Hindawi
Journal of Electrical and Computer Engineering
Volume 2018, Article ID 6270816, 10 pages
https://doi.org/10.1155/2018/6270816

Research Article

Parallel Nonnegative Matrix Factorization with Manifold Regularization

Fudong Liu, Zheng Shan, and Yihang Chen

State Key Laboratory of Mathematical Engineering and Advanced Computing, Zhengzhou, Henan 450001, China

Correspondence should be addressed to Fudong Liu; lwfydy@126.com

Received 12 November 2017; Revised 7 February 2018; Accepted 15 March 2018; Published 2 May 2018

Academic Editor: Tongliang Liu

Nonnegative matrix factorization (NMF) decomposes a high-dimensional nonnegative matrix into the product of two reduced dimensional nonnegative matrices. However, conventional NMF neither qualifies large-scale datasets as it maintains all data in memory nor preserves the geometrical structure of data which is needed in some practical tasks. In this paper, we propose a parallel NMF with manifold regularization method (PNMF-M) to overcome the aforementioned deficiencies by parallelizing the manifold regularized NMF on distributed computing system. In particular, PNMF-M distributes both data samples and factor matrices to multiple computing nodes instead of loading the whole dataset in a single node and updates both factor matrices locally on each node. In this way, PNMF-M succeeds to resolve the pressure of memory consumption for large-scale datasets and to speed up the computation by parallelization. For constructing the adjacency matrix in manifold regularization, we propose a two-step distributed graph construction method, which is proved to be equivalent to the batch construction method. Experimental results on popular text corpora and image datasets demonstrate that PNMF-M significantly improves both scalability and time efficiency of conventional NMF thanks to the parallelization on distributed computing system; meanwhile it significantly enhances the representation ability of conventional NMF thanks to the incorporated manifold regularization.

1. Introduction

Data representation is a fundamental problem in data analysis. A good representation typically uncovers the latent structure of a dataset by reducing the dimensionality of data. Several methods including principal component analysis (PCA), linear discriminant analysis (LDA), and vector quantization (VQ) have addressed this issue. Recently, nonnegative matrix factorization (NMF) [1] incorporates nonnegativity constraint to obtain parts-based representation of data, and thus it has been widely applied in many applications, such as document clustering [2, 3], image recognition [4, 5], audio processing [6], and video processing [7].

However, conventional NMF suffers from a few deficiencies: (1) conventional NMF usually works in batch mode and requires all data to reside in memory, and this leads to tremendous storage overhead as the increase of the data samples and (2) conventional NMF ignores the geometrical structure embedded in data and causes unsatisfactory representation ability. To overcome the first deficiency, either parallel or distributed algorithms have been proposed for NMF to fit for large-scale datasets. Kanjani [8] utilized multithreading to develop a parallel NMF (PNMF) based on multicore machine. Robila and Maciak [9] introduced two thread-level parallel versions for traditional multiplicative solution and adaptive projected gradient method. However, their methods are prohibited for large-scale datasets due to the memory limitation of a single computer. Liu et al. [10] proposed a distributed NMF (DNMF) to analyze large-scale web dyadic data and verified its effectiveness on distributed computing systems. Dong et al. [11] also attempted to design a PNMF based on the distributed memory platform with the message passing interface library. Although all the above parallel NMF algorithms achieve a considerable speedup in terms of scalability, they cannot consider the geometric structure in dataset. To overcome the second deficiency, Cai et al. [12] proposed graph-regularized NMF (GNMF) which extended conventional NMF by constructing an affinity graph [13] to encode the geometrical information of data and enhanced representation ability. Gu et al. [14] further

extended GNMF to avoid trivial solution and scale transfer problems by imposing a normalized cut-like constraint on the cluster assignment matrix. Lu et al. [15] incorporated manifold regularization into NMF for hyperspectral unmixing and obtained desirable unmixing performance. These improved algorithms get better representation ability but work inefficiently for large-scale datasets. Liu et al. [16] introduced the geometric structure of data into incremental NMF [17, 18] and utilized two efficient sparse approximations, buffering and random projected tree, to process large-scale datasets. Yu et al. [19] also presented an incremental GNMF algorithm to improve scalability. But these algorithms only performed well for incremental or streaming datasets and could not deal with large-scale batch datasets. In addition, Guan et al. [20] and Liu et al. [21], respectively, introduce Manhattan distance and large-cone penalty for NMF to improve representation and generalization ability.

In conclusion, none of the above works can simultaneously overcome both deficiencies due to the great computation for calculating the decomposition and the storage requirement of the adjacency matrix. In this paper, we take the best of advantages of parallel NMF and manifold regularized NMF and design a parallel NMF with manifold regularization method (PNMF-M) by parallelizing manifold regularized NMF on distributed computing systems. In particular, PNMF-M distributes both data samples and factor matrices to multiple computing nodes in a balanced way and parallelizes the update for both factor matrices locally on each node. Since the graph construction is the bottleneck of the computation of PNMF-M, we adopt a two-step distributed graph construction method to compute the adjacency matrix and obtain an adjacent graph equivalent to that constructed in batch mode. Experimental results on popular text corpora and image datasets show that PNMF-M not only outperforms conventional NMF in terms of both scalability and time efficiency by parallelization, but also significantly enhances the representation ability of conventional NMF by incorporating manifold regularization.

2. Related Work

This section gives a brief review of PNMF, GNMF, and their corresponding popular algorithms. Details are as follows.

2.1. PNMF. Due to working in batch mode, conventional NMF requires to process masses of large matrix operations, which is inevitably confronted with high time overhead. Parallel processing is often an efficient method to speed up computing intensive algorithms. In general, the parallelism of the algorithms is embodied in two aspects. The first aspect is computation flow, but unfortunately conventional NMF is highly correlated among the computing steps and could not be decomposed into uncorrelated tasks. So the second aspect, data structure, is taken into account. A feasible plan is to divide the data into small blocks and assign them to different processing units.

Robila and Maciak [9] implement a common example of parallel NMF. Let the input data matrix $X = [x_1, \ldots, x_n] \in \mathbb{R}^{m \times n}$, the basis matrix $U \in \mathbb{R}^{m \times k}$, and the coefficient matrix $V \in \mathbb{R}^{k \times n}$. Given that there are p processing units, both k and n are exactly divisible by p. They divide the above three matrices into p equally sized submatrices by column; that is, $X = [X_1, \ldots, X_p]$, $U = [U_1, \ldots, U_p]$, and $V = [V_1, \ldots, V_p]$. They still minimize the cost function of conventional NMF and obtain the update rules for V and U as follows:

$$\left(V_s^{t+1}\right)_{rj} = \left(V_s^t\right)_{rj} \frac{\left(U^{t\mathrm{T}} X_s\right)_{rj}}{\left(U^{t\mathrm{T}} U^t V_s^t\right)_{rj}}$$
$$\left(U_s^{t+1}\right)_{ir} = \left(U_s^t\right)_{ir} \frac{\left(X V_s^{t+1\mathrm{T}}\right)_{ir}}{\left(U^t V_s^{t+1} V_s^{t+1\mathrm{T}}\right)_{ir}}, \quad (1)$$

where t denotes the iteration number, $s = 1, \ldots, p$, $i = (s-1)*(k/p)+1, \ldots, s*(k/p)$, $j = (s-1)*(n/p)+1, \ldots, s*(n/p)$, and $r = 1, \ldots, k$.

