DSANLS: Accelerating Distributed Nonnegative Matrix Factorization via Sketching

Yuqiu Qian[†], Conghui Tan[‡], Nikos Mamoulis[♯], David W. Cheung[†]

[†]The University of Hong Kong [‡]The Chinese University of Hong Kong [♯]University of Ioannina

[†]{yqqian, dcheung}@cs.hku.hk [‡]chtan@se.cuhk.edu.hk [♯]nikos@cs.uoi.gr

ABSTRACT

Nonnegative matrix factorization (NMF) has been successfully applied in different fields, such as text mining, image processing, and video analysis. NMF is the problem of determining two nonnegative low rank matrices U and V, for a given input matrix M, such that $M \approx UV^\top$. There is an increasing interest in parallel and distributed NMF algorithms, due to the high cost of centralized NMF on large matrices. In this paper, we propose a *distributed sketched alternating nonnegative least squares* (DSANLS) framework for NMF, which utilizes a matrix sketching technique to reduce the size of nonnegative least squares subproblems in each iteration for U and V. We design and analyze two different random matrix generation techniques and two subproblem solvers. Our theoretical analysis shows that DSANLS converges to the stationary point of the original NMF problem and it greatly reduces the computational cost in each subproblem as well as the communication cost within the cluster. DSANLS is implemented using MPI for communication, and tested on both dense and sparse real datasets. The results demonstrate the efficiency and scalability of our framework, compared to the state-of-art distributed NMF MPI implementation.

ACM Reference Format:
Yuqiu Qian, Conghui Tan, Nikos Mamoulis, David W. Cheung. 2018. DSANLS: Accelerating Distributed Nonnegative Matrix Factorization via Sketching. In *Proceedings of WSDM'18.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3159652.3159662

1 INTRODUCTION

Nonnegative matrix factorization (NMF) is a technique for discovering nonnegative latent factors and/or performing dimensionality reduction. NMF finds applications in text mining [30], image/video processing [19], and analysis of social networks [34]. Unlike general matrix factorization (MF), NMF restricts the two output matrix factors to be nonnegative. Nonnegativity is inherent in the feature space of many real-world applications, therefore the resulting factors can have a natural interpretation. Specifically, the goal of NMF is to decompose a huge matrix $M \in \mathbb{R}_+^{m\times n}$ into the product of two matrices $U \in \mathbb{R}_+^{m\times k}$ and $V \in \mathbb{R}_+^{n\times k}$ such that $M \approx UV^\top$. $\mathbb{R}_+^{m\times n}$ denotes the set of $m \times n$ matrices with nonnegative real values, and k is a user-specified dimensionality, where typically $k \ll m, n$.

Generally, NMF can be defined as an optimization problem [21] as follows:

$$\min_{U\in\mathbb{R}_+^{m\times k}, V\in\mathbb{R}_+^{n\times k}} \left\|M - UV^\top\right\|_F, \tag{1}$$

where $\|X\|_F = \left(\sum_{ij} x_{ij}^2\right)^{1/2}$ is the Frobenius norm of X. However, Problem (1) is hard to solve directly because it is non-convex. Therefore, almost all NMF algorithms leverage two-block coordinate descent schemes: they optimize over one of the two factors, U or V, while keeping the other fixed [8]. By fixing V, we can optimize U by solving a nonnegative least squares (NLS) subproblem:

$$\min_{U\in\mathbb{R}_+^{m\times k}} \left\|M - UV^\top\right\|_F. \tag{2}$$

Modern data analysis tasks apply on big matrix data with increasing scale and dimensionality. Examples include community detection in a billion-node social network, background separation on a 4K video in which every frame has approximately 27 million rows [15], text mining on a bag-of-words matrix with millions of words. The volume of data is anticipated to increase in the 'big data' era, making it impossible to store the whole matrix in the main memory throughout NMF. Therefore, there is a need for high-performance and scalable distributed NMF algorithms.

In this paper, we propose a distributed framework for NMF. We choose Message Passing Interface using C (MPI/C) for our distributed implementation for efficiency, generality and privacy reasons. MPI/C does not require reading/writing data to/from disk or global shuffles of data matrix entries, as what Spark or MapReduce do. Nodes can collaborate without sharing their local input data, which is important for applications that involve sensitive data and have privacy considerations. Besides, high performance numerical computing routines like Intel® Math Kernel Library can be leveraged. The state-of-art implementation of distributed NMF is MPI-FAUN [15], a general framework that iteratively solves NLS subproblems for U and V. The main idea behind MPI-FAUN is to exploit the independence of local updates for rows of U and V, in order to minimize the communication requirements of matrix multiplication operations within the NMF algorithms.

Our idea is to speed up distributed NMF in a new, orthogonal direction: by reducing the problem size of each NLS subproblem within NMF, which in turn decreases the overall computation cost. In a nutshell, we reduce the size of each NLS subproblem, by employing a *matrix sketching* technique: the involved matrices in the

WSDM'18, February 5–9, 2018, Marina Del Rey, CA, USA

ACM ISBN 978-1-4503-5581-0/18/02…$15.00
https://doi.org/10.1145/3159652.3159662

subproblem are multiplied by a specially designed random matrix at each iteration, which greatly reduces their dimensionality. As a result, the computational cost of each subproblem drops.

However, applying matrix sketching comes with several issues. First, although the size of each subproblem is significantly reduced, sketching involves matrix multiplication which brings computational overhead. Second, unlike in a single machine setting, the data are distributed to different nodes, which may have to communicate extensively in a poorly designed solution. In particular, each node only retains part of both the input matrix and the generated approximate matrices, causing difficulties due to data dependencies in the computation process. Besides, the generated random matrices should be the same for all nodes in every iteration, while broadcasting the random matrix to all nodes brings severe communication overhead and can become the bottleneck of distributed NMF. Furthermore, after reducing each original subproblem to a sketched random new subproblem, it is not clear whether the algorithm still converges and whether it converges to stationary points of the original NMF problem.

Our *distributed sketched alternating nonnegative least squares* (DSANLS) overcomes these problems. First, the extra computation cost due to sketching is reduced with a proper choice of the random matrices. Second, the same random matrices used for sketching are generated independently at each node, thus there is no need for transferring them between nodes during distributed NMF. Having the complete random matrix at each node, an NMF iteration can be done locally with the help of a matrix multiplication rule with proper data partitioning. Therefore, our matrix sketching approach reduces not only the computational, but also the communication cost. Moreover, due to the fact that sketching also *shifts* the optimal solution of each original NMF subproblem, we propose subproblem solvers paired with theoretical guarantees of their convergence to a stationary point of the original subproblems.

Our contributions can be summarized as follows:

- We propose DSANLS, a novel high-performance distributed NMF algorithm. DSANLS is the first distributed NMF algorithm that leverages matrix sketching to reduce the problem size of each NLS subproblem and can be applied to both dense and sparse input matrices with a convergence guarantee.
- We propose a novel and specially designed subproblem solver (*proximal coordinate descent*), which helps DSANLS to converge faster. We also discuss the use of *projected gradient descent* as subproblem solver, showing that it is equivalent to stochastic gradient descent (SGD) on the original (non-sketched) NLS subproblem.
- We present a detailed theoretical analysis of DSANLS, and prove that DSANLS converges to a stationary point of the original NMF problem. This convergence proof is novel and non-trivial because of the involvement of matrix sketching at each iteration.
- We conduct an experimental evaluation using several (dense and sparse) real datasets, which demonstrates the efficiency and scalability of DSANLS.

