counter i to record passing times (iterations) among participants and set j_p to mark the last participant's index, where the (i, j_p) will be transferred along with the updated matrix factors so that the participant who receives the message can randomly select the next one excluding participant j_p. When the initialization ends, each participant (I_j) in the predefined set L (*batch*) will be assigned a pair of starting matrix factors $\hat{P}_j$ and $\hat{Q}_j$.

4.3.1.2 Phase II: Distributed Compressive Community Sensing via Parallelized Low-Rank Approximation

Given the mapped local data matrix R^j on j^{th} participant, *CSWA* intends to approximate the optimal estimation of matrix factors $\hat{P}_j$ and $\hat{Q}_j$ via parallelized stochastic gradient descent on top of non-negative matrix factorization algorithm. Specifically, the initialized $(\hat{P}_j, \hat{Q}_j, \mathbf{0}, \mathbf{null})$ has been allocated on the jth participant, where $\mathbf{0}$ refers to the fact that no update has been executed and "**null**" refers to there is no previous participant (coming from the organizer) which has updated

Algorithm: Initializing Batch and Matrix Factors $(\hat{P}, \hat{Q})$ on Organizer (Algorithm 1)

Data:
$R_{|S|\times w}$ — the target data matrix
Parameter:
```
/* Subareas covered by per participant                                  */
```
$|S|$ — the maximum numbers of subareas
w — the size of windows
l — the size of latent space
begin
```
    /* Predefine a set of participants                                  */
```
    Randomly Draw *N Participants* into **Set** L
```
    /* L = {I_1, I_2, ..., I_N}                                         */
```
    for *each* $I_j \in L$ **do**
```
        /* Initialize matrix factors P,Q on I_j                         */
```
        $\hat{P}_j \leftarrow |S| \times l$ Gaussian Random Matrix
        $\hat{Q}_j \leftarrow l \times w$ Gaussian Random Matrix
```
        /* Initialize the counter and the previous participant index    */
```
        SEND $(\hat{P}_j, \hat{Q}_j, \mathbf{0}, \mathbf{null})$ to **L**;
    end
end

the matrix factors (the index of previous participant is empty). Then the algorithm processes the updating task on each participant from the predefined *batch* (L) in parallel.

Suppose two dense matrix factors are $P \in \mathbb{R}^{|S|\times l}$ and $Q \in \mathbb{R}^{l\times w}$, the target minimization loss function over m participants through parallelized stochastic gradient descent is as follow:

$$\hat{P}, \hat{Q} \leftarrow \underset{P\in\mathbb{R}^{|S|\times l}, Q\in\mathbb{R}^{l\times w}}{argmin} \left\{ \frac{1}{m}\sum_{j=1}^{m} F_j \circ (R^j - PQ)_F^2 + \lambda_P P_F^2 + \lambda_Q Q_F^2 \right\}, \tag{4.1}$$

where l is the size of latent space, "$\circ$" means element-wise matrix multiplication, $\cdot_F$ is the Frobenius norm, λ_P and λ_Q are regularization parameters. Particularly, parallelly starting on each participant I_j, Algorithm 2 first receives the input $(\hat{P}_j, \hat{Q}_j)$ from the last involved participant in the secure network (or initialized from the organizer in the first run). Next it updates the $(\hat{P}_j, \hat{Q}_j)$ using the mapped local data matrix R^j with the missing-value filter matrix F_j, and randomly picks up the next participant except the previous one (j_p) from the secure participants network and sends the

updated $(\hat{P}_j, \hat{Q}_j)$ to this chosen participant. The matrix F_j is a matrix filling with 0 (missing) and 1 (collected) which can set the missing elements in matrix R^j to zero by the element-wise multiplication. We mainly use it to prevent the missing value in the local data matrix R^j from affecting the gradient updating in $(\hat{P}_j, \hat{Q}_j)$. In addition, we leverage the *Truncate()* function, where the negative values in matrix factors $(\hat{P}_j, \hat{Q}_j)$ will be set to zero, then ensuring the non-negativeness of $(\hat{P}_j, \hat{Q}_j)$ when finishing each update.

Algorithm: Parallelized Optimization on the jth Participant (Algorithm 2)

Data:
R^j — the local data matrix on the jth participant
F_j — the filter matrix on the jth participant
Parameter:
i — the number iterations
j_p, j — the index of previous and current participant
η — step size
Δ_{min} — the minimum allowed perturbation
t_{max} — the maximum number of allowed updates
λ_P, λ_Q — regularization parameter on P and Q matrices
begin
  `/* On receiving the message from the previous participant */`
  RECEIVE $(\hat{P}_j, \hat{Q}_j, t, j_p)$
  `/* Noting that "`$A \circ B$`" means element-wise matrix multiplication */`
  $g_p \leftarrow (F_j \circ (R^j - \hat{P}_j \hat{Q}_j))\hat{Q}_j^T - \lambda_P \cdot \hat{P}_j$
  $g_q \leftarrow \hat{P}_j^T (F_j \circ (R^j - \hat{P}_j \hat{Q}_j)) - \lambda_Q \cdot \hat{Q}_j$
  $\hat{P}_j \leftarrow \hat{P}_j - \eta \cdot g_p$
  $\hat{Q}_j \leftarrow \hat{Q}_j - \eta \cdot g_q$
  `/* Set the negative elements to zero */`
  $\hat{P}_j, \hat{Q}_j \leftarrow Truncate(\hat{P}_j, \hat{Q}_j)$
  $i \leftarrow i + 1$
  `/* Checking convergence conditions */`
  $\Delta = max\{|g_p|_\infty, |g_q|_\infty\}$
  if $\Delta \geq \Delta_{max}$ ***AND*** $i \leq t_{max}$ **then**
    `/* Not converged, continuing the algorithm */`
    $j_{next} \leftarrow$ **Draw a random number from** 1 **to** m **except** j_p;
    SEND $(\hat{P}_j, \hat{Q}_j, i, j)$ **to the** j_{next}^{th} **Participant**;
  else
    `/* Converged, find out the optimal estimates */`
    SEND $(\hat{P}_j, \hat{Q}_j)$ **to the Organizer**;
  end
end

Algorithm 2 keeps picking up the next participant for updating, until the times of updates i exceeds the maximal number of updates, or the updating process converges (i.e., $max\left\{\left|g_p\right|_\infty, \left|g_q\right|_\infty\right\} \leq \Delta_{max}$). Similar procedures are starting on each participant I_j and the related matrix factors keep updating independently. Once the updating process completes on each participant, Algorithm 2 sends $(\hat{P}_j, \hat{Q}_j)$ where $j = 1, 2, ..., N$ to the organizer. When all the parallel processes are finished, the organizer has received N pairs of the estimated $(\hat{P}, \hat{Q})$ for recovery of the target data matrix.

Algorithm: Mobile Sensing Recovery on the Organizer (Algorithm 3)

Data:
$\hat{P}_j, \hat{Q}_j$ — the received matrix factors from the *batch*
begin
```
/* Average all P̂_j, Q̂_j on organizer                                  */
```
$\bar{P} \leftarrow \frac{1}{N}\sum_{j=1}^{N}\hat{P}_j$
$\bar{Q} \leftarrow \frac{1}{N}\sum_{j=1}^{N}\hat{Q}_j$
```
/* Recover the target overall data matrix                              */
```
$\hat{R} \leftarrow \bar{P}\bar{Q}$
end

4.3.1.3 Phase III: Spatial-Temporal Data Recovery

As we have introduced in the Preliminaries, the organizer can recover the target data matrix $\hat{R}$ based on the optimal estimated matrix factors $(\hat{P}, \hat{Q})$.

Given the received matrix factors $(\hat{P}_j, \hat{Q}_j)$ which are from the *batch*, Algorithm 3 first separately average the $\hat{P}$ and $\hat{Q}$ from $j = 1$ to N. Then, to recover the target data matrix, the algorithm multiplies the averaged matrix factors $(\bar{P}, \bar{Q})$ and obtains the well-estimated target data matrix $\hat{R}$.

4.3.1.4 Algorithm Analysis

In this section, we brief the analytical results of the proposed algorithms.

Given the overall set of subareas (S), the size of the latent space (l), the size of the windows (w), in each iteration, N participants in the system would send out messages, while each participant sends a $|S| \times l$ matrix and a $l \times w$ matrix (i.e., P and Q matrices). In this way, the system-wide communication complexity in the worst-case (after the completion of t_{max} iterations of message-passing) should be $\mathscr{O}\left(\left(|S| \cdot l + l \cdot w\right) \cdot t_{max} \cdot N\right)$.

Suppose P^* and Q^* are the optimal solutions of the problem listed in Eq. 1, while $\bar{P}$ and $\bar{Q}$ (appeared in Algorithm 3) are two approximation results obtained by our algorithm. According to [96], the approximation error of $||P^* - \bar{P}||_F \to 0$ and $||Q^* - \bar{Q}||_F \to 0$, when $t_{max \to +\infty}$ and N is sufficiently large. Note that with a larger N, the proposed algorithm can achieve a faster rate of convergence of the approximation error with increasing t_{max}. For more theoretical analysis, please refer to [96].

4.3.2 Evaluation

In order to evaluate the *CSWA* algorithm, we use the *Temperature (TEMP)* and *PM 2.5 air quality (PM25)* dataset, where the Experimental Setup section will cover all the settings and assumptions. Based on the above dataset, we first introduce the baseline algorithms which are commonly used in sensor data recovery. Specifically, the baseline algorithms adopt the ***matrix completion*** method and leverage the ***centralized computing*** patterns to recover the target sensing data. Then, we compare the performance of *CSWA* with baseline algorithms on two real-world datasets.

4.3.2.1 Experimental Setup

For *TEMP* [122] and *PM25* [123] datasets, the sensing value of temperature (°C)/PM2.5 (air quality index) are located on each participant's mobile sensor in varying time slots (sensing cycle) and at different subareas. In details, the *TEMP* dataset contains the temperature readings in 57 cells

(Subareas) and each sensing cycle lasts for 30 minutes. The *PM25* dataset includes the PM2.5 air quality values on 36 stations (Subareas) with the same sensing cycle.

In order to simulate the settings of the centralized computing patterns, we aggregate the collected sensing data from each participant. In details, we follow the aforementioned three phases to set the appropriate value of four key factors: *the Number of Participants (m), the Number of Subareas that each participant covers in each sensing cycle, the Size of Windows (w)* and *the Size of Latent Space (l)*. Note that each participant can sense the temperature/PM2.5 at a subset of subarea. Specifically, we use the maximum number of subareas s ($1 \leq s \leq |S|$) in the experiments, assuming the participant can cover no more than s subareas. To simulate the scenario that each participant can cover various number of subareas, the actual number of subareas covered by the participant will follow the discrete uniform distribution $\mathrm{U}\{1, s\}$.

4.3.2.2 Baseline Algorithms

In this section, we briefly introduce three baseline algorithms, where their advantages and drawbacks are listed as compared to *CSWA* algorithm.

- ***Spatio-Temporal Compressive Sensing (STCS)*** – *STCS* [111,119] leverages the sparsity regularized matrix factorization to fill in the missing values in a certain matrix accounting for spatial-temporal properties. Based on the low-rank nature of real-world data matrices, STCS first exploits global and subarea structures in the data metrics. Then, it recovers the original matrices through matrix factorization under spatial-temporal constraints. Indeed, STCS advances ideas from compressive sensing and provides a highly effective (high accuracy and robustness) approach to solve the problem of missing data interpolation.
- ***Robust Principle Component Analysis (RPCA)*** and ***Truncated Singular Value Decompo-***

sition (TSVD) – *RPCA* [124] is derived from a widely used statistical procedure of principal component analysis (PCA), where RPCA performs well on solving the problem of matrices recovering. With respect to a mass of missing observations, RPCA aims to recover a low-rank matrix through random sampling techniques [125]. *TSVD* [126] is also commonly used to approximate a low-rank matrix. Different from the traditional singular value decomposition, TSVD sets all but the first k largest singular values equal to zero and use only the first k columns of the corresponding unitary matrices. With the optimality property, this method provides an efficient way to recover the target sensing matrix.

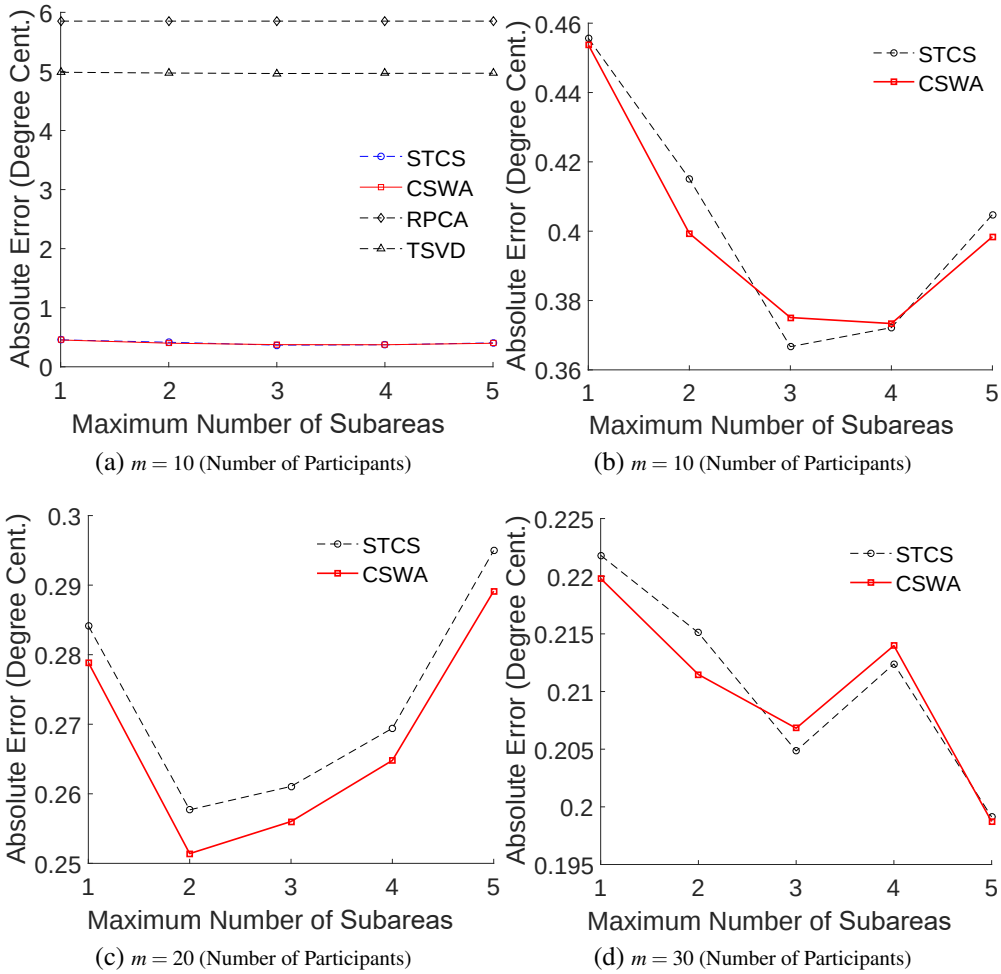
4.3.2.3 Experimental Results

In this section, we report the performance of *CSWA* and other three baselines on *TEMP* and *PM25* datasets. Specifically, we use the *Absolute Error*, which is the averaged element-wise difference $\left(\sum_{a=1}^{|S|}\sum_{t=1}^{|T|}\left|\widehat{R}_{a,t}-R^*_{a,t}\right|/(|S|\cdot|T|)\right)$ between the recovered matrix ($\widehat{R}$) and the original data matrix (R^*), as the indicator of the performance.

