Robust Decentralized Low-Rank Matrix Decomposition

ISTVÁN HEGEDŰS and ÁRPÁD BERTA, University of Szeged, Hungary
LEVENTE KOCSIS and ANDRÁS A. BENCZÚR, Institute for Computer Science and Control, Hungarian Academy of Sciences (MTA SZTAKI)
MÁRK JELASITY, University of Szeged, and MTA-SZTE Research Group on AI, Hungary

Low-rank matrix approximation is an important tool in data mining with a wide range of applications including recommender systems, clustering, and identifying topics in documents. When the matrix to be approximated originates from a large distributed system—such as a network of mobile phones or smart meters—this is a very challenging problem due to the strongly conflicting, yet essential requirements of efficiency, robustness, and privacy preservation. We argue that while collecting sensitive data in a centralized fashion may be efficient, it is not an option when considering privacy and efficiency at the same time. Thus, we do not allow any sensitive data to leave the nodes of the network. The local information at each node (personal attributes, documents, media ratings, etc.) defines one row in the matrix. This means that all computations have to be performed at the edge of the network. Known parallel methods that respect the locality constraint—such as synchronized parallel gradient search or distributed iterative methods—require synchronized rounds or have inherent issues with load balancing, thus they are not robust to failure. Our distributed stochastic gradient descent algorithm overcomes these limitations. During the execution, any sensitive information remains local, while the global features (e.g., the factor model of movies) converge to the correct value at all nodes. We present a theoretical derivation, as well as a thorough experimental evaluation of our algorithm. We demonstrate that the convergence speed of our method is competitive while not relying on synchronization and being robust to extreme and realistic failure scenarios. To demonstrate the feasibility of our approach, we present trace-based simulations, real smartphone user behavior analysis, and tests over real movie recommender system data.

CCS Concepts: •**Human-centered computing** → **Mobile computing;** •**Information systems** → *Recommender systems;* •**Computing methodologies** → *Factor analysis;* •**Software and its engineering** → *Peer-to-peer architectures;* •**Networks** → Network privacy and anonymity;

Additional Key Words and Phrases: data mining; decentralized matrix factorization; decentralized recommender systems; online learning; stochastic gradient descent; singular value decomposition; privacy

ACM Reference Format:
István Hegedűs, Árpád Berta, Levente Kocsis, András A. Benczúr, and Márk Jelasity, 2016. Robust Decentralized Low-Rank Matrix Decomposition. *ACM Trans. Intell. Syst. Technol.* 7, 4, Article 62 (May 2016), 24 pages.
DOI: http://dx.doi.org/10.1145/2854157

1. INTRODUCTION

Finding a low-rank decomposition of a matrix is an essential tool in data mining and information retrieval [Azar et al. 2001]. Prominent applications include recom-

This work was partially supported by the grant OTKA NK 105645, and the "Momentum - Big Data" grant of the Hungarian Academy of Sciences.
Author's addresses: I. Hegedűs, Á. Berta, and M. Jelasity, MTA-SZTE Research Group on Artificial Intelligence, University of Szeged, PO Box 652, H-6701 Szeged, Hungary, L. Kocsis and A. A. Benczúr, Institute for Computer Science and Control, Hungarian Academy of Sciences (MTA SZTAKI), H-1111 Budapest, Lágymányosi u. 11.
 2157-6904/2016/05-ART62 $15.00
DOI: http://dx.doi.org/10.1145/2854157

mender systems [Drineas et al. 2002], information retrieval via Latent Semantic Indexing [Berry et al. 1995; Papadimitriou et al. 2000], Kleinberg's HITS algorithm for graph centrality [Kleinberg 1999], clustering [Drineas et al. 2004; McSherry 2001], and learning mixtures of distributions [Kannan et al. 2005; Achlioptas and McSherry 2005].

Low rank matrix approximation can naturally be applied for collaborative filtering. Given n items and m users, one can generate recommendations based on the low-rank decomposition of the $m \times n$ dimensional ratings matrix A with rows assigned to users and columns to items. First, the essence of the ratings matrix A is computed by finding a low-rank decomposition $A \approx XY^T$, where matrices X and Y^T are of dimension $m \times k$ and $k \times n$, respectively, and where k is small [Koren et al. 2009]. A row of X and a column of Y can be interpreted as a compressed representation of the corresponding user and item, respectively, in a k-dimensional feature space. The low-rank representation can then be applied for making recommendations by predicting all the unknown ratings in A by using the corresponding values in XY^T. Additional applications of low-rank approximation are found for adjacency matrices in social network analysis and term-document matrices in text classification.

Our targeted application environments are large-scale networked systems of user devices such as smartphones or smart meters. We identify three conflicting requirements. First, data on these devices such as movie ratings, lists of friends, personal documents, or sensor readings are often very sensitive, hence *privacy* is a key consideration in our algorithms. At the same time, the application also has to be *robust*. Devices will leave or fail frequently, and they can join as well at any time. Lastly, the solution has to be *efficient* enough to be practical. That is, fast convergence must be achieved, while keeping the amount of messages sent affordable for realistic communication networks.