The remainder of the paper is organized as follows. Section 2 briefly reviews NMF algorithms, discusses distributed NMF techniques, and introduces the matrix sketching technique. Our DSANLS algorithm is presented in Section 3. Detailed theoretical analysis of DSANLS algorithm is discussed in Section 4. Section 5 evaluates DSANLS. Finally, Section 6 concludes the paper.

2 BACKGROUND AND RELATED WORK

2.1 NMF Algorithms

Within two-block coordinate descent schemes (exact or inexact, shown as Algorithm 1), different subproblem solvers are proposed.

Algorithm 1: Two-Block Coordinate Descent: Framework of Most NMF Algorithms

initialize $U_0 \geq 0$, $V_0 \geq 0$;
for $t = 0$ ***to*** $T - 1$ **do**
  $U_{t+1} \leftarrow \text{update}(M, U_t, V_t)$;
  $V_{t+1} \leftarrow \text{update}(M, U_{t+1}, V_t)$;
end
return U_T *and* V_T

The first widely used update rule is Multiplicative Updates (MU), which was first applied for solving NLS problems in [6]. Later, MU was rediscovered and used for NMF in [21]. MU is based on the majorization-minimization framework. Its application guarantees that the objective function monotonically decreases [6, 21].

Another extensively studied method is alternating nonnegative least squares (ANLS), which represents a class of methods where the subproblems for U and V are solved exactly following the framework described in Algorithm 1. ANLS is guaranteed to converge to a stationary point [10] and has been shown to perform very well in practice with active set [16, 18], projected gradient [24], quasi-Newton [41], or accelerated gradient [12] methods as the subproblem solver.

Hierarchical alternating least squares (HALS) [4] solves each NLS subproblem using an exact coordinate descent method that updates one individual column of U at a time. The optimal solutions of the corresponding subproblems can be written in a closed form.

2.2 Distributed NMF

Parallel NMF algorithms are well studied in the literature [13, 36]. However, different from a parallel and single machine setting, data sharing and communication have considerable cost in a distributed setting. Therefore, we need specialized NMF algorithms for massive scale data handling in a distributed environment. The first method in this direction [25] is based on the MU algorithm. It mainly focuses on sparse matrices and applies a careful partitioning of the data in order to maximize data locality and parallelism. Later, Cloud-NMF [23], a MapReduce-based NMF algorithm similar to [25], was implemented and tested on large-scale biological datasets. Another distributed NMF algorithm [40] leverages block-wise updates for local aggregation and parallelism. It also performs frequent updates using whenever possible the most recently updated data, which is more efficient than traditional concurrent counterparts. Apart from MapReduce implementations, Spark is also attracting attention for its advantage in iterative algorithms, e.g., using MLlib [28]. Finally, there are implementations using X10 [11] and on GPU [27].

The most recent and related work in this direction is MPI-FAUN [14, 15], which is the first implementation of NMF using MPI for interprocessor communication. MPI-FAUN is flexible and can be utilized for a broad class of NMF algorithms that iteratively solve NLS subproblems including MU, HALS, and ANLS/BPP. MPI-FAUN exploits the independence of local update computation for rows of U and V

to apply communication-optimal matrix multiplication. In a nutshell, the full matrix M is split across a two-dimensional grid of processors and multiple copies of both U and V are kept at different nodes, in order to reduce the communication between nodes during the iterations of NMF algorithms.

2.3 Matrix Sketching

Matrix sketching is a technique that has been previously used in numerical linear algebra [9], statistics [32] and optimization [33]. Its basic idea is described as follows. Suppose we need to find a solution x to the equation:

$$Ax = b, \quad (A \in \mathbb{R}^{m\times n},\ b \in \mathbb{R}^m). \tag{3}$$

Instead of solving this equation directly, in each iteration of matrix sketching, a random matrix $S \in \mathbb{R}^{d\times m}$ ($d \ll m$) is generated, and we instead solve the following problem:

$$(SA)x = Sb. \tag{4}$$

Obviously, the solution of (3) is also a solution to (4), but not vice versa. However, the problem size has now decreased from $m \times n$ to $d\times n$. With a properly generated random matrix S and an appropriate method to solve subproblem (4), it can be guaranteed that we will progressively approach the solution to (3) by iteratively applying this sketching technique.

To the best of our knowledge, there is only one piece of previous work [38] which incorporates dual random projection into the NMF problem, in a centralized environment, sharing similar ideas as SANLS, the centralized version of our DSANLS algorithm. However, Wang et al. [38] did not provide an efficient subproblem solver, and their method was less effective than non-sketched methods in practical experiments. Besides, data sparsity was not taken into consideration in their work. Furthermore, no theoretical guarantee was provided for NMF with dual random projection. In short, SANLS is not same as [38] and DSANLS is much more than a distributed version of [38]. The methods that we propose in this paper are efficient in practice and have strong theoretical guarantees.

3 DSANLS: DISTRIBUTED SKETCHED ANLS

3.1 Notations

For a matrix A, we use $A_{i:j}$ to denote the entry at the i-th row and j-th column of A. Besides, either i or j can be omitted to denote a column or a row, i.e., $A_{i:}$ is the i-th row of A, and $A_{:j}$ is its j-th column. Furthermore, i or j can be replaced by a subset of indices. For example, if $I \subset \{1, 2, \ldots, m\}$, $A_{I:}$ denotes the sub-matrix of A formed by all rows in I, whereas $A_{:J}$ is the sub-matrix of A formed by all columns in a subset $J \subset \{1, 2, \ldots, n\}$.

3.2 Data Partitioning

Assume there are N computing nodes in the cluster. We partition the row indices $\{1, 2, \ldots, m\}$ of the input matrix M into N disjoint sets $I_1, I_2, \ldots, I_N$, where $I_r \subset \{1, 2, \ldots, m\}$ is the subset of rows assigned to node r, as in [25]. Similarly, we partition the column indices $\{1, 2, \ldots, n\}$ into disjoint sets $J_1, J_2, \ldots, J_N$ and assign column set J_r to node r. The number of rows and columns in each node are near the same in order to achieve load balancing, i.e., $|I_r| \approx m/N$ and $|J_r| \approx n/N$ for each node r. The factor matrices U and V are also assigned to nodes accordingly, i.e., node r stores and updates $U_{I_r:}$ and $V_{J_r:}$ as shown in Figure 1.

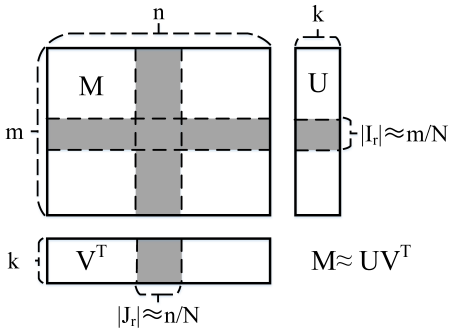


Figure 1: Data partitioning to N nodes

Data partitioning in distributed NMF differs from that in parallel NMF. Previous works on parallel NMF [13, 36] choose to partition U and V along the long dimension, but we adopt the row-partitioning of U and V as in [25]. To see why, take the U-subproblem (2) as an example and observe that it is row-independent in nature, i.e., the r-th row block of its solution $U_{I_r:}$ is given by

$$U_{I_r:} = \underset{U_{I_r:}\in\mathbb{R}_+^{|I_r|\times k}}{\arg\min} \left\|M_{I_r:} - U_{I_r:}V^\top\right\|_F^2 \tag{5}$$

and thus can be solved independently without referring to any other row blocks of U. The same holds for the V-subproblem. In addition, no communication is needed concerning M when solving (5) because $M_{I_r:}$ is already present in node r.