***TEMP* Datasets**. First, we present a comparison of algorithms with the settings of the maximum number of subareas (covered by each participant) ranging from 1 to 5 in Fig. 4.2. Due to the overall better performances of *CSWA* and STCS, we present the entire comparison in (a) and only compare *CSWA* with STCS in other three settings (the same in Figures 4.3, 4.4 and 4.5 as well). Specifically, in Fig. 4.2(a), 10 participants are involved. Then we vary the number of participants from 10 to 30 in the increment of 10 in Figs. 4.2(b), (c) and (d). We observe that the error is around 0.2 to 0.45 with varying maximum number of subareas from 1 to 5. It is noteworthy that *CSWA* can compete to STCS under these settings.

Second, we also compare *CSWA* with baseline algorithms by varying the number of participants in the secure P2P network. In Fig.3(a), the maximum number of subareas is 1. Then we increase it

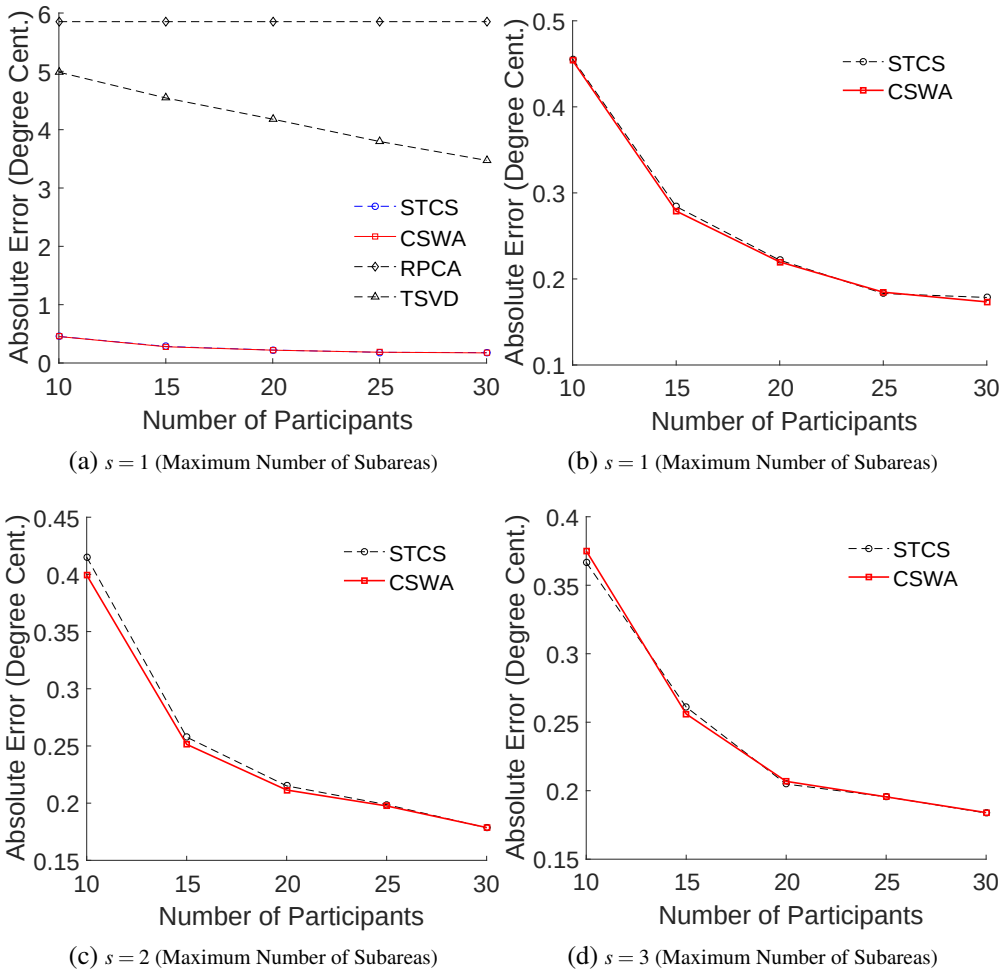
from 1 to 3 in the increment of 1 in Figs. 4.3(b), (c) and (d). In each comparison between *CSWA* and STCS, the error decreases when the number of participants increases for both of these two algorithms. This demonstrates that the larger group of participants can improve the performance of the matrix recovery, where intuitively the participants can cover more subareas and sensing cycles. Similar to the previous setting, *CSWA* can approximate the performance of STCS as well.



(a) $m = 10$ (Number of Participants)
(b) $m = 10$ (Number of Participants)
(c) $m = 20$ (Number of Participants)
(d) $m = 30$ (Number of Participants)

Figure 4.2: Performance Comparison with Varying *Maximum Number of Subareas* (s) per Participant per Cycle on *TEMP* Datasets.

Further, we alter the values of two aforementioned key factors, such as *Size of Windows* and *Size of Latent Space*, to observe the variation of the error. Fig. 4.4 shows that the error decreases when



(a) $s = 1$ (Maximum Number of Subareas)

(b) $s = 1$ (Maximum Number of Subareas)

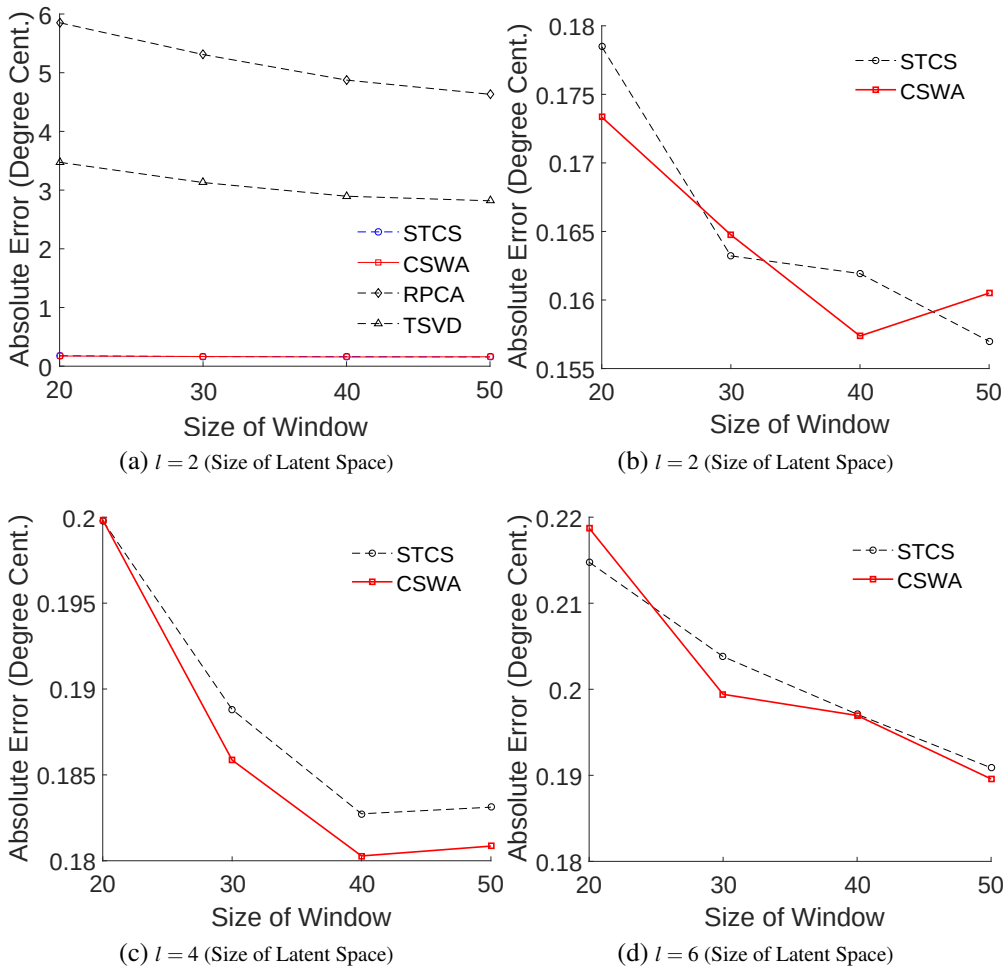
(c) $s = 2$ (Maximum Number of Subareas)

(d) $s = 3$ (Maximum Number of Subareas)

Figure 4.3: Performance Comparison with Varying *Number of Participants* (m) on *TEMP* Datasets.

the window size increases from 20 to 50. Note that for each size of latent space in Figs. 4.4(b), (c) and (d), the decreasing trends of the error are almost the same and the performance of *CSWA* still can compete with STCS. Fig. 4.5 exhibits that the error increases when the size of the latent space increases from 2 to 10. Thus, for *TEMP* datasets, the small size of latent space can better approximate the original data matrix when it is low-rank. Thus the performance of *CSWA* is still competitive to STCS, as shown in Figs. 4.5(b), (c) and (d).

***PM25* Datasets**. We conduct experiments with similar settings as *TEMP* datasets. Since the per-



(a) $l = 2$ (Size of Latent Space)

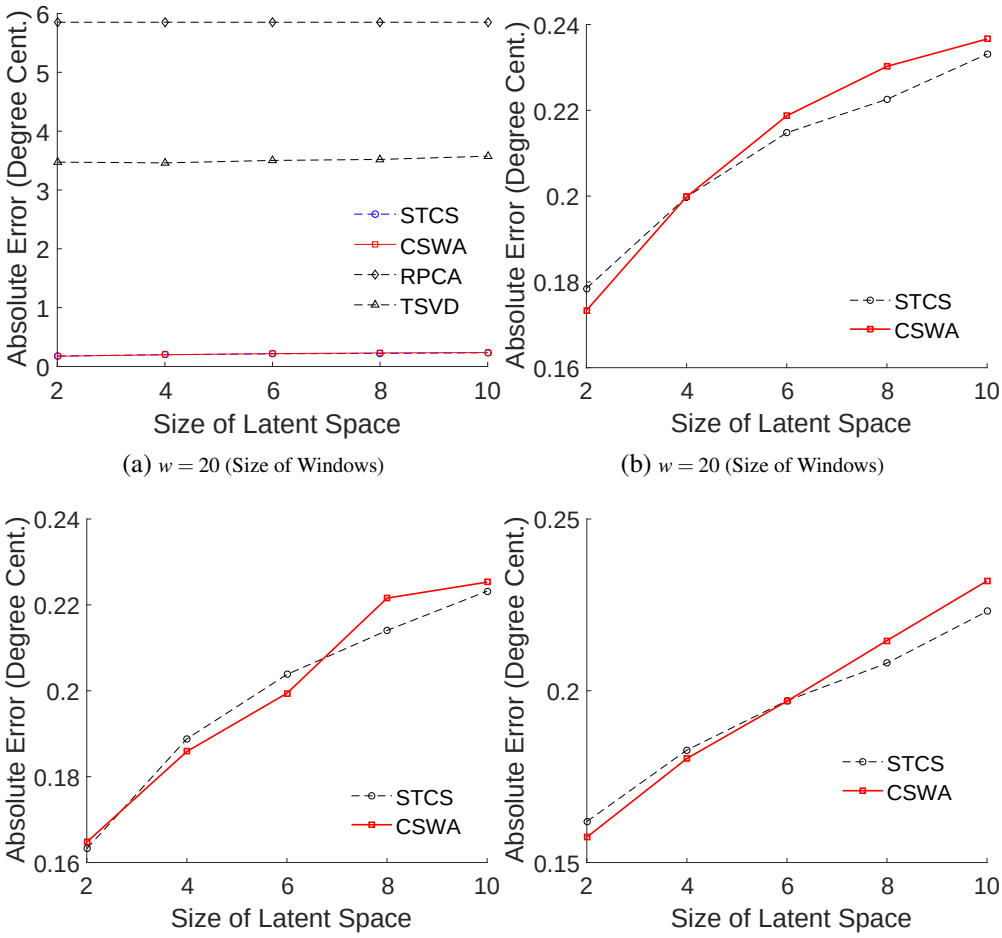
(b) $l = 2$ (Size of Latent Space)

(c) $l = 4$ (Size of Latent Space)

(d) $l = 6$ (Size of Latent Space)

Figure 4.4: Performance Comparison with Varying *Size of Window* (w) on *TEMP* Datasets.

formances of RPCA and TSVD are still not as good as the other two algorithms, we only present the comparison between the proposed *CSWA* and STCS here. Specifically, in Table. 4.1, we list the *Absolute Error* of these two algorithms with varying number of participants (m) and the window size (w). When the number of participants increases, the error is decreasing intuitively. On the contrary, the error increases with increased size of the window. However, *CSWA* performs comparably to STCS, sometimes even better (e.g., for $m = 20$). In Table. 4.2, we show the performance with varying size of latent space and the number of subareas covered by each participant. The re-



(a) $w = 20$ (Size of Windows)

(b) $w = 20$ (Size of Windows)

(c) $w = 30$ (Size of Windows)

(d) $w = 40$ (Size of Windows)

Figure 4.5: Performance Comparison with Varying *Size of Latent Space* (l) on *TEMP* Datasets.

sults reveal that the number of subareas does not affect the error significantly, while with the larger latent space the error is smaller with *PM25* datasets. Under these two settings, the performance of *CSWA* can still compete with STCS. Note that for each setting, we present the performance on the varying factor while keeping the other factor at optimal value. Also it is worth noting that the overall error is small on the average (10 with PM2.5 index ranging from 1 to 500) in both of *CSWA* and STCS.

Summary: With two real-world datasets, we compared the proposed *CSWA* with the baseline

Table 4.1: Performance Comparison (*Absolute Error*) with Varying *Number of Participants* (m) and *Size of Windows* (w) on *PM25* Datasets.

	Number of Participants (m)			Size of Windows (w)		
	10	20	30	20	30	40
CSWA	15.563	11.686	9.561	8.844	10.232	12.028
STCS	15.185	11.864	9.353	8.517	10.090	11.955

Table 4.2: Performance Comparison (*Absolute Error*) with Varying *Size of Latent Space* (l) and *Maximum Number of Subareas* (s) on *PM25* Datasets.

	Size of Latent Space (l)			Maximum Number of Subareas (s)		
	2	4	6	1	2	3
CSWA	11.777	9.561	8.945	8.166	8.945	8.844
STCS	11.518	9.353	8.719	8.220	8.719	8.516

algorithms STCS, RPCA and TSVD. For both of the datasets, *CSWA* significantly outperforms RPCA and TSVD in most cases. Moreover, compared to the centralized algorithm STCS, *CSWA* also presents its competitiveness, with a low approximation error (0.2° in city-wide temperature and 10 units of PM2.5 index in urban air quality). Even in some settings, the *CSWA* has a lower approximation error than STCS, which demonstrates the superiority of *CSWA* .