By parallelizing conventional NMF, PNMF can achieve the same representation ability, while reducing the time complexity.

2.2. GNMF. In common, conventional NMF performs the factorization in the Euclidean space, which cannot get satisfactory solution. Based on manifold assumption [22], GNMF embeds the intrinsic geometric structure of the data space in conventional NMF to overcome the above defect.

One of the representative algorithms for GNMF is proposed by Cai et al. [12]. They define the graph weight matrix $W \in \mathbb{R}^{n \times n}$, which is obtained by computing the weight relationship between any two data samples in input data matrix X. Furthermore, another two matrices are derived from W. One is the diagonal matrix D whose entries are row or column sum of W, that is, $D_{ii} = \sum_{j=1}^{n} W_{ij} = \sum_{i=1}^{n} W_{ij}$, and the other is $L = D - W$, which is called graph Laplacian. They utilize Lagrangian multiplier method to solve the above problem and derive the following update rules:

$$\left(V^{t+1}\right)_{rj} = \left(V^t\right)_{rj} \frac{\left(U^{t\mathrm{T}} X + \lambda V^t W\right)_{rj}}{\left(U^{t\mathrm{T}} U^t V^t + \lambda V^t D\right)_{rj}} \quad (2)$$

$$\left(U^{t+1}\right)_{ir} = \left(U^t\right)_{ir} \frac{\left(X V^{t+1\mathrm{T}}\right)_{ir}}{\left(U^t V^{t+1} V^{t+1\mathrm{T}}\right)_{ir}}, \quad (3)$$

where $\lambda \geq 0$ is the regularization parameter, t denotes the iteration number, $i = 1, \ldots, m$, $j = 1, \ldots, n$, and $r = 1, \ldots, k$.

By incorporating the geometric structure, GNMF can significantly improve representation ability in comparison to conventional NMF.

3. Parallel Manifold Regularized Nonnegative Matrix Factorization

In conventional GNMF, the intrinsic low dimensional manifold structure embedded in the dataset is expressed by n by n graph weight matrix W. From the update rule (2), we can

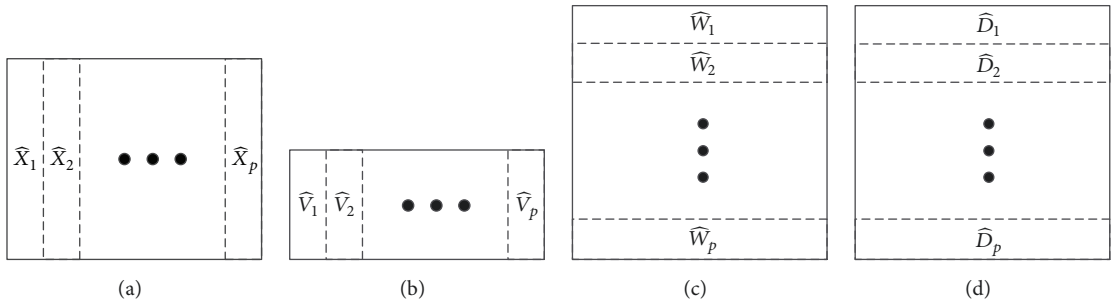

Figure 1: Data partitioning strategy for updating V in PNMF-M.

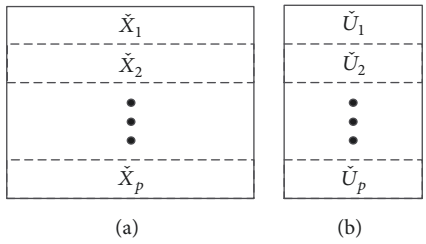

Figure 2: Data partitioning strategy for updating U in PNMF-M.

easily see that the manifold terms in conventional GNMF give rise to a lot of additional computations compared with NMF. Therefore, we describe the proposed PNMF-M in this section, which improves this issue by distributing both data samples and factor matrices to multiple computing nodes while parallelizing the update for both factor matrices and manifold construction.

3.1. Data Partition and Distributed Computing. We still assume $X \in \mathbb{R}^{m\times n}$, $U \in \mathbb{R}^{m\times k}$, $V \in \mathbb{R}^{k\times n}$, $W \in \mathbb{R}^{n\times n}$, and p as the input data matrix, basis matrix, coefficient matrix, graph weight matrix, and number of processors, respectively.

PNMF-M divides X and V into p blocks in the column direction, namely, $X = [\widehat{X}_1, \widehat{X}_2, \ldots, \widehat{X}_p]$, $V = [\widehat{V}_1, \widehat{V}_2, \ldots, \widehat{V}_p]$, as shown in Figures 1(a) and 1(b); thus each of the first $p-1$ blocks consists of $\lceil n/p \rceil$ columns and the last one consists of $n - \lceil n/p \rceil \times (p-1)$ columns. Similarly, PNMF-M divides X and U into p blocks in the row direction as shown in Figures 2(a) and 2(b); that is, $X = [\check{X}_1^{\mathrm{T}}, \check{X}_2^{\mathrm{T}}, \ldots, \check{X}_p^{\mathrm{T}}]^{\mathrm{T}}$, $U = [\check{U}_1^{\mathrm{T}}, \check{U}_2^{\mathrm{T}}, \ldots, \check{U}_p^{\mathrm{T}}]^{\mathrm{T}}$, so each of the first $p-1$ blocks consists of $\lceil m/p \rceil$ rows and the last one consists of $m - \lceil m/p \rceil \times (p-1)$ rows.

As the scale of W is much larger than those of X, U, and V, the weight matrix W should be distributed across p processors. The same is true for D. To this end, PNMF-M divides W and D into p blocks in the row direction, namely, $W = [\widehat{W}_1^{\mathrm{T}}, \widehat{W}_2^{\mathrm{T}}, \ldots, \widehat{W}_p^{\mathrm{T}}]^{\mathrm{T}}$, $D = [\widehat{D}_1^{\mathrm{T}}, \widehat{D}_2^{\mathrm{T}}, \ldots, \widehat{D}_p^{\mathrm{T}}]^{\mathrm{T}}$. In this way, each of the first $p-1$ blocks is a $\lceil n/p \rceil \times n$-dimensional matrix and the last one is a $(n - \lceil n/p \rceil \times (p-1)) \times n$-dimensional matrix, as described in Figures 1(c) and 1(d).

According to [12], to take the advantage of manifold regularization, PNMF-M optimizes the following objective:

$$F = \|X - UV\|_F^2 + \lambda \mathrm{Tr}\left(VLV^{\mathrm{T}}\right), \tag{4}$$

where λ denotes the regularization parameter and $L = D - W$.

Since each update for U (or V) needs the latest value of V (or U), it is difficult to distribute the updates for them on p processors. According to the aforementioned data partitioning strategy, PNMF-M reformulates the multiplicative update rules in [12], at the tth iteration, as follows:

$$\left(\widehat{V}_s^{t+1}\right)_{rj} = \left(\widehat{V}_s^{t}\right)_{rj} \frac{\left(U^{t\,\mathrm{T}} \widehat{X}_s\right)_{rj} + \lambda \sum_{f=1}^{p} \left(\widehat{V}_f^t \widehat{W}_f\right)_{rj}}{\left(U^{t\,\mathrm{T}} U^t \widehat{V}_s^t\right)_{rj} + \lambda \sum_{f=1}^{p} \left(\widehat{V}_f^t \widehat{D}_f\right)_{rj}} \tag{5}$$

$$\left(\check{U}_s^{t+1}\right)_{ir} = \left(\check{U}_s^{t}\right)_{ir} \frac{\left(\check{X}_s V^{t+1\,\mathrm{T}}\right)_{ir}}{\left(\check{U}_s^t V^{t+1} V^{t+1\,\mathrm{T}}\right)_{ir}} \tag{6}$$

where t denotes the iteration number, $s = 1, \ldots, p$, $i = (s-1) \times \lceil m/p \rceil + 1, \ldots, s \times \lceil m/p \rceil$, $j = (s-1) \times \lceil n/p \rceil + 1, \ldots, s \times \lceil n/p \rceil$ (note that when $s = p$, $i = (p-1) \times \lceil m/p \rceil + 1, \ldots, m$, $j = (p-1) \times \lceil n/p \rceil + 1, \ldots, n$), with $r = 1, \ldots, k$.

In each iteration round, PNMF-M updates each block of V and U locally by (5) and (6), respectively, and inserts the reduction for the whole factors among processors between (5) and (6). Since our algorithm is developed on a distributed computing system with 40 Gb/s Infiniband interconnection network whose communication overloads among processors are relatively low, this updating strategy significantly speeds up the calculation of manifold regularized NMF by the distributed computing.

3.2. Manifold Construction. Besides data partition and distributed computing, manifold construction is another key factor in PNMF-M. In the update rule (5), updating $\widehat{V}_s$ in the

Input: data matrix $X \in \mathbb{R}^{m\times n}$, reduced dimensionality $k \in \mathbb{N}$, regularization parameter $\lambda \in \mathbb{R}$, tolerance $\tau \in \mathbb{R}$, parallelism $p \in \mathbb{N}$
Output: basis matrix $U \in \mathbb{R}^{m\times k}$, coefficient matrix $V \in \mathbb{R}^{k\times n}$
(1) Assign $\widehat{X}_i$, $\check{X}_i$, $\check{U}_i$, and $\widehat{V}_i$ to P_i as described in Figures 1 and 2
(2) Compute idx(i) from $\widehat{X}_i$
(3) Send $\widehat{X}_i$ to the other $p-1$ processes P_j
(4) Receive $\widehat{X}_j$ from the other $p-1$ processes P_j to obtain idx(j)
(5) Merge all p index sets to construct $\widehat{W}_i$
(6) **repeat**
(7) Apply (5) to update $\widehat{V}_i^{t+1}$
(8) Send $\widehat{V}_i^{t+1}$ to the other $p-1$ processes P_j
(9) Receive $\widehat{V}_j^{t+1}$ from the other $p-1$ processes P_j to constitute V^{t+1}
(10) Apply (6) to update $\check{U}_i^{t+1}$
(11) Send $\check{U}_i^{t+1}$ to the other $p-1$ processes P_j
(12) Receive $\check{U}_j^{t+1}$ from the other $p-1$ processes P_j to constitute U^{t+1}
(13) **until** convergence
(14) **return** $\check{U}_i$ and $\widehat{V}_i$

Algorithm 1: PNMF-M procedure.

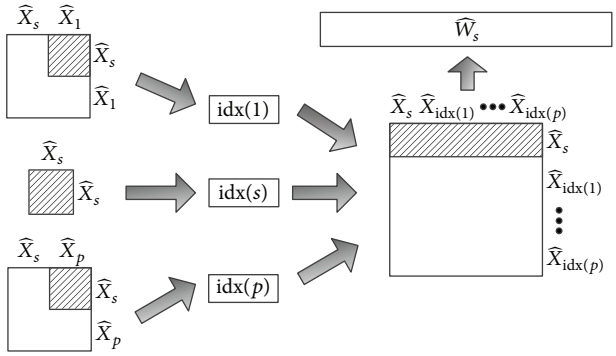

Figure 3: Manifold construction in PNMF-M.

sth processing unit requires the products of $\widehat{V}_f$ and $\widehat{W}_f$ in all p processing units. However, directly constructing the global adjacency graph from the input data matrix X costs $\Theta(n^2)$ time in manifold learning and heavily prohibits manifold regularized NMF for large-scale datasets. In this subsection, we design an efficient construction method by means of the communications between different processing units.

In Figure 1(c), we can see that the sth processor simply holds $\widehat{W}_s$ which is a part of the global manifold W and represents the weight relationship between data samples in $\widehat{X}_s$ and all the data samples in X. Originally, it needs the entire data matrix X to construct $\widehat{W}_s$, and the tremendous storage requirement becomes the bottleneck which restricts the scalability of the algorithm. It is therefore critical to construct the manifold in an efficient way. To overcome this drawback, in this paper, we design a two-step manifold construction method by means of high-throughput interconnection network.

Figure 3 depicts the details of the construction method. Without loss of generality, we take the sth processor P_s as an example to demonstrate both steps. In the first step, PNMF-M first constructs locally an adjacency subgraph from $\widehat{X}_s$ in P_s relative to the adjacency graph constructed from X and thus obtains an index set idx(s) which contains local neighbor indices of each data sample in $\widehat{X}_s$. Then, P_s utilizes high-throughput interconnection network to take turns to receive data samples from the other $s-1$ remote processors. When $\widehat{X}_f$ comes, where $f = 1, 2, \ldots, p$ except s, PNMF-M reconstructs an adjacency subgraph from $\widehat{X}_s$ and $\widehat{X}_f$ and obtains an index set idx(f) which contains remote neighbor indices of each data sample in $\widehat{X}_s$. In the second step, PNMF-M computes the weight relationship between $\widehat{X}_s$ and data sample matrix $\widehat{X}_{\text{idx}(f)}$ indicated by each index set idx(f) and picks out the final neighbor indices of each data sample in $\widehat{X}_s$ to construct a manifold $\widehat{W}_s$ in P_s.

In our method, we select k-nearest neighbors (k-NN) mode in [22] to construct the graph, and thus the proposed construction method can be proved to get a manifold equivalent to the batch construction method. Since each processor can discard data sample matrix received from remote processors after constructing adjacency subgraph, PNMF-M greatly reduces the storage overhead and improves the scalability.

3.3. PNMF-M Algorithm. We are now ready to alternatively apply the update rules (5) and (6) to solve PNMF-M. Algorithm 1 shows the pseudocode on distributed computing systems. Given that p processes exist in PNMF-M, Algorithm 1 first partitions data samples and factor matrices according to Figures 1 and 2 and assigns the associated submatrices to each process P_i (Statement (1)). Then, P_i begins to locally construct its private weight matrix in parallel. It first computes idx(i) from $\widehat{X}_i$ in the local process (Statement (2)) and then receives $\widehat{X}_j$ from the other $p-1$ processes P_j to obtain idx(j) (Statement (4)). Next, it merges all p index sets to construct $\widehat{W}_i$ (Statement (5)). Then, P_i alternatively employs the update

Table 1: Statistics of benchmark datasets.

	Reuters	TDT2	MNIST	COIL
Number of samples	21578	64527	70000	7200
Number of samples used	8067	9394	10000	7200
Number of attributes	18933	36771	784	1024
Number of clusters	135	100	10	100
Number of clusters used	30	30	10	100
Max. cluster size	3713	1844	1135	72
Min. cluster size	18	52	892	72
Med. cluster size	45	131	1009	72
Avg. cluster size	269	313	1000	72

rules (5) and (6) to obtain its private coefficient submatrix $\widehat{V}_i$ (Statement (7)) and basis submatrix $\check{U}_i$ (Statement (10)) until the termination condition for the iteration is satisfied.

Notice that updating $\widehat{V}_i$ requires the basis matrix U which consists of p submatrices $\check{U}_j$ distributed across each process P_j, where $j = 1, 2, \ldots, p$, as shown in (5). Hence, Algorithm 1 has to insert the interprocess communication to implement the reduction for U before updating $\widehat{V}_i$ (Statement (11)-(12)). Similarly, Algorithm 1 also requires increasing the interprocess communication to achieve the reduction for V before updating $\widehat{U}_i$ (Statements (8) and (9)).

4. Experimental Evaluation

In this section, we verify the proposed PNMF-M by comparing it with PNMF in [11] and GNMF in [12], which incorporate parallel computing and manifold regularization into NMF, respectively.

4.1. Datasets and Evaluation Metrics. We choose Reuters and TDT2 text corpora widely used for document clustering and MNIST and COIL100 image databases used for pattern recognition and machine learning as our benchmark datasets. Table 1 lists their detailed statistics.

In this experiment, we utilize canonical k-means clustering method to transform the coefficient matrix obtained by our algorithm into the labels of data samples, then compare them with the labels provided by the benchmark datasets, and finally use the accuracy (AC) and the normalized mutual information (NMI) to evaluate the clustering performance of each tested algorithm. For the way to compute AC and NMI, Xu et al. [2] and Cai et al. [12] have given detailed description, so we need not explain them here.

To evaluate the scalability of our proposed algorithm, we record the execution time of each tested algorithm and utilize the speedup (SP) and time cost (TC) as metrics, where SP denotes the speedup obtained by PNMF-M relative to GNMF and TC denotes the time cost introduced by PNMF-M relative to PNMF; that is,

$$\begin{aligned} \mathrm{SP} &= \frac{\mathrm{Time}_{\mathrm{GNMF}}}{\mathrm{Time}_{\mathrm{PNMF\text{-}M}}} \\ \mathrm{TC} &= \frac{\mathrm{Time}_{\mathrm{PNMF\text{-}M}}}{\mathrm{Time}_{\mathrm{PNMF}}}. \end{aligned} \tag{7}$$

Table 2: Experimental parameters for clustering performance evaluation.

Experimental parameter	Value
Regularization parameter (λ)	100
Tolerance (τ)	10^{-4}
Neighbor mode	k-NN ($k = 5$)
Parallelism (p)	ceil(n/1000)

4.2. Clustering Performance Comparisons. We collect data samples in multiple clusters randomly selected from the benchmark datasets and provide them and the corresponding cluster number to each tested algorithm. All the tested algorithms are run 50 times on different data subsets each of which consists of some part of clusters randomly selected from the whole clusters. We compute the mean of them for each given cluster number as the final clustering performance. At the same time, we also compute the mean of SP and TC to verify the effectiveness of PNMF-M.

Table 2 lists the values of main experimental parameters for clustering performance evaluation. In our experiment, we set regularization parameter to 100 and the tolerance to 10^{-4} as the termination condition of each iteration. In addition, we employ k-NN definition to construct the graph weight matrix and set k to 5. For parallel algorithms, we adaptively select the parallelism p according to the number of samples n and set $p = \mathrm{ceil}(n)$, where ceil(x) means rounding up x to an integer.

Figures 4–7 report AC and NMI versus the cluster number on four benchmark datasets, respectively. Figure 8 reports SP and TC of our algorithm versus the cluster number on four benchmark datasets.

Through comparative analysis, we can obtain the following observations:

(1) Since the manifold constructed in PNMF-M is equivalent to that constructed in GNMF, both obtain the same clustering performance on all four benchmark datasets.

(2) PNMF-M gets better clustering performance than PNMF in terms of both AC and NMI, because PNMF-M incorporates the intrinsic geometric structure of data and obtains the same clustering performance as GNMF, while PNMF ignores the manifold structure of the dataset.

(3) GNMF directly constructs the manifold from the entire data matrix in a single processing node, which incurs

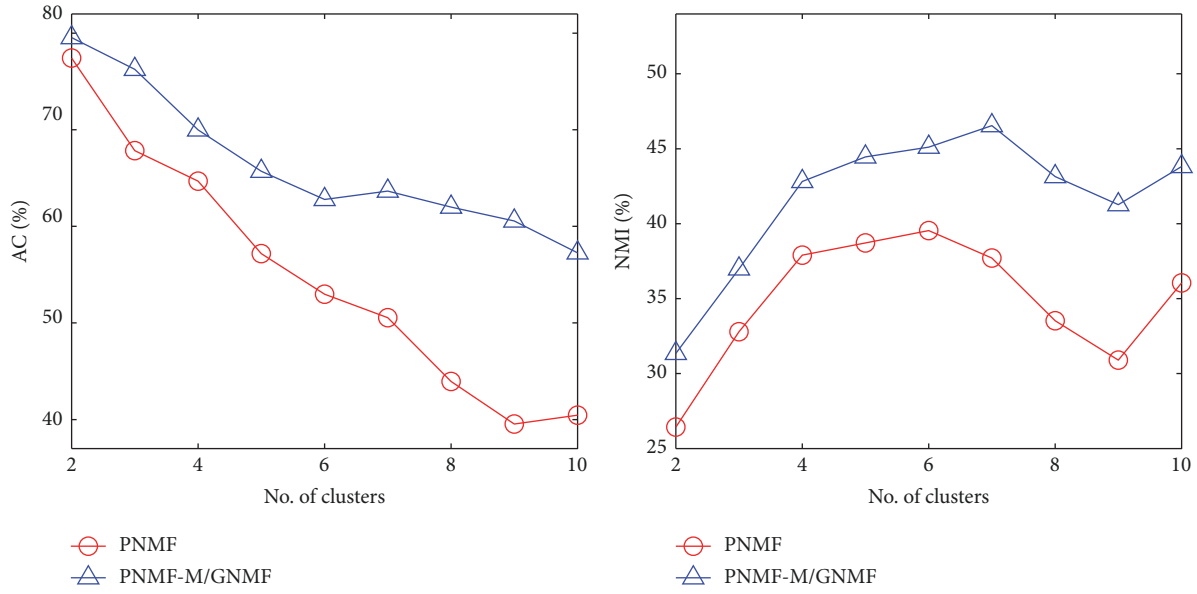

Figure 4: AC and NMI on Reuters.

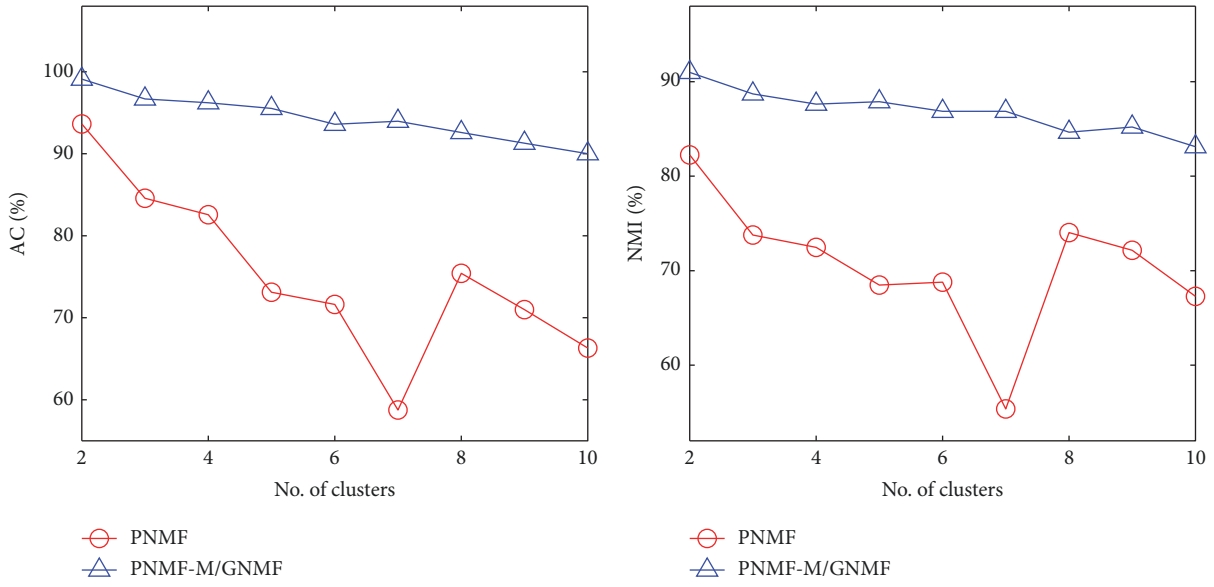

Figure 5: AC and NMI on TDT2.

tremendous computation and storage overhead. In contrast, PNMF-M designs a two-step distributed method to accelerate the manifold construction. In addition, PNMF-M parallelizes the update procedure to eliminate large size matrix operations in the update rules and further reduces the computation and storage overhead. According to Figure 8, PNMF-M acquires about 4–10 times speedup on four benchmark datasets relative to GNMF.

(4) Compared with PNMF, PNMF-M requires to construct the manifold in parallel and introduce the manifold term into the update rule for the coefficient matrix, which inevitably increases the computation overhead of the algorithm to some extent. PNMF-M pays about 2–4 times cost on four benchmark datasets relative to PNMF, as shown in Figure 8.

4.3. Scalability Comparisons. We collect increasing scale data subsets in all the clusters randomly selected from the benchmark datasets as the input datasets of each tested algorithm, run them in the same hardware configuration and software environment as listed in Table 3, and verify the scalability by comparing their execution time and evaluating SP achieved by PNMF-M. In our experiments, we set parallelism to 2 (PNMF-M2), 4 (PNMF-M4), 8 (PNMF-M8), and 16 (PNMF-M16) for PNMF-M, respectively.

Figure 9 reports SP achieved by PNMF-M with different parallelism on four benchmark datasets relative to GNMF. We can draw the conclusions from it as follows:

(1) When the number of input data samples is small (below about $5 * 10^3$), SP obtained by PNMF-M is less than the corresponding parallelism. This is because all processing

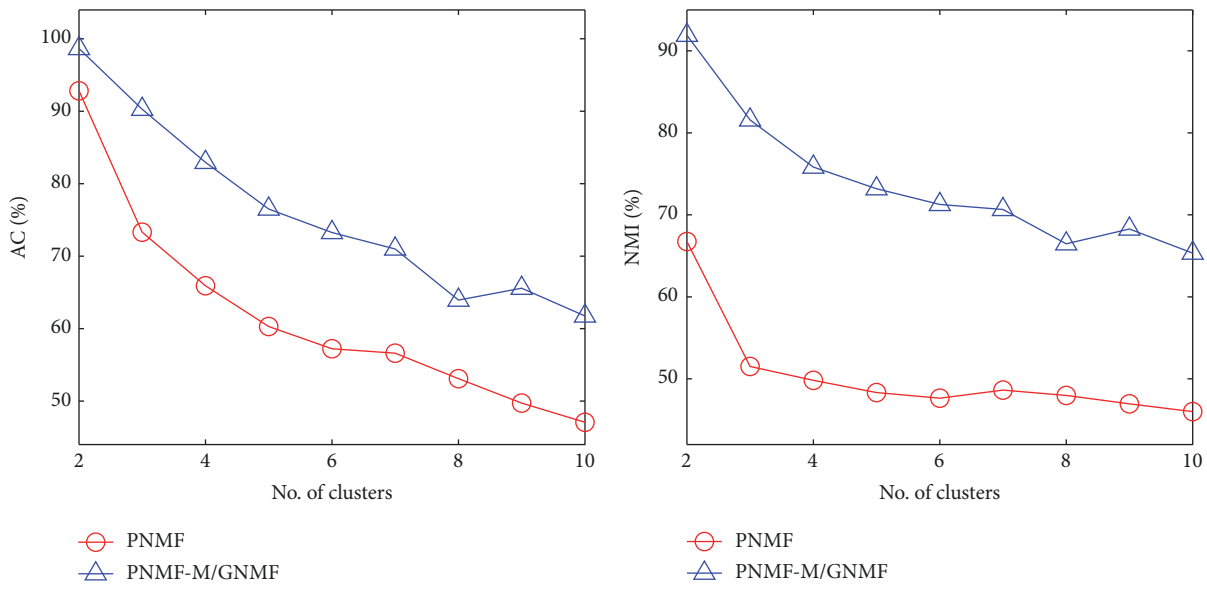

Figure 6: AC and NMI on MNIST.

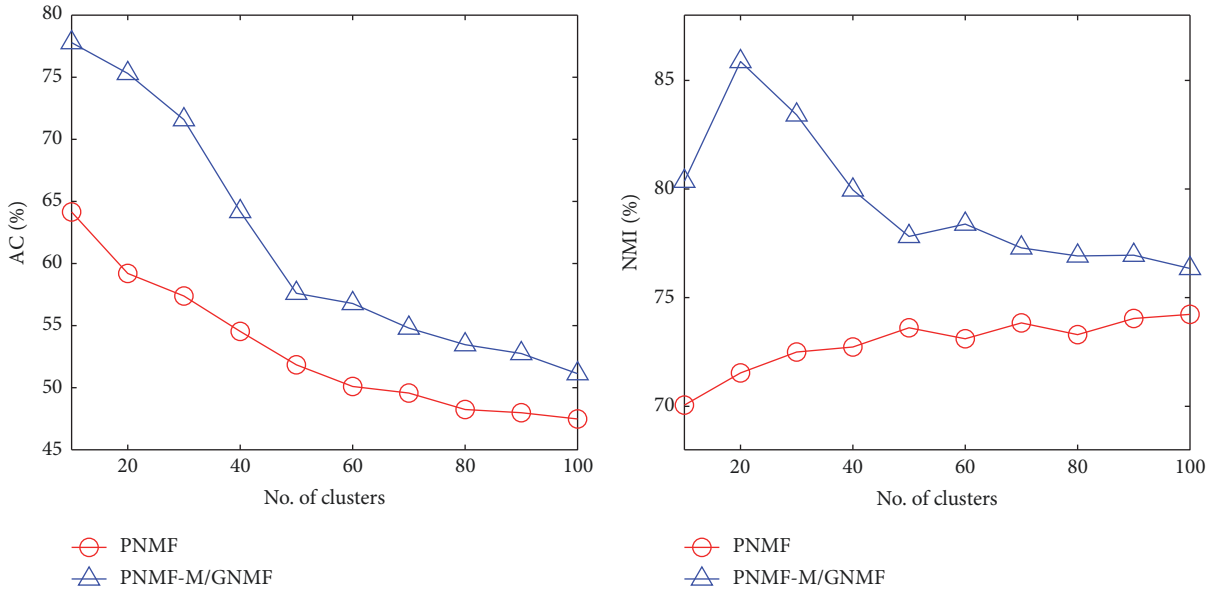

Figure 7: AC and NMI on COIL100.

Table 3: Experimental environment for scalability evaluation.

Number of nodes	16
Memory capacity on a single node	16 TB
Maximum number of threads	24
Operating system	Ubuntu
Interconnection network	Infiniband
Peak bandwidth of network	40 Gb/s

nodes need to communicate with each other to construct the manifold during the execution of PNMF-M, which introduces communication latency into parallel computation.

(2) When the number of input data samples is large (above about $5 * 10^3$), SP achieved by PNMF-M is more than the corresponding parallelism. The reason is that all the input data samples reside in external storage before the execution of each tested algorithm. But GNMF requires all data to be loaded into memory during its execution, which goes beyond the memory capacity in a single processing node and has to load large amounts of data samples by virtual memory. To address the problem, PNMF-M divides the input data into multiple subsets and stores them in different processing node, which reduces memory access overhead caused by loading data.

(3) In the case of small input data samples, PNMF-M achieves better SP on MNIST and COIL100 than on Reuters

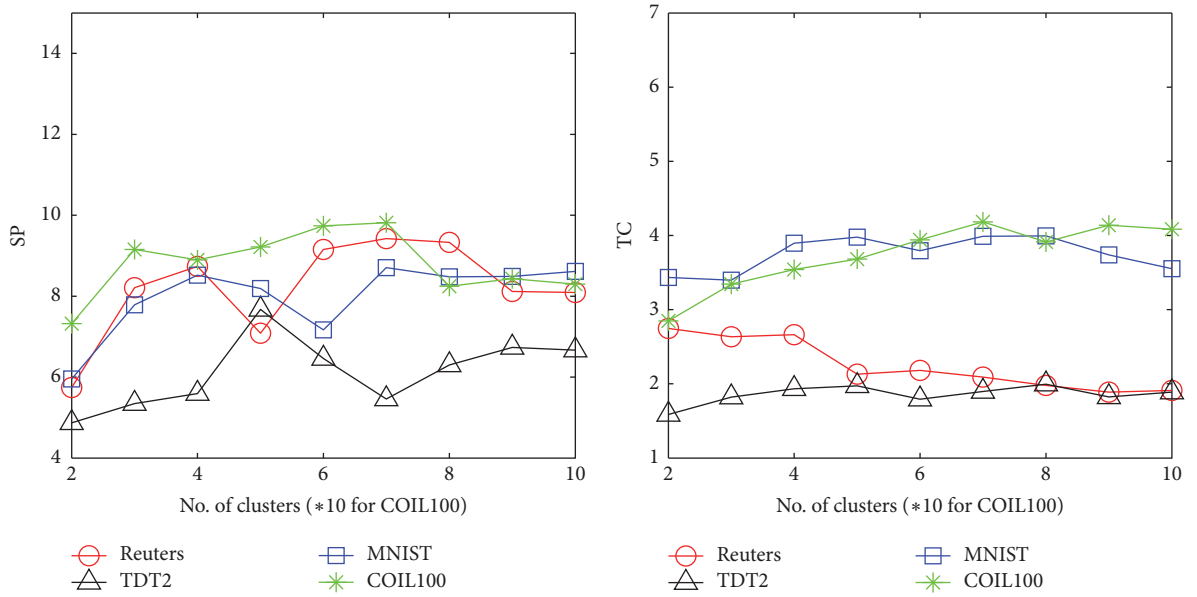

Figure 8: SP and TC on four benchmark datasets.

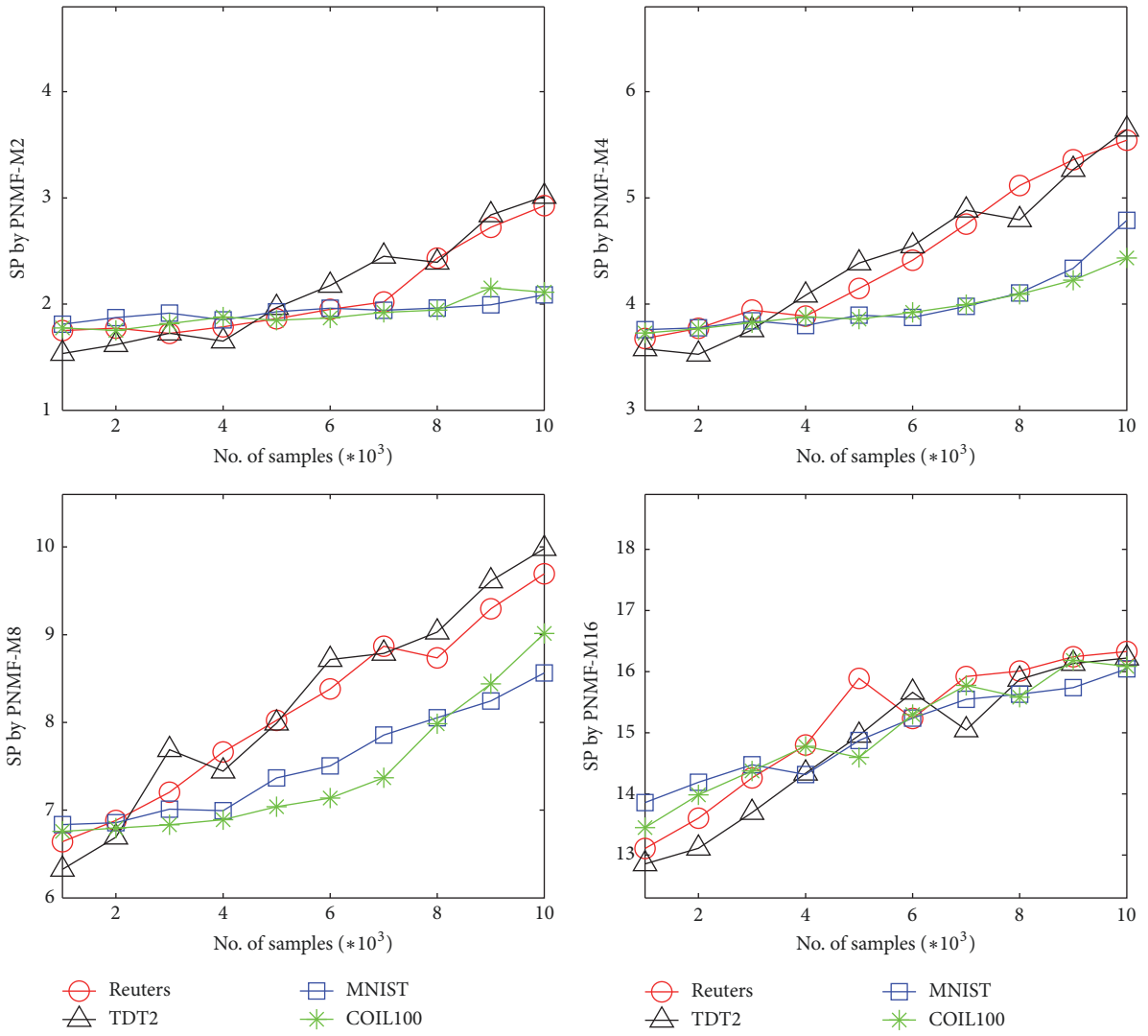

Figure 9: Speedup by PNMF-M on four benchmark datasets.

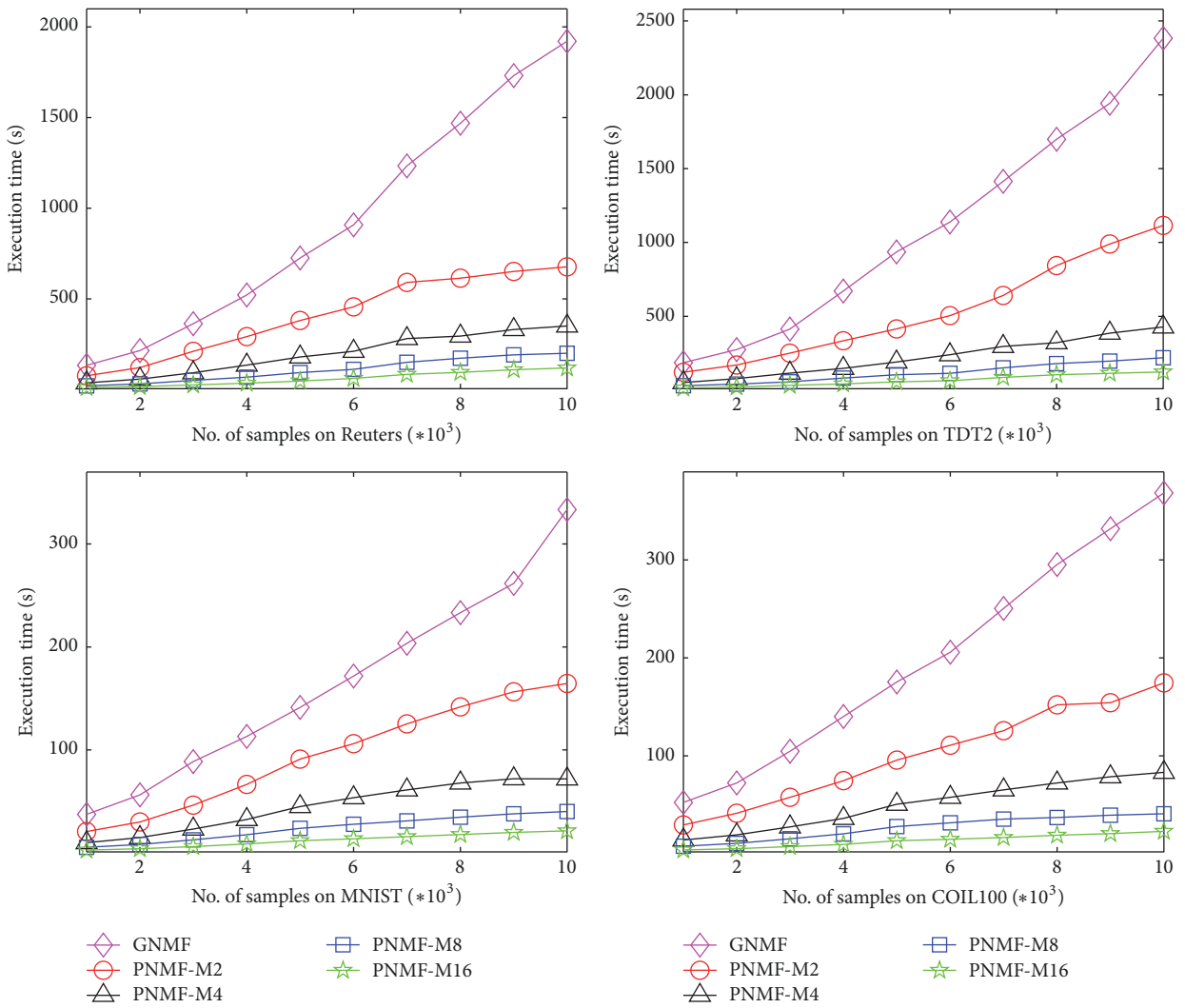

Figure 10: Execution time on four benchmark datasets.

and TDT2 when parallelism for PNMF-M is equal to 2, 4, and 8. But the result is the opposite with the increase of the number of input data samples. The main reason for this result is twofold: one is based on the two analyses mentioned in (1) and (2) and the other is that memory and communication overhead for a single data sample in Reuters and TDT2 are larger because the dimensionality of data in them is higher than that in MNIST and COIL100. But when parallelism for PNMF-M increases to 16, the overhead caused by high dimensionality is spread over each processing node. Hence, SP obtained by PNMF-M16 has little difference among four benchmark datasets.

In addition, we also compare the execution time of all the tested algorithms on four benchmark datasets, as shown in Figure 10. It is obvious that the increasingly widening gap of the execution time exists between serial algorithm (GNMF) and parallel algorithms (PNMF-M2, PNMF-M4, PNMF-M8, and PNMF-M16). It is because GNMF involves lots of large size matrix operations during the manifold construction and updating procedure, and the time consumed by them grows nonlinearly as the number of data samples and attributes increase, while PNMF-M transforms them into relatively small size matrix operations and utilizes multiple processing nodes to perform them in parallel. Hence, PNMF-M achieves better scalability than GNMF.

5. Conclusion

In this paper, we proposed a parallel nonnegative matrix factorization with regularization method (PNMF-M) which introduced manifold regularization into conventional NMF and parallelized it on distributed computing systems. In PNMF-M, besides the balanced data partition strategy, we conducted the update procedure in parallel. In particular, we presented a two-step distributed graph construction method and obtained a manifold equivalent to that constructed in batch mode. Experimental results showed that PNMF-M not only outperforms conventional NMF in terms of both scalability and time efficiency thanks to the parallelization, but also significantly enhances the representation ability of conventional NMF thanks to the manifold regularization.

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

References

[1] 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, 1999.

[2] W. Xu, X. Liu, and Y. Gong, *Document clustering based on non-negative matrix factorization*, 2003, SIGIR, 2003.

[3] F. Shahnaz, M. W. Berry, V. P. Pauca, and R. J. Plemmons, "Document clustering using nonnegative matrix factorization," *Information Processing & Management*, vol. 42, no. 2, pp. 373–386, 2006.

[4] L. Miao and H. Qi, "Endmember extraction from highly mixed data using minimum volume constrained nonnegative matrix factorization," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 45, no. 3, pp. 765–777, 2007.

[5] B. Shen and L. Si, "Non-negative matrix factorization clustering on multiple manifolds," in *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence*, p. 575, 2010.

[6] R. Rui and C.-C. Bao, "Projective non-negative matrix factorization with Bregman divergence for musical instrument classification," in *Proceedings of the 2012 2nd IEEE International Conference on Signal Processing, Communications and Computing, ICSPCC 2012*, pp. 415–418, China, August 2012.

[7] O. Jenkins and M. Mataric, "Deriving action and behavior primitives from human motion data," in *Proceedings of the IROS 2002: IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 2551–2556, Lausanne, Switzerland.

[8] K. Kanjani, "Parallel non-negative matrix factorization for document clustering," Tech. Rep., Texas AM University, 2007.

[9] S. A. Robila and L. G. Maciak, "Considerations on parallelizing nonnegative matrix factorization for hyperspectral data unmixing," *IEEE Geoscience and Remote Sensing Letters*, vol. 6, no. 1, pp. 57–61, 2009.

[10] C. Liu, H. Yang, J. Fan, L. He, and Y. Wang, "Distributed nonnegative matrix factorization for web-scale dyadic data analysis on MapReduce," in *WWW 2010*, pp. 681–690, USA.

[11] C. Dong, H. Zhao, and W. Wang, "Parallel nonnegative matrix factorization algorithm on the distributed memory platform," *International Journal of Parallel Programming*, vol. 38, no. 2, pp. 117–137, 2010.

[12] D. Cai, X. He, J. Han, and T. S. Huang, "Graph regularized nonnegative matrix factorization for data representation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 33, no. 8, pp. 1548–1560, 2011.

[13] M. Belkin, V. Sindhwani, and P. Niyogi, "Manifold regularization: a geometric framework for learning from labeled and unlabeled examples," *Journal of Machine Learning Research*, vol. 7, pp. 2399–2434, 2006.

[14] Q. Gu, C. Ding, and J. Han, "On trivial solution and scale transfer problems in graph regularized NMF," in *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI '11)*, pp. 1288–1293, Barcelona, Spain, July 2011.

[15] X. Lu, H. Wu, Y. Yuan, P. Yan, and X. Li, "Manifold regularized sparse NMF for hyperspectral unmixing," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 51, no. 5, pp. 2815–2826, 2013.

[16] F. Liu, X. Yang, N. Guan, and X. Yi, "Online graph regularized non-negative matrix factorization for large-scale datasets," *Neurocomputing*, vol. 204, pp. 162–171, 2016.

[17] S. S. Bucak and B. Gunsel, "Video content representation by incremental non-negative matrix factorization," in *Proceedings of the 14th IEEE International Conference on Image Processing (ICIP '07)*, pp. 113–116, San Antonio, Tex, USA, September 2007.

[18] S. S. Bucak and B. Gunsel, "Incremental subspace learning via non-negative matrix factorization," *Pattern Recognition*, vol. 42, no. 5, pp. 788–797, 2009.

[19] Z.-Z. Yu, Y.-H. Liu, B. Li, S.-C. Pang, and C.-C. Jia, "Incremental graph regulated nonnegative matrix factorization for face recognition," *Journal of Applied Mathematics*, vol. 2014, Article ID 928051, 2014.

[20] N. Guan, D. Tao, Z. Luo, and J. Shawe-Taylor, "MahNMF: Manhattan non-negative matrix factorization," https://arxiv.org/abs/1207.3438.

[21] T. Liu, M. Gong, and D. Tao, "Large-cone nonnegative matrix factorization," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 28, no. 9, pp. 2129–2142, 2017.

[22] M. Belkin and P. Niyogi, "Laplacian eigenmaps and spectral techniques for embedding and clustering," in *Advances in Neural Information Processing Systems*, vol. 14, pp. 585–591, 2001.