Distributed Bayesian Matrix Decomposition for Big Data Mining and Clustering

Chihao Zhang, Yang Yang, Wei Zhou and Shihua Zhang*

Abstract—Matrix decomposition is one of the fundamental tools to discover knowledge from big data generated by modern applications. However, it is still inefficient or infeasible to process very big data using such a method in a single machine. Moreover, big data are often distributedly collected and stored on different machines. Thus, such data generally bear strong heterogeneous noise. It is essential and useful to develop distributed matrix decomposition for big data analytics. Such a method should scale up well, model the heterogeneous noise, and address the communication issue in a distributed system. To this end, we propose a distributed Bayesian matrix decomposition model (DBMD) for big data mining and clustering. Specifically, we adopt three strategies to implement the distributed computing including 1) the accelerated gradient descent, 2) the alternating direction method of multipliers (ADMM), and 3) the statistical inference. We investigate the theoretical convergence behaviors of these algorithms. To address the heterogeneity of the noise, we propose an optimal plug-in weighted average that reduces the variance of the estimation. Synthetic experiments validate our theoretical results, and real-world experiments show that our algorithms scale up well to big data and achieves superior or competing performance compared to two typical distributed methods including Scalable-NMF and scalable k-means++.

Index Terms—Distributed algorithm, Bayesian matrix decomposition, clustering, big data, data mining

1 Introduction

Big data emerge from various disciplines of sciences with the development of technologies. For example, different types of satellite platforms have generated vast amounts of remote sensing data, and high-throughput sequencing technologies provide large-scale transcriptomic data. Such data are usually organized in the matrix form and are generally redundant and noisy. Therefore, matrix decomposition becomes one of the fundamental tools to explore the data. The history of matrix decomposition dates back to more than one century ago when Pearson invented principal component analysis (PCA) [1]. Since then, matrix decomposition has been extensively studied due to its effectiveness and is still an active topic today. The conceptual idea of matrix decomposition is that the primitive big and noisy data matrix can be approximated by the product of two or more compact low-rank matrices. Mathematically, given the observed data matrix $\boldsymbol{X} \in R^{m\times n}$, matrix decomposition methods consider

$$\min_{\boldsymbol{W},\boldsymbol{H}} \mathrm{D}(\boldsymbol{X}||\boldsymbol{W}\boldsymbol{H}), \tag{1}$$

where $\boldsymbol{W} \in R^{m\times r}$, $\boldsymbol{H} \in R^{r\times n}$, and D is a divergence function. Typically, $r \ll \min(m,n)$, and thus $\boldsymbol{W}$ and $\boldsymbol{H}$ are the compact basis and coefficient matrices, respectively.

Matrix decomposition methods are flexible by imposing different restrictions or regularizers on $\boldsymbol{W}$, $\boldsymbol{H}$, and choosing different divergence functions. From the view of minimization of the reconstruction error [2], PCA is merely the matrix decomposition method with the Frobenius norm as the divergence function $\|\boldsymbol{X}-\boldsymbol{W}\boldsymbol{H}\|_F^2$. It is known that PCA is sensitive to gross errors. Shen *et al*. [3] developed a robust PCA by replacing the Frobenius norm with L_1-norm. Min *et al* [4]. imposed group-sparse penalties on SVD, accounting for prior group effects group. The famous clustering algorithm k-means can also be understood as a matrix decomposition method. If we choose the Frobenius norm as the divergence function and restrict $\boldsymbol{H}$ such that its each column indicates the cluster membership by a single one, then Eq. (1) is exactly the k-means method. Moreover, it is common that the primitive data are naturally nonnegative, such as images and text corpus. Hence, the nonnegative matrix factorization (NMF) has been explored [5], [6]. NMF restricts both $\boldsymbol{W}$ and $\boldsymbol{H}$ to be nonnegative. The negativeness enhances the interpretability of the model and leads to part-based feature extraction and sparseness [6]. Therefore, NMF gets popular and has many variants developed, including sparse NMF [7], [8] and graph regularized NMF [9], [10], [11].

However, the standard matrix decomposition methods are prone to overfitting the observed matrix that is noisy and has missing values. The imposed regularizers can reduce the risk of overfitting, but their parameters require careful tuning. Bayesian matrix decomposition addresses this problem by incorporating priors into the model parameters. Tipping and Bishop [12] first proposed the probabilistic PCA (PPCA) and showed that PPCA is a latent variable model with independent and identical distribution (IID) Gaussian noise. The probabilistic treatment permits various extensions of PCA [13], [14], [15], [16]. One of the appealing characteristics of the Bayesian approach is that it gives the flexibility to design different matrix decomposition methods by choosing appropriate distributions for priors and noise. Multi-view data collected from different sources are now ubiquitous, and bear distinctly heterogeneous noise [17]. Such data from different sources are complementary, and thus computational methods for integrative analysis are urgently needed. Some matrix decomposition methods for multi-view data integration have been explored [18], [19],

- *Chihao Zhang and Shihua Zhang are with the NCMIS, CEMS, RCSDS, Academy of Mathematics and Systems Science, Chinese Academy of Sciences, Beijing 100190, School of Mathematical Sciences, University of Chinese Academy of Sciences, Beijing 100049, and Center for Excellence in Animal Evolution and Genetics, Chinese Academy of Sciences, Kunming 650223, China.*
- *Yang Yang and Wei Zhou are with the School of Software, Yunnan University, Kunming 650504, China.*
**To whom correspondence should be addressed. Email: zsh@amss.ac.cn.*

[20], [21], [22]. Typically, most of those algorithms assume that the data matrices share a common basis matrix (or coefficient matrix), enabling the methods to perform an integrative analysis. However, few of the existing methods consider the heterogeneity of the noise, and thus the data view of high noise may affect the analysis of that of low noise. To address this problem, Zhang and Zhang [23] employed the Bayesian approach and proposed Bayesian joint matrix decomposition (BJMD). Their experiments showed that considering the heterogeneous noise leads to superior performance in clustering. But theoretical analysis is still lacking.

The matrix decomposition methods mentioned above have little relevance to the underlying computational architecture. They assumed that the program is running on a single machine, and an arbitrary number of data points are accessible instantaneously. However, the huge size of data often makes it impossible to handle all of them on a single machine. Many applications collect data distributedly from different sources (e.g., labs, hospitals). The communication between them is expensive due to the limited bandwidth, and direct data sharing also raise privacy concern. Moreover, data collected from different sources often bear strong heterogeneous noise. Therefore, developing efficient matrix decomposition methods in a distributed system is essential. The commonly used computation architecture is that the overall data $\boldsymbol{X} \in R^{m\times n}$ is distributed onto C nodes that are connected to a central processor. The desired methods should scale up well to distributed big data, communicate efficiently, and adequately tackle the heterogeneity of the noise. Researchers have developed many distributed matrix decomposition methods including distributed k-means [24], [25], distributed BPMF [26], [27], [28], distributed NMF [29], [30], [31], and so on. Developing efficient distributed algorithms for matrix decomposition methods should account for the partition strategy of the distributed data and then adopt an appropriate optimization strategy. For example, when the number of instances n is vast, and the number of features is small or moderate, i.e., the transposition of the data matrix $\boldsymbol{X}$ is tall-and-skinny, it is usually convenient to split $\boldsymbol{X}$ over columns. We should then consider which optimization strategy is suitable for current partitioned data. However, few studies explored different optimization strategies and elaborated their difference. Even more serious is that few methods tackle the heterogeneity of noise among the distributed data.

To this end, we propose a distributed Bayesian matrix decomposition model (DBMD) that extends the BJMD for big data clustering and mining. We limit our scope to the data matrix whose transposition is tall-and-skinny, distribute it by columns onto nodes, and then focus on the optimization strategies for solving DBMD. Specifically, we adopt three strategies to implement the distributed computing, including 1) the accelerated gradient descent (AGD), 2) the alternating direction method of multipliers (ADMM), and 3) the communication-efficient accurate statistical estimation (CEASE), and then investigate their convergence behavior in the distributed setting. To tackle the heterogeneous noise, we propose an optimal plug-in weighted average that minimizes the variance of the estimation. Extensive experiments verify our theoretical results, and the real-world experiments show the scalability and the effectiveness of our methods.

The contributions of this paper are as follows: 1) we propose a scalable distributed Bayesian matrix decomposition model for big data matrix whose transposition is tall-and-skinny. 2) we adopt three optimization strategies and elaborate their differences in both empirical and theoretical perspectives. 3) We propose a flexible weighted average to tackle the heterogeneous noise and provide the theoretical result that is lacked in [23].

2 Preliminaries and Notations

Throughout this paper, we use three standard mathematical notations including lightface lowercase (x), boldface lowercase ($\boldsymbol{x}$), boldface uppercase ($\boldsymbol{X}$) characters to represent scalars, vectors and matrices, respectively. $\boldsymbol{x}_{i\cdot}$, $\boldsymbol{x}_{\cdot j}$, x_{ij} represent the i-th row, the j-th column, and the entry of the i-th row and the j-th of the matrix $\boldsymbol{X}$, respectively. Given a sequence of matrices $\{\boldsymbol{X}_c\}_{c=1}^{C}$, we use the following notations: $(\boldsymbol{x}_{i\cdot})_c$, $(\boldsymbol{x}_{\cdot j})_c$, $(x_{ij})_c$ to denote the corresponding row, column and entry in the c-th matrix $\boldsymbol{X}_c$. $\bar{\boldsymbol{X}}_c = \sum_{c=1}^{C} \boldsymbol{X}_c/C$ is the average of the matrix sequence $\{\boldsymbol{X}_c\}_{c=1}^{C}$. $c \in [C]$ indicates that $c \in \{1, 2, \ldots, C\}$.

Suppose a matrix variate function $f : R^{m\times n} \rightarrow R$ is convex and differentiable, and let ∇f denote the gradient of f. We say that f is strongly convex with parameter $\mu_f > 0$, if for all $\boldsymbol{X}$, $\boldsymbol{Y} \in R^{m\times n}$

$$f(\boldsymbol{X}) \geq f(\boldsymbol{Y}) + \langle \boldsymbol{Y}, \boldsymbol{X} - \boldsymbol{Y} \rangle + \frac{\mu_f}{2} \left\| \boldsymbol{X} - \boldsymbol{Y} \right\|_F^2 . \quad (2)$$

where $\langle \boldsymbol{Y}, \boldsymbol{X} - \boldsymbol{Y} \rangle = \mathrm{tr}(\boldsymbol{Y}^T(\boldsymbol{X} - \boldsymbol{Y}))$ indicates the Frobenius inner product of matrices. When ∇f is Lipschitz continuous with parameter L_f, we then have

$$\left\| \nabla f(\boldsymbol{X}) - \nabla f(\boldsymbol{Y}) \right\|_F \leq L_f \left\| f(\boldsymbol{X}) - f(\boldsymbol{X}) \right\|_F . \quad (3)$$

We denote the ratio of L_f to μ_f as $\kappa_f = L_f/\mu_f$ if it exists.

3 Related Work

3.1 Bayesian Matrix Decomposition

Due to the flexibility and the effectiveness of Bayesian matrix decomposition, there have been a number of studies since PPCA was developed in 1999 [12]. At the same year, Bishop proposed the Bayesian PCA [13], which can automatically determine the number of retained principal components. To account for the complex type of noise in the real-world, PCA with exponential family noise [14], [16] and its Bayesian variants have also been proposed [15].