CHAPTER 5: MISCELLANEOUS LEARNING ALGORITHMS IN DISTRIBUTED AND INTELLIGENT SYSTEMS

Nowadays, statistical and machine learning algorithms are used more frequently and intensively to solve problems in a wide range of applications, e.g., smart home, medical diagnosis, and environment analysis. These algorithms are often highly parameterized and their performances are sensitive to hyper-parameter settings. For example, the well-known Multi-Layer Perceptron (MLP) [45] suffers from a large variance of prediction accuracy with different hyper-parameter settings for the same task, where the hyper-parameters include the number of layers, the number of neurons in each layer, the type of the activation functions, the learning strategies, etc. All of these settings should be well configured before a machine learning model is applied to a real application.

Hyper-parameter tuning is essential to achieve good predictive performance, while it quickly becomes expensive as the data size and/or search space grows. In the past decades, many hyper-parameter tuning algorithms have been developed and analyzed. As the state-of-the-art, Model-Based Optimization (MBO), also known as *Bayesian optimization* [127], solves the expensive optimization problem by fitting a Gaussian process regression to approximate the predictive performance in dependence of the hyper-parameters. The performance of MBO has been shown in [128]. Normally, such tuning requires the dedicated machine learning model to be trained and evaluated on centralized data to obtain a performance estimate.

Distributed embedded (as well as edge computing) systems are widely utilized to run various machine learning algorithms due to their high flexibility (mobility), scalability, and low energy consumption in real-world applications. For example, modern air quality monitoring systems consist of multiple nodes located around the target area, in order to increase robustness and eliminate possible bias. Each node can be regarded as an individual system. It has a sensor module used

for monitoring the environment and collecting data, and a processing module, which is able to load light-weighted machine learning tasks based on locally collected data, and supports efficient training and fast inference. These distributed embedded systems are more powerful and intelligent than traditional sensors that are only used for collecting data.

In such distributed settings, the original design of a centralized hyper-parameter tuning process is no longer suitable and efficient. If data is transferred through low bandwidth connections, merging all sub-data sets to one central node consumes a large amount of communication resources and leads to large overheads, and, hence reduces the available time for tuning. In some scenarios, it is impossible to collect and store the raw data due to privacy concerns or limited storage of the central node. In addition, distributed nodes have overlapping sensing areas and the redundant data (repeatedly uploaded) causes further burdens to the central node. Moreover, the execution time of machine learning algorithms is usually sensitive to the hardware platforms. In an extensive study of unsupervised methods, the impact of particular implementations, frameworks, programming languages and libraries on the run-time performance has been shown in [129]. Particularly for run-time considerations, it has been stated that caching behaviour determines the performance of implemented algorithms even more than algorithmic differences [130]. For example, the run-time of a random forest in [131] is optimized for different platforms using different settings due to the different hardware designs, e.g., cache size. Therefore, if the objective of the tuning is to speed up the algorithm, the optimal setting on the central node may not be optimal for the dedicated distributed embedded systems due to different hardware architectures.

As an alternative to such global data integration, each node can conduct hyper-parameter tuning independently based on its local data. However, for each node the storage and detecting area are limited. Hence each node can only keep one part of the whole data set collected in this area. If each node tunes the hyper-parameter independently using its local sub-data set, the performance of the machine learning algorithm will vary due to the small size of the training data. The main challenge

of the hyper-parameter tuning on a distributed embedded systems lies on how to utilize these decentralized sub-data sets to generate a universal hyper-parameter setting, which can be applied to all the nodes in this system. Towards this, two potential requirements for the new method are raised, i.e., 1) increase the accuracy of prediction; and 2) improve the run-time efficiency.

5.1 Preliminaries

In this section, we first introduce several hyper-parameter tuning algorithms. Afterwards, the *Model-Parallelism* and *Federated Learning* are discussed briefly, which motivate our work.

5.1.1 Hyper-parameter Tuning Algorithms

The most direct and easy to implement tuning algorithm is Grid Search [132] which discretizes the hyper-parameter search space and exhaustively evaluates all possible combinations in a Cartesian grid to find the setting with the best performance. Another variation is Random Search [133], which randomly samples hyper-parameter settings from the search space. The drawback of both tuning methods is that they do not make use of information obtained from previous tries, which implies a waste of computational resources. In contrast, Sequential Model-Based Optimization [127] takes advantage of the previous search trajectory and has been proven to optimize hyper-parameters more efficiently [134]. In the classical approach, Gaussian process regression, also called Kriging, is used as its regression model [135]. For certain scenarios and hierarchical search spaces, tree-based surrogates, such as the Tree-structured Parzen Estimator (TPE) [136] or random forests [137], have been proved to be beneficial. In order to extend MBO with parallel evaluations, various techniques have been developed [138–140]. They can propose and evaluate multiple points in each iteration. To account for heterogeneous run-times of different proposals, asynchronous parallel

strategies [141] as well as scheduling methods [142] have been developed.

5.1.2 Model-Parallelism and Federated Learning

Due to the increasing demands of distributed data collection, storage, and processing as well as the privacy-preserved concerns in many applications, federated learning [143, 144] has become one of the popular computing paradigms, where a machine learning model is trained across multiple decentralized edge devices or servers with their local data. In most federated computing platforms, "no raw data sharing" is an important requirement, where a machine learning algorithm should be trained using all data stored in all the distributed machines (i.e., nodes), but without any cross-machine raw data sharing. Specifically, the aforementioned hyper-parameter tuning algorithms (e.g. MBO) can be accelerated by federated learning and typically be divided into two types: *Data-Parallelism* [145] and *Model-Parallelism* [11] methods. On each embedded system (node), the *Data-Parallelism* algorithm first trains the model by using the local data. Afterwards, a global model is obtained via model-averaging [146]. The aggregated model is considered as the trained model based on the overall data (from multiple nodes). Due to the construction of Data-Parallelism, parallel computing method can be easily applied. The *Model-Parallelism* requires multiple nodes to learn a shared prediction model collaboratively. Such an algorithm has to commonly update parameters synchronously or asynchronously across all nodes, which causes additional overheads. In many applications, parameters updating can be a tough nut.

Both aforementioned approaches keep all the training data local on corresponding nodes. Compared with the Data-Parallelism (as the chosen baseline algorithm *MBO-S*), the Model-Parallelism (which *MODES* adopts) usually can achieve better performance, as it globally optimizes the performance of the model [11]. However, as far as we know, no previous studies have been carried out with respect to Model-Parallelism in connection with MBO on embedded systems. Addition-

ally, since hyper-parameters are optimized in parallel based on local data for each node, Model-Parallelism-based methods appear to be more efficient, compared to the traditional (centralized) MBO.

5.2 *MODES*: Model-based Optimization on Distributed Embedded Systems

We propose *MODES*, a Model Based Optimization method to tune hyper-parameters for machine learning algorithms on Distributed Embedded Systems **locally** and **efficiently**. Each node is treated as a small black box. It trains an individual model based on its local data. The whole distributed embedded system is considered as a big black box, and the goal is to optimize the performance of this black box, with respect to the accuracy of prediction and/or run-time efficiency. The novel features are as follows:

- We design a framework *MODES* to apply MBO on resource-constrained distributed embedded systems, which not only speeds up the tuning process to obtain the optimal hyper-parameters efficiently, but also improves the generalization ability of the obtained hyper-parameter setting. The proposed *MODES* tremendously mitigates the data communication cost by only transferring hyper parameter settings and performance values, i.e., accuracy of classifications.

- In order to meet different requirements, we further categorize *MODES* into two optimization modes: (1) the Black-box mode (*MODES*-B) recognizes the whole ensemble as a single black box and optimizes the hyper-parameters of each individual model jointly by considering the weights for different nodes and (2) the Individual mode (*MODES*-I) recognizes all models as clones of the same black box which allows it to efficiently parallelize the optimization in a distributed setting.

- We conduct extensive evaluations to compare our proposed two modes of *MODES* with two baselines, i.e., applying MBO for tuning hyper-parameter setting on each single node using its local data independently (MBO-S), and tuning based on centralized data (MBO-C). The results show that: 1) *MODES*-B has slightly worse performance than MBO-C but without raw data aggregation, and outperforms MBO-S in most of cases. 2) *MODES*-I highly improves the run-time efficiency, where the improvement depends upon the number of nodes in the distributed system, at a cost of slightly performance degradation comparing with MBO-S in some cases. The implementation of *MODES* and corresponding experiments are released in [147].

5.3 Model Based Optimization

Model-Based Optimization (MBO) solves the optimization problem:

$$x^* = \arg\max_{x \in \mathscr{X}} f(x)$$

for a given function $f(x)\colon \mathscr{X} \to \mathbb{R}$ with $\mathscr{X} \subset \mathbb{R}^p$. We assume that the true expensive black box function can be approximated through a surrogate. This surrogate is a regression method that is comparably inexpensive to be evaluated. For MBO, typically a Gaussian process regression is chosen. To start the optimization, an initial design $\mathscr{D}$ of k points, laid out in a Latin hypercube design, is evaluated on the expensive function and yields the outcomes y. In the following, the sequential model-based optimization iteratively repeats the following steps until a predefined budget is exhausted:

1. A Gaussian process is fitted to all past evaluations, serving as a surrogate to estimate f globally.

2. An acquisition function is optimized to determine the most promising point $\hat{x}$:

$$\hat{x} = \arg\max_{x \in \mathscr{X}} acq(x).$$

3. $y = f(\hat{x})$ is evaluated, $\hat{x}$ and y are added to $\mathscr{D}$ and y.

The acquisition function has to balance exploration (evaluate points where the surrogates prediction is uncertain) and exploitation (evaluate points that are predicted to be optimal by the surrogate). The final optimal result $\hat{x}^*$ is the input that leads to the maximal observed objective value, e.g., prediction accuracy.

In the original formulation, MBO only proposes a single point in each iteration. It is necessary to obtain multiple proposals in each iteration in order to make use of parallel computing infrastructures. Snoek et. al. [135] proposed the qCB as a computational simple acquisition function for multiple proposals:

$$\text{qCB}(x, \lambda_j) = \hat{\mu}(x) + \lambda_j \hat{s}(x) \text{ with } \lambda_j \sim \text{Exp}(\lambda), \tag{5.1}$$

with $\hat{\mu}(x)$ as the mean prediction and $\hat{s}(x)$ as the uncertainty prediction of the surrogate for point x. To obtain n proposals we first sample n values of λ_j from an exponential distribution with an expected value of λ, yielding n different acquisition functions $\text{qCB}(x, \lambda_j)$. Functions with a low value of λ_j will result in optima close to points where the surrogate predicts an optimum (exploitation). A high λ_j leads to optima in areas where the surrogate is uncertain (exploration). Because each acquisition function is comparably cheap to evaluate, we can obtain each optimum by an exhaustive iterative random search. Each optimum $\hat{x}_j$ is the hyper-parameter configuration that will be evaluated on node j. The combination of exploitative and exploratory configuration proposals leads to an effective optimization of the given black box. The parallelized Bayesian Optimization in [140] outperforms the state-of-the-art CMA-ES on most of the test functions.

In this work, single proposal MBO is applied for *MODES*-B while parallelization through multiple proposals using the qCB criterion is applied for *MODES*-I.

5.4 Distributed Model-Based Optimization

In this section, the model of the distributed embedded system is introduced at first. Afterwards, two categories of proposed *MODES* with different structures are explained in detail. *MODES*-B is developed in order to improve the overall accuracy of prediction by considered the whole distributed system as a black box. Meanwhile, the difference among nodes and corresponding local sub-data set has been taken into consideration. Nevertheless, *MODES*-I tries to improve the run-time efficiency by evaluating multiple hyper-parameter settings in different nodes at the same time, with the assumption that the difference among nodes is negligible.

5.4.1 System Model

In a distributed embedded system, also denoted as a *cluster*, several embedded systems cooperate towards a common objective. In this work, we assume a homogeneous cluster[1], in which all the nodes have identical characteristics. For this cluster, we assume:

- It consists of n nodes, denoted as $ES_1, ES_2, ES_3 \ldots ES_n$. Each node is one embedded system.
- Each node has limited storage and can only store a certain amount of data.
- Data collected by different nodes are (at least partially) different and can be treated as subsets of a completed data set.

[1]The proposed method can also be applied on heterogeneous clusters, with the effort to synchronize the execution of different nodes, e.g., assign the heavy workloads to nodes with more resources and better computational performance, which is out of the scope in this work.

- Connections among nodes are of low bandwidth and only the tiny data can be transferred, i.e., hyper-parameter settings and performance results (accuracy of classifications).

In our setting, a master-slave model is applied on all the available nodes. Although all nodes run a dedicated machine learning algorithm, only one node runs the MBO algorithm. The node where the MBO is deployed, is called *master*, which runs MBO and the dedicated machine learning algorithm at the edge at the same time. The remaining nodes, called *slaves*, only run the dedicated machine learning algorithm. Due to our setting of limited computational power of embedded systems, only light-weighted machine learning algorithms are applied, which results in a relatively small search space for hyper-parameters. Hence, the optimization workload of MBO does not affect the execution of other applications running on the *master* node. In addition, the number of hyper-parameters of the machine learning algorithm is denoted by p.

5.4.2 Black-box Mode MODES-B

In *MODES*-B, the whole distributed system is treated as a single black box. The hyper-parameter setting of each individual node is optimized jointly in order to improve the performance as a way of ensemble learning. The whole system only generates one prediction at a time. Such a method can be utilized in a wide range of applications, e.g., air quality prediction in one area utilizing all the embedded sensors in that area [17], and object recognition by using images taken from different angles [148].

The structure of *MODES*-B is shown in Figure 5.1, and the corresponding workflow is presented in Algorithm . MBO runs initial setup at first to construct the surrogate denoted as $\mathscr{S}$. At the beginning of each iteration, MBO only generates one set of hyper-parameters with the highest expected improvement with respect to the current surrogate, which consists of $(n \times p + n)$ parameters, i.e,

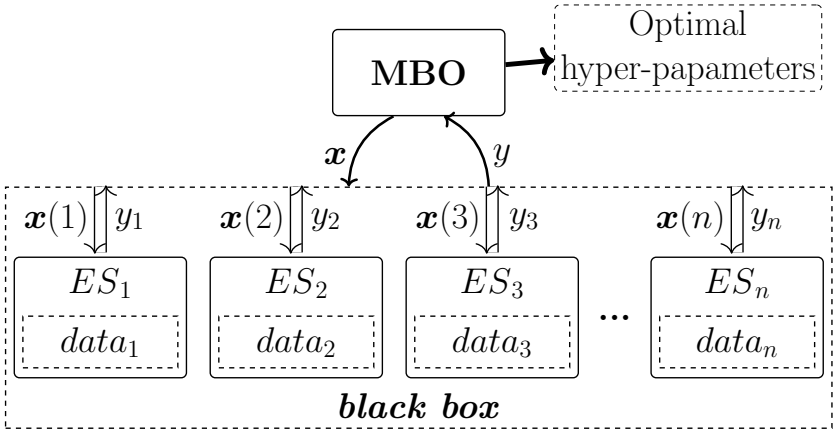


Figure 5.1: *MODES*-B: The distributed embedded system is treated as a single black box

$x \in \mathbb{R}^{n \times p+n}$. In each setting, first p parameters represent the hyper-parameters for the first node, second p parameters represent the hyper-parameters for the second node and so on. Moreover, n weights indicating the importance of each node and its local data are represented through x as well.