On the other hand, solving (5) requires the entire V of size $n \times k$, meaning that every node needs to gather V from all other nodes. This process can easily be the bottleneck of a naive distributed ANLS implementation. As we will explain shortly, our DSALNS algorithm alleviates this problem, since we use a sketched matrix of reduced size instead of the original complete matrix V.

3.3 SANLS: Sketched ANLS

To better understand DSANLS, we first introduce the Sketched ANLS (SANLS), i.e., a centralized version of our algorithm. Recall that in section 2.1, at each step of ANLS, either U or V is fixed and we solve a nonnegative least square problem (2) over the other variable. Intuitively, it is unnecessary to solve this subproblem with high accuracy, because we may not have reached the optimal solution for the fixed variable so far. Hence, when the fixed variable changes in the next step, this accurate solution from the previous step will not be optimal anymore and will have to be re-computed. Our idea is to apply matrix sketching for each subproblem, in order to obtain an approximate solution for it at a much lower computational and communication cost.

Specifically, suppose we are at the t-th iteration of ANLS, and our current estimations for U and V are U^t and V^t respectively. We must solve subproblem (2) in order to update U^t to a new matrix U^{t+1}. We apply matrix sketching to the residual term of subproblem (2). The subproblem now becomes:

$$\min_{U\in\mathbb{R}_+^{m\times k}} \left\|MS^t - U\left(V^{t\top}S^t\right)\right\|_F^2, \tag{6}$$

where $S^t \in \mathbb{R}^{n\times d}$ is a randomly-generated matrix. Hence, the problem size decreases from $n \times k$ to $d \times k$. d is chosen to be much smaller than n, in order to sufficiently reduce the computational cost[1]. Similarly, we transform the V-subproblem into

$$\min_{V\in\mathbb{R}_+^{n\times k}} \left\|M^\top S'^t - V\left(U^{t\top} S'^t\right)\right\|_F^2, \tag{7}$$

where $S'^t \in \mathbb{R}^{m\times d'}$ is also a random matrix with $d' \ll m$.

3.4 DSANLS: Distributed SANLS

Now, we come to our proposal: the distributed version of SANLS called DSANLS. Since the U-subproblem (6) is the same as the V-subproblem (7) in nature, here we restrict our attention to the U-subproblem. The first observation about subproblem (6) is that it is still row-independent, thus node r only needs to solve

$$\min_{U_{I_r:}\in\mathbb{R}_+^{|I_r|\times k}} \left\|\left(MS^t\right)_{I_r:} - U_{I_r:}\left(V^{t\top} S^t\right)\right\|_F^2.$$

For simplicity, we denote

$$A_r^t \triangleq \left(MS^t\right)_{I_r:} \quad \text{and} \quad B^t \triangleq V^{t\top} S^t, \tag{8}$$

and the above subproblem can be written as:

$$\min_{U_{I_r:}\in\mathbb{R}_+^{|I_r|\times k}} \left\|A_r^t - U_{I_r:}B^t\right\|_F^2. \tag{9}$$

Thus, node r needs to know matrices A_r^t and B^t in order to solve the subproblem.

For A_r^t, by applying matrix multiplication rules, we get

$$A_r^t = \left(MS^t\right)_{I_r:} = M_{I_r:}S^t$$

Therefore, if S^t is stored at node r, A_r^t can be computed without any communication.

On the other hand, computing $B^t = \left(V^{t\top} S^t\right)$ requires communication across the whole cluster, since the rows of V^t are distributed across different nodes. Fortunately, if we assume that S^t is stored at all nodes again, we can compute B^t in a much cheaper way. Following block matrix multiplication rules, we can rewrite B^t as:

$$\begin{aligned} B^t &= V^{t\top} S^t \\ &= \begin{bmatrix} \left(V_{J_1:}^t\right)^\top & \cdots & \left(V_{J_N:}^t\right)^\top \end{bmatrix} \begin{bmatrix} S_{J_1:}^t \\ \vdots \\ S_{J_N:}^t \end{bmatrix} \\ &= \sum_{r=1}^{N} \left(V_{J_r:}^t\right)^\top S_{J_r:}^t. \end{aligned}$$

Note that the summand $\bar{B}_r^t \triangleq \left(V_{J_r:}^t\right)^\top S_{J_r:}^t$ is a matrix of size $k \times d$ and can be computed locally. As a result, communication is only needed for summing up the matrices $\bar{B}_r^t$ of size $k \times d$ by using MPI all-reduce operation, which is much cheaper than transmitting the whole V_t of size $n \times k$.

[1]However, we should not choose an extremely small d, otherwise the the size of sketched subproblem would become so small that it can hardly represent the original subproblem, preventing NMF from converging to a good result. In practice, we can set $d = 0.1n$ for medium-sized matrices and $d = 0.01n$ for large matrices if $m \approx n$. When m and n differ a lot, e.g., $m \ll n$ without loss of generality, we should not apply sketching technique to the V subproblem (since solving the U subproblem is much more expensive) and simply choose $d = m \ll n$.

Now, the only remaining problem is the transmission of S^t. Since S^t can be dense, even larger than V^t, broadcasting it across the whole cluster can be quite expensive. However, it turns out that we can avoid this. Recall that S^t is a randomly-generated matrix; each node can generate exactly the same matrix, if we use the same pseudo-random generator and the same seed. Therefore, we only need to broadcast the random seed, which is just an integer, at the beginning of the whole program. This ensures that each node generates exactly the same random number sequence and hence the same random matrices S^t at each iteration.

In short, the communication cost of each node is reduced from $O(nk)$ to $O(dk)$ by adopting our sketching technique for the U-subproblem. Likewise, the communication cost of each V-subproblem is decreased from $O(mk)$ to $O(d'k)$. The general framework of our DSANLS algorithm is listed in Algorithm 2.

Algorithm 2: Distributed SANLS on Node r

Initialize $U_{I_r:}^0$, $V_{J_r:}^0$
Broadcast the random seed
for $t = 0$ ***to*** $T-1$ **do**
  Generate random matrix $S^t \in \mathbb{R}^{n\times d}$
  Compute $A_r^t \leftarrow M_{I_r:}S^t$
  Compute $\bar{B}_r^t \leftarrow \left(V_{J_r:}^t\right)^\top S_{J_r:}^t$
  All-Reduce: $B^t \leftarrow \sum_{i=1}^N \bar{B}_i^t$
  Update $U_{I_r:}^{t+1}$ by solving $\min_{U_{I_r:}} \|A_r^t - U_{I_r:}B^t\|$
  Generate random matrix $S'^t \in \mathbb{R}^{m\times d'}$
  Compute $A_r'^t \leftarrow \left(M_{:J_r}\right)^\top S'^t$
  Compute $\bar{B}_r'^t \leftarrow \left(U_{I_r:}^t\right)^\top S_{I_r:}'^t$
  All-Reduce: $B'^t \leftarrow \sum_{i=1}^N \bar{B}_i'^t$
  Update $V_{J_r:}^{t+1}$ by solving $\min_{V_{J_r:}} \|A_r'^t - V_{J_r:}B'^t\|$
end
return $U_{I_r:}^T$ *and* $V_{J_r:}^T$

3.5 Generation of Random Matrices

A key problem in Algorithm 2 is how to generate random matrices $S^t \in \mathbb{R}^{n\times d}$ and $S'^t \in \mathbb{R}^{m\times d'}$. Here we focus on generating a random $S^t \in \mathbb{R}^{d\times n}$ satisfying Assumption 1. The reason for choosing such a random matrix is that the corresponding sketched problem would be equivalent to the original problem on expectation; we will prove this in Section 3.6.