Analogous to Bayesian PCA, the Bayesian treatments for k-means [32], NMF [33], [34] have also been explored. In collaborative filtering, Salakhutdinov and Mnih [35] proposed the Bayesian probabilistic matrix factorization (BPMF). Saddiki *et al.* [36] proposed a mixed-membership model named GLAD that employs priors of Laplace and Dirichlet distribution on $\boldsymbol{W}$ and columns of $\boldsymbol{H}$, respectively. Bayesian k-means has also been proposed and it can automatically determine the number of clusters [32]. Salakhutdinov and Mnih proposed the Bayesian probabilistic matrix factorization (BPMF) for predicting user preference for movies [35], which places the Gaussian priors over both the basis and coefficient matrices.

Very recently, Zhang and Zhang extended GLAD to BJMD to tackle the heterogeneous noise of multi-view data [23]. Let's denote the multi-view data as $\boldsymbol{X}_c$, $c \in [C]$, where $\boldsymbol{X}_c \in R^{m\times n_c}$ indicates the data collected is from the c-th source. BJMD assumes that the observed data matrices $\boldsymbol{X}_c$ share the same basis matrix $\boldsymbol{W}$ and different coefficient matrices $\boldsymbol{H}_c$ and use the Gaussian distributions of different variances to model the heterogeneity of the noise. Similar to GLAD, BJMD puts the Laplace prior onto the basis matrix $\boldsymbol{W}$ to pursue sparsity and the Dirichlet prior onto the columns of $\boldsymbol{H}_c$ to enhance the interpretability. Moreover, two efficient algorithms via variational inference and maximum a posterior respectively have been developed for solving it. They are much faster than GLAD, and thus are applicable to relatively large data. But BJMD is still a single machine methodology.

3.2 Distributed Matrix Decomposition

It is known that a proper initialization for the k-means algorithm is crucial for obtaining a good final solution. But the typical single machine initialization algorithms such as k-means++ [37] are sequential, which limits its applicability to big data. Generally, scaling the k-means algorithm to distributed data is relatively easy due to its iterative nature. Distributed k-means algorithms often split data by samples. The distributed versions of k-means often focus on reducing the number of passes needed to obtain a good initialization by sampling, e.g., DKEM [24] and scalable k-means++ [25].

Recently, Yu *et al.* proposed a distributed BPMF by splitting the data by samples and employed a stochastic alternating direction method of multipliers (ADMM) to solve it [26]. But it is common in the filtering collaborative that the user-item data $\boldsymbol{X} \in R^{m\times n}$ are very spare and of large m and n. Splitting $\boldsymbol{X}$ by rows is only efficient for the tall-and-skinny matrix due to the communication load. To reduce the communication load, some studies split the data matrix $\boldsymbol{X}$ over both columns and rows, and then store the blocks of $\boldsymbol{X}$ distributedly on nodes [27], [28]. Then they employed distributed Monte Carlo Markov Chain methods (MCMC) for inference. There also exist distributed NMF variants that split $\boldsymbol{X}$ into blocks [29], [31]. Moreover, Benson *et al.* proposed an approximated and scalable NMF algorithm for tall-and-skinny matrices. Inspired by Donoho and Stodden [38], they assume that the matrix $\boldsymbol{X}$ is nearly separable, i.e.

$$\boldsymbol{X} = \boldsymbol{X}(:,\mathcal{K})\boldsymbol{H} + \boldsymbol{E}, \tag{4}$$

where $\mathcal{K}$ is an index set with size r, $\boldsymbol{X}(:,\mathcal{K})$ is the submatrix of $\boldsymbol{X}$ restricted to the columns indexed by $\mathcal{K}$, and E is a noise matrix. This algorithm only requires one round iteration [30].

4 DBMD

4.1 Model Construction

Suppose that the data matrix $\boldsymbol{X} \in R^{m\times n}$, where m is the number of features, n is the number of samples and $n \gg m$. Since n is very large, $\boldsymbol{X}$ cannot be handled by a single machine. Therefore, $\boldsymbol{X}$ is split by columns and distributedly stored on C machines: $\{\boldsymbol{X}_c\}_{c=1}^C$, where $\boldsymbol{X}_c \in R^{m\times n_c}$ and $\sum_{c=1}^C n_c = n$. Inspired by a recent study [23], we assume that $\{\boldsymbol{X}_c\}_{c=1}^C$ share the same basis matrix $\boldsymbol{W}$ and are generated as follows

$$\boldsymbol{X}_c = \boldsymbol{W}\boldsymbol{H}_c + \boldsymbol{E}_c, \tag{5}$$

where $\boldsymbol{W} \in R^{m\times r}$, $\boldsymbol{H}_c \in R^{r\times n_c}$ and $\boldsymbol{E}_c$ is the IID Gaussian noise, $(e_{ij})_c \sim N(0,\sigma_c^2)$. We further put a zero-mean Laplace prior on $\boldsymbol{W}$ to enforce its sparsity

$$w_{ik} \sim p(w_{ik}|0,\lambda_0), \tag{6}$$

where the density function

$$p(y|\mu,\lambda) = \frac{1}{2\lambda}\exp\left(-\frac{|y-\mu|}{\lambda}\right). \tag{7}$$

We put a Dirichlet prior $\text{Dir}(\boldsymbol{\alpha}_0)$ on each column $(h_{\cdot j})_c$ of $\boldsymbol{H}_c$

$$(\boldsymbol{h}_{\cdot \boldsymbol{j}})_c \sim p((\boldsymbol{h}_{\cdot \boldsymbol{j}})_c|\boldsymbol{\alpha_0}), \tag{8}$$

where $\boldsymbol{\alpha}_0 > 0$ is a r-dimensional vector and the density function of $\text{Dir}(\boldsymbol{\alpha}_0)$ is

$$p(\boldsymbol{y}|\boldsymbol{\alpha}_0) = \frac{\Gamma(\sum_{i=1}^r \alpha_i)}{\prod_{i=1}^r \Gamma(\alpha_{0i})}\prod_{i=1}^r y_i^{\alpha_i - 1}. \tag{9}$$

The support of the Dirichlet is $y_1,\ldots,y_r$, where $y_i \in (0,1)$ and $\sum_{i=1}^r y_i = 1$, which is a unit simplex. Note that the Dirichlet prior restricts the $\boldsymbol{H}_c$ to be non-negative and the columns sum of $\boldsymbol{H}_c$ all equal one. Therefore, $(h_{\cdot j})_c$ can be interpreted as a vector indicates the membership of clusters. The Gaussian noise ϵ_c models the noise level in the corresponding $\boldsymbol{X}_c$

$$\epsilon_c \sim p(\epsilon_c|0,\sigma_c^2), \tag{10}$$

where

$$p(y|\mu,\sigma_c^2) = \frac{1}{\sqrt{2\pi}\sigma_c}\exp\left(-\frac{(y-\mu)^2}{2\sigma_c^2}\right). \tag{11}$$

We are interested in the posterior of $\boldsymbol{W}$ and $\boldsymbol{H}_c$. By the Bayes' theorem, it is proportional to the complete likelihood, which can be written as

$$\begin{aligned} &p(\boldsymbol{W},\boldsymbol{H}_1,\ldots,\boldsymbol{H}_C,\sigma_1^2,\ldots,\sigma_C^2,\boldsymbol{X}_1,\ldots,\boldsymbol{X}_C;\lambda_0,\boldsymbol{\alpha_0}) \\ =&p(\boldsymbol{W};\lambda)\prod_{c=1}^C p(\boldsymbol{X}_c|\boldsymbol{W},\boldsymbol{H}_c,\sigma_c^2)p(\boldsymbol{H}_c;\boldsymbol{\alpha_0}). \end{aligned} \tag{12}$$

Maximum a posterior can then be formulated as minimizing the negative log likelihood

$$\begin{aligned} \min_{\boldsymbol{W},\boldsymbol{H}_c} &\sum_{c=1}^C \frac{1}{2\sigma_c^2}||\boldsymbol{X}_c - \boldsymbol{W}\boldsymbol{H}_c||_F^2 + \frac{1}{\lambda_0}||\boldsymbol{W}||_1 \\ &- \sum_{c=1}^C\sum_{k,j}(\alpha_{0k}-1)\ln(h_{kj})_c + \sum_{c=1}^C mn_c\ln\sigma_c \\ \text{s.t.} &\sum_{k=1}^r (h_{kj})_c = 1, (h_{kj})_c > 0, \end{aligned} \tag{13}$$

where the first term is essentially the weighted sum of the goodness of the approximations across the C nodes. Intuitively, the weight $1/2\sigma_c^2$ gives higher importance to the clean data (σ_c^2 is small) and lower importance to the noisy data. The second term is the L_1-norm regularizer, which enforces the sparsity of the basis matrix $\boldsymbol{W}$. The third term is due to the Dirichlet prior and regularizes the coefficient matrices $\{\boldsymbol{H}_c\}_{c=1}^C$. Specifically, the third term is minimized when $(h_{kj})_c = \alpha_{0k}/\sum_{k=1}^r \alpha_{0k}$, which is the expectation of the Dirichlet prior. Therefore, the third term enforces $(\boldsymbol{h}_{\cdot \boldsymbol{j}})_c$ to the prior and reduces the risk of overfitting.

For the ease of presentation, we denote $\boldsymbol{\alpha} = \boldsymbol{\alpha_0} - \boldsymbol{1}$, $\lambda = 1/\lambda_0$ and set $\sigma_c = 1$ (we will discuss the situation where σ_c is inferred later). Let's rewrite the optimization problem in Eq. (13) as follows

$$\begin{aligned}\min_{\boldsymbol{W},\boldsymbol{H}_c} & \sum_{c=1}^{C}\frac{1}{2}||\boldsymbol{X}_c-\boldsymbol{W}\boldsymbol{H}_c||_F^2+\lambda||\boldsymbol{W}||_1 \\ & -\sum_{c=1}^{C}\sum_{k,j}\alpha_k\ln(h_{kj})_c \qquad (14)\\ \mathrm{s.t.} & \sum_{k=1}^{r}(h_{kj})_c=1,(h_{kj})_c>0.\end{aligned}$$

Note that Eq. (14) is bi-convex to $\boldsymbol{W}$ and $\boldsymbol{H}_c$. A common approach is to update $\boldsymbol{W}$ and $\boldsymbol{H}_c$ alternatively.

4.2 Accelerated Gradient Decent Optimization

Recall that $\{\boldsymbol{X}_c\}_{c=1}^{C}$ is distributedly stored on C nodes. We store the corresponding $\boldsymbol{H}_c$ on the c-th nodes and store $\boldsymbol{W}$ on the central processor. Then we can easily adopt the maximum a posterior algorithm in [23] to solve Eq. (14) by updating $\boldsymbol{W}$ and $\{\boldsymbol{H}_c\}_{c=1}^{C}$ alternatively. Specifically, we update $\boldsymbol{W}$ with other parameters fixed

$$\min_{\boldsymbol{W}}\frac{1}{2}||\boldsymbol{X}_c-\boldsymbol{W}\boldsymbol{H}_c||_F^2+\lambda||\boldsymbol{W}||_1. \qquad (15)$$

Let's denote $f(\boldsymbol{W})=\sum_{c=1}^{C}f_c(\boldsymbol{W})$, $f_c(\boldsymbol{W})=\frac{1}{2}\|\boldsymbol{X}_c-\boldsymbol{W}\boldsymbol{H}_c\|_F^2$ and $g(\boldsymbol{W})=\lambda\|\boldsymbol{W}\|_1$. The objective function Eq. (15) consists of the non-smooth L_1-norm regularizer $g(\boldsymbol{W})$ and the quadratic loss terms. Therefore, it can be efficiently solved by the fast iterative shrinkage-thresholding algorithm (FISTA) [39]. FISTA is an accelerated gradient decent (AGD) algorithm and enjoys a quadratic convergence rate. Specifically, we construct two sequences $\{\boldsymbol{Y}^k\}$ and $\{\boldsymbol{W}^k\}$, and alternatively update them

$$\boldsymbol{W}^k=\arg\min_{\boldsymbol{W}} g(\boldsymbol{W})+\frac{L}{2}\left\|\boldsymbol{W}-\left(\boldsymbol{Y}^k-\frac{1}{L}\nabla\tilde{f}_c(\boldsymbol{Y}^k)\right)\right\|_F^2 \qquad (16)$$

and

$$\boldsymbol{Y}^{k+1}=\boldsymbol{W}^k+\frac{\nu_k-1}{\nu_{k+1}}(\boldsymbol{W}^k-\boldsymbol{W}^{k-1}), \qquad (17)$$

where $L_f=\left\|\sum_{c=1}^{C}H_cH_c^T\right\|_2>0$ is the Lipschitz constant of $\sum_c\nabla f_c(\boldsymbol{W})$. $\boldsymbol{W}^k$ contains the approximate solution by minimizing the proximal function. Eq. (16) has the closed-form solution

$$\boldsymbol{W}^k=\mathcal{S}_{\lambda/L}\left(\boldsymbol{Y}^k-\frac{1}{L}\sum_c\nabla\boldsymbol{f}_c(\boldsymbol{Y}^k)\right).$$

where

$$\mathcal{S}_{\lambda/L}(\boldsymbol{X})=\mathrm{sign}(\boldsymbol{X})\circ\max(|\boldsymbol{X}|-\lambda/L,0),$$

is the soft thresholding operator, and $\circ$ is the Hadamard product. $\boldsymbol{Y}^{k+1}$ stores the search point constructed by linearly combining the latest two approximate solutions, i.e., $\boldsymbol{W}^{k-1}$ and $\boldsymbol{W}^k$. The combination coefficient ν_{t+1} was carefully designed in [39] as follows

$$\nu_{k+1}=\frac{1+\sqrt{4\nu_k^2+1}}{2}. \qquad (18)$$

At each iteration, we broadcast the current $\boldsymbol{Y}^k$ to all nodes and compute the gradients. Then we collect the computed gradients to the central processor and update $\boldsymbol{W}^k$. We iteratively update $\boldsymbol{W}^k$ and $\boldsymbol{Y}^k$ until it converges.

Algorithm 1 Updating $\boldsymbol{W}$ with AGD

```
Input: Initial $\boldsymbol{W}^0$, $\boldsymbol{Y}^0=\boldsymbol{W}^0$, $k=0$, $\nu_0=1$
1: repeat
2:     Each node machine computes $\nabla f_c(\boldsymbol{Y}^{\boldsymbol{k}})$ and sends to the central processor
3:     The central processor computes
           $\boldsymbol{W}^k=\mathcal{S}_{\lambda/L}\left(\boldsymbol{Y}^k-\frac{1}{L}\sum_c\nabla\boldsymbol{f}_c(\boldsymbol{Y}^k)\right).$
4:     The central processor computes $\nu_{k+1}=\frac{1+\sqrt{4\nu_k^2+1}}{2}$
5:     The central processor computes
           $\boldsymbol{Y}^{k+1}=\boldsymbol{W}_c^k+\frac{\nu_k-1}{\nu_{k+1}}(\boldsymbol{W}_c^k-\boldsymbol{W}_c^{k-1})$
       and broadcasts to the nodes
6:     $k\leftarrow k+1$
7: until Convergence.
```

Updating $\{\boldsymbol{H}_c\}_{c=1}^{C}$ is straightforward. We broadcast $\boldsymbol{W}$ from the central processor to the nodes, and then we can adopt the same algorithm in [23] to solve the following problem

$$\begin{aligned}\min_{\boldsymbol{H}_c} & \frac{1}{2}||\boldsymbol{X}_c-\boldsymbol{W}\boldsymbol{H}_c||_F^2-\sum_{c=1}^{C}\sum_{k,j}\alpha_k\ln(h_{kj})_c \qquad (19)\\ \mathrm{s.t.} & \sum_{k=1}^{r}(h_{kj})_c=1,(h_{kj})_c>0,\end{aligned}$$

with respect to $\boldsymbol{H}_c$ in parallel.

Note that solving Eq. (19) requires no data communication. Therefore, it has no difference from the single machine algorithm. However, one concern arises when we update $\boldsymbol{W}$ in a distributed system, broadcasting $\boldsymbol{W}$ to the nodes and collecting the gradients from the nodes can be very time-consuming. The speed of inter-node communication can be much slower than that of intra-node computation in the distributed system [40]. Therefore, the data communication is often the bottleneck of the distributed algorithm and updating $\boldsymbol{W}$ requires carefully consideration. The algorithm of updating $\boldsymbol{W}$ with AGD is summarized in Fig. 1A and Algorithm 1. In the following, we discuss how to design two efficient algorithms from optimization and statistical perspectives, respectively.

4.3 Efficient Distributed Optimization

We adopt the ADMM to implement the algorithm of updating $\boldsymbol{W}$. Note that $\boldsymbol{W}$ in Eq. (15) is a global variable. We formulate the following optimization problem

$$\begin{aligned}\min_{\boldsymbol{W},\boldsymbol{W}_c} & \sum_{c=1}^{C}\frac{1}{2}||\boldsymbol{X}_c-\boldsymbol{W}_c\boldsymbol{H}_c||_F^2+\lambda||\boldsymbol{W}||_1 \qquad (20)\\ \mathrm{s.t.} & \boldsymbol{W}_c=\boldsymbol{W},\end{aligned}$$

where $\boldsymbol{W}$ is the consensus variable. One can easily verify that Eq. (20) is equivalent to Eq. (15). Let's write down its augmented

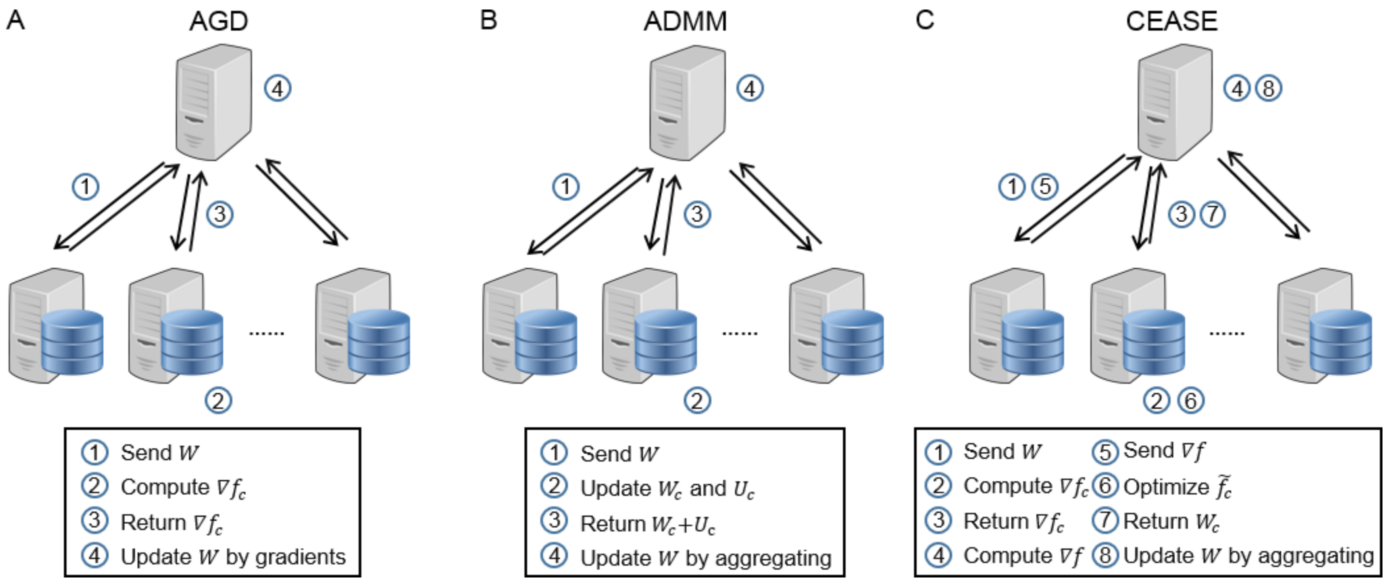

Fig. 1. Illustration of updating $\boldsymbol{W}$. AGD optimizes $\boldsymbol{W}$ on the central processor. ADMM and CEASE update $\boldsymbol{W}_c$ on the nodes and aggregate them on the central processor. The steps at each round of iteration are numbered sequentially.

Algorithm 2 Updating $\boldsymbol{W}$ with ADMM

```
Input: initial $\boldsymbol{W}^0$, $\boldsymbol{W}_c^0$, $\boldsymbol{U}_c^0$, $\rho$, $k = 0$
1: repeat
2:     Computes $\boldsymbol{W}_c^k$ by Eq. (25) on each node machine
3:     Computes $\boldsymbol{W}_c - \boldsymbol{U}_c^k/\rho$ in each node machine and sends to the central processor
4:     The central processor obtains $\boldsymbol{W}^{k+1}$ by aggregating
           $\boldsymbol{W}^{k+1} = \mathcal{S}_{\lambda/C\rho}(\bar{\boldsymbol{W}}_c^{k+1} - \bar{\boldsymbol{U}}_c^k/\rho)$
5:     $k \leftarrow k+1$
6: until Convergence
```

Lagrangian

$$\begin{aligned}L_\rho(\boldsymbol{W}, \boldsymbol{W}_c, \boldsymbol{U}_c) = \sum_{c=1}^{C} \frac{1}{2} \|\boldsymbol{X}_c - \boldsymbol{W}_c\boldsymbol{H}_c\|_F^2 + \lambda \|\boldsymbol{W}\|_1 \\ + \sum_{c=1}^{C} \langle \boldsymbol{U}_c, \boldsymbol{W} - \boldsymbol{W}_c\rangle + \frac{1}{2}\rho \sum_{c=1}^{C} \|\boldsymbol{W} - \boldsymbol{W}_c\|_F^2,\end{aligned} \tag{21}$$

where $\rho > 0$ is the penalty parameter, and $\boldsymbol{U}_c$ is the corresponding dual variables. $\boldsymbol{W}$ is the global variable stored on the central processor, and $\boldsymbol{W}_c$, $\boldsymbol{U}_c$, $\boldsymbol{H}_c$ are locally stored on the nodes. The ADMM at the $(k+1)$-th iteration consists of the following steps

$$\boldsymbol{W}_c^{k+1} = \arg\min_{\boldsymbol{W}_c} L_\rho(\boldsymbol{W}^k, \boldsymbol{W}_c, \boldsymbol{U}_c^k), \tag{22}$$

$$\boldsymbol{W}^{k+1} = \arg\min_{\boldsymbol{W}} L_\rho(\boldsymbol{W}, \boldsymbol{W}_c^{k+1}, \boldsymbol{U}_c^k), \tag{23}$$

$$\boldsymbol{U}_c^{k+1} = \boldsymbol{U}_c^k + \rho(\boldsymbol{W}^{k+1} - \boldsymbol{W}_c^{k+1}). \tag{24}$$

Both Eq. (22) and Eq. (23) have the closed-form solutions

$$\boldsymbol{W}_c^{k+1} = \left(\frac{\boldsymbol{X}_c\boldsymbol{H}_c^T + \boldsymbol{U}_c^k}{\rho} + \boldsymbol{W}^k\right)\left(\boldsymbol{I}_r + \frac{\boldsymbol{H}_c\boldsymbol{H}_c^T}{\rho}\right)^{-1}, \tag{25}$$

$$\boldsymbol{W}^{k+1} = \mathcal{S}_{\lambda/C\rho}(\bar{\boldsymbol{W}}_c^{k+1} - \bar{\boldsymbol{U}}_c^k/\rho), \tag{26}$$

where $\mathcal{S}_{\lambda/C\rho}$ is the soft thresholding operator with parameter $\lambda/C\rho$. Now, $\boldsymbol{W}_c$ is optimized locally on the nodes in parallel and requires no data communication. $\boldsymbol{U}_c$ is also locally optimized in parallel. The only step that involving data communication is updating the global variable $\boldsymbol{W}$. It is simply $\bar{\boldsymbol{W}}_c^{k+1} - \bar{\boldsymbol{U}}_c^k/\rho$ and taking soft thresholding operation. We compute $\boldsymbol{W}_c^k - \boldsymbol{U}_c^k$ on each node machine in parallel and then aggregate the results on the central node. We then apply the thresholding operator to obtain the new $\boldsymbol{W}$ and broadcast it to all nodes. Note that at each iteration, we only need to collect and broadcast a matrix of size $m \times r$. Therefore, the data communication load has been significantly reduced. The algorithm of updating $\boldsymbol{W}$ with ADMM is summarized in Fig. 1B and Algorithm 2.

4.4 Efficient Distributed Statistical Inference

We can also solve Eq. (15) from a statistical perspective. Recent advances on distributed statistical inference [41], [42] provide us with powerful tools. Here we use the CEASE to develop an efficient distributed statistical procedure due to its effectiveness. Let $\boldsymbol{W}$ at the k-th iteration be $\boldsymbol{W}^k$. Following the scheme of CEASE, each node machine computes

$$\boldsymbol{W}_c^k = \arg\min_{\boldsymbol{W}} \tilde{f}_c(\boldsymbol{W}), \tag{27}$$

where

$$\begin{aligned}\tilde{f}_c(\boldsymbol{W}) = f_c(\boldsymbol{W}) - \langle \nabla f_c(\boldsymbol{W}^k) - \nabla f(\boldsymbol{W}^k), \boldsymbol{W}\rangle \\ + \frac{\gamma}{2}\left\|\boldsymbol{W} - \boldsymbol{W}^k\right\|_F^2 + g(\boldsymbol{W}),\end{aligned} \tag{28}$$

where $\gamma \geq 0$ is the parameter of the proximal point algorithm. It is notably that the $f_c(\boldsymbol{W}) - \langle \nabla f_c(\boldsymbol{W}^k) - \nabla f(\boldsymbol{W}^k), \boldsymbol{W}\rangle$ is referred as the gradient-enhanced loss (GEL) function, in which the loss of the local data $\boldsymbol{X}_c$ is enhanced by the global gradient $\nabla f(\boldsymbol{W}^k)$. Conceptually, the global gradient adaptively enhances the similarity of the f_c and thus accelerates the convergence. This idea of using GEL has also been explored in [41], [43]. When $\lambda = 0$, there exists a closed-form solution. While $\lambda > 0$, Eq. (27) consists of the non-smooth L_1-norm regularizer $g(\boldsymbol{W})$ and the remaining smooth terms that are merely sums of the quadratic loss term and the linear terms. It can also be efficiently solved by FISTA. The optimizing of $\boldsymbol{W}_c^k$ requires no data communication. The central processor collects $\boldsymbol{W}_c^k$ and aggregates by taking average $\boldsymbol{W}^{k+1} = \frac{1}{C}\sum_{c=1}^{C}\boldsymbol{W}_c^k$. The whole algorithm of updating W with CEASE is summarized in Fig. 1C and Algorithm 3. The data communication load of CEASE is twice of the ADMM due to additionally broadcasting and sending $\nabla f(\boldsymbol{W}^k)$.

Algorithm 3 Updating $\boldsymbol{W}$ with CEASE

```
Input: initial $\boldsymbol{W}^0$, $\boldsymbol{W}_c^0$, $\gamma$, $k = 0$
1: repeat
2:     Computes $\nabla f_k(\boldsymbol{W}^k)$ on each node machine and sends to the central processor
3:     The central processor computes $\nabla f(\boldsymbol{W}^k) = 1/C\sum_{c=1}^{C}\nabla f_k(\boldsymbol{W}^k)$ and broadcasts to nodes.
4:     Computes $\boldsymbol{W}_c^k$ on each node machine by FISTA and sends to the central processor
5:     The central processor aggregates $\boldsymbol{W}^{k+1} = \bar{\boldsymbol{W}}_c^k$
6:     $k \leftarrow k + 1$
7: until Convergence
```

4.5 Tackle the Heterogeneous Noise

ADMM and CEASE do not account for the heterogeneity of the noise. Fortunately, they can be easily extended by plugging in the weighted average to achieve that. Given $\boldsymbol{W}_c$ and $\boldsymbol{H}_c$, the variance of the noise of each $\boldsymbol{X}_c$ is computed by $\sigma_c^2 = \|\boldsymbol{X}_c - \boldsymbol{W}_c\boldsymbol{H}_c\|_F^2/mn_c$, which can be derived by the maximum likelihood as showed in [23]. Then $\boldsymbol{H}_c$ can be separately updated on each node machine, and thus different noise levels will not influence the results. However, aggregating $\boldsymbol{W}_c$ corresponding to different levels of noise by taking average is problematic. Let's consider the ADMM algorithm. Intuitively, $\boldsymbol{W}_c$ inferred from $\boldsymbol{X}_c$ of lower noise is more believable. Inspired by [23], we adopt the weighted average to aggregate $\boldsymbol{W}_c$

$$\tilde{\boldsymbol{W}}_c = \sum_{c=1}^{C}\frac{1/\sigma_c^2}{\sum_{c=1}^{C}1/\sigma_c^2}\boldsymbol{W}_c. \tag{29}$$

Note that the weight of $\boldsymbol{W}_c$ with small variance σ_c^2 is higher. CEASE can also be easily modified by plugging in the weighted average. Note that CEASE takes the average of both the gradients $\nabla f_k(\boldsymbol{W}^k)$ (Algorithm 3, line 3) and $\boldsymbol{W}_c$ (Algorithm 3, line 5) on the central processor. Thus, we can also use the weighted average versions to aggregate the gradients and $\boldsymbol{W}_c$, respectively.

4.6 Computational Remarks on Updating W

4.6.1 The Optimality of the Weighted Average

Assumption 1. *$\boldsymbol{X}_c$ follows the matrix normal distribution with isotropic covariances, $\boldsymbol{X}_c \sim \mathcal{MN}(\boldsymbol{X}_c^*, \sigma_c I, \sigma_c I)$.*

Assumption 2. *$\{\boldsymbol{H}_c\}_{c=1}^{C}$ are of the same size and each column of H_c follows the same distribution and the expectation $E(\boldsymbol{H}_c\boldsymbol{H}_c^T)$ exists.*

Note that the Assumption 1 is equivalent to the generation process Eq. (5). The estimated $\tilde{\boldsymbol{W}}$ and $\bar{\boldsymbol{W}}$ are also random matrices. We use the sum of the entry-wise variances to measure the variances, e.g., $\mathrm{var}(\tilde{\boldsymbol{W}}) = \sum_{i,k}\mathrm{var}(\tilde{w}_{ik})$. For convenience, we assume that $\{\boldsymbol{H}_c\}_{c=1}^{C}$ are of the same size.

Theorem 1. *Let Assumptions 1 and 2 hold and let $\lambda = 0$ and $k \to \infty$. The variance ratio*

$$\frac{\mathrm{var}(\tilde{\boldsymbol{W}})}{\mathrm{var}(\bar{\boldsymbol{W}})} = \frac{\sum_c C/(1/\sigma_c^2)}{\sum_c \sigma_c^2/C} \le 1 \tag{30}$$

The equality reaches if and only if all σ_c^2 are equal. Moreover, the $\tilde{\boldsymbol{W}}$ is the optimal weighted average that minimizes the variance.

Proof. We first consider the ADMM algorithm. Both f and g are **closed, proper and convex**. A previous study [44, Section 3.2.1] has shown that the dual variable $\boldsymbol{U}_c$ converges to $\boldsymbol{U}_c^*$ with $k \to \infty$. So we treat $\boldsymbol{U}_c^*$ as a constant matrix. Note that

$$\boldsymbol{X}_c\boldsymbol{H}_c^T/\rho \sim \mathcal{MN}(\boldsymbol{X}_c^*\boldsymbol{H}_c^T/\rho, \sigma_c I/\rho, \sigma_c\boldsymbol{H}_c^T\boldsymbol{H}_c/\rho). \tag{31}$$

Based on the Eq. (22) and the property of the matrix normal distribution, we have

$$\mathrm{var}(\boldsymbol{W}_c) = \sigma_c^2\,\mathrm{tr}(I \otimes \boldsymbol{\Lambda}_c\boldsymbol{H}_c^T\boldsymbol{H}_c\boldsymbol{\Lambda}_c), \tag{32}$$

where $\Lambda_c = (\boldsymbol{I} + \boldsymbol{H}_c\boldsymbol{H}_c^T)^{-1}$. The variance ratio

$$\begin{aligned}\frac{\mathrm{var}(\tilde{\boldsymbol{W}})}{\mathrm{var}(\bar{\boldsymbol{W}})} &= \frac{\sum_c\frac{1/\sigma_c^4}{(\sum_c 1/\sigma_c^2)^2}\mathrm{var}(\boldsymbol{W}_c)}{\sum_c\frac{1}{C^2}\mathrm{var}(\boldsymbol{W}_c)}\\ &= \frac{\sum_c\frac{1/\sigma_c^2}{(\sum_c 1/\sigma_c^2)^2}\mathrm{tr}(\boldsymbol{\Lambda}_c\boldsymbol{H}_c^T\boldsymbol{H}_c^T\boldsymbol{\Lambda}_c)}{\sum_c\frac{\sigma_c^2}{C^2}\mathrm{tr}(\boldsymbol{\Lambda}_c\boldsymbol{H}_c^T\boldsymbol{H}_c^T)}\\ &= \frac{\sum_c C/(1/\sigma_c^2)}{\sum_c\sigma_c^2/C} \le 1\end{aligned} \tag{33}$$

Based on the inequality of arithmetic and geometric means, the ration is not greater than 1.

Then we prove the optimality. Consider minimizing the variance of the weighted average $v_c\boldsymbol{W}_c$.

$$\min\sum_c v_c^2\sigma_c^2 a, \tag{34}$$

with $\boldsymbol{v} \ge 0, \sum_c v_c = 1$. Denote $a = \mathrm{tr}(\boldsymbol{\Lambda}_c\boldsymbol{H}_c^T\boldsymbol{H}_c^T)$. Eq. (34) is a constrained quadratic programming. The KKT condition

$$v_c\sigma_c^2 a - \mu = 0, c \in [C] \tag{35}$$

where $\mu > 0$ is the dual variable of the Lagrangian. Then $v_i = \frac{1/\sigma_c^2}{\sum_c 1/\sigma_c^2}$ and $\mu = \frac{a}{\sum_c 1/\sigma_c^2}$ is the solution of the system of equations. □

Note that the numerator is the harmonic average of σ_c^2 and the denominator is the arithmetic average of σ_c^2. Theorem 1 shows that the weighted average $\tilde{\boldsymbol{W}}$ can reduce the variance of $\bar{\boldsymbol{W}}$, and the weights are optimal. The result holds for both ADMM and CEASE algorithms. When $\{\boldsymbol{H}_c\}_{c=1}^{C}$ are of different size, there exists similar result that can be proved with the same procedure.

4.6.2 *Convergence Rate*

In this section, we discuss the convergence rate of updating $\boldsymbol{W}$. In particular, we concern about the effect of the number of instances increasing on the convergence rate.

Lemma 1. *$f(\boldsymbol{W})$ is strongly convex with parameter $\mu_f = \sigma_{min}(\sum_{c=1}^{C} \boldsymbol{H}_c\boldsymbol{H}_c^T)$, and $\nabla f(\boldsymbol{W})$ is Lipschitz continuous with parameter $L_f = \sigma_{max}(\sum_{c=1}^{C} \boldsymbol{H}_c\boldsymbol{H}_c^T)$, where $\sigma_{min}(\boldsymbol{A})$ and $\sigma_{max}(\boldsymbol{A})$ indicate the smallest and the largest eigenvalue of matrix A, respectively. The ration $\kappa_f = L_f/\mu_f$ exists.*

Proof. $f(\boldsymbol{W})$ is twice differentiable and we have

$$\nabla f(\boldsymbol{W}) = \sum_{c=1}^{C}(\boldsymbol{W}\boldsymbol{H}_c - \boldsymbol{X}_c)\boldsymbol{H}_c^T, \nabla^2 f(\boldsymbol{W}) = \sum_{c=1}^{C}(\boldsymbol{H}_c\boldsymbol{H}_c^T)\otimes I \tag{36}$$

Note that

$$\nabla^2 f(\boldsymbol{W}) \succeq \sigma_{\min}\left(\sum_{c=1}^{C}\boldsymbol{H}_c\boldsymbol{H}_c^T\right) I \tag{37}$$

It implies that $f(\boldsymbol{W})$ is strongly convex with parameter $\mu_f = \sigma_{\min}(\sum_{c=1}^{C} \boldsymbol{H}_c\boldsymbol{H}_c^T)$. For $\forall \boldsymbol{W}, \boldsymbol{Y} \in R^{m\times r}$, we have

$$\begin{aligned}\|\nabla f(\boldsymbol{W}) - \nabla f(\boldsymbol{Y})\|_F &= \left\|(\boldsymbol{W}-\boldsymbol{Y})\sum_{c=1}^{C}\boldsymbol{H}_c\boldsymbol{H}_c^T\right\|_F \\ &\le L_f \|\boldsymbol{W}-\boldsymbol{Y}\|_F ,\end{aligned} \tag{38}$$

where $L_f = \sigma_{\max}(\sum_{c=1}^{C} \boldsymbol{H}_c\boldsymbol{H}_c^T)$. $\nabla f(\boldsymbol{W})$ is Lipschitz continuous with parameter L_f. □

Lemma 2. *There exists $\delta > 0$, such that $\left\|\nabla^2 f(\boldsymbol{W}) - \nabla^2 f_c(\boldsymbol{W})\right\|_F \le \delta$ holds for all $c \in [C]$ and $\boldsymbol{W} \in R^{m\times r}$.*

Lemma 1 and 2 characterize the smoothing part of the objective function. Suppose that the Assumption 2 holds. It is easy to verify that L_f and μ_f grow linearly with the increasing of the number of instances (n_c) in each node machine.

Proof. Note

$$\left\|\nabla^2 f(\boldsymbol{W}) - \nabla^2 f_c(\boldsymbol{W})\right\|_2 = \left\|\sum_{l\neq c}\boldsymbol{H}_l\boldsymbol{H}_l^T \otimes I\right\|_2 = \delta_c. \tag{39}$$

Thus, there exists and $\delta = \max\{\delta_c\}_{c=1}^{C}$. □

The FISTA algorithm is known to have a quadratic convergence rate [39, Theorem 4.4]. But the contraction factor was not given in this work. Tao *et al* [45, Thereom 5.5] provided the contraction factor τ_1 of the local convergence. The contraction factor of FISTA, τ_1 depends on the structural parameter κ_f. The contraction factor of the CEASE algorithm was given in [42, Theorem 3.1].

Theorem 2. *Consider $\{\boldsymbol{W}^k\}$ generated by Algorithm 3. Suppose that $\delta^2/(\mu_f+\gamma)^2 < \mu_f/(\mu_f+2\gamma)$. We have:*

$$\left\|\boldsymbol{W}^{k+1} - \boldsymbol{W}^*\right\|_F \le \left\|\boldsymbol{W}^{k} - \boldsymbol{W}^*\right\|_F \tau_3, \tag{40}$$

where $\boldsymbol{W}^*$ is the KKT point, $\tau_3 = \frac{\delta\sqrt{\mu_f^2+2\gamma\mu_f}+\gamma}{(\mu_f+\gamma)^2} < 1$ is the contraction factor.

Theorem 2 implies that the CEASE algorithm converges faster with n_c increasing. It is intuitive that the estimation of $\boldsymbol{W}$ in the node machine, i.e., $\boldsymbol{W}_c^k$, is more accurate when n_c is sufficiently large. Therefore, it takes fewer rounds of aggregation before the algorithm converges. But it is not the case for the AGD algorithm, the structural parameter κ_f remains stable with n_c increasing (Fig. 3, right). So we do not expect Algorithm 1 converges faster with n_c increasing.

TABLE 1
Complexity and Communication Load of Updating $\boldsymbol{W}$

	Complexity	Communication load
AGD	$O(q_1C(mnr+mr^2))$	$2q_1mr$
ADMM	$O(q_2C(mnr+mr^2+r^3))$	$2q_2mr$
CEASE	$O(tq_3C(mnr+mr^2))$	$4q_3mr$

4.6.3 *Computational Complexity and Communication Load*

We focus on the computational complexity and communication load of updating $\boldsymbol{W}$ until convergence of the three algorithms. Suppose that the AGD, ADMM and CEASE algorithms stop in q_1, q_2 and q_3 iterations, respectively. Computing the gradients on C machines is $O(C(mnr+mr^2))$, and then the complexity of AGD is $O(C(mnr+mr^2))$. AGD needs to collect the gradient and then broadcast the updated $\boldsymbol{W}$, so the communication load is $2q_1mr$. One iteration of ADMM involves matrix multiplication, soft thresholding and computation of the inverse of matrices of size $r\times r$, and thus has the complexity $O(C(mnr+mr^2+r^3))$. The total complexity is $O(q_2C(mnr+mr^2+r^3))$. ADMM only collects and broadcasts a matrix of $m\times r$. So, the communication load of ADMM is $2q_2mr$. CEASE computes the $\boldsymbol{W}_c$ by FISTA. Suppose FISTA stops in t iterations, and then the complexity of CEASE is $O(tq_3C(mnr+mr^2))$. CEASE broadcasts and collects both the gradients and $\boldsymbol{W}_c$, so the communication load is $4q_3mr$. Table 1 summarizes the complexity and the communication load of updating $\boldsymbol{W}$ for all the tree algorithms. Both ADMM and CEASE introduce the auxiliary variable $\boldsymbol{W}_c$ on the nodes to reduce the communication load, and they have to pay the extra computational cost.

5 Experimental Results

We first evaluated the three algorithms on synthetic data to verify the theoretical analysis of updating $\boldsymbol{W}$. Then we applied the proposed methods to real-world datasets for clustering and compared them with the distributed k-means and Scalable-NMF. The synthetic experiments were performed on a desktop computer with a 2GHz Intel Xeon E5-2683 v3 CPU, a GTX 1080 GPU card, 16GB memory, and the real-world experiments were performed on a small Spark cluster with eight machines (one central processor and the rest are nodes). Each machine is equipped with an Intel i7 CPU and 16GB memory. The source code is available at https://github.com/zhanglabtools/dbmd. Readers may also refer to Supplementary Materials for implementation details.

5.1 Synthetic Experiments

We generate the basis matrix $\boldsymbol{W} \in R^{m\times r}$ inspired by [46],

$$w_{ik} = \begin{cases} a, & 1+(k-1)(l-coh) \le i \le l+(k-1)(l-coh) \\ & k \in [r] \\ 0, & \text{otherwise}\end{cases} \tag{41}$$

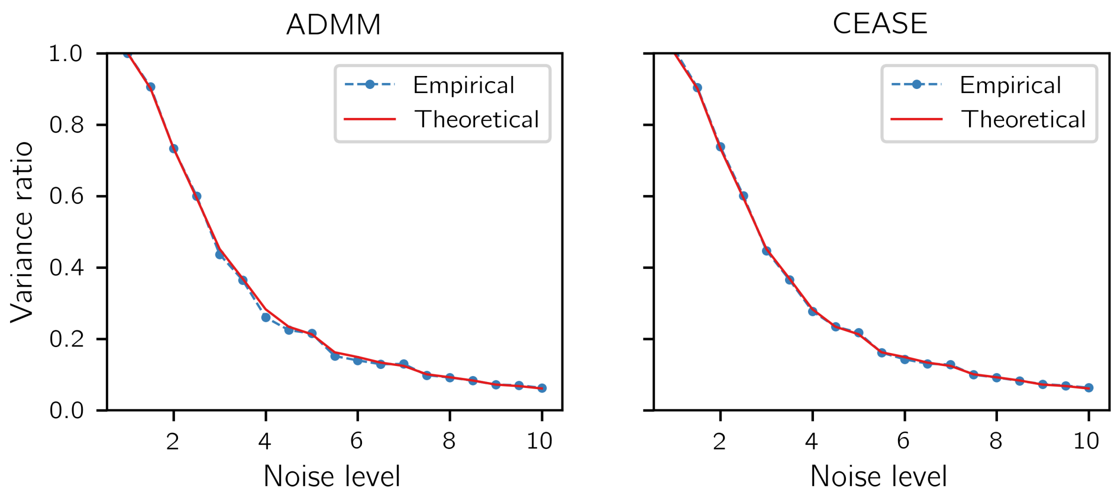

Fig. 2. The variance ratio $\mathrm{var}(\tilde{\boldsymbol{W}})/\mathrm{var}(\bar{\boldsymbol{W}})$ on a series of synthetic datasets $\{\boldsymbol{X}_c\}_{c=1}^5$, where the noise level $\sigma_c = 1, c \in [4]$, and σ_5 increases from 1 to 10.

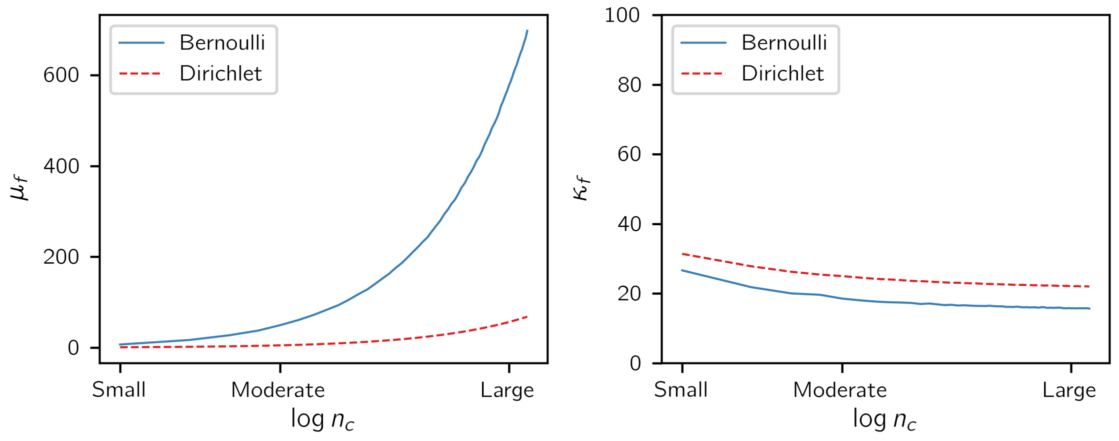

Fig. 3. The $\log n_c$ versus the largest eigenvalue $\sigma_{\max}$ and the condition number κ of $\sum_{c=1}^C \boldsymbol{H}_c\boldsymbol{H}_c^T$, respectively. $\boldsymbol{H}_c$ are drawn from the Bernoulli and Dirichlet distributions respectively. n_c ranges from 100 to 6000, and $\sigma_{\max}$ and κ are the average of 100 times for a given n_c.

where a is a constant, l denotes the number of non-zero entries in each column of $\boldsymbol{W}$, and coh denotes the length of coherence between basis $\boldsymbol{w}_{i-1,\cdot}$ and $\boldsymbol{w}_{i,\cdot}$. We generated the coefficient matrices $\{\boldsymbol{H}_c\}_{c=1}^C$ in two different ways: 1) draw entries of $\boldsymbol{H}_c$ from the Bernoulli distribution $(h_{kj})_c \sim \mathrm{B}(1,p)$. We set the last entries of all zero columns of $\boldsymbol{H}_c$ to 1, and then we normalize $\boldsymbol{H}_c$ such that the sum of column equals one; 2) draw columns of $\boldsymbol{H}_c$ from the Dirichlet distribution with a parameter $\boldsymbol{\alpha}$. Then the observed data matrices $\{\boldsymbol{X}_c\}_{c=1}^C$ are generated by $\boldsymbol{X}_c = \boldsymbol{W}\boldsymbol{H}_c + \boldsymbol{E}_c$, where $(e_{ij})_c \sim N(0, \sigma_c^2)$. We suppose that $\boldsymbol{H}_c$ is known in this subsection.

To verify the effectiveness of the weighted averages, we generated a series of datasets $\{\boldsymbol{X}_c\}_{c=1}^5$ with $a = 1.5$, $l = 20$, $n_c = 100$, $r = 10$ and $coh = 2$. $\{\boldsymbol{H}_c\}_{c=1}^5$ were drawn from the Bernoulli distribution with $p = 0.1$. We set the noise level $\sigma_c = 1, c \in [4]$ and σ_5 ranges from 1 to 10. Consequently, $\boldsymbol{X}_c \in R^{182\times100}$. We applied the ADMM with $\rho = 50$ and CEASE to the $\{\boldsymbol{X}_c\}_{c=1}^5$ with known $\boldsymbol{H}_c$. The theoretical variance ratio is given in Theorem 1. We also computed the empirical variance ratio by repeating the experiments for 100 times. The result confirms the correctness of our theoretical analysis in Theorem 1. The empirical variance ratio fits the theoretical line well for both the ADMM and CEASE algorithms (Fig. 2). With σ_c increasing, the estimated variance of $\boldsymbol{W}$ by the weighted average is smaller than that of the simple average (variance ratio approaches 0). Therefore, the plug-in weighted average can significantly reduce the variance of the estimated $\boldsymbol{W}$ when the heterogeneous noise exists.

We then investigated the convergence behaviors of the proposed methods with small, moderate, and large n_c on nodes. To facilitate the comparison, we generated the first synthetic data A $\{\boldsymbol{X}_c\}_{c=1}^5$ with $a = 1.5$, $l = 20$, $r = 20$, $coh = 2$ $\sigma_c = 1$ and $n_c = 100, 500, 5000$, respectively. $\boldsymbol{H}_c$ were drawn from the Bernoulli distribution with $p = 1/20$. Synthetic data A contains 3 datasets with different n_c. We generated another synthetic data B with the same parameters, but $\boldsymbol{H}_c$ were drawn from the Dirichlet distribution with $\boldsymbol{\alpha} = \boldsymbol{1}$. Given n_c, $\sigma_{\max}(\sum_{c=1}^C \boldsymbol{H}_c\boldsymbol{H}_c^T)$ of $\boldsymbol{H}_c$ drawn from the Bernoulli distribution is greater than that of $\boldsymbol{H}_c$ drawn from the Dirichlet distribution; $\sigma_{\max}(\sum_{c=1}^C \boldsymbol{H}_c\boldsymbol{H}_c^T)$ grows linearly with n_c increasing (Fig. 3, left). But the condition number $\kappa(\sum_{c=1}^C \boldsymbol{H}_c\boldsymbol{H}_c^T)$ doesn't change a lot with n_c increasing (Fig. 3, right).

We plot the curves of the number of iterations versus the objective function values (Fig. 4). The convergence behaviors confirm our analysis: 1) ADMM and CEASE converge faster when n_c gets larger, because $\sigma_{\max}(\sum_{c=1}^C \boldsymbol{H}_c\boldsymbol{H}_c^T)$ gets larger

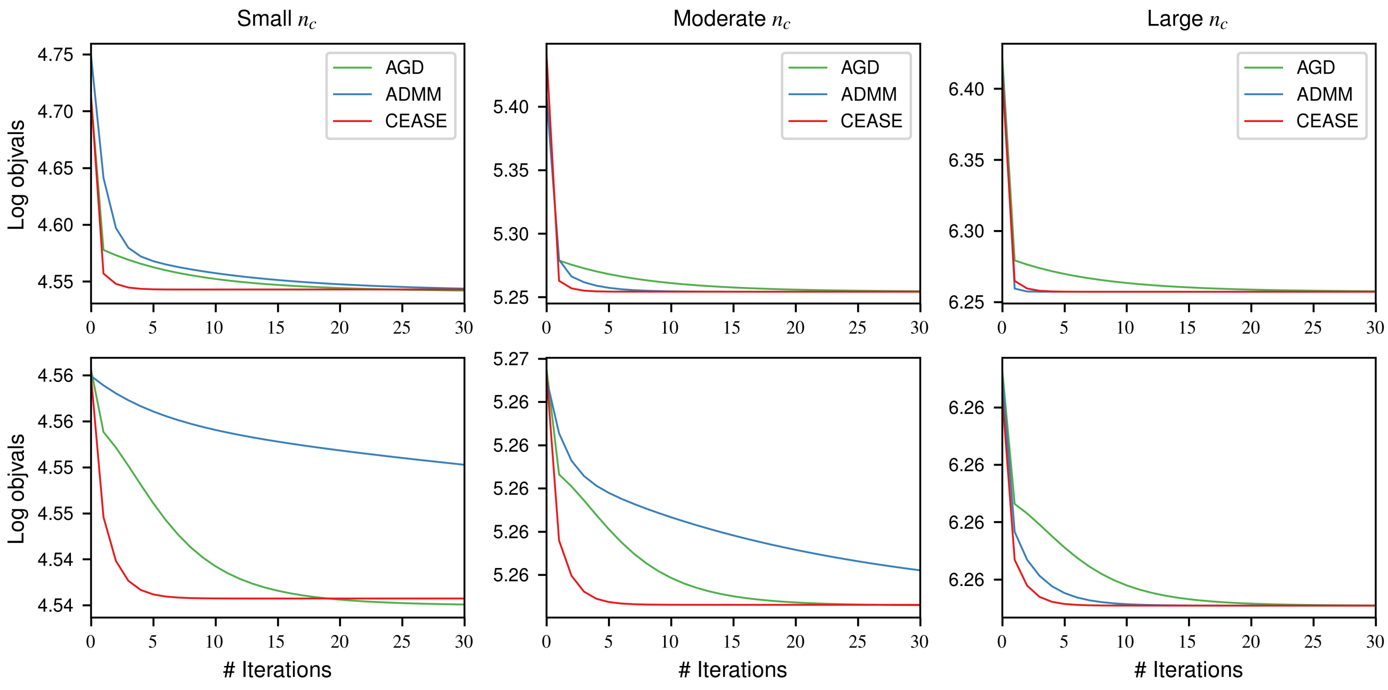

Fig. 4. Log objective function values (Log objvals) versus the number of iterations of AGD, ADMM and CEASE for updating $\boldsymbol{W}$, respectively. Top: results on synthetic data A; bottom: results on synthetic data B.From left to right: $n_c = 100, 500, 5000$, respectively.

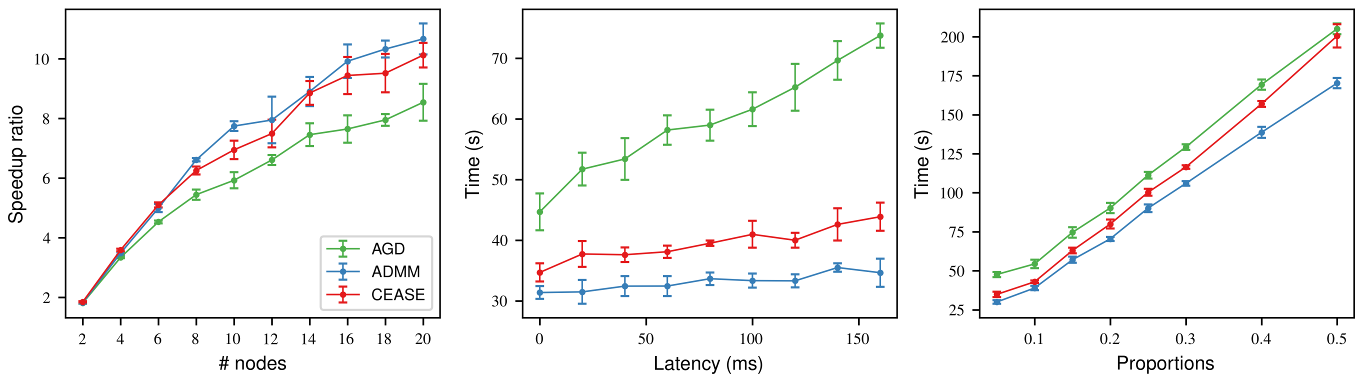

Fig. 5. Scalability of the proposed methods on the simulated data. Error bars show the standard deviations of 5 runs. Left to right: speedup ratios versus the number of nodes; latency between central machines and nodes versus running time, #nodes $= 20$; proportions of instances in first node versus running time, #nodes $= 20$.

when n_c increases. 2) ADMM converges faster when $\boldsymbol{H}_c$ are drawn from the Bernoulli distribution (Fig. 4, top row), because $\sigma_{\max}(\sum_{c=1}^{C} \boldsymbol{H}_c \boldsymbol{H}_c^T)$ is larger than that drawn from the Dirichlet distribution. It is the same for CEASE. 3) The convergence of AGD takes around the same number of steps for small, moderate, and large size of n_c. Unlike ADMM and CEASE, the convergence speed of AGD doesn't change with n_c increasing. There are also some other interesting observations: 1) The gradient-enhanced loss of CEASE accelerates its convergence. CEASE converges faster when n_c is small and moderate. 2) ADMM is very slow when n_c is small. But when n_c is sufficiently large, ADMM may take fewer steps than AGD. 3) Because we can compute the Lipschitz constant L of $\nabla f(\boldsymbol{W})$ directly, and $1/L$ is the largest step size. Therefore, AGD is fast on this problem, and it converges within 30 steps. The experimental results show that CEASE reduces the number iterations regardless of n_c. But ADMM reduces the number iterations when n_c is sufficiently large.

Scalability is one of the core issues of distributed algorithms. To demonstrate the scalability, we evaluated the proposed algorithms on simulated data in various scenarios. In particular, we generated the simulated data matrix with $m = 100$, $n = 1,000,000$ and $r = 5$. We increased the number of nodes from 1 to 20 and computed the speedup ratios. All of the proposed algorithms scale up well when the number of nodes is smaller than 10 (Fig. 5, left panel). However, the speedup ratios decay when the number of nodes > 10 due to the increase of the communication and computation load on the central machine (see Sec. 4.6.3). Specifically, compared to the other two algorithms, AGD suffers more speedup ratios decay because it requires more rounds of communications for updating $\boldsymbol{W}$. For the same reason, AGD will spend more time when the latency between the central machine and nodes is high (Fig. 5, middle panel, latency is measured in milliseconds). When the instances are not evenly distributed across nodes, ADMM and CEASE are slightly faster than AGD (Fig. 5,

TABLE 2
Summary of the Datasets

Dataset	# Instances	# Features	# Classes
CoverType	~540,000	54	7
KDD-99	~4,900,000	41	11
MNIST	~400,000	784	10
RCV1	~420,000	100	11
Optical-Radar	~320,000	174	7
SUSY	50,00,000	18	2

right panel).

5.2 Real-World Experiments

We further applied the proposed algorithms to real-world large-scale datasets for clustering.

Datasets. There are six datasets in total (Table 2). In particular, RCV1 was downloaded from [47], and the remaining datasets were downloaded from the UCI Machine Learning Repository (https://archive.ics.uci.edu/ml/datasets.php). Among the six datasets, the classes of CoverType, KDD-99, RCV1, and Optical-Radar are unbalanced. In particular, CoverType is the forest cover type data. It contains seven classes, and 85% instances are distributed between two classes. KDD-99 contains 23 classes. We removed classes that occurred less than 100 times, and 11 classes were retained. The largest three classes account for 97% instances (57%, 21% and 19%, respectively). RCV1 contains 53 classes, and we removed classes that are less than 2%. There are 11 classes left. The largest four classes account for 70%, and the proportions of the remaining classes range from 2% to 10%. Optical-Radar is very unbalanced too. 98% of the instances belong to the largest five classes. We computed z-score of each feature. The classes of MNIST and SUSY are evenly distributed. MNIST is the handwritten digits data. Each image is of size 28×28 and thus can be represented by a 784-dimensional vector. We normalized all features to [0, 1]. We used a subset of MNIST. SUSY is a physic dataset and only have two classes.

Experiment settings. For all experiments, we set $\rho = 150$, $\gamma = 0.001$, $\boldsymbol{\alpha} = 1.5$. We set the L_1-norm regularizer parameter $\lambda = 2000, 4000, 500, 100, 500, 500$ for CoverType, KDD-99, MNIST, RCV1, Optical-Radar, SUSY, respectively. We always set r equals the true number of classes for convenience. To facilitate the comparison of time costs of ADMM and CEASE, we stop the procedure of updating of $\boldsymbol{W}$ at the $(k+1)$-th iteration, if $||\boldsymbol{W}^{k+1} - \boldsymbol{W}^k|| \leq \left\|\boldsymbol{W}^0\right\|_F \times 10^{-2}$. AGD will stop immediately, because the step size $1/L_f$ is typically very small. Therefore, we ensured that the AGD algorithm iterates at least 30 rounds. The column of H_c indicates the membership of the corresponding instance. We assigned an instance $(\boldsymbol{x}_{\cdot j})_c$ to the class corresponding to the largest entry of the $(\boldsymbol{h}_{\cdot j})_c$. We then evaluated the performance of the clustering by accuracy and F-measure which is more suitable for unbalanced classes. The true classes and the predicted ones were matched by the Hungarian algorithm [48].

We compared the proposed algorithms with two distributed clustering algorithms based on matrix decomposition, including scalable k-means++ [49] and Scalable-NMF [50] implemented with Spark. Generally speaking, our methods achieve competitive or superior performance compared to them in terms of accuracy and F-measure on most of the datasets (Table 3). In particular, the

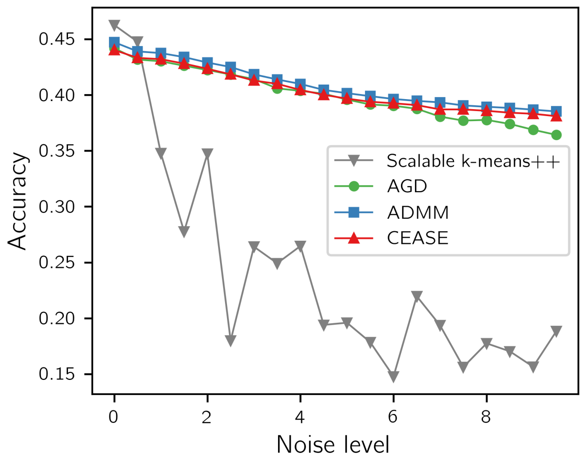

Fig. 6. The clustering performance on the noisy MNIST data. The noise level of 20% instances increasing from 0 to 9.5. The accuracy is the average of 10 runs.

proposed algorithms tend to have higher accuracy. Scalable-NMF does not perform well. The separable assumption of Scalable-NMF may be violated on real-world data. As for the running time, Scalable-NMF is very fast because it only requires one round iteration. Our methods are slower for involving alternatively updating $\boldsymbol{W}$ and $\boldsymbol{H}_c$. In particular, the interior algorithm for updating $\boldsymbol{H}_c$ is computationally expensive. AGD is slower than CEASE and ADMM because it requires more rounds of communications. CEASE reduces the communication load by paying more computational cost on the nodes. Among the three proposed algorithms, ADMM is the fastest because it has closed-form solutions at each step of updating $\boldsymbol{W}$, and n_c is sufficiently large. The results are consistent with that of the simulated data. One may consider to remove the Dirichlet prior constraints to obtain a faster algorithm of updating $\boldsymbol{H}_c$. Nevertheless, the experimental results show that the Dirichlet prior is essential to achieve better clustering performance (see Supplementary Materials).

To verify the robustness of the proposed methods to noise, we created a series of noisy MNIST data. Specifically, we added Gaussian noises of $\sigma_1 = 0.1$, $\sigma_2 = 1.0$ to 20% and 60% of the instances of MNIST data. We then added Gaussian noises of the standard deviation σ_3 varying from 0 to 9.5 in step of 0.5 to the remaining 20% instances. In consequence, we generated the semi-synthetic MNIST datasets under 20 different noise settings. We then applied the scalable k-means++ and the proposed methods to them. Scalable-NMF is omitted for its poor performance. Scalable k-means++ suffers from the increasing noise, and its performance drops down sharply and is very unstable (Fig. 6). On the contrary, all of the proposed methods show a mild performance decline while the noisy level increases. It implies that the Bayesian priors reduce the risk of overfitting to the highly noisy data. Moreover, the performance of ADMM and CEASE are slightly better than that of AGD when the noisy level is sufficiently large ($\sigma_3 > 6.5$). It reminds us that considering the heterogeneity of the noise also contributes to the robustness of the model.

6 Discussion and Conclusion

We proposed a distributed Bayesian matrix decomposition model for big data mining and clustering. Three distributed strategies (i.e., AGD, ADMM and CEASE) were adopted to solve it. The

TABLE 3
Clustering Performance on Real-world Datasets

	CoverType			KDD-99			MNIST		
	Accuracy	F-Measure	Time	Accuracy	F-Measure	Time	Accuracy	F-Measure	Time
Scalable-NMF	33.79	30.08	3.33(0.41)	79.42	40.72	10.53(0.60)	20.55	23.01	35.26(4.10)
Scalable k-means++	29.90(4.46)	22.29(3.82)	13.41(1.34)	84.30(2.44)	35.59(5.89)	28.61(2.56)	47.34(1.93)	48.08(1.55)	78.61(8.73)
DBMD-AGD	42.83(0.18)	28.80(0.96)	81.68(3.02)	90.56(0.98)	40.16(2.06)	507.83(9.00)	43.60(0.75)	43.59(1.06)	280.03(20.14)
DBMD-ADMM	42.87(0.06)	28.45(2.45)	64.56(2.19)	90.14(0.86)	42.08(3.82)	419.28(38.53)	45.20(0.53)	45.07(0.47)	175.78(2.18)
DBMD-CEASE	42.96(0.06)	29.15(1.72)	65.04(2.74)	91.52(0.53)	40.58(2.15)	394.50(29.47)	45.06(1.54)	45.01(1.64)	198.81(19.32)
	RCV1			Optical-Radar			SUSY		
Scalable-NMF	29.19	25.07	3.55(0.36)	-	-	-	-	-	-
Scalable k-means++	39.41(2.67)	32.15(4.08)	23.82(1.62)	70.01(5.08)	55.52(6.07)	47.22(5.29)	63.57(0.00)	61.82(0.00)	74.17(5.54)
DBMD-AGD	42.74(2.16)	33.50(3.47)	112.86(2.46)	75.41(4.26)	63.37(2.37)	101.74(21.81)	65.42(0.03)	62.71(0.40)	128.85(2.75)
DBMD-ADMM	43.16(1.72)	33.41(2.94)	78.05(2.08)	75.91(2.35)	58.96(3.37)	78.25(19.69)	65.42(0.04)	62.90(0.27)	131.37(35.82)
DBMD-CEASE	43.75(3.19)	34.86(2.83)	84.95(1.17)	76.71(1.96)	64.31(3.97)	78.66(22.50)	65.13(0.62)	62.67(0.92)	118.87(5.13)

The means and the standard deviations of 5 runs are shown here. Time is in seconds (s).The standard deviations of the Scalable-NMF are not reported, because it is a deterministic algorithm. Scalable-NMF is not suitable for Optical-Radar and SUSY because they contain negative entries.

convergence rates of AGD and CEASE depend on different structural parameters (μ_f and κ_f) and thus have different behaviors. In short, CEASE converges faster with the number of instances on each node machine increasing, but the convergence rate of AGD doesn't change much. Empirically, ADMM also converges faster with the number of instances growing. To tackle the heterogeneous noise in the data, we propose an optimal plug-in weighted average scheme that significantly reduces the variance of the estimation. The proposed algorithms scale up well. The real-world experiments demonstrate that the proposed algorithms achieve superior or competitive performance. Both the Bayesian prior and the weighted average strategies reduce the influence of the highly noisy data.

There are several questions worth investigating in future studies. First, the concept of the weighted average can be generalized to other algorithms. Second, we assume that the transposition of the data matrix is tall-and-skinny, which is limited. It is commonplace for modern applications that the data matrix is fat and tall, i.e., the numbers of rows and columns are both vast. Finally, we observed that ADMM converges faster when n_c is larger. The convergence rate of this algorithm needs to be further investigated.

Acknowledgment

This work has been supported by the National Key Research and Development Program of China [2019YFA0709501]; the National Natural Science Foundation of China [11661141019, 61621003, 61762089]; the National Ten Thousand Talent Program for Young Top-notch Talents, and the CAS Frontier Science Research Key Project for Top Young Scientist [QYZDB-SSW-SYS008].

References

[1] K. Pearson, " LIII. no lines and planes of closest fit to systems of points in space ," *London, Edinburgh, Dublin Philos. Mag. J. Sci.*, vol. 2, no. 11, pp. 559–572, nov 1901.
[2] C. Bishop, *Pattern recognition and machine learning*. Springer, 2006.
[3] Y. Shen, Z. Wen, and Y. Zhang, "Augmented Lagrangian alternating direction method for matrix separation based on low-rank factorization," *Optim. Methods Softw.*, vol. 29, no. 2, pp. 239–263, mar 2014.
[4] W. Min, J. Liu, and S. Zhang, "Group-sparse svd models via l_1-and l_0-norm penalties and their applications in biological data," *IEEE Trans. Knowl. Data Eng.*, 2019.
[5] P. Paatero and U. Tapper, "Positive matrix factorization: A non-negative factor model with optimal utilization of error estimates of data values," *Environmetrics*, vol. 5, no. 2, pp. 111–126, jun 1994.
[6] D. D. Lee and H. S. Seung, "Learning the parts of objects by non-negative matrix factorization," *Nature*, vol. 401, no. 6755, pp. 788–791, oct 1999.
[7] P. O. Hoyer, "Non-negative sparse coding," in *Neural Networks Signal Process. - Proc. IEEE Work.*, vol. 2002-January, 2002, pp. 557–565.
[8] H. Kim and H. Park, "Sparse non-negative matrix factorizations via alternating non-negativity-constrained least squares for microarray data analysis," *Bioinformatics*, vol. 23, no. 12, pp. 1495–1502, jun 2007.
[9] D. Cai, X. He, X. Wu, and J. Han, "Non-negative matrix factorization on manifold," in *Proc. IEEE Int. Conf. Data Mining*, 2008, pp. 63–72.
[10] D. Cai, X. He, J. Han, and T. S. Huang, "Graph regularized nonnegative matrix factorization for data representation," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 33, no. 8, pp. 1548–1560, 2011.
[11] S. Zhang, Q. Li, J. Liu, and X. J. Zhou, "A novel computational framework for simultaneous integration of multiple types of genomic data to identify microrna-gene regulatory modules," *Bioinformatics*, vol. 27, no. 13, pp. i401–i409, 2011.
[12] M. E. Tipping and C. M. Bishop, "Probabilistic principal component analysis," *J. R. Stat. Soc. Ser. B (Statistical Methodol.)*, vol. 61, no. 3, pp. 611–622, aug 1999.
[13] C. M. Bishop, "Bayesian PCA," in *Adv. Neural Inf. Process. Syst.*, 1999, pp. 382–388.
[14] M. Collins, S. Dasgupta, and R. E. Schapire, "A generalization of principal component analysis to the exponential family," in *Adv. Neural Inf. Process. Syst.*, 2001, pp. 617–624.
[15] S. Mohamed, Z. Ghahramani, and K. A. Heller, "Bayesian exponential family PCA," in *Adv. Neural Inf. Process. Syst.*, 2009, pp. 1089–1096.
[16] J. Li and D. Tao, "Simple exponential family PCA," *IEEE Trans. Neural Networks Learn. Syst.*, vol. 24, no. 3, pp. 485–497, 2013.
[17] C. Xu, D. Tao, and C. Xu, "A survey on multi-view learning," *arXiv preprint arXiv:1304.5634*, 2013.
[18] S. Zhang, C.-C. Liu, W. Li, H. Shen, P. W. Laird, and X. J. Zhou, "Discovery of multi-dimensional modules by integrative analysis of cancer genomic data," *Nucleic Acids Res.*, vol. 40, no. 19, pp. 9379–9391, 2012.
[19] L. Jing, C. Zhang, and M. K. Ng, "SNMFCA: supervised NMF-based image classification and annotation," *IEEE Trans. Image Process.*, vol. 21, no. 11, pp. 4508–4521, 2012.
[20] J. Liu, C. Wang, J. Gao, and J. Han, "Multi-view clustering via joint nonnegative matrix factorization," in *Proc. SIAM Int. Conf. Data Min.* SIAM, 2013, pp. 252–260.
[21] L. Zhang and S. Zhang, "A General Joint Matrix Factorization Framework for Data Integration and its Systematic Algorithmic Exploration," *IEEE Trans. Fuzzy Syst.*, 2019.
[22] ——, "Learning common and specific patterns from data of multiple interrelated biological scenarios with matrix factorization," *Nucleic Acids Res.*, vol. 47, no. 13, pp. 6606–6617, 2019.
[23] C. Zhang and S. Zhang, "Bayesian joint matrix decomposition for data integration with heterogeneous noise," *IEEE Trans. Pattern Anal. Mach. Intell.*, pp. 1–1, 2019.

[24] R. Jin, A. Goswami, and G. Agrawal, "Fast and exact out-of-core and distributed k-means clustering," *Knowl. Inf. Syst.*, vol. 10, no. 1, pp. 17–40, 2006.
[25] B. Bahmani, B. Moseley, A. Vattani, R. Kumar, and S. Vassilvitskii, "Scalable k-means++," *Proc. VLDB Endow.*, vol. 5, no. 7, pp. 622–633, 2012.
[26] Z.-Q. Yu, X.-J. Shi, L. Yan, and W.-J. Li, "Distributed stochastic ADMM for matrix factorization," in *Proc. ACM Int. Conf. Conf. Inf. Knowl. Manag.* ACM, 2014, pp. 1259–1268.
[27] S. Ahn, A. Korattikara, N. Liu, S. Rajan, and M. Welling, "Large-scale distributed Bayesian matrix factorization using stochastic gradient MCMC," in *Proc. ACM SIGKDD Int. Conf. Knowl. Discov. Data Min.* ACM, 2015, pp. 9–18.
[28] X. Qin, P. Blomstedt, E. Leppäaho, P. Parviainen, S. Kaski, J. Davis, E. Fromont, D. Greene, and B. B. Bringmann Xiangju Qin, "Distributed Bayesian matrix factorization with limited communication," *Mach. Learn.*, vol. 108, pp. 1805–1830, 2019.
[29] C. Liu, H.-c. Yang, J. Fan, L.-W. He, and Y.-M. Wang, "Distributed nonnegative matrix factorization for web-scale dyadic data analysis on mapreduce," in *Proc. Int. Conf. World Wide Web.* ACM, 2010, pp. 681–690.
[30] A. R. Benson, J. D. Lee, B. Rajwa, and D. F. Gleich, "Scalable methods for nonnegative matrix factorizations of near-separable tall-and-skinny matrices," in *Adv. Neural Inf. Process. Syst.*, Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2014, pp. 945–953.
[31] R. Zdunek and K. Fonal, "Distributed nonnegative matrix factorization with HALS algorithm on MapReduce," in *Lect. Notes Comput. Sci.*, vol. 10393 LNCS. Springer Verlag, 2017, pp. 211–222.
[32] M. Welling and K. Kurihara, "Bayesian k-means as a "maximization-expectation" algorithm," in *Proc. SIAM Int. Conf. Data Min.*, vol. 2006, 2006, pp. 474–478.
[33] M. N. Schmidt, O. Winther, and L. K. Hansen, "Bayesian non-negative matrix factorization," in *Int. Conf. Indep. Compon. Anal. Signal Sep.* Springer, 2009, pp. 540–547.
[34] A. T. Cemgil, "Bayesian inference for nonnegative matrix factorisation models," *Comput. Intell. Neurosci.*, vol. 2009, pp. 1–17, 2009.
[35] R. Salakhutdinov and A. Mnih, "Bayesian probabilistic matrix factorization using Markov chain Monte Carlo," in *Proc. Int. Conf. Mach. Learn.* ACM, 2008, pp. 880–887.
[36] H. Saddiki, J. McAuliffe, and P. Flaherty, "GLAD: a mixed-membership model for heterogeneous tumor subtype classification," *Bioinformatics*, vol. 31, no. 2, pp. 225–232, jan 2015.
[37] D. Arthur and S. Vassilvitskii, "k-means++: The advantages of careful seeding," in *Proc. Annu. ACM-SIAM Symp. Discret. Algorithms.* Society for Industrial and Applied Mathematics, 2007, pp. 1027–1035.
[38] D. Donoho and V. Stodden, "When does non-negative matrix factorization give a correct decomposition into parts?" in *Adv. Neural Inf. Process. Syst.*, S. Thrun, L. K. Saul, and B. Schölkopf, Eds. MIT Press, 2004, pp. 1141–1148.
[39] A. Beck and M. Teboulle, "A fast iterative shrinkage-thresholding algorithm for linear inverse problems," *SIAM J. Imag. Sciences*, vol. 2, no. 1, pp. 183–202, 2009.
[40] G. Lan, S. Lee, and Y. Zhou, "Communication-efficient algorithms for decentralized and stochastic optimization," *Math. Program.*, 2018.
[41] M. I. Jordan, J. D. Lee, and Y. Yang, "Communication-efficient distributed statistical inference," *J. Am. Stat. Assoc.*, vol. 114, no. 526, pp. 668–681, 2019.
[42] J. Fan, Y. Guo, and K. Wang, "Communication-efficient accurate statistical estimation," *arXiv preprint arXiv:1906.04870*, 2019.
[43] O. Shamir, N. Srebro, and T. Zhang, "Communication-efficient distributed optimization using an approximate newton-type method," in *Int. Conf. Mach. Learn.*, 2014, pp. 1000–1008.
[44] S. Boyd, N. Parikh, E. Chu, B. Peleato, J. Eckstein *et al.*, "Distributed optimization and statistical learning via the alternating direction method of multipliers," *Foundation News and Trends® in Machine learning*, vol. 3, no. 1, pp. 1–122, 2011.
[45] S. Tao, D. Boley, and S. Zhang, "Local linear convergence of ISTA and FISTA on the LASSO problem," *SIAM J. Optim.*, vol. 26, no. 1, pp. 313–336, 2016.
[46] S. Wu, A. Joseph, A. S. Hammonds, S. E. Celniker, B. Yu, and E. Frise, "Stability-driven nonnegative matrix factorization to interpret spatial gene expression and build local gene networks," *Proc. Natl. Acad. Sci.*, vol. 113, no. 16, pp. 4290–4295, 2016.
[47] C.-C. Chang and C.-J. Lin, "LIBSVM: A library for support vector machines," *ACM Trans. Intell. Syst. Technol.*, vol. 2, no. 3, pp. 1–27, 2011.
[48] H. W. Kuhn, "The hungarian method for the assignment problem," *Nav. Res. Logist. Q.*, vol. 2, no. 1–2, pp. 83–97, 1955.
[49] X. Meng, J. Bradley, B. Yavuz, E. Sparks, S. Venkataraman, D. Liu, J. Freeman, D. B. Tsai, M. Amde, S. Owen, and Others, "Mllib: machine learning in Apache Spark," *J. Mach. Learn. Res.*, vol. 17, no. 1, pp. 1235–1241, 2016.
[50] A. Gittens, A. Devarakonda, E. Racah, M. Ringenburg, L. Gerhardt, J. Kottalam, J. Liu, K. Maschhoff, S. Canon, J. Chhugani, and Others, "Matrix factorizations at scale: a comparison of scientific data analytics in Spark and C++ MPI using three case studies," in *IEEE Int. Conf. Big Data.* IEEE, 2016, pp. 204–213.

Chihao Zhang is a PhD candidate in the Academy of Mathematics and Systems Science, Chinese Academy of Sciences. His research interests include data science, data mining and machine learning.

Yang Yang Yang Yang is a Master strudent in the School of Software, Yunnan University. His research interests include distributed computing and machine learning.

Wei Zhou received the PhD degree from the Chinese Academy of Science. Now he is a full professor at Software School of Yunnan University. His current research interests include distributed data intensive computing and bioinformatics.

Shihua Zhang received the PhD degree in applied mathematics and bioinformatics from the Academy of Mathematics and Systems Science, Chinese Academy of Sciences in 2008 with the highest honor. He joined the same institute as an Assistant Professor in 2008, and is currently Professor. His research interests are mainly in bioinformatics and computational biology, data mining, pattern recognition and machine learning. He has won various awards and honors including Ten Thousand Talent Program—Young top-notch talent (2018), NSFC for Excellent Young Scholars (2014), Outstanding Young Scientist Program of CAS (2014) and Youth Science and Technology Award of China (2013). Now he serves as an Editorial Board Member of BMC Genomics, Frontiers in Genetics, and so on. He is a member of the IEEE, ISCB and SIAM.