The dedicated machine learning model *ML* is trained on each node by using the given hyper-parameter setting and the local sub-data set. Each node generates one local performance result (accuracy of classification) of the trained machine learning model by using an evaluation test set that is shared across all nodes. The final result y is averaged according to the weights of results from all the nodes, i.e., $y = \sum_{i=1}^{n} w_i \times y_i$, where y_i is the local performance result of node i, and $\sum_{i=1}^{n} w_i = 1$. Afterwards, the final result is utilized to update the surrogate of MBO. The process is repeated until the maximum number of iterations is reached or the time budget is exhausted.

In this mode, the number of dimensions of the search space is $n \times p + n$. Therefore, the large number of nodes (n) in the dedicated cluster and/or the large number of hyper-parameters (p) of the dedicated machine learning model can result in a search space with a large number of dimensions. The computation power that MBO needs to update the surrogate and to propose new setting(s) is

Algorithm: Workflow of *MODES*-B

Input: number of nodes n, dedicated machine learning model ML, number of hyper parameters p, time budget T, and maximum tuning iterations Itr;

1: **output** (O)ptimal hyper-parameter setting: $HP\text{-}B$;
2: Initialize: MBO surrogate $\mathscr{S}$, iteration $i \leftarrow 0$, time $t \leftarrow 0$;
3: **while** $i \leq Itr$ and $t \leq$ T
4: $x \leftarrow$ MBO $(\mathscr{S}, n, p)$;
5: **for** i from 1 to n
6: $y_i = ML(x(i), ES_i, data_i)$
7: $y \leftarrow \sum_{i=1}^{n} w_i \times y_i$;
8: Update surrogate according to (x, y);
9: i ++;
10: Accumulate consumed time t;
11: MBO generates the optimized $HP\text{-}B$ according to current surrogate;
12: **return** ()$HP\text{-}B$;

proportional to the size of search space. However, due to the limited computational capability, embedded systems may not be able to find the optimal hyper-parameter setting from such a huge search space within a certain time budget.

Against this limitation, we enforce all the nodes to share the same setting of hyper-parameters but different weights, i.e., $\forall i, j \leq n, i \neq j : x_i = x_j$ and $\exists i, j \leq n, i \neq j : w_i \neq w_j$. As a result, the search space is significantly reduced to $(p+n)$ dimensions. In each MBO iteration, all the nodes receive the same set of hyper-parameters, and train the dedicated machine learning model using their local data sets independently. Afterwards, the shared evaluation test set is utilized to evaluate the performance of these trained machine learning models on different nodes, and the weighted mean is returned to the *master* node, which is used to update MBO surrogate. In the end, one set of optimized hyper-parameters along with the weights of nodes are obtained.

Please note, the proposed *MODES*-B with different hyper-parameters for each node, i.e., $\exists i, j \leq n, i \neq j : x_i \neq x_j$, can also be applied on powerful distributed systems. However, that is out of the scope in this work.

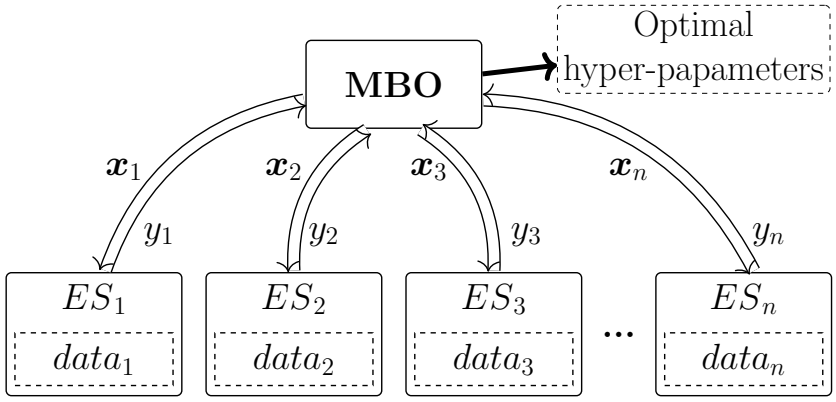


Figure 5.2: *MODES*-I: Each embedded system acts as an individual black box.

Algorithm: Workflow of *MODES*-I

Input: number of nodes n, dedicated machine learning model ML, number of hyper parameters p, time budget T, and maximum tuning iterations Itr;

1: **output** (O)ptimal hyper-parameter setting: $HP\text{-}I$;
2: Initialize: MBO surrogate $\mathscr{S}$, iteration $i \leftarrow 0$, time $t \leftarrow 0$;
3: **while** $i \leq Itr$ and $t \leq$ T
4: $\{x_1, x_2, \ldots, x_n\} \leftarrow$ MBO $(\mathscr{S}, n, p)$;
5: **for** j from 1 to n
6: $y_j \leftarrow$ ML($\{x_j, ES_j, data_j\}$);
7: Update surrogate according to $\{(x_1, y_1), \ldots, (x_n, y_n)\}$;
8: $i \leftarrow i +$ n;
9: Accumulate consumed time t;
10: MBO generates the optimized $HP\text{-}I$ according to current surrogate;
11: **return** ()$HP\text{-}I$;

5.4.3 Individual Mode MODES-I

In *MODES*-I, each node is treated as an instance of the same black box. The whole cluster acts like a multi-processor system and each node is a single processor. This enables us to apply MBO in a parallelized manner. In this scenario, the performance of multiple proposed hyper-parameter settings can be evaluated at the same time, i.e., each node trains a dedicated machine learning model using one set of the proposed hyper-parameter settings.

The structure of *MODES*-I is shown in Figure 5.2, the workflow is presented in Algorithm . In each iteration, MBO proposes n different hyper-parameter settings based on the knowledge obtained from the current surrogate, using the qCB criterion as explained in Section 5.3. Each node uses one hyper-parameter setting to independently train the dedicated machine learning model using their local data. Afterwards, these trained models are evaluated by using an identical evaluation test set, which is shared among different nodes. The individual performance measures, i.e., the accuracy of classification, are sent back to the *master* node. In our setting, synchronized updating of surrogate is applied, where the MBO updates the surrogate, once all nodes finished their evaluation. Therefore, the execution time of each iteration equals to the longest execution time of all these nodes. The iterations are repeated until the time budget is exhausted or the maximum number of iterations is reached. The optimization result is one hyper-parameters setting that can be utilized for all the nodes. The whole system can generate the prediction by a simple average with equal weights from different nodes. Alternatively, a single node can do the prediction itself with a lack of robustness.

MODES-I significantly improves the run-time efficiency of the hyper-parameter tuning process, by fully utilizing the computational power of all the nodes inside the distributed system, i.e., it evaluates n proposed settings in parallel by considering all the information from the local data in different nodes. Although the performance of the tuned hyper-parameters may not be improved significantly, due to the fact that different data in different nodes creates noisy results, it is still practical in running time sensitive applications on distributed embedded systems. For example, real-time traffic flow prediction needs real-time responses from the embedded systems (e.g., mobile devices), which makes the tuning speed more important than the accuracy improvement. Another representative example is application for human activity recognition on mobile devices, i.e., mobile phone or smart watch, which needs fast response (recognition time) according to the sensor's signal and the computation power is restricted.

5.4.4 Comparison between MODES-B and MODES-I

The aforementioned *MODES*-B and *MODES*-I focus on different requirements with different assumptions. *MODES*-B tries to improve the performance of the whole system by considering the difference among different nodes. While *MODES*-I tries to improve the run-time efficiency of the tuning process by assuming the nodes and its local sub-data sets are with high similarity.

In *MODES*-B, the whole distributed embedded system is treated as an ensemble. Each hyper-parameter setting involves not only the hyper-parameter for the dedicated machine learning model, but also the weights for different model. In each iteration of optimization process, only one single proposal is trained and evaluated in the entire system. In the end, the obtained optimized hyper-parameter setting is applied for the whole ensemble, and only one classification result is generated by the system. Theoretically, since the tuned weights represent the importance of different nodes and corresponding sub-data sets, *MODES*-B can outperform other hyper-parameter tuning algorithms if sub-data sets held by different nodes are imbalanced or some sub-data sets have great noise. Well tuned weights can eliminate these drawbacks of the original system.

In *MODES*-I, multiple nodes in a distributed embedded system are treated as multiple clones of a single node. In addition, the local sub-data sets are considered as subsets of a consistent data set. This treatment relies on an assumption that the optimal hyper-parameters of the dedicated machine learning model for different nodes are with high similarity. Therefore, multiple proposals are trained and evaluated on all the available nodes at the same time, in order to accelerate the optimization of the corresponding surrogate. Ideally, the tuning process can be sped up by n times, where n is the number of nodes in the dedicated distributed embedded system. However, when there are many nodes, the resulting surrogate may not be able to generate a sufficient number of valuable proposals for evaluating the machine learning algorithms in parallel in the next iteration. That is, some of the proposed hyper-parameter settings to be evaluated have to be generated ran-

domly without any contributions to the corresponding surrogate. Moreover, since each node can make the prediction independently, *MODES*-I is more scalable, compared to *MODES*-B. Hence, node(s) can be easily added or removed without affecting the functionality of the distributed system.

5.5 Evaluation

To validate the performance of *MODES*, we consider a distributed embedded system with four ODROID-N2 boards [149]. Each of them integrates a quad-core ARM Cortex-A73 CPU, a dual-core Cortex-A53 CPU with a Mali-G52 GPU, and 32GB storage. The ODROID-N2's DDR4 RAM is running at 1320Mhz with 1.2 Volt low power consumption. One of these boards serves as the *master* which runs the mlrMBO [150], which is an R implementation of MBO. All four boards, including the master, act as *slave* nodes which run the machine learning algorithms for a specific task. These nodes are connected with each other, which makes the data transmission possible. For distributed systems with more nodes, we present emulation results for 16 nodes in Section 5.6.

Due to the limited computation power of the constructed distributed embedded system, we adopt 5 popular real-world data sets with reasonable size, i.e., at most $60,000$ instances, to evaluate the proposed *MODES* framework:

1. The MNIST [151] data set: it contains $60,000$ handwritten digits (from 0 to 9) images with 28×28 grey-scale resolution. The MNIST data set is widely used for evaluating the performance of machine learning algorithms. Here, we fit our learning task as an image classification problem on the MNIST data set.

2. The Fashion-MNIST [152] data set: it consists of Zalando's article images, where the statistics are exactly the same as the original MNIST data set, i.e., with the same number of

instances, the same image size, and the same distribution of different classes. The Fashion-MNIST is more representative for modern computer vision tasks. It usually serves as a replacement for the original MNIST data set when benchmarking machine learning algorithms, since the original MNIST classification task is easy (e.g., MLP can easily achieve the accuracy of 95%) and overused in the machine learning domain.

3. The Gas Sensor Array Drift [153] data set, denoted as Gas-drift data set: unlike vision-based data sets (e.g., MNIST-like image data sets), the Gas-drift data set is measured from 16 chemical sensors exposed to 5 distinct pure gaseous substances at different concentration levels. The resulting data set contains $13,910$ instances, each instance contains 129 attributes (dimensions), and the whole data set is divided into 5 unbalanced classes. To scale the value of features from different ranges, we normalized the data to the range of $[-1,1]$.

4. The Covertype [154] data set: it is a non-vision data set as well, coming from the US Forest Service inventory information. This data set is originally used to predict forest cover type from cartographic variables, and it is sensitive for the model settings (parameter tuning) of some popular machine learning algorithms (e.g., MLP, SVM and Random Forest). The original data set contains $581,012$ instances and 7 classes. However, the number of instances for different classes are extremely unbalanced, i.e., 100 times difference. Hence, we downsized the data set according to the size of the smallest class, i.e., each class now contains $2,747$ instances, and in total $19,229$ instances.

5. The HAR [155] data set: it consists of $10,299$ instances, which are built from the recordings of 30 subjects performing activities of daily living while carrying a waist-mounted smartphone with embedded inertial sensors. Therefore, the HAR data set naturally fits the distributed embedded systems scenario and it satisfies the assumptions of *MODES* well. As a sensing data set, six human activities are included, i.e., walking, climbing the stairs, walking down the stairs, sitting, standing, and laying.

Based on the selected data sets and the computational power of the platform, two machine learning algorithms that represent the state-of-the-art are selected as the optimization objects: 1) Multi-Layer Perceptron (MLP) [45] and 2) Random Forest (RF) [156]. The performance of these two benchmark machine learning algorithms have been well-reported on the aforementioned detests, where they can be used as the references for the performance of our *MODES*. Moreover, the performances of MLP and RF are both sensitive to the hyper-parameters, which makes MBO tuning necessary. Please note that we do not execute multiple algorithms on one node at the same time although multiprocessors are available, since it may introduce memory lack, cache miss, and execution interference.

5.5.1 Experimental Setup

The detailed configurations and settings of machine learning algorithms and further operations of data sets are introduced as follows:

5.5.1.1 Hyper-parameter Settings

To efficiently evaluate the performance of fine-tuned machine learning algorithms, for the most accuracy-sensitive hyper-parameters among all adjustable hyper-parameters in MLP and Random Forest, we select values based on experience.

For MLP, 5 hyper-parameters are tuned, i.e., the number of layers $\in [1,\ 15]$, units per layer $\in [10,\ 150]$ in steps of 10, activation $\in$ {identity, logistic, tanh, relu}, L2 penalty $\in \{10^{-5},\ 10^{-4},\ 10^{-3},\ 10^{-2}\}$, and initial learning rate for Adam $\in \{10^{-4},\ 10^{-3},\ 10^{-2},\ 10^{-1}\}$.

For Random Forest, 7 hyper-parameters are tuned, i.e., number of trees $\in [5,\ 150]$ in steps of 5, maximal number of features to consider at every split $\in$ {auto, sqrt, log2}, maximal number of

levels in trees $\in [2,40]$ and `None` represents auto mode, minimal number of samples required to split a node $\in [2,20]$ in steps of 2, minimal number of samples required at leaf node $\in [1,20]$, function to measure the quality of a split $\in$ {gini, entropy}, and whether bootstrap samples are used when building trees $\in$ {True, False}.

5.5.1.2 Pre-processing of Data Sets

Firstly, each data set is randomly split into a test set and a training set with a ratio of 1:5. Afterwards, the test set is equally divided into an evaluation test set and an unseen final test set. The evaluation test set is only used for hyper-parameter tuning, i.e., verify the performance of proposed hyper-parameter setting and the result is used to update the MBO surrogate. The unseen final test set is used to evaluate the final performance of hyper-parameters optimized by different methods and their data storage situations accordingly. Finally, in order to simulate the situation of data storage on real distributed embedded systems, four sub-data sets are generated from the overall training set by applying the following strategies:

- **Uniform Split (D1)**: Equally divide the training set into four parts.
- **Duplicated Split (D2)**: Each of the four training sets from D1 is extended with 30% data randomly selected from the remaining three parts. Therefore, each sub-data set has overlapping data with the other sub-data sets.
- **Unbalanced Split (D3)**: Divide the training set unequally with shares of $20\%, 20\%, 30\%$, and 30%.