Assumption 1. *Assume the random matrices are normalized and have bounded variance, i.e., there exists a constant σ^2 such that*

$$\mathbb{E}\left[S^t S^{t\top}\right] = I \quad \textit{and} \quad \mathbb{V}\left[S^t S^{t\top}\right] \le \sigma^2$$

for all t, where I is the identity matrix.

Different options exist for such matrices, which have different computation costs in forming sketched matrices $A_r^t = M_{I_r:}S^t$ and $\bar{B}_r^t = \left(V_{J_r:}^t\right)^\top S_{J_r:}^t$. Since $M_{I_r:}$ is much larger than $V_{J_r:}^t$ and thus computing A_r^t is more expensive, we only consider the cost of constructing A_r^t here.

The most classical choice for a random matrix is one with i.i.d. Gaussian entries having mean 0 and variance $1/d$. It is easy to show that $\mathbb{E}\left[S^t S^{t\top}\right] = I$. Besides, Gaussian random matrix has bounded

variance because Gaussian distribution has finite fourth-order moment. However, since each entry of such a matrix is totally random and thus no special structure exists in S^t, matrix multiplication will be expensive. That is, when given $M_{I_r:}$ of size $|I_r| \times n$, computing its sketched matrix $A_r^t = M_{I_r:}S^t$ requires $O(|I_r|nd)$ basic operations.

A seemingly better choice for S^t would be a *subsampling* random matrix. Each column of such random matrix is uniformly sampled from $\{e_1, e_2, \ldots, e_n\}$ without replacement, where $e_i \in \mathbb{R}^n$ is the i-th canonical basis vector (i.e., a vector having its i-th element 1 and all others 0). We can easily show that such an S^t also satisfies $\mathbb{E}\left[S^t S^{t\top}\right] = I$ and the variance $\mathbb{V}\left[S^t S^{t\top}\right]$ is bounded, but this time constructing the sketched matrix $A_r^t = M_{I_r:}S^t$ only requires $O(|I_r|d)$. Besides, subsampling random matrix can preserve the sparsity of original matrix. Hence, a subsampling random matrix would be favored over a Gaussian random matrix by most applications, especially for very large-scale or sparse problems. On the other hand, we observed in our experiments that a Gaussian random matrix can result in a faster per-iteration convergence rate, because each column of the sketched matrix A_r^t contains entries from multiple columns of the original matrix and thus is more informative. Hence, it would be better to use a Gaussian matrix when the sketch size d is small and thus a $O(|I_r|nd)$ complexity is acceptable, or when the network speed of the cluster is poor, hence we should trade more local computation cost for less communication cost.

Although we only test two representative types of random matrices (i.e., Gaussian and subsampling random matrices), our framework is readily applicable for other choices, such as subsampled randomized Hadamard transform (SRHT) [1, 26] and count sketch [3, 5, 31]. The choice of random matrices is not the focus of this paper and left for future investigation.

3.6 Solving Subproblems

Before describing how to solve subproblem (9), let us make an important observation. As discussed in Section 2.3, the sketching technique has been applied in solving linear systems before. However, the situation is different in matrix factorization. Note that for the distributed matrix factorization problem we usually have

$$\min_{U_{I_r:}\in\mathbb{R}_+^{|I_r|\times k}} \left\|M_{I_r:} - U_{I_r:}V^{t\top}\right\|_F^2 \neq 0.$$

So, for the sketched subproblem (9), which can be equivalently written as

$$\min_{U_{I_r:}\in\mathbb{R}_+^{|I_r|\times k}} \left\|\left(M_{I_r:} - U_{I_r:}V^{t\top}\right)S^t\right\|_F^2,$$

the non-zero entries of the residual matrix $\left(M_{I_r:} - U_{I_r:}V^{t\top}\right)$ will be scaled by the matrix S^t at different levels. As a consequence, the optimal solution will be shifted because of sketching. This fact alerts us that for SANLS, we need to update U^{t+1} by exploiting the sketched subproblem (9) to step towards the true optimal solution and avoid convergence to the solution of the sketched subproblem.

3.6.1 Projected Gradient Descent. A natural method is to use *one step*[2] of projected gradient descent for the sketched subproblem:

$$\begin{aligned} U_{I_r:}^{t+1} &= \max\left\{U_{I_r:}^t - \eta_t\ \nabla_{U_{I_r:}}\left\|A_r^t - U_{I_r:}B^t\right\|_F^2\Big|_{U_{I_r:}=U_{I_r:}^t},\ 0\right\} \\ &= \max\left\{U_{I_r:}^t - 2\eta_t\left[U_{I_r:}^t B^t B^{t\top} - A_r^t B^{t\top}\right],\ 0\right\}, \end{aligned} \tag{10}$$

where $\eta_t > 0$ is the step size and $\max\{\cdot,\cdot\}$ denotes the entry-wise maximum operation. In the gradient descent step (10), the computational cost mainly comes from two matrix multiplications: $B^t B^{t\top}$ and $A_{t,r}B^{t\top}$. Note that A_r^t and B^t are of sizes $|I_r|\times d$ and $k\times d$ respectively, thus the gradient descent step takes $O\left(kd(|I_r| + k)\right)$ in total.

To exploit the nature of this algorithm, we further expand the gradient:

$$\begin{aligned} \nabla_{U_{I_r:}}\left\|A_r^t - U_{I_r:}B^t\right\|_F^2 &= 2\left[U_{I_r:}B^t B^{t\top} - A_r^t B^{t\top}\right] \\ \overset{(8)}{=} 2\left[U_{I_r:}\left(V^{t\top}S^t\right)\left(V^{t\top}S^t\right)^\top - \left(M_{I_r:}S^t\right)\left(V^{t\top}S^t\right)^\top\right] \\ = 2\left[U_{I_r:}V^{t\top}\left(S^t S^{t\top}\right)V^t - M_{I_r:}\left(S^t S^{t\top}\right)V^t\right]. \end{aligned}$$

By taking the expectation of the above equation, and using the fact $\mathbb{E}\left[S^t S^{t\top}\right] = I$, we have:

$$\begin{aligned} \mathbb{E}\left[\nabla_{U_{I_r:}}\left\|A_r^t - U_{I_r:}B^t\right\|_F^2\right] &= 2\left[U_{I_r:}V^{t\top}V^t - M_{I_r:}V^t\right] \\ = \nabla_{U_{I_r:}}\left\|M_{I_r:} - U_{I_r:}V^{t\top}\right\|_F^2, \end{aligned}$$

which means that the gradient of the sketched subproblem is equivalent to the gradient of the original problem on expectation. Therefore, such a step of gradient descent can be interpreted as a (generalized) *stochastic gradient descent* (SGD) [29] method on the original subproblem. Thus, according to the theory of SGD, we naturally require the step sizes $\{\eta_t\}$ to be diminishing, i.e., $\eta_t \to 0$ as t increases.