Achieving cryptographically sound privacy is impractically expensive in our target systems. For example, a state-of-the-art secure multiparty computing solution by Nikolaenko et al. requires powerful central servers, and its scalability is demonstrated only for small examples [Nikolaenko et al. 2013]. This implies that in any robust and scalable server-based solution the central server will have access to some or all the sensitive data of the nodes. Since we want to avoid this, in our solution we opt for storing personal data only on personal devices that we process in a fully distributed way. While this choice does not in itself offer privacy in a cryptographic sense, it is a necessary condition of privacy in a practical setup.

Out of the three required properties, in this paper we focus on robustness and efficiency while using strictly local data, to the point that our solution is practically applicable in realistic privacy sensitive scenarios. In Section 7 we outline a few ideas on how to exploit the locality constraint to maximize privacy, but a detailed discussion is beyond the scope of the paper.

1.1. Our contribution

We implement a family of efficient and robust distributed matrix decomposition techniques. One of our algorithms applies for sparse matrices with undefined entries allowed, such as user rating data sets in recommender systems. The algorithm supports regularization. With another algorithm, we may compute the singular value decomposition (SVD) for complete matrices. Note that SVD is a special low-rank decomposition that exists for any fully defined matrix A with the attractive property that the decomposition consists of orthogonal matrices.

Our algorithms use stochastic gradient descent (SGD) to find a low-rank matrix decomposition $A \approx XY^T$, where we maintain several instances of the matrix Y that we update at the nodes. For privacy considerations, nodes store their own rows of the

matrices A and X. Each instance of Y performs a random walk over the network. When Y visits a node, it gets updated based on the local row of A, and at the same time the local row of X gets updated as well. We may assume that the matrix Y visible over the network carries no sensitive information, as Y represents the publicly known items of the recommender system.

To the best of our knowledge, we present the first fully distributed SGD algorithm that keeps X and A strictly local. As a special feature, our algorithm updates several versions of Y by sending them around the network, resulting in a convergence much faster than a single instance of SGD. The algorithm is fully asynchronous: messages can be delayed or lost. All we require from the network is a random walk mechanism that can perform every transition in bounded time. We show that the only stable fixed points of the search algorithm correspond to the desired decomposition.

In addition, we perform an experimental analysis to demonstrate the convergence speed and the scalability of the protocol.We also present a thorough experimental study over a trace of real smartphone user behavior. We demonstrate that our method is practical in a realistic networking environment and a realistic application setup.

1.2. Related work

Calculating SVD and other matrix decompositions is a well-studied problem. One approach is based on treating it as an optimization problem (see Section 2) and using *gradient search* to solve it [Gorrell 2006]. Guan et al. also follow this approach adapted for the non-negative case and propose an efficient gradient algorithm [Guan et al. 2012a].

In general, parallel versions of gradient search often assume the MapReduce framework [Chu et al. 2007] or a similar, less constrained, but still centralized model [Le et al. 2012]. In these approaches, partial gradients are calculated over batches of data in parallel, and these are either applied to blocks of X and Y in the map phase or summed up in the reduce phase. Zinkevich et al. propose a different approach in which SGD is applied on batches of data and the resulting models are then combined [Zinkevich et al. 2010]. Petroni et al. propose performance optimizations based on graph partitioning in a similar framework [Petroni and Querzoni 2014]. Gemulla et al. [Gemulla et al. 2011] propose an efficient parallel technique in which blocks of X and Y are iteratively updated while only blocks of Y are exchanged. In contrast to these approaches, we work with fully distributed data: we do not wish to make X public and we do not rely on central components or synchronization—a set of requirements ruling out the direct application of earlier approaches.

Another possibility is using fully distributed *iterative methods*. GraphLab [Low et al. 2010] supports iterative methods for various problems including SVD. In these approaches, the communication graph in the network is defined by the non-zero elements of matrix A; in other words, A is stored as edge-weights in the network. This is feasible only if A is (very) sparse and well balanced; a constraint rarely met in practice. In addition, iterative methods need access to A^T as well, which violates our constraint that the rows of A are not shared. Using the same edge-weight representation of A, one can implement another optimization approach for matrix decomposition: an iterative optimization of subproblems over overlapping local subnetworks [Liao et al. 2010]. The drawback of this approach is, again, that the structure of A defines the communication network and access to A^T is required. The approach of Ling et al. [Ling et al. 2012] also needs global information: in each iteration step, a global average needs to be calculated.

The first fully distributed algorithm for spectral analysis was given in [Kempe and McSherry 2004] where data is kept at the compute nodes and partial results are sent through the network. That algorithm, however, only computes the user side singular

vectors but not the item side ones and hence is insufficient, for example, to provide recommendations. Similarly, [Korada et al. 2011] computes the low-rank approximation but not the decomposition. This drawback is circumvented in [Isaacman et al. 2011] by assigning compute nodes not just for users but for items as well. In their gradient descent algorithm, item vectors are also stored at peers, which means that all items must know the ratings they receive, and this violates privacy.

We overcome the limitations of earlier fully distributed and gossip SVD algorithms by providing an algorithm without item-based processing elements, i.e. our algorithm is fully distributed in the sense that processing is carried out exclusively at the user devices, and the users may access their own ratings and the approximations of their own user factor vector as well as all the item factor vectors locally. Our approach is based on a stochastic gradient algorithm similar to the online algorithm presented in [Guan et al. 2012b], which calculates Y under an additional non-negativity constraint with the help of a constant-sized buffer of samples. We, however, work in a different, non-streaming setting: we are given a fixed decentralized database where we wish to calculate X as well. Also, in order to preserve data privacy, only one row of A is processed in each step and X is strictly decentralized, so no buffering can be implemented.

2. PROBLEM DEFINITION

The rank k matrix approximation problem is defined as follows. We are given a matrix $A \in \mathbb{R}^{m\times n}$ that holds our raw data, such as ratings of movies by users, or word statistics in documents. We are looking for matrices $X \in \mathbb{R}^{m\times k}$ and $Y \in \mathbb{R}^{n\times k}$ such that the error function

$$J(X,Y) = \frac{1}{2}\|A - XY^T\|_F^2 = \frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{n}(a_{ij} - \sum_{l=1}^{k} x_{il}y_{jl})^2 \tag{1}$$

is minimized. We say that the matrix XY^T that minimizes this function is an optimal rank-k approximation of A. Clearly, matrices X and Y^T—and hence XY^T—have a rank of at most k. Normally we select k such that $k \ll \min(m,n)$ in order to achieve a significant compression of the data. As a result, matrices X and Y^T can be thought of as high level features (such as topics of documents or semantic categories for words) that can be used to represent the original raw data in a compact way.

Singular value decomposition (SVD) is closely related to the above matrix decomposition problem. The SVD of a matrix $A \in \mathbb{R}^{m\times n}$ involves two orthogonal matrices $U \in \mathbb{R}^{m\times m}$ and $V \in \mathbb{R}^{n\times n}$ such that

$$A = U\Sigma V^T = \sum_{i=1}^{r} \sigma_i u_i v_i^T, \tag{2}$$

where the columns of the matrices $U = [u_1 u_2 \cdots u_m]$ and $V = [v_1 v_2 \cdots v_n]$ are the left and right singular vectors, and $\Sigma \in \mathbb{R}^{m\times n}$ is a diagonal matrix containing the singular values $\sigma_1, \sigma_2, \ldots, \sigma_r \geq 0$ of A ($r = \min(m,n)$). The relationship between SVD and-low rank decomposition is that $U_k\Sigma_k V_k^T$ is an optimal rank-k approximation of A, where the matrices $U_k \in \mathbb{R}^{m\times k}$ and $V_k \in \mathbb{R}^{n\times k}$ are derived from U and V by keeping the first k columns, and $\Sigma_k \in \mathbb{R}^{k\times k}$ is derived from Σ by keeping the top left $k \times k$ rectangular area, assuming without loss of generality that $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_r$ [Srebro and Jaakkola 2003]. In order to unify the notation, we use an alternate definition of SVD as a matrix factorization $A \approx X^* {Y^*}^T$ with

$$X^* = U_k\Sigma_U, \quad Y^* = V_k\Sigma_V, \tag{3}$$

such that Σ_U and Σ_V are diagonal and $\Sigma_U \Sigma_V = \Sigma_k$. In other words, although X^* and Y^* are not uniquely defined, we require that they contain orthogonal columns that are scaled versions of left and right singular vectors of A, respectively.

An important practical extension of the problem is considering undefined or missing values in A. In practice our knowledge of A is partial, yet we are interested in a decomposition that reproduces at least the known values. A common approach to follow is to define

$$J(X,Y) = \frac{1}{2} \sum_{(i,j)\in I} (a_{ij} - \sum_{l=1}^{k} x_{il} y_{jl})^2 + \frac{\alpha}{2}\|X\|_F^2 + \frac{\alpha}{2}\|Y\|_F^2, \tag{4}$$

where I contains those indices that are defined in A. The additional terms serve the purpose of regularization and are multiplied by the regularization parameter α.

In practical applications, another technique is to include bias explicitly in the model. Bias is expressed by the vector $b \in \mathbb{R}^{m\times 1}$ that is included in the error function as

$$J(X,Y,b) = \frac{1}{2} \sum_{(i,j)\in I} (a_{ij} - b_i - \sum_{l=1}^{k} x_{il} y_{jl})^2 + \frac{\alpha}{2}\|X\|_F^2 + \frac{\alpha}{2}\|Y\|_F^2, \tag{5}$$

where we have kept the regularization terms as well for completeness. For example, in recommender systems, bias represents the differences between users: some users tend to give high scores, others tend to give lower scores. Intuitively, X and Y represent relative differences in this context, and bias represents the average score. This often leads to a better prediction performance. With bias, the approximation of matrix A is thus given by $XY^T + b\mathbb{1}