Therefore, together with the original complete training set, we have four different training set settings, which mimic possible patterns of distributed data storage.

5.5.2 Selection of Baselines

In order to compare the performance of our proposed methods, two baselines, i.e., MBO-C and MBO-S are evaluated. MBO-S optimizes the setting of hyper-parameters for each node individually using its local data, so that each node obtains its own local optimal hyper-parameters. In MBO-C, the optimal setting of hyper-parameters is tuned by MBO for the original complete training data set. Note that MBO-C here only shows the performance of centralized hyper-parameter tuning, where the result can be regarded as the reference for other distributed tuning methods.

To be fair, each MBO tuning procedure has the same budget of maximal 100 iterations and 12 hours run-time. For *MODES*-I, only 25 iterations and 3 hours run-time are assigned, since it can evaluate four different hyper-parameter settings at the same time in each iteration, and in total 100 proposals are evaluated in the end. Afterwards, the optimized hyper-parameters are applied to train the dedicated machine learning algorithms. The training data sets are the same as those used during hyper-parameter tuning. At last, the identical testing data is adopted which is unseen for all methods.

5.5.3 Experimental Results

We evaluated all combinations with respect to the different data sets, machine learning algorithms, and data split methods. The results are shown in Tables 5.1, 5.2, 5.3, 5.4, and 5.5. We report the accuracy of the classification results for two machine learning algorithms separately for the different data sets. Since MLP and RF are modularized and standardized, the randomness from the algorithm itself in training can be ignored. This implies that even a tiny accuracy improvement is only incurred by a better hyper-parameter setting. Due to the space limitations, only the average results of *MODES*-I and MBO-S are shown in Tables, i.e., simple average accuracy of classifications

Table 5.1: The accuracy of two machine learning algorithms using different hyper-parameter tuning methods on **MNIST** data set.

Algo.	Data	*MODES*-B	*MODES*-I	MBO-S	MBO-C
MLP	D1	**0.9562**	0.9510	0.9530	0.9712
	D2	**0.9588**	0.9600	0.9582	
	D3	**0.9573**	0.9500	0.9534	
RF	D1	**0.9382**	0.9362	0.9380	0.9574
	D2	**0.9436**	0.9423	0.9420	
	D3	**0.9399**	0.9380	0.9362	

Table 5.2: The accuracy of two machine learning algorithms using different hyper-parameter tuning methods on **Fashion-MNIST** data set.

Algo.	Data	*MODES*-B	*MODES*-I	MBO-S	MBO-C
MLP	D1	**0.8610**	0.8584	0.8601	0.8882
	D2	0.8645	0.8660	0.8657	
	D3	**0.8614**	0.8578	0.8601	
RF	D1	0.8590	0.8601	0.8598	0.8792
	D2	**0.8650**	0.8637	0.8639	
	D3	**0.8623**	0.8601	0.8601	

for 4 nodes. To clearly show the comparisons, we use bold text for accuracy values to represent superiority when comparing *MODES*-B with MBO-S, and use gray background color to represent better performance obtained from *MODES*-I or MBO-S. Please note, MBO-C always outperforms the other methods, since the complete training data set is utilized for both hyper-parameter tuning and machine learning algorithm training. The results of MBO-C only show the upper bounds of the machine learning algorithms on the dedicated data sets, but is not comparable with other distributed-based methods.

The results on the MNIST data set are shown in Table 5.1. The proposed *MODES*-B outperforms MBO-S in all three data sets for both machine learning algorithms. Meanwhile, *MODES*-I is comparable with MBO-S with respect to accuracy, i.e., *MODES*-I outperforms MBO-S in 3 settings

Table 5.3: The accuracy of two machine learning algorithms using different hyper-parameter tuning methods on **Gas-drift** data set.

Algo.	Data	*MODES*-B	*MODES*-I	MBO-S	MBO-C
MLP	D1	**0.9887**	0.9875	0.9873	0.9942
	D2	0.9634	0.9897	0.9904	
	D3	**0.9898**	0.9862	0.9769	
RF	D1	0.9860	0.9825	0.9861	0.9921
	D2	0.9851	0.9857	0.9864	
	D3	**0.9860**	0.9837	0.9827	

Table 5.4: The accuracy of two machine learning algorithms using different hyper-parameter tuning methods on **Covertype** data set.

Algo.	Data	*MODES*-B	*MODES*-I	MBO-S	MBO-C
MLP	D1	0.5521	0.5454	0.5842	0.7978
	D2	0.6782	0.7011	0.7082	
	D3	0.7017	0.6605	0.7094	
RF	D1	**0.8581**	0.8562	0.8561	0.9869
	D2	0.8683	0.8684	0.8686	
	D3	**0.8581**	0.8540	0.8563	

and performs slightly worse in the other 3 settings ($< 0.35\%$), but it is much faster in hyper-parameter tuning, i.e., on average 2.7 times for both MLP and RF.

Table 5.2 shows the results for the Fashion-MNIST data set. The *MODES*-B outperforms the MBO-S in most of the cases for both machine learning algorithms. The *MODES*-I is comparable with MBO-S in all the cases with respect to the accuracy, i.e., the difference is less than 0.23%, but much faster in hyper-parameter tuning, i.e., on average 3.7 times for MLP and 2.7 times for RF.

Table 5.3 demonstrates the results of the Gas-drift data set. This data set is too easy to be predicted, since the accuracy is higher than 98% in most of the cases. This makes the efforts of *MODES* insignificant.

Table 5.5: The accuracy of two machine learning algorithms using different hyper-parameter tuning methods on **HAR** data set.

Algo.	Data	*MODES*-B	*MODES*-I	MBO-S	MBO-C
MLP	D1	0.8190	0.8428	0.8500	0.8700
	D2	**0.9168**	0.8895	0.8898	
	D3	0.8207	0.8600	0.8365	
RF	D1	0.8413	0.8503	0.8585	0.8880
	D2	**0.8788**	0.8750	0.8787	
	D3	**0.8520**	0.8395	0.8463	

The results of the Covertype data set are shown in Table 5.4. The performance of RF is much better than that of MLP. This means that RF should be applied to handle the Covertype data set. Using RF, *MODES*-B outperforms MBO-S in most of the cases, i.e., in 2/3 of the cases. Although *MODES*-I is slightly worse than MBO-S in 2/3 of the cases, it is 2.2 times faster in hyper-parameter tuning for both MLP and RF.

As demonstrated in Table 5.5, *MODES* also shows great competitiveness compared to the MBO-C and MBO-S on the HAR data set. Since the HAR data set has fewer dimensions than MNIST (562:784), considering the much smaller sample size (1:6), HAR is more difficult to predict and learn by MLP and RF algorithms. In this case, *MODES*-B suffers from a performance decrease [157] especially in D1 and D3 training set settings, where the honorable weight tuning in *MODES*-B is trivial since the size of data on each node is considerably small. It explains why the results of *MODES*-B for MLP on D1 and D3 and RF on D1 are slightly worse than with MBO-S. However, on D2 *MODES*-B outperforms MBO-S for both MLP and RF, which indicates that *MODES*-B is in a good stand when data is sufficient (D2 is supplemented by more data). Besides, *MODES*-I shows its stability. It is still comparable with MBO-S ($< 0.73\%$). Moreover, *MODES*-I is on average 3.1 times faster than MBO-S in hyper-parameter tuning for both MLP and RF.

5.5.4 Statistical Analysis

Since MBO itself has randomized decisions (including the selection of the initialized points and the proposals based on the surrogates), it is necessary to analyze the variance to verify the correctness of our evaluation results. However, due to the limited computation power of the real distributed embedded system, the experiments are extremely time consuming. Depending on the size of different data sets, one set of experiments, i.e., each table, takes 80 to 245 CPU hours. Reporting statistic analysis on all of them would require a lot of tests.

To demonstrate that MBO can be applied for those data sets with good statistical stability, we consider the most unstable experiment, namely, Covertype data set.[2] We tested *only* RF with MBO-C on Covertype data set for 50 times. The repeated experiments took 35 hours on our evaluation platform. The result shows that the variance is less than 0.13%. As a result, the evaluation in our work is considered stable and reproducible.

5.5.5 Discussion

For the applicability, the results in Table 5.1, 5.2, 5.3, 5.4, and 5.5 show that *MODES*-B outperforms MBO-S in most of the evaluated cases. Although *MODES*-I shows less competitiveness in classification accuracy, it significantly improves the run-time efficiency, which is even much more important than accuracy in some real world timing-sensitive applications, e.g., autonomous driving systems [159]. In addition, *MODES*-B can handle the situations much better than MBO-S if the data size is unbalances in different nodes, i.e., D3 in our evaluations. *MODES*-B for MLP outperforms MBO-S when D3 is applied in 3 over 4 cases (The case for Covertype data set in Table 5.4 is not considered here, since MLP is not the best suitable algorithm for Covertype data set compared

[2]The Covertype data set is unstable, reported in [158].

to RF). *MODES*-B for RF outperforms MBO-S in all the five cases when D3 is applied.

In summary, for a great variety of data sets and/or applications without data aggregation, the method *MODES*, with two different modes, outperforms the traditional approach MBO-S in terms of either accuracy (*MODES*-B) or run-time efficiency (*MODES*-I) without much accuracy degradation.

5.6 Scalability

In order to investigate the scalability of the *MODES*, an emulation platform is established by using a cache-coherent SMP, consisting of two 64-bit Intel Xeon Processor E5-2650Lv4, with 35 MB cache and 64 GB main memory. There are 28 physical cores in total, and each core is considered as a virtual node.

The size of data sets in Section 5.5 is too small as the data has to be distributed to a large number of nodes. To demonstrate the scalability of the *MODES*, we evaluated data sets chosen from the following two data sets on 16 nodes:

1. The Infinite-MNIST [160] is also known as MNIST8M data set: it produces an infinite supply of digit images derived from the well-known MNIST data set using pseudo-random deformations and translations.

2. The SVHN [161]: a real-world image data set for developing machine learning and object recognition algorithms. It can be seen as similar in favor of MNIST, but incorporates an order of magnitude more labeled data (over 600,000 digit images) and comes from a significantly harder, unsolved, real world problem (recognizing digits and numbers in natural scene images). SVHN is obtained from house numbers in Google Street View images, where each

Table 5.6: The accuracy of two machine learning algorithms using different hyper-parameter tuning methods on **Infinite MNIST** data set.

Algo.	Data	*MODES*-B		*MODES*-I		MBO-S	MBO-C
		original	scaled-np	scaled-n	scaled-4		
MLP	D1	**0.9676**	0.9658	0.9634	0.9619	0.9662	0.9966
	D2	0.9683	**0.9723**	0.9695	0.9673	0.9710	
	D3	0.9604	0.9623	0.9598	0.9615	0.9637	
RF	D1	0.9396	**0.9407**	0.9393	0.9390	0.9378	0.9832
	D2	0.9444	**0.9468**	0.9404	0.9440	0.9450	
	D3	**0.9458**	0.9377	0.9369	0.9369	0.9371	

image is 32-by-32 digit ranging from 0 to 9.

To eliminate the influence from the size of sub-data set in each node, following the size of MNIST data set used in Section 5.5 (i.e., 60,000 training samples for 4 nodes), we enlarge the size of data set linearly. That is, only 240,000 training samples in total for both data sets were chosen individually in our experiments. Meanwhile, similar sub-data sets generation strategies are applied: (1) equally divided the training set into 16 parts, denoted as **D1**; (2) each sub-data set from D1 is extended with 5,000 samples randomly selected from the remaining samples, denoted as **D2**; (3) divide the training samples unequally, i.e., 8 sets with 5% share and 8 sets with 7.5% share, denoted as **D3**.

For *MODES*-B, two stop conditions are considered: (a) *original* and (b) *scaled-np*. In the *original* mode, the tuning procedure has the same budget with aforementioned experiments, i.e., maximal 100 iterations and 12 hours run-time. In the *scaled-np* mode, the budget is scaled according to the number of hyper-parameters. Although only 5 hyper-parameters for the MLP model have to be tuned for MBO-S and MBO-C, since each node also has one weight parameter to be tuned, we need to tune 21 hyper-parameters for *MODES*-B when deploying MLP on 16 nodes. Therefore, the tuning procedure budget is 420 iterations or 50.4 hours run-time accordingly (4.2 times of

Table 5.7: The accuracy of two machine learning algorithms using different hyper-parameter tuning methods on **SVHN** data set.

Algo.	Data	*MODES*-B		*MODES*-I		MBO-S	MBO-C
		original	scaled-np	scaled-n	scaled-4		
MLP	D1	0.7285	0.7351	0.7295	0.7308	0.7429	0.8626
	D2	0.7551	0.7579	0.7544	0.7567	0.7621	
	D3	0.7227	0.7398	0.7329	0.7309	0.7420	
RF	D1	**0.6435**	0.6433	0.6375	0.6398	0.6370	0.755
	D2	0.6556	**0.6580**	0.6559	0.6552	0.6554	
	D3	**0.6454**	0.6451	0.6313	0.6371	0.6357	

the original budget). For RF, 3.3 times of the original budget is set. The stop condition (b) is added, since the dimension of search space for hyper-parameter optimizing becomes larger when the number of nodes increases. Therefore, more tuning iterations and time budget are needed to achieve an optimal solution.

The parallelism of optimizing a surrogate model has its bottleneck, as described earlier. Even when the number of nodes is increased, the surrogate may not be able to be used to generate a sufficient number of valuable hyper-parameter settings to be evaluated in the next iteration. This results in a situation that some of the generated hyper-parameter settings for the next iteration can be useless. Therefore, more iterations and time are expected. To evaluate this situation, two stop conditions for *MODES*-I are designed: (a) *scaled-n* and (b) *scaled-4*. In the *scaled-n* mode, the budget of tuning procedure is scaled according to the number of nodes, i.e., $\frac{1}{n}$ of the original budget, that is, 7 iterations and 1 hour for 16 nodes. In the *scaled-4* mode, the budget is 25 iterations or 3 hours, which is the same as the experiments on 4 nodes in Section 5.5.

The results of the Infinite MNIST data set are shown in Table 5.6. *MODES*-B outperforms MBO-S in most of the evaluated cases for both MLP and RF algorithms. *MODES*-I significantly improves the run-time efficiency without much accuracy degradation. Table 5.7 shows the results of the

SVHN data set. Both MLP and RF cannot make decent predictions in a distributed setting, i.e., the accuracy of classification is less than 80% for all the evaluated cases. This is due to the architecture limitation [162] of MLP and RF themselves on complicated digit image data set. Notice that when MLP is applied, MBO-S outperforms both *MODES*-B and *MODES*-I in all the evaluated cases. On the contrary, *MODES*-B and *MODES*-I outperform MBO-S in all the evaluated cases if RF is applied.

Regarding to different stop conditions for both *MODES*-B and *MODES*-I, extension of the budget for tuning cannot significantly improve the performance of the method. Sometimes, the over-fitting due to more tuning iterations even worsen the accuracy of classifications. We applied two different test sets, one evaluation test set is only used for hyper-parameter tuning, and another unseen final test set is used for final performance evaluation. If the hyper-parameters of a machine learning model are over tuned for the first evaluation test set, it may have bad performance when a new test set is applied, due to the weak generalization of the trained model.