3.6.2 Proximal Coordinate Descent. However, it is well known that the gradient descent method converges slowly, while the coordinate descent method, namely the HALS method for NMF, is quite efficient [8]. Still, because of its very fast convergence, HALS should not be applied to the sketched subproblem directly because it shifts the solution away from the true optimal solution. Therefore, we would like to develop a method which resembles HALS but will not converge towards the solutions of the sketched subproblems.

To achieve this, we add a regularization term to the sketched subproblem (9). The new subproblem becomes:

$$\min_{U_{I_r:}\in\mathbb{R}_+^{|I_r|\times k}} \left\|A_r^t - U_{I_r:}B^t\right\|_F^2 + \mu_t\left\|U_{I_r:} - U_{I_r:}^t\right\|_F^2, \tag{11}$$

where $\mu_t > 0$ is a parameter. Such regularization is reminiscent to the proximal point method [37] and parameter μ_t controls the step size as $1/\eta_t$ in projected gradient descent. We therefore require $\mu_t \to +\infty$ to enforce the convergence of the algorithm, e.g., $\mu_t = t$.

At each step of proximal coordinate descent, only one column of $U_{I_r:}$, say $U_{I_r,j}$ where $j \in \{1, 2, \ldots, k\}$, is updated:

$$\min_{U_{I_r:j}\in\mathbb{R}_+^{|I_r|}} \left\|A_r^t - U_{I_r:j}B_{j:}^t - \sum_{l\neq j}U_{I_r:l}B_{l:}^t\right\|_F^2 + \mu_t\left\|U_{I_r:j} - U_{I_r:j}^t\right\|_2^2.$$

[2]Note that we only apply one step of projected gradient descent here to avoid solution shifted.

It is not hard to see that the above problem is still row-independent, which means that each entry of the row vector $U_{I_r:j}$ can be solved independently at each node. For example, for any $i \in I_r$, the solution of $U_{i:j}^{t+1}$ is given by:

$$U_{i:j}^{t+1} = \arg\min_{U_{i:j}\geq 0} \left\| \left(A_r^t\right)_{i:} - U_{i:j}B_{j:}^t - \sum_{l\neq j} U_{i:l}B_{l:}^t \right\|_2^2 + \mu_t \left\| U_{i:j} - U_{i:j}^t \right\|_2^2$$
$$= \max\left\{ \frac{\mu_t U_{i:j}^t + \left(A_r^t\right)_{i:} B_{j:}^{t\top} - \sum_{l\neq j} U_{i:l}B_{l:}^t B_{j:}^{t\top}}{B_{j:}^t B_{j:}^{t\top} + \mu_t}, 0 \right\}. \quad (12)$$

At each step of coordinate descent, we choose the column j from $\{1, 2, \ldots, k\}$ successively. When updating column j at iteration t, the columns $l < j$ have already been updated and thus $U_{I_r:l} = U_{I_r:l}^{t+1}$, while the columns $l > j$ are old so $U_{I_r:l} = U_{I_r:l}^t$.

The complete proximal coordinate descent algorithm for the U-subproblem is summarized in Algorithm 3. When updating column j, computing the matrix-vector multiplication $A_r^t B_{j:}^{t\top}$ takes $O(d|I_r|)$. The whole inner loop takes $O\left(k\left(d + |I_r|\right)\right)$ because one vector dot product of length d is required for computing each summand and the summation itself needs $O\left(k|I_r|\right)$. Considering that there are k columns in total, the overall complexity of coordinate descent is $O\left(k((k+d)\,|I_r| + kd)\right)$. Typically, we choose $d > k$, so the complexity can be simplified to $O\left(kd\left(|I_r| + k\right)\right)$, which is the same as that of gradient descent.

Since proximal coordinate descent is much more efficient than projected gradient descent, we adopt it as the default subproblem solver within DSANLS.

Algorithm 3: Proximal Coordinate Descent for Local Subproblem (9) on Node r

Parameter: $\mu_t > 0$
for $j = 1$ ***to*** k **do**
  $T \leftarrow \mu_t U_{I_r:j}^t + A_r^t B_{j:}^{t\top}$
  for $l = 1$ ***to*** $j-1$ **do**
    $T \leftarrow T - \left(B_{l:}^t B_{j:}^{t\top}\right) U_{I_r:l}^{t+1}$
  end
  for $l = j+1$ ***to*** k **do**
    $T \leftarrow T - \left(B_{l:}^t B_{j:}^{t\top}\right) U_{I_r:l}^t$
  end
  $U_{I_r:j}^{t+1} \leftarrow \max\left\{T / \left(B_{j:}^t B_{j:}^{t\top} + \mu_t\right), 0\right\}$
end
return $U_{I_r:}^{t+1}$

4 THEORETICAL ANALYSIS

4.1 Complexity Analysis

We now analyze the computational and communication costs of our DSANLS algorithm, when using subsampling random sketch matrices. The computational complexity at each node is:

$$O\Big(\overbrace{d}^{\text{generating } S^t} + \overbrace{|I_r|d}^{\text{constructing } A_r^t \text{ and } B^t} + \overbrace{kd(|I_r|+k)}^{\text{solving subproblem}} \Big)$$
$$= O\left(kd(|I_r| + k)\right) \approx O\left(kd\left(\frac{m}{N} + k\right)\right) \quad (13)$$

Moreover, as we have shown in Section 3.4, the communication cost of DSANLS is $O\left(kd\right)$.

On the other hand, for a classical implementation of distributed HALS [7], the computational cost is

$$O\left(kn\left(|I_r| + k\right)\right) \approx O\left(kn\left(\frac{m}{N} + k\right)\right) \quad (14)$$

and the communication cost is $O\left(kn\right)$ due to the all-gathering of V^t's.

Comparing the above quantities, we observe an $n/d \gg 1$ speedup of our DSANLS algorithm over HALS in both computation and communication. However, we empirically observed that DSANLS has a slower per-iteration convergence rate (i.e., it needs more iterations to converge). Still, as we will show in the next section, in practice, DSANLS is superior to alternative distributed NMF algorithms, after taking all factors into account.

4.2 Convergence Analysis

Here we provide theoretical convergence guarantees for the proposed SANLS and DSANLS algorithms. We show that SANLS and DSANLS converge to a stationary point.

To establish convergence result, Assumption 2 is needed first.

Assumption 2. *Assume all the iterates U^t and V^t have uniformly bounded norms, which means that there exists a constant R such that*

$$\|U^t\|_F \leq R \quad \textit{and} \quad \|V^t\|_F \leq R$$

for all t.

We experimentally observed that this assumption holds in practice, as long as the step sizes used are not too large. Besides, Assumption 2 can also be enforced by imposing additional constraints, e.g.:

$$U_{i:l} \leq \sqrt{2\|M\|_F} \quad \text{and} \quad V_{j:l} \leq \sqrt{2\|M\|_F} \quad \forall i, j, l, \quad (15)$$

with which we have $R = \max\{m, n\}k\sqrt{2\|M\|_F}$. Such constraints can be very easily handled by both of our projected gradient descent and regularized coordinate descent solvers. Lemma 4.1 shows that imposing such extra constraints does not prevent us from finding the global optimal solution.

Lemma 4.1. *If the optimal solution to the original problem* (1) *exists, there is at least one global optimal solution in the domain* (15).

Based on Assumptions 1 (see Section 3.5) and Assumption 2, we now can formally show our main convergence result:

Theorem 4.2. *Under Assumptions 1 and 2, if the step sizes satisfy*

$$\sum_{t=1}^{\infty} \eta_t = \infty \quad \textit{and} \quad \sum_{t=1}^{\infty} \eta_t^2 < \infty,$$

for projected gradient descent, or

$$\sum_{t=1}^{\infty} 1/\mu_t = \infty \quad \textit{and} \quad \sum_{t=1}^{\infty} 1/\mu_t^2 < \infty,$$

for regularized coordinate descent, then SANLS and DSANLS with either sub-problem solver will converge to a stationary point of problem (1) *with probability 1.*

The proofs of Lemma 4.1 and Theorem 4.2 can be found in our technical report [35].

Table 1: Statistics of Datasets

Task	Dataset	#Rows	#Columns	Non-zero values	Sparsity
Video analysis	BOATS	216,000	300	64,800,000	0%
Image processing	MIT CBCL FACE	2,429	361	876,869	0%
	MNIST	70,000	784	10,505,375	80.86%
	GISETTE	13,500	5,000	8,770,559	87.01%
Text mining	Reuters(RCV1)	804,414	47,236	60,915,113	99.84%
Community detection	DBLP Collaboration Network	317,080	317,080	2,416,812	99.9976%

5 EXPERIMENTAL EVALUATION

This section includes an experimental evaluation of our algorithm on both dense and sparse real data matrices. The code can be downloaded from https://github.com/qianyuqiu79/DSANLS.

5.1 Datasets

We use real public datasets corresponding to different NMF tasks in our evaluation. Their statistics are summarized in Table 1.

Video Analysis. NMF can be used on video data for background subtraction (i.e., to detect moving objects) [17]. We here use BOATS[3] video dataset [2], which includes boats moving through water. The video has 15 fps and it is saved as a sequence of png files, whose format is RGB with a frame size of 360×200. We use 'Boats2' which contains one boat close to the camera for 300 frames and reshape the matrix such that every RGB frame is a column of our matrix; the final matrix is dense with size $216,000 \times 300$.

Image Processing. The first dataset we use for this application is MIT CBCL FACE DATABASE[4] as in [20]. To form the vectorized matrix, we use all 2,429 face images (each with 19×19 pixels) in the original training set. The second dataset is MINST[5], which is a widely used handwritten digits dataset. All 70,000 samples including both training and test set are used to form the vectorized matrix. The third one is GISETTE[6], another widely used dataset in handwritten digit recognition problem. We use all 13,500 pictures in the training, validation, and test datasets and form the vectorized matrix.

Text Mining. We use the Reuters document corpora[7] as in [39]. Reuters Corpus Volume I (RCV1) [22] is an archive of 804,414 manually categorized newswire stories made available by Reuters, Ltd. for research purposes. The official LYRL2004 chronological split is utilized. Non-zero values contain cosine-normalized, log TF-IDF vectors.

Community Detection. We convert the DBLP collaboration network[8] into its adjacency matrix. It is a co-authorship graph where two authors are connected if they have published at least one paper together.

5.2 Setup

We conduct our experiments on the Linux cluster of our institute with a total of 96 nodes. Each node contains 8-core Intel® Core™ i7-3770 CPU @ 1.60GHz cores and 16 GB of memory. Our algorithm is implemented in C using the Intel® Math Kernel Library (MKL) and Message Passing Interface (MPI). We use 10 nodes by default. Since tuning the factorization rank k is outside the scope of this paper, we use 100 as default value of k. Because of the large sizes of RCV1 and DBLP, we only use subsampling random matrices for them, as the use of Gaussian random matrices is too slow.

We evaluate DSANLS with subsampling and Gaussian random matrices, denoted by DSANLS/S and DSANLS/G, respectively, using proximal coordinate descent as the default subproblem solver. As mentioned in [14, 15], it is unfair to compare with a Hadoop implementation. We only compare DSANLS with MPI-FAUN[9] (all MPI-FAUN-MU, MPI-FAUN-HALS, and MPI-FAUN-ABPP implementations), which is the first and the state-of-the-art C++/MPI implementation with MKL and Armadillo. For parameters pc and pr in MPI-FAUN, we use the optimal values for each dataset, according to the recommendations in [14, 15].

5.3 Results

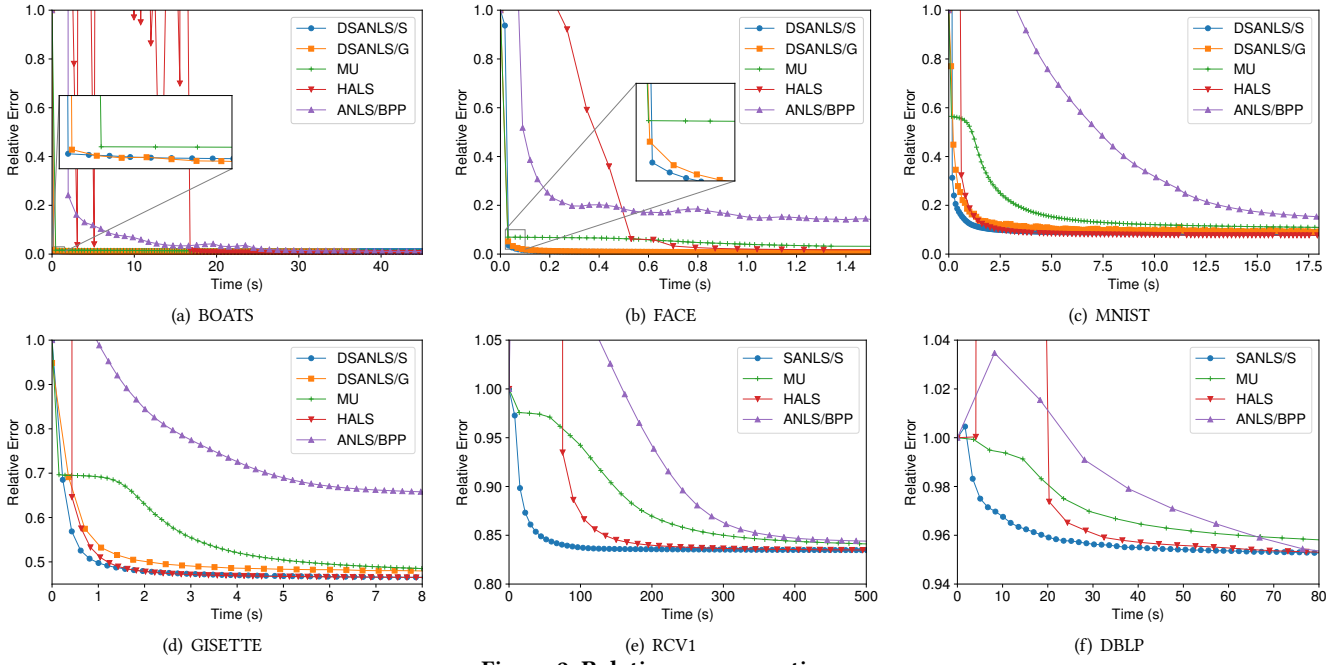
5.3.1 Performance Comparison. We use the relative error of the low rank approximation compared to the original matrix to measure the effectiveness of NMF by DSANLS with MPI-FAUN. This error measure was been widely used in previous work [14, 15, 17] and is formally defined as

$$\left\|M - UV^\top\right\|_F / \|M\|_F .$$

Since the time for each iteration is significantly reduced by our proposed DSANLS compared to MPI-FAUN, in Figure 2, we show the relative error over time for DSANLS and MPI-FAUN implementations of MU, HALS, and ANLS/BPP on the 6 real public datasets. Observe that DSANLS/S performs best in all 6 datasets, although DSANLS/G has faster per-iteration convergence rate. MU converges relatively slowly and usually has a bad convergence result; on the other hand HALS may oscillate in the early rounds[10], but converges quite fast and to a good solution. Surprisingly, although ANLS/BPP is considered to be the state-of-art NMF algorithm, it does not perform well in all 6 datasets. As we will see, this is due to its high per-iteration cost.

5.3.2 Scalability Comparison. We vary the number of nodes used in the cluster from 2 to 16 and record the average time for 100 iterations of each algorithm. Figure 3 shows the reciprocal of per-iteration time as a function of the number of nodes used. All algorithms exhibit good scalability for all datasets (nearly a straight line), except for FACE (i.e., Figure 3(a)). FACE is the smallest dataset, whose number of columns is 300, while k is set to 100 by default. When n/N is smaller than k, the complexity is dominated by k, hence, increasing the number of nodes does not reduce the computational cost, but may increase the communication overhead.

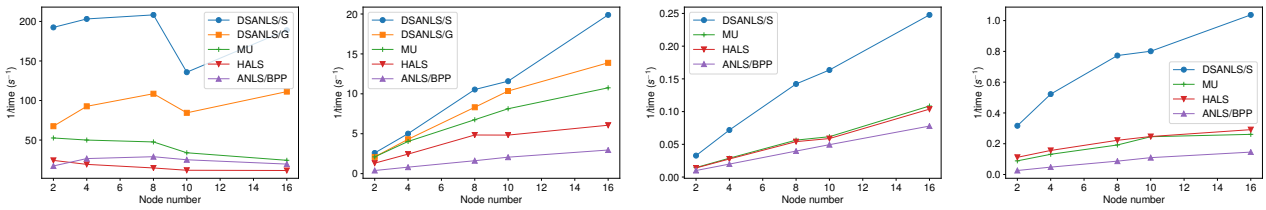
[3] http://visal.cs.cityu.edu.hk/downloads/
[4] http://cbcl.mit.edu/software-datasets/FaceData2.html
[5] http://yann.lecun.com/exdb/mnist/
[6] http://clopinet.com/isabelle/Projects/NIPS2003/#challenge
[7] we use the second version RCV1-v2, which can be found in http://jmlr.csail.mit.edu/papers/volume5/lewis04a/
[8] http://snap.stanford.edu/data/com-DBLP.html
[9] public code available at https://github.com/ramkikannan/nmflibrary
[10] HALS does not guarantee the objective function to decrease monotonically.



(a) BOATS (b) FACE (c) MNIST

(d) GISETTE (e) RCV1 (f) DBLP

Figure 2: Relative error over time



(a) FACE (b) MNIST (c) RCV1 (d) DBLP

Figure 3: Reciprocal of per-iteration time as a function of cluster size

In general, we can observe that DSANLS/Subsampling has the lowest per-iteration cost compared to all other algorithms, and DSANLS/Gaussian has similar cost to MU and HALS. ANLS/BPP has the highest per-iteration cost, explaining the bad performance of ANLS/BPP in Figure 2.

5.3.3 Performance Varying the Value of k. Although tuning the factorization rank k is outside the scope of this paper, we compare the performance of DSANLS with MPI-FAUN varying the value of k from 20 to 500 on RCV1. Observe from Figure 4 and Figure 2(e) that DSANLS outperforms the state-of-art algorithms for all values of k. Naturally, the relative error of all algorithms decreases with the increase of k, but they also take longer to converge.

5.3.4 Comparison with Projected Gradient Descent. In Section 3.6, we claimed that our proximal coordinate descent approach (denoted as DSANLS-RCD) is faster than projected gradient descent (also presented in the same section, denoted as DSANLS-PGD). Figure 5 confirms the difference in the convergence rate of the two approaches regardless of the random matrix generation approach.

6 CONCLUSION

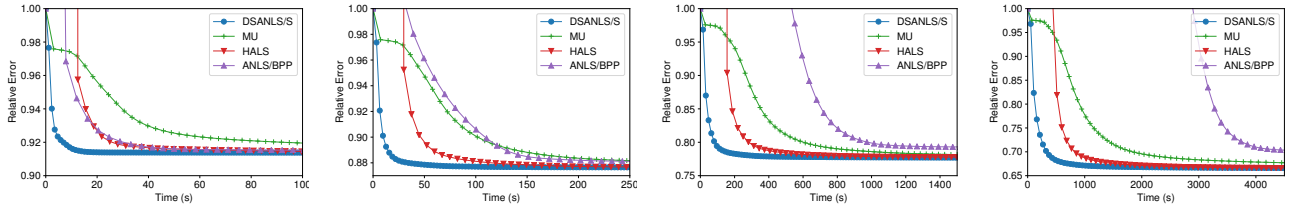
In this paper, we presented a novel distributed NMF algorithm that can be used for scalable analytics of high dimensional matrix data. Our approach follows the general framework of ANLS, but utilizes matrix sketching to reduce the problem size of each NLS subproblem. We discussed and compared two different approaches for generating random matrices (i.e. Gaussian and subsampling random matrices). Then, we presented two subproblem solvers for our general framework, and theoretically proved that our algorithm is convergent. We analyzed the per-iteration computational and communication cost of our approach and its convergence, showing its superiority compared to the previous state-of-the-art. Our experiments on several real datasets show that our method converges fast to an accurate solution and scales well with the number of cluster nodes used. In the future, we plan to study the application of DSANLS to dense or sparse tensors.

ACKNOWLEDGEMENTS

This work is partially supported by GRF Grants 17201414 and 17205015 from Hong Kong Research Grant Council. It has also received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 657347.

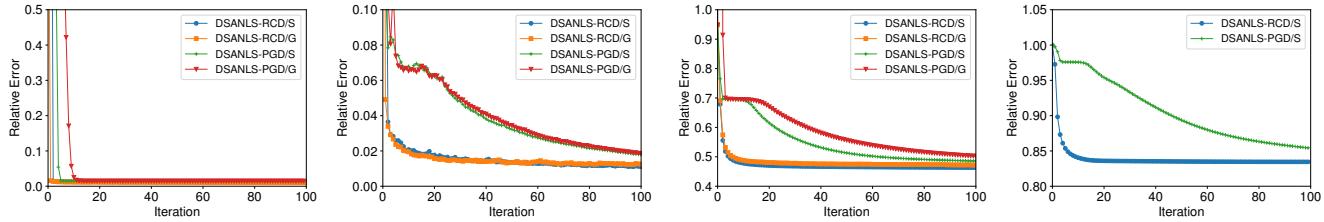
REFERENCES

[1] N. Ailon and B. Chazelle. Approximate nearest neighbors and the fast johnson-lindenstrauss transform. In *STOC*, pages 557–563. ACM, 2006.



(a) k=20 (b) k=50 (c) k=200 (d) k=500

Figure 4: Relative error over time, varying k value



(a) BOATS (b) FACE (c) GISETTE (d) RCV1

Figure 5: Relative error per-iteration of different subproblem solvers

[2] A. B. Chan, V. Mahadevan, and N. Vasconcelos. Generalized stauffer–grimson background subtraction for dynamic scenes. *Machine Vision and Applications*, 22(5):751–766, 2011.

[3] M. Charikar, K. Chen, and M. Farach-Colton. Finding frequent items in data streams. *Theoretical Computer Science*, 312(1):3–15, 2004.

[4] A. Cichocki and P. Anh-Huy. Fast local algorithms for large scale nonnegative matrix and tensor factorizations. *IEICE transactions on fundamentals of electronics, communications and computer sciences*, 92(3):708–721, 2009.

[5] K. L. Clarkson and D. P. Woodruff. Low rank approximation and regression in input sparsity time. In *STOC*, pages 81–90. ACM, 2013.

[6] M. E. Daube-Witherspoon and G. Muehllehner. An iterative image space reconstruction algorithm suitable for volume ect. *IEEE transactions on medical imaging*, 5(2):61–66, 1986.

[7] J. P. Fairbanks, R. Kannan, H. Park, and D. A. Bader. Behavioral clusters in dynamic graphs. *Parallel Computing*, 47:38–50, 2015.

[8] N. Gillis. The why and how of nonnegative matrix factorization. *Regularization, Optimization, Kernels, and Support Vector Machines*, 12(257), 2014.

[9] R. M. Gower and P. Richtárik. Randomized iterative methods for linear systems. *SIAM Journal on Matrix Analysis and Applications*, 36(4):1660–1690, 2015.

[10] L. Grippo and M. Sciandrone. On the convergence of the block nonlinear gauss–seidel method under convex constraints. *Operations research letters*, 26(3):127–136, 2000.

[11] D. Grove, J. Milthorpe, and O. Tardieu. Supporting array programming in x10. In *ARRAY*, page 38. ACM, 2014.

[12] N. Guan, D. Tao, Z. Luo, and B. Yuan. Nenmf: an optimal gradient method for nonnegative matrix factorization. *IEEE Transactions on Signal Processing*, 60(6):2882–2898, 2012.

[13] K. Kanjani. Parallel non negative matrix factorization for document clustering. *CPSC-659 (Parallel and Distributed Numerical Algorithms) course. Texas A&M University, Tech. Rep*, 2007.

[14] R. Kannan, G. Ballard, and H. Park. A high-performance parallel algorithm for nonnegative matrix factorization. In *PPoPP*, page 9. ACM, 2016.

[15] R. Kannan, G. Ballard, and H. Park. Mpi-faun: An mpi-based framework for alternating-updating nonnegative matrix factorization. *arXiv preprint arXiv:1609.09154*, 2016.

[16] H. Kim and H. Park. Nonnegative matrix factorization based on alternating nonnegativity constrained least squares and active set method. *SIAM journal on matrix analysis and applications*, 30(2):713–730, 2008.

[17] J. Kim, Y. He, and H. Park. Algorithms for nonnegative matrix and tensor factorizations: A unified view based on block coordinate descent framework. *Journal of Global Optimization*, 58(2):285–319, 2014.

[18] J. Kim and H. Park. Fast nonnegative matrix factorization: An active-set-like method and comparisons. *SIAM Journal on Scientific Computing*, 33(6):3261–3281, 2011.

[19] I. Kotsia, S. Zafeiriou, and I. Pitas. A novel discriminant non-negative matrix factorization algorithm with applications to facial image characterization problems. *IEEE Transactions on Information Forensics and Security*, 2(3):588–595, 2007.

[20] D. D. Lee and H. S. Seung. Learning the parts of objects by non-negative matrix factorization. *Nature*, 401(6755):788–791, 1999.

[21] D. D. Lee and H. S. Seung. Algorithms for non-negative matrix factorization. In *NIPS*, pages 556–562, 2001.

[22] D. D. Lewis, Y. Yang, T. G. Rose, and F. Li. Rcv1: A new benchmark collection for text categorization research. *JMLR*, 5(Apr):361–397, 2004.

[23] R. Liao, Y. Zhang, J. Guan, and S. Zhou. Cloudnmf: a mapreduce implementation of nonnegative matrix factorization for large-scale biological datasets. *Genomics, proteomics & bioinformatics*, 12(1):48–51, 2014.

[24] C.-J. Lin. Projected gradient methods for nonnegative matrix factorization. *Neural computation*, 19(10):2756–2779, 2007.

[25] 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 *WWW*, pages 681–690. ACM, 2010.

[26] Y. Lu, P. Dhillon, D. P. Foster, and L. Ungar. Faster ridge regression via the subsampled randomized hadamard transform. In *NIPS*, pages 369–377, 2013.

[27] E. Mejía-Roa, D. Tabas-Madrid, J. Setoain, C. García, F. Tirado, and A. Pascual-Montano. Nmf-mgpu: non-negative matrix factorization on multi-gpu systems. *BMC bioinformatics*, 16(1):1, 2015.

[28] X. Meng, J. Bradley, B. Yuvaz, E. Sparks, S. Venkataraman, D. Liu, J. Freeman, D. Tsai, M. Amde, S. Owen, et al. Mllib: Machine learning in apache spark. *JMLR*, 17(34):1–7, 2016.

[29] A. Nemirovski, A. Juditsky, G. Lan, and A. Shapiro. Robust stochastic approximation approach to stochastic programming. *SIAM Journal on optimization*, 19(4):1574–1609, 2009.

[30] V. P. Pauca, F. Shahnaz, M. W. Berry, and R. J. Plemmons. Text mining using non-negative matrix factorizations. In *SDM*, pages 452–456. SIAM, 2004.

[31] N. Pham and R. Pagh. Fast and scalable polynomial kernels via explicit feature maps. In *ACM SIGKDD*, pages 239–247. ACM, 2013.

[32] M. Pilanci and M. J. Wainwright. Iterative hessian sketch: Fast and accurate solution approximation for constrained least-squares. *JMLR*, pages 1–33, 2015.

[33] M. Pilanci and M. J. Wainwright. Newton sketch: A linear-time optimization algorithm with linear-quadratic convergence. *arXiv preprint arXiv:1505.02250*, 2015.

[34] I. Psorakis, S. Roberts, M. Ebden, and B. Sheldon. Overlapping community detection using bayesian non-negative matrix factorization. *Physical Review E*, 83(6):066114, 2011.

[35] Y. Qian, C. Tan, N. Mamoulis, and D. W. Cheung. Dsanls: Accelerating distributed nonnegative matrix factorization via sketching. Technical report, HKU CS, 2017.

[36] S. A. Robila and L. G. Maciak. A parallel unmixing algorithm for hyperspectral images. In *Optics East*, pages 63840F–63840F. International Society for Optics and Photonics, 2006.

[37] R. T. Rockafellar. Monotone operators and the proximal point algorithm. *SIAM journal on control and optimization*, 14(5):877–898, 1976.

[38] F. Wang and P. Li. Efficient nonnegative matrix factorization with random projections. In *SDM*, pages 281–292. SIAM, 2010.

[39] W. Xu, X. Liu, and Y. Gong. Document clustering based on non-negative matrix factorization. In *SIGIR*, pages 267–273. ACM, 2003.

[40] J. Yin, L. Gao, and Z. M. Zhang. Scalable nonnegative matrix factorization with block-wise updates. In *ECML-PKDD*, pages 337–352. Springer, 2014.

[41] R. Zdunek and A. Cichocki. Non-negative matrix factorization with quasi-newton optimization. In *ICAISC*, pages 870–879. Springer, 2006.