For the scalability, *MODES* can still work well when the number of nodes increases, with the dedicated machine learning algorithm and the addressable data set, i.e., the evaluation in Table 5.6. In Table 5.7, *MODES* for MLP does not work at all, only because that the SVHN data set is over complicated for both MLP and RF. Hence, both MLP and RF show poor prediction results. Even so, we still observe that *MODES* has relative better performance than MBO-S with RF which demonstrates the superiority of *MODES*.

CHAPTER 6: CONCLUSION AND FUTURE DIRECTION

We summary the contribution of the proposed statistical and stochastic learning algorithm in three aspects. **Firstly**, we study and formulate the problem of statistical 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 sparse discriminant analysis, 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 [163] 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. **Secondly**, to achieve the above goals, we design the stochastic algorithm which leverages the direct estimation of learning model to derive a distributed loss function of specific statistical learning, parameterizes the distributed loss function with local/global estimates through bootstrapping, and approximates a global estimation of key learning vector by optimizing the "distributed bootstrapping loss function" and further improving the estimation through parallel computing. **Finally**, we demonstrate the performance gain of our improved learning algorithms on both pseudo-random simulation and real-world applications. The real-world applications ranges from intelligent medical systems to distributed environment monitoring systems.

Although we has begun to address the challenges of applying statistical and stochastic learning algorithm in distributed intelligent systems discussed in this dissertation, there are a number of critical open directions in distributed/federated learning that are yet to be explored. We briefly list

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

some open problems below.

- Extreme communication schemes: It remains to be seen how much communication is necessary in distributed learning. For example, can we gain a deeper theoretical and empirical understanding of one-shot/few-shot communication schemes in massive and statistically heterogeneous networks?

- Novel models of asynchrony: Two communication schemes most commonly studied in distributed optimization are bulk synchronous and asynchronous approaches. However, in distributed networks, each device is often undedicated to the task at hand and most devices are not active on any given iteration. Can we devise device-centric communication models beyond synchronous and asynchronous training, where each device can decide when to interact with the server (rather than being dedicated to the workload)?

- Heterogeneity diagnostics: Recent works have aimed to quantify statistical heterogeneity through various metrics, though these metrics must be calculated during training. This motivates the following questions: Are there simple diagnostics that can be used to quantify systems and statistical heterogeneity before training? Can these diagnostics be exploited to further improve the convergence of federated optimization methods?

- Productionizing distributed learning: There are a number of practical concerns that arise when running federated learning in production. For example, how can we handle issues such as concept drift (when the underlying data-generation model changes over time); diurnal variations (when the devices exhibit different behavior at different times of the day or week); and cold start problems (when new devices enter the network)?

These challenging problems (and more) will require collaborative efforts from a wide range of research communities.

APPENDIX A: LIST OF PUBLISHED PAPERS

Referred Conference, Workshop and Poster Papers (* equal contribution)

- Ashkan Farhangi*, **Jiang Bian***, Jun Wang, and Zhishan Guo, A Deep Learning Strategy for I/O Scheduling in Storage Systems, *Proceedings of the 40th IEEE Real-Time Systems Symposium, BP Session (**RTSS'19 BP**).*

- **Jiang Bian**, Weibo Wang, Xiang Zhang, Wei Wang, Arthur Huang, Zhishan Guo. On Generating Dominators of Customer Preferences, *Proceedings of IEEE International Conference on Big Data, 5th Special Session on Intelligent Data Mining (**BigData'19**).*

- Haoyi Xiong*, Kafeng Wang*, **Jiang Bian***, Zhanxin Zhu, Chengzhong Xu, Zhishan Guo, Jun Huan. SpHMC: Spectral Hamiltonian Monte Carlo, *Proceedings of the 33nd AAAI Conference on Artificial Intelligence (**AAAI'19**).*

- Haoyi Xiong, Wei Cheng, Yanjie Fu, Wenqing Hu, **Jiang Bian**, Zhishan Guo. De-Biasing Covariancce-Regularized Discriminant Analysis, *Proceedings of the 28th International Joint Conferences on Artificial Intelligence (**IJCAI'18**).*

- **Jiang Bian***, Haoyi Xiong*, Yanjie Fu, Sajal K. Das. CSWA: Aggregation-Free Spatial-Temporal Community Sensing, *Proceedings of the 32nd AAAI Conference on Artificial Intelligence (**AAAI'18**).*

- **Jiang Bian**, Haoyi Xiong*, Wei Cheng, Yanjie Fu, Wenqing Hu, Zhishan Guo. Multi-Party Sparse Discriminant Learning, *Proceedings of the 17th IEEE International Conference on Data Mining (**ICDM'17**).*

- Haoyi Xiong*, Wei Cheng, Wenqing Hu, **Jiang Bian**, Zhishan Guo. AWDA: An Adaptive Wishart Discriminant Analysis, *Proceedings of the 17th IEEE International Conference on Data Mining (**ICDM'17**).*

- **Jiang Bian**, Laura E. Barnes, Guanling Chen, Haoyi Xiong*. Early Detection of Diseases using Electronic Health Records Data and Covariance-Regularized Linear Discriminant Analysis, *Proceedings of IEEE International Conference on Biomedical and Health Informatics (**BHI'17**)*.

Referred Journal Articles

- Kafeng Wang, Haoyi Xiong, **Jiang Bian**, Zhanxing Zhu, Qian Gao, Zhishan Guo, Cheng-Zhong Xu, and Jun Huan. Sampling Sparse Representations with Randomized Measurement Langevin Dynamics, *IEEE Transactions on Knowledge Discovery from Data (**TKDD**)*, 2020, to appear.

- Baoxin Zhao*, Haoyi Xiong*, **Jiang Bian**, Zhishan Guo, Cheng-Zhong Xu, and Dejing Dou. COMO: Widening Deep Neural Networks with COnvolutional MaxOut, *IEEE Transactions on Multimedia (**TMM**)*, 2020.

- **Jiang Bian**, Haoyi Xiong, Yanjie Fu, Zhishan Guo. *MP^2SDA*: Multi-Party Parallelized Sparse Discriminant Learning, *IEEE Transactions on Knowledge Discovery from Data (**TKDD**)*, 14.3 (2020): 1-22.

- **Jiang Bian**, Sijia Yang, Lichen Wang, Haoyi Xiong, Yanjie Fu, Zeyi Sun, Zhishan Guo, Jun Wang. CRLEDD: Regularized Causalities Learning for Early Detection of Diseases using Electronic Health Record (EHR) Data, *IEEE Transactions on Emerging Topics in Computational Intelligence (**TETCI**)*, 2020.

- Haoyi Xiong, Wei Cheng, **Jiang Bian**, Wenqing Hu, Zhishan Guo. DBSDA: Lowering the Error Bound of Sparse Linear Discriminant Analysis via Model De-Biasing, *IEEE Transactions on Neural Networks and Learning Systems (**TNNLS**), 99(2018): 1-11*.

APPENDIX B: PERMISSION TO REUSE PUBLISHED MATERIAL

IEEE Requesting permission to reuse content from an IEEE publication

Multi-party Sparse Discriminant Learning

Conference Proceedings: 2017 IEEE International Conference on Data Mining (ICDM)
Author: Jiang Bian
Publisher: IEEE
Date: Nov. 2017

Thesis / Dissertation Reuse

The IEEE does not require individuals working on a thesis to obtain a formal reuse license, however, you may print out this statement to be used as a permission grant:

Requirements to be followed when using any portion (e.g., figure, graph, table, or textual material) of an IEEE copyrighted paper in a thesis:

1) In the case of textual material (e.g., using short quotes or referring to the work within these papers) users must give full credit to the original source (author, paper, publication) followed by the IEEE copyright line © 2011 IEEE.
2) In the case of illustrations or tabular material, we require that the copyright line © [Year of original publication] IEEE appear prominently with each reprinted figure and/or table.
3) If a substantial portion of the original paper is to be used, and if you are not the senior author, also obtain the senior author's approval.

Requirements to be followed when using an entire IEEE copyrighted paper in a thesis:

1) The following IEEE copyright/ credit notice should be placed prominently in the references: © [year of original publication] IEEE. Reprinted, with permission, from [author names, paper title, IEEE publication title, and month/year of publication]
2) Only the accepted version of an IEEE copyrighted paper can be used when posting the paper or your thesis on-line.
3) In placing the thesis on the author's university website, please display the following message in a prominent place on the website: In reference to IEEE copyrighted material which is used with permission in this thesis, the IEEE does not endorse any of [university/educational entity's name goes here]'s products or services. Internal or personal use of this material is permitted. If interested in reprinting/republishing IEEE copyrighted material for advertising or promotional purposes or for creating new collective works for resale or redistribution, please go to http://www.ieee.org/publications_standards/publications/rights/rights_link.html to learn how to obtain a License from RightsLink.

If applicable, University Microfilms and/or ProQuest Library, or the Archives of Canada may supply single copies of the dissertation.

BACK | CLOSE WINDOW

Copyright Clearance Center | RightsLink®

Home | Help | Email Support | Sign in | Create Account

IEEE Requesting permission to reuse content from an IEEE publication

CRLEDD: Regularized Causalities Learning for Early Detection of Diseases Using Electronic Health Record (EHR) Data

Author: Jiang Bian
Publication: IEEE Transactions on Emerging Topics in Computational Intelligence
Publisher: IEEE
Date: Dec 31, 1969

Thesis / Dissertation Reuse

The IEEE does not require individuals working on a thesis to obtain a formal reuse license, however, you may print out this statement to be used as a permission grant:

Requirements to be followed when using any portion (e.g., figure, graph, table, or textual material) of an IEEE copyrighted paper in a thesis:

1) In the case of textual material (e.g., using short quotes or referring to the work within these papers) users must give full credit to the original source (author, paper, publication) followed by the IEEE copyright line © 2011 IEEE.
2) In the case of illustrations or tabular material, we require that the copyright line © [Year of original publication] IEEE appear prominently with each reprinted figure and/or table.
3) If a substantial portion of the original paper is to be used, and if you are not the senior author, also obtain the senior author's approval.

Requirements to be followed when using an entire IEEE copyrighted paper in a thesis:

1) The following IEEE copyright/ credit notice should be placed prominently in the references: © [year of original publication] IEEE. Reprinted, with permission, from [author names, paper title, IEEE publication title, and month/year of publication]
2) Only the accepted version of an IEEE copyrighted paper can be used when posting the paper or your thesis on-line.
3) In placing the thesis on the author's university website, please display the following message in a prominent place on the website: In reference to IEEE copyrighted material which is used with permission in this thesis, the IEEE does not endorse any of [university/educational entity's name goes here]'s products or services. Internal or personal use of this material is permitted. If interested in reprinting/republishing IEEE copyrighted material for advertising or promotional purposes or for creating new collective works for resale or redistribution, please go to http://www.ieee.org/publications_standards/publications/rights/rights_link.html to learn how to obtain a License from RightsLink.

If applicable, University Microfilms and/or ProQuest Library, or the Archives of Canada may supply single copies of the dissertation.

BACK | CLOSE WINDOW

© 2020 Copyright - All Rights Reserved | Copyright Clearance Center, Inc. | Privacy statement | Terms and Conditions
Comments? We would like to hear from you. E-mail us at customercare@copyright.com

LIST OF REFERENCES

[1] L. Rosasco, J. Bouvrie, R. Rifkin, and T. Poggio, "Statistical learning theory and applications," *McGovern Institute for Brain Research, Center for Biological and Computational Learning, MIT, Cambridge, MA*, 2008.

[2] L. Bottou, "Large-scale machine learning with stochastic gradient descent," in *Proceedings of COMPSTAT'2010*, pp. 177–186, Springer, 2010.

[3] Y. Zhou, D. Wilkinson, R. Schreiber, and R. Pan, "Large-scale parallel collaborative filtering for the netflix prize," in *International Conference on Algorithmic Applications in Management*, pp. 337–348, Springer, 2008.

[4] J. H. Friedman, "On bias, variance, 0/1—loss, and the curse-of-dimensionality," *Data mining and knowledge discovery*, vol. 1, no. 1, pp. 55–77, 1997.

[5] K. Bhaduri, R. Wolff, C. Giannella, and H. Kargupta, "Distributed decision-tree induction in peer-to-peer systems," *Statistical Analysis and Data Mining*, vol. 1, no. 2, pp. 85–103, 2008.

[6] J.-P. Zhang, Z.-W. Li, and J. Yang, "A parallel svm training algorithm on large-scale classification problems," in *Proceedings of 2005 International Conference on Machine Learning and Cybernetics*, vol. 3, pp. 1637–1641, IEEE, 2005.

[7] F. J. Provost and D. N. Hennessy, "Scaling up: Distributed machine learning with cooperation," in *AAAI/IAAI, Vol. 1*, pp. 74–79, 1996.

[8] G.-J. Qi, C. Aggarwal, D. Turaga, D. Sow, and P. Anno, "State-driven dynamic sensor selection and prediction with state-stacked sparseness," in *Proceedings of the 21th ACM*

SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 945–954, ACM, 2015.

[9] H. Chen and R. Cramer, "Algebraic geometric secret sharing schemes and secure multi-party computations over small fields," in *Annual International Cryptology Conference*, pp. 521–536, Springer, 2006.

[10] Z. Fengy, H. Xiong, C. Song, S. Yang, B. Zhao, L. Wang, Z. Chen, S. Yang, L. Liu, and J. Huan, "Securegbm: Secure multi-party gradient boosting," *arXiv preprint arXiv:1911.11997*, 2019.

[11] E. P. Xing, Q. Ho, W. Dai, J. K. Kim, J. Wei, S. Lee, X. Zheng, P. Xie, A. Kumar, and Y. Yu, "Petuum: A new platform for distributed machine learning on big data," *IEEE Transactions on Big Data*, vol. 1, no. 2, pp. 49–67, 2015.

[12] J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, M. Mao, A. Senior, P. Tucker, K. Yang, Q. V. Le, *et al.*, "Large scale distributed deep networks," in *Advances in neural information processing systems*, pp. 1223–1231, 2012.

[13] K. I. Tsianos, S. Lawlor, and M. G. Rabbat, "Consensus-based distributed optimization: Practical issues and applications in large-scale machine learning," in *Communication, Control, and Computing (Allerton), 2012 50th Annual Allerton Conference on*, pp. 1543–1550, IEEE, 2012.

[14] P. Smyth, M. Welling, and A. U. Asuncion, "Asynchronous distributed learning of topic models," in *Advances in Neural Information Processing Systems*, pp. 81–88, 2009.

[15] R. Ormándi, I. Hegedűs, and M. Jelasity, "Gossip learning with linear models on fully distributed data," *Concurrency and Computation: Practice and Experience*, vol. 25, no. 4, pp. 556–571, 2013.

[16] Z. L. H. L. L. Z. S. Y. Xiangbo Shu, Jinhui Tang, "Personalized age progression with bi-level aging dictionary learning," *IEEE transactions on pattern analysis and machine intelligence*, vol. 40, no. 4, pp. 905–917, 2017.

[17] J. Bian, H. Xiong, Y. Fu, and S. K. Das, "Cswa: Aggregation-free spatial-temporal community sensing," in *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[18] J. Tang, R. Hong, S. Yan, T.-S. Chua, G.-J. Qi, and R. Jain, "Image annotation by k nn-sparse graph-based label propagation over noisily tagged web images," *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 2, no. 2, p. 14, 2011.

[19] L. Tian and Q. Gu, "Communication-efficient distributed sparse linear discriminant analysis," *arXiv preprint arXiv:1610.04798*, 2016.

[20] T. Cai and W. Liu, "A direct estimation approach to sparse linear discriminant analysis," *Journal of the American Statistical Association*, vol. 106, no. 496, pp. 1566–1577, 2011.

[21] E. P. Xing, Q. Ho, P. Xie, and D. Wei, "Strategies and principles of distributed machine learning on big data," *Engineering*, vol. 2, no. 2, pp. 179–195, 2016.

[22] R. O. Duda, P. E. Hart, and D. G. Stork, *Pattern Classification (2nd Ed)*. Wiley, 2001.

[23] B. Kulis *et al.*, "Metric learning: A survey," *Foundations and Trends in Machine Learning*, vol. 5, no. 4, pp. 287–364, 2013.

[24] R. Peck and J. Van Ness, "The use of shrinkage estimators in linear discriminant analysis," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, no. 5, pp. 530–537, 1982.

[25] T. T. Cai, Z. Ren, H. H. Zhou, *et al.*, "Estimating structured high-dimensional covariance and precision matrices: Optimal rates and adaptive estimation," *Electronic Journal of Statistics*, vol. 10, no. 1, pp. 1–59, 2016.

[26] V. A. Marčenko and L. A. Pastur, "Distribution of eigenvalues for some sets of random matrices," *Mathematics of the USSR-Sbornik*, vol. 1, no. 4, p. 457, 1967.

[27] A. Zollanvari and E. R. Dougherty, "Random matrix theory in pattern classification: An application to error estimation," in *2013 Asilomar Conference on Signals, Systems and Computers*, 2013.

[28] W. Krzanowski, P. Jonathan, W. McCarthy, and M. Thomas, "Discriminant analysis with singular covariance matrices: methods and applications to spectroscopic data," *Applied Statistics*, pp. 101–115, 1995.

[29] J. Ye, R. Janardan, and Q. Li, "Two-dimensional linear discriminant analysis," in *NIPS*, (Cambridge, MA, USA), pp. 1569–1576, 2004.

[30] Z. Zhang *et al.*, "Learning metrics via discriminant kernels and multidimensional scaling: Toward expected euclidean representation," in *ICML*, vol. 2, pp. 872–879, 2003.

[31] S. Kaski and J. Peltonen, "Informative discriminant analysis," in *ICML*, pp. 329–336, 2003.

[32] D. M. Witten and R. Tibshirani, "Covariance-regularized regression and classification for high dimensional problems," *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, vol. 71, no. 3, pp. 615–636, 2009.

[33] R. J. Durrant and A. Kabán, "Random projections as regularizers: learning a linear discriminant from fewer observations than dimensions," *Machine Learning*, vol. 99, no. 2, pp. 257–286, 2015.

[34] P. J. Bickel, E. Levina, *et al.*, "Some theory for fisher's linear discriminant function,naive bayes', and some alternatives when there are many more variables than observations," *Bernoulli*, vol. 10, no. 6, pp. 989–1010, 2004.

[35] J. Friedman, T. Hastie, and R. Tibshirani, "Sparse inverse covariance estimation with the graphical lasso," *Biostatistics*, vol. 9, no. 3, pp. 432–441, 2008.

[36] C.-H. Zhang and S. S. Zhang, "Confidence intervals for low dimensional parameters in high dimensional linear models," *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, vol. 76, no. 1, pp. 217–242, 2014.

[37] Z. Tirandaz, G. Akbarizadeh, and H. Kaabi, "Polsar image segmentation based on feature extraction and data compression using weighted neighborhood filter bank and hidden markov random field-expectation maximization," *Measurement*, vol. 153, p. 107432, 2020.

[38] F. Samadi, G. Akbarizadeh, and H. Kaabi, "Change detection in sar images using deep belief network: a new training approach based on morphological images," *IET Image Processing*, vol. 13, no. 12, pp. 2255–2264, 2019.

[39] C. Oliver and S. Quegan, *Understanding synthetic aperture radar images*. SciTech Publishing, 2004.

[40] G. Akbarizadeh, "A new statistical-based kurtosis wavelet energy feature for texture recognition of sar images," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 50, no. 11, pp. 4358–4368, 2012.

[41] Z. Tirandaz and G. Akbarizadeh, "A two-phase algorithm based on kurtosis curvelet energy and unsupervised spectral regression for segmentation of sar images," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 9, no. 3, pp. 1244–1264, 2015.

[42] G. Akbarizadeh and Z. Tirandaz, "Segmentation parameter estimation algorithm based on curvelet transform coefficients energy for feature extraction and texture description of sar

images," in *2015 7th conference on information and knowledge technology (IKT)*, pp. 1–4, IEEE, 2015.

[43] F. Sharifzadeh, G. Akbarizadeh, and Y. S. Kavian, "Ship classification in sar images using a new hybrid cnn–mlp classifier," *Journal of the Indian Society of Remote Sensing*, vol. 47, no. 4, pp. 551–562, 2019.

[44] S. Lawrence, C. L. Giles, A. C. Tsoi, and A. D. Back, "Face recognition: A convolutional neural-network approach," *IEEE transactions on neural networks*, vol. 8, no. 1, pp. 98–113, 1997.

[45] M. W. Gardner and S. Dorling, "Artificial neural networks (the multilayer perceptron)—a review of applications in the atmospheric sciences," *Atmospheric environment*, vol. 32, no. 14-15, pp. 2627–2636, 1998.

[46] F. Wang and J. Sun, "Psf: A unified patient similarity evaluation framework through metric learning with weak supervision," *Biomedical and Health Informatics, IEEE Journal of*, vol. 19, pp. 1053–1060, May 2015.

[47] J. Sun, F. Wang, J. Hu, and S. Edabollahi, "Supervised patient similarity measure of heterogeneous patient records," *ACM SIGKDD Explorations Newsletter*, vol. 14, no. 1, pp. 16–24, 2012.

[48] K. Ng, J. Sun, J. Hu, and F. Wang, "Personalized predictive modeling and risk factor identification using patient similarity," *AMIA Summit on Clinical Research Informatics (CRI)*, 2015.

[49] J. Zhang, H. Xiong, Y. Huang, H. Wu, K. Leach, and L. E. Barnes, "MSEQ: Early detection of anxiety and depression via temporal orders of diagnoses in electronic health data," in *BigData*, IEEE, 2015.

[50] S. Jensen and U. SPSS, "Mining medical data for predictive and sequential patterns: Pkdd 2001," in *Proceedings of the 5th European Conference on Principles and Practice of Knowledge Discovery in Databases*, 2001.

[51] C. Liu, F. Wang, J. Hu, and H. Xiong, "Temporal Phenotyping from Longitudinal Electronic Health Records: A Graph Based Framework," in *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '15, (New York, NY, USA), pp. 705–714, ACM, 2015.

[52] A. Tarantola, *Inverse problem theory and methods for model parameter estimation*. SIAM, 2005.

[53] D. Panda, S. Singh, S. Mukherjee, and S. Chakraborty, "Comparing and analysis of different optimization techniques on sparse multi-class data," in *2019 International Conference on Computational Intelligence and Knowledge Economy (ICCIKE)*, pp. 528–531, IEEE, 2019.

[54] F. O'Sullivan, "A statistical perspective on ill-posed inverse problems," *Statistical science*, pp. 502–518, 1986.

[55] Z. Qiao, L. Zhou, and J. Z. Huang, "Effective linear discriminant analysis for high dimensional, low sample size data," in *Proceeding of the World Congress on Engineering*, vol. 2, pp. 2–4, Citeseer, 2008.

[56] F. Scholz, "Maximum likelihood estimation," *Encyclopedia of statistical sciences*, 1985.

[57] S. Puntanen and G. P. Styan, "The equality of the ordinary least squares estimator and the best linear unbiased estimator," *The American Statistician*, vol. 43, no. 3, pp. 153–161, 1989.

[58] H. W. Engl and C. W. Groetsch, *Inverse and ill-posed problems*, vol. 4. Elsevier, 2014.

[59] V. N. Vapnik and V. Vapnik, *Statistical learning theory*, vol. 1. Wiley New York, 1998.

[60] V. Vapnik, "Principles of risk minimization for learning theory," in *NIPS*, pp. 831–838, 1991.

[61] V. Vapnik, E. Levin, and Y. Le Cun, "Measuring the vc-dimension of a learning machine," *Neural computation*, vol. 6, no. 5, pp. 851–876, 1994.

[62] K. van den Doel and U. M. Ascher, "On level set regularization for highly ill-posed distributed parameter estimation problems," *Journal of Computational Physics*, vol. 216, no. 2, pp. 707–723, 2006.

[63] J. A. Suykens and J. Vandewalle, "Least squares support vector machine classifiers," *Neural processing letters*, vol. 9, no. 3, pp. 293–300, 1999.

[64] R. Tibshirani, "Regression shrinkage and selection via the lasso," *Journal of the Royal Statistical Society. Series B (Methodological)*, pp. 267–288, 1996.

[65] X. L. Tan, "Optimal estimation of slope vector in high-dimensional linear transformation model," *arXiv preprint arXiv:1604.07087*, 2016.

[66] D. W. Hosmer Jr, S. Lemeshow, and R. X. Sturdivant, *Applied logistic regression*, vol. 398. John Wiley & Sons, 2013.

[67] L. K. Hansen and P. Salamon, "Neural network ensembles," *IEEE transactions on pattern analysis and machine intelligence*, vol. 12, no. 10, pp. 993–1001, 1990.

[68] E. R. Dubberke, K. A. Reske, L. C. McDonald, and V. J. Fraser, "Icd-9 codes and surveillance for clostridium difficile–associated disease," *Emerging infectious diseases*, vol. 12, no. 10, p. 1576, 2006.

[69] HCUP, "Appendix a - clinical classification software-diagnoses," 2014.

[70] O. Banerjee, L. E. Ghaoui, and A. d'Aspremont, "Model selection through sparse maximum likelihood estimation for multivariate gaussian or binary data," *Journal of Machine learning research*, vol. 9, no. Mar, pp. 485–516, 2008.

[71] S. Boyd and L. Vandenberghe, *Convex optimization*. Cambridge university press, 2004.

[72] J. Friedman, T. Hastie, H. Höfling, R. Tibshirani, *et al.*, "Pathwise coordinate optimization," *The Annals of Applied Statistics*, vol. 1, no. 2, pp. 302–332, 2007.

[73] B. Bayar, N. Bouaynaya, and R. Shterenberg, "Smurc: High-dimension small-sample multivariate regression with covariance estimation," *IEEE journal of biomedical and health informatics*, vol. 21, no. 2, pp. 573–581, 2017.

[74] J. C. Turner and A. Keller, "College Health Surveillance Network: Epidemiology and Health Care Utilization of College Students at U.S. 4-Year Universities," *Journal of American College Health: J of ACH*, p. 0, June 2015.

[75] K. S. Kendler, J. M. Hettema, F. Butera, C. O. Gardner, and C. A. Prescott, "Life event dimensions of loss, humiliation, entrapment, and danger in the prediction of onsets of major depression and generalized anxiety," *Archives of general psychiatry*, vol. 60, no. 8, pp. 789–796, 2003.

[76] J. H. Friedman, "Regularized discriminant analysis," *Journal of the American statistical association*, vol. 84, no. 405, pp. 165–175, 1989.

[77] S. H. Huang, P. LePendu, S. V. Iyer, M. Tai-Seale, D. Carrell, and N. H. Shah, "Toward personalizing treatment for depression: predicting diagnosis and severity," *Journal of the American Medical Informatics Association*, vol. 21, no. 6, pp. 1069–1075, 2014.

[78] K. Kroenke and R. L. Spitzer, "The PHQ-9: a new depression diagnostic and severity measure," *Psychiatr Ann*, vol. 32, no. 9, pp. 1–7, 2002.

[79] J. Jankova, S. van de Geer, *et al.*, "Confidence intervals for high-dimensional inverse covariance estimation," *Electronic Journal of Statistics*, vol. 9, no. 1, pp. 1205–1229, 2015.

[80] R. S. Bucks, Y. Gidron, P. Harris, J. Teeling, K. A. Wesnes, and V. H. Perry, "Selective effects of upper respiratory tract infection on cognition, mood and emotion processing: A prospective study," *Brain, Behavior, and Immunity*, vol. 22, pp. 399–407, Mar. 2008.

[81] S. Cohen, "Psychological Stress and Susceptibility to Upper Respiratory Infections," *Am J Respir Crit Care Med*, vol. 152, pp. S53–S58, Oct. 1995.

[82] A. Javanmard and A. Montanari, "Confidence intervals and hypothesis testing for high-dimensional regression.," *Journal of Machine Learning Research*, vol. 15, no. 1, pp. 2869–2909, 2014.

[83] A. J. Rothman, E. Levina, and J. Zhu, "Sparse multivariate regression with covariance estimation," *Journal of Computational and Graphical Statistics*, vol. 19, no. 4, pp. 947–962, 2010.

[84] A. DasGupta, *Asymptotic theory of statistics and probability*. Springer Science & Business Media, 2008.

[85] C.-J. Lin, "Libsvm data: Classification (binary class)." `https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary.html`, 2017.

[86] R. Tibshirani, T. Hastie, B. Narasimhan, and G. Chu, "Diagnosis of multiple cancer types by shrunken centroids of gene expression," *Proceedings of the National Academy of Sciences*, vol. 99, no. 10, pp. 6567–6572, 2002.

[87] L. Clemmensen, T. Hastie, D. Witten, and B. Ersbøll, "Sparse discriminant analysis," *Technometrics*, vol. 53, no. 4, 2011.

[88] J. Shao, Y. Wang, X. Deng, S. Wang, *et al.*, "Sparse linear discriminant analysis by thresholding for high dimensional data," *The Annals of Statistics*, vol. 39, no. 2, pp. 1241–1265, 2011.

[89] G.-J. Qi, J. Tang, Z.-J. Zha, T.-S. Chua, and H.-J. Zhang, "An efficient sparse metric learning in high-dimensional space via l 1-penalized log-determinant regularization," in *Proceedings of the 26th Annual International Conference on Machine Learning*, pp. 841–848, ACM, 2009.

[90] X. Shu, J. Tang, G.-J. Qi, Z. Li, Y.-G. Jiang, and S. Yan, "Image classification with tailored fine-grained dictionaries," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 28, no. 2, pp. 454–467, 2016.

[91] M. H. Pesaran, Y. Shin, and R. P. Smith, "Pooled mean group estimation of dynamic heterogeneous panels," *Journal of the American Statistical Association*, vol. 94, no. 446, pp. 621–634, 1999.

[92] S. Raudys and R. P. Duin, "Expected classification error of the fisher linear classifier with pseudo-inverse covariance matrix," *Pattern Recognition Letters*, vol. 19, no. 5, pp. 385–392, 1998.

[93] W. Hoeffding, H. Robbins, *et al.*, "The central limit theorem for dependent random variables," *Duke math. J*, vol. 15, no. 3, pp. 773–780, 1948.

[94] C. Z. Mooney, R. D. Duval, and R. Duvall, *Bootstrapping: A nonparametric approach to statistical inference*. No. 94-95, Sage, 1993.

[95] P. A. Mykland, "Asymptotic expansions and bootstrapping distributions for dependent variables: a martingale approach," *The Annals of Statistics*, pp. 623–654, 1992.

[96] M. Zinkevich, M. Weimer, L. Li, and A. J. Smola, "Parallelized stochastic gradient descent," in *Advances in neural information processing systems*, pp. 2595–2603, 2010.

[97] D. M. Witten, J. H. Friedman, and N. Simon, "New insights and faster computations for the graphical lasso," *Journal of Computational and Graphical Statistics*, vol. 20, no. 4, pp. 892–900, 2011.

[98] J. O. Berger, *Statistical decision theory and Bayesian analysis*. Springer Science & Business Media, 2013.

[99] J. C. Platt, "12 fast training of support vector machines using sequential minimal optimization," *Advances in Kernel Methods*, pp. 185–208, 1999.

[100] D. Zhang, L. Wang, H. Xiong, and B. Guo, "4w1h in mobile crowd sensing," *IEEE Communications Magazine*, vol. 52, no. 8, pp. 42–48, 2014.

[101] S. Ji, Y. Zheng, and T. Li, "Urban sensing based on human mobility," in *Proceedings of the 2016 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, pp. 1040–1051, ACM, 2016.

[102] J. Yoon, B. Noble, and M. Liu, "Surface street traffic estimation," in *Proceedings of the 5th international conference on Mobile systems, applications and services*, pp. 220–232, ACM, 2007.

[103] R. Herring, A. Hofleitner, S. Amin, T. Nasr, A. Khalek, P. Abbeel, and A. Bayen, "Using mobile phones to forecast arterial traffic through statistical learning," in *89th Transportation Research Board Annual Meeting*, pp. 10–14, 2010.

[104] S. Hachem, A. Pathak, and V. Issarny, "Probabilistic registration for large-scale mobile participatory sensing," in *Pervasive Computing and Communications (PerCom), 2013 IEEE International Conference on*, pp. 132–140, IEEE, 2013.

[105] B. Pan, Y. Zheng, D. Wilkie, and C. Shahabi, "Crowd sensing of traffic anomalies based on human mobility and social media," in *Proceedings of the 21st ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pp. 344–353, ACM, 2013.

[106] K. Aberer, S. Sathe, D. Chakraborty, A. Martinoli, G. Barrenetxea, B. Faltings, and L. Thiele, "Opensense: open community driven sensing of environment," in *Proceedings of the ACM SIGSPATIAL International Workshop on GeoStreaming*, pp. 39–42, ACM, 2010.

[107] T. Liu, Y. Zheng, L. Liu, Y. Liu, and Y. Zhu, "Methods for sensing urban noises," *Tec. Rep. MSR-TR-2014-66*, 2014.

[108] H. Xiong, D. Zhang, L. Wang, and H. Chaouchi, "Emc 3: Energy-efficient data transfer in mobile crowdsensing under full coverage constraint," *IEEE Transactions on Mobile Computing*, vol. 14, no. 7, pp. 1355–1368, 2015.

[109] X. Sheng, J. Tang, and W. Zhang, "Energy-efficient collaborative sensing with mobile phones," in *INFOCOM, 2012 Proceedings IEEE*, pp. 1916–1924, IEEE, 2012.

[110] D. Zhang, H. Xiong, L. Wang, and G. Chen, "Crowdrecruiter: selecting participants for piggyback crowdsensing under probabilistic coverage constraint," in *Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, pp. 703–714, ACM, 2014.

[111] L. Wang, D. Zhang, A. Pathak, C. Chen, H. Xiong, D. Yang, and Y. Wang, "Ccs-ta: Quality-guaranteed online task allocation in compressive crowdsensing," in *Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, pp. 683–694, ACM, 2015.

[112] L. Wang, D. Zhang, Y. Wang, C. Chen, X. Han, and A. M'hamed, "Sparse mobile crowdsensing: challenges and opportunities," *IEEE Communications Magazine*, vol. 54, no. 7, pp. 161–167, 2016.

[113] L. Wang, D. Zhang, P. Wang, C. Animesh, X. Han, and W. Xiong, "Space-ta: Cost-effective task allocation exploiting intra- and inter-data correlations in sparse crowdsensing," *ACM Transactions on Intelligent Systems and Technology*, 2017.

[114] L. Wang, D. Yang, X. Han, T. Wang, D. Zhang, and X. Ma, "Location privacy-preserving task allocation for mobile crowdsensing with differential geo-obfuscation," in *Proceedings of the 26th International Conference on World Wide Web*, pp. 627–636, International World Wide Web Conferences Steering Committee, 2017.

[115] L. Wang, D. Zhang, D. Yang, B. Y. Lim, and X. Ma, "Differential location privacy for sparse mobile crowdsensing," in *Data Mining (ICDM), 2016 IEEE 16th International Conference on*, pp. 1257–1262, IEEE, 2016.

[116] T. Schneider, "Analysis of incomplete climate data: Estimation of mean values and covariance matrices and imputation of missing values," *Journal of Climate*, vol. 14, no. 5, pp. 853–871, 2001.

[117] D. Kondrashov and M. Ghil, "Spatio-temporal filling of missing points in geophysical data sets," *Nonlinear Processes in Geophysics*, vol. 13, no. 2, pp. 151–159, 2006.

[118] L. Kong, M. Xia, X.-Y. Liu, M.-Y. Wu, and X. Liu, "Data loss and reconstruction in sensor networks," in *INFOCOM, 2013 Proceedings IEEE*, pp. 1654–1662, IEEE, 2013.

[119] Y. Zhang, M. Roughan, W. Willinger, and L. Qiu, "Spatio-temporal compressive sensing and internet traffic matrices," in *ACM SIGCOMM Computer Communication Review*, vol. 39, pp. 267–278, ACM, 2009.

[120] D. D. Lee and H. S. Seung, "Algorithms for non-negative matrix factorization," in *Advances in neural information processing systems*, pp. 556–562, 2001.

[121] Y. Mao and L. K. Saul, "Modeling distances in large-scale networks by matrix factorization," in *Proceedings of the 4th ACM SIGCOMM conference on Internet measurement*, pp. 278–287, ACM, 2004.

[122] F. Ingelrest, G. Barrenetxea, G. Schaefer, M. Vetterli, O. Couach, and M. Parlange, "Sensorscope: Application-specific sensor network for environmental monitoring," *ACM Transactions on Sensor Networks (TOSN)*, vol. 6, no. 2, p. 17, 2010.

[123] Y. Zheng, F. Liu, and H.-P. Hsieh, "U-air: When urban air quality inference meets big data," in *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 1436–1444, ACM, 2013.

[124] H. Gao, J.-F. Cai, Z. Shen, and H. Zhao, "Robust principal component analysis-based four-dimensional computed tomography," *Physics in medicine and biology*, vol. 56, no. 11, p. 3181, 2011.

[125] M. A. Fischler and R. C. Bolles, "Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography," *Communications of the ACM*, vol. 24, no. 6, pp. 381–395, 1981.

[126] S. Isam, I. Kanaras, and I. Darwazeh, "A truncated svd approach for fixed complexity spectrally efficient fdm receivers," in *Wireless Communications and Networking Conference (WCNC), 2011 IEEE*, pp. 1584–1589, IEEE, 2011.

[127] D. R. Jones, M. Schonlau, and W. J. Welch, "Efficient Global Optimization of Expensive Black-Box Functions," *Journal of Global Optimization*, vol. 13, no. 4, pp. 455–492, 1998.

[128] F. Hutter, H. Hoos, and K. Leyton-Brown, "An evaluation of sequential model-based optimization for expensive blackbox functions," in *Proceedings of the 15th annual conference companion on Genetic and evolutionary computation*, pp. 1209–1216, 2013.

[129] H.-P. Kriegel, E. Schubert, and A. Zimek, "The (black) art of runtime evaluation: Are we comparing algorithms or implementations?," *Knowledge and Information Systems*, vol. 52, no. 2, pp. 341–378, 2017.

[130] S. Nijssen and J. Kok, "Frequent subgraph miners: runtimes don\'t say everything," in *Proceedings of the Workshop on Mining and Learning with Graphs*, pp. 173–180, 2006.

[131] S. Buschjager, K.-H. Chen, J.-J. Chen, and K. Morik, "Realization of random forest for real-time evaluation through tree framing," in *IEEE International Conference on Data Mining (ICDM)*, 2018.

[132] Y. A. LeCun, L. Bottou, G. B. Orr, and K.-R. Müller, "Efficient backprop," in *Neural networks: Tricks of the trade*, pp. 9–48, Springer, 2012.

[133] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *Journal of Machine Learning Research*, 2012.

[134] J. Berk, V. Nguyen, S. Gupta, S. Rana, and S. Venkatesh, "Exploration enhanced expected improvement for bayesian optimization," in *Machine Learning and Knowledge Discovery in Databases - ECML/PKDD Proceedings*, vol. 11052 of *Lecture Notes in Computer Science*, pp. 621–637, Springer, 2018.

[135] J. Snoek, H. Larochelle, and R. P. Adams, "Practical bayesian optimization of machine learning algorithms," in *Advances in neural information processing systems*, pp. 2951–2959, 2012.

[136] J. Bergstra, D. Yamins, and D. Cox, "Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures," in *Proceedings of the 30th International Conference on Machine Learning*, vol. 28, (Atlanta, Georgia, USA), pp. 115–123, PMLR, 17–19 Jun 2013.

[137] F. Hutter, H. H. Hoos, and K. Leyton-Brown, "Sequential model-based optimization for general algorithm configuration," in *International conference on learning and intelligent optimization*, Springer, 2011.

[138] D. Ginsbourger, R. Le Riche, and L. Carraro, "Kriging is well-suited to parallelize optimization," in *Computational Intelligence in Expensive Optimization Problems*, pp. 131–162, Springer, 2010.

[139] F. Hutter, H. H. Hoos, and K. Leyton-Brown, "Parallel Algorithm Configuration," in *Learning and Intelligent Optimization* (Y. Hamadi and M. Schoenauer, eds.), no. 7219 in Lecture Notes in Computer Science, pp. 55–70, Springer Berlin Heidelberg, 2012.

[140] M. A. R. Coy, F. Rehbach, A. E. Eiben, and T. Bartz-Beielstein, "Parallelized bayesian optimization for problems with expensive evaluation functions," in *GECCO '20: Genetic and Evolutionary Computation Conference, Companion Volume, Cancún, Mexico, July 8-12, 2020* (C. A. C. Coello, ed.), pp. 231–232, ACM, 2020.

[141] J. Janusevskis, R. Le Riche, D. Ginsbourger, and R. Girdziusas, "Expected improvements for the asynchronous parallel global optimization of expensive functions: Potentials and challenges," in *International Conference on Learning and Intelligent Optimization*, Springer, 2012.

[142] J. Richter, H. Kotthaus, B. Bischl, P. Marwedel, J. Rahnenführer, and M. Lang, "Faster Model-Based Optimization Through Resource-Aware Scheduling Strategies," in *Learning and Intelligent Optimization*, pp. 267–273, Springer International Publishing, May 2016.

[143] J. Konečnỳ, H. B. McMahan, F. X. Yu, P. Richtárik, A. T. Suresh, and D. Bacon, "Federated learning: Strategies for improving communication efficiency," *arXiv preprint arXiv:1610.05492*, 2016.

[144] L. Li, H. Xiong, J. Wang, C.-Z. Xu, and Z. Guo, "Smartpc: Hierarchical pace control in real-time federated learning system,"

[145] O. K. Baek, "Data-centric distributed computing." US Patent 8,060,464.

[146] G. Claeskens, N. L. Hjort, *et al.*, "Model selection and model averaging," *Cambridge Books*, 2008.

[147] J. Shi, J. Bian, and J. Richter, "Model-based Optimization on Distributed Embedded System." `https://github.com/Strange369/MODES-public`, 2020.

[148] Y. Gu, H. Do, Y. Ou, and W. Sheng, "Human gesture recognition through a kinect sensor," in *2012 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, pp. 1379–1384, IEEE, 2012.

[149] "ODROID-N2." https://www.hardkernel.com/shop/odroid-n2-with-4gbyte-ram/. visit on 25.10.2019.

[150] B. Bischl, J. Richter, J. Bossek, D. Horn, J. Thomas, and M. Lang, "mlrMBO: A Modular Framework for Model-Based Optimization of Expensive Black-Box Functions," *arXiv:1703.03373 [stat]*, 2018.

[151] Y. LeCun, C. Cortes, and C. J. Burges, "The mnist database of handwritten digits, 1998," *URL http://yann. lecun. com/exdb/mnist*, 1998.

[152] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms," *arXiv preprint arXiv:1708.07747*, 2017.

[153] A. Vergara, S. Vembu, T. Ayhan, M. A. Ryan, M. L. Homer, and R. Huerta, "Chemical gas sensor drift compensation using classifier ensembles," *Sensors and Actuators B: Chemical*, vol. 166, pp. 320–329, 2012.

[154] J. A. Blackard and D. J. Dean, "Comparative accuracies of artificial neural networks and discriminant analysis in predicting forest cover types from cartographic variables," *Computers and electronics in agriculture*, vol. 24, no. 3, pp. 131–151, 1999.

[155] D. Anguita, A. Ghio, and et al., "A public domain dataset for human activity recognition using smartphones.," in *Esann*, 2013.

[156] A. Liaw, M. Wiener, *et al.*, "Classification and regression by randomforest," *R news*, vol. 2, no. 3, pp. 18–22, 2002.

[157] Y.-B. Chan and P. Hall, "Scale adjustments for classifiers in high-dimensional, low sample size settings," *Biometrika*, vol. 96, no. 2, pp. 469–478, 2009.

[158] J. Gama, R. Rocha, and P. Medas, "Accurate decision trees for mining high-speed data streams," in *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 523–528, 2003.

[159] J. Levinson, J. Askeland, J. Becker, J. Dolson, D. Held, S. Kammel, J. Z. Kolter, D. Langer, O. Pink, V. Pratt, *et al.*, "Towards fully autonomous driving: Systems and algorithms," in *2011 IEEE Intelligent Vehicles Symposium (IV)*, pp. 163–168, IEEE, 2011.

[160] G. Loosli, S. Canu, and L. Bottou, "Training invariant support vector machines using selective sampling," in *Large Scale Kernel Machines* (L. Bottou, O. Chapelle, D. DeCoste, and J. Weston, eds.), pp. 301–320, Cambridge, MA.: MIT Press, 2007.

[161] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng, "Reading digits in natural images with unsupervised feature learning," 2011.

[162] I. J. Goodfellow, Y. Bulatov, J. Ibarz, S. Arnoud, and V. Shet, "Multi-digit number recognition from street view imagery using deep convolutional neural networks," *arXiv preprint arXiv:1312.6082*, 2013.

[163] D. Bogdanov, M. Niitsoo, T. Toft, and J. Willemson, "High-performance secure multi-party computation for data mining applications," *International Journal of Information Security*, vol. 11, no. 6, pp. 403–418, 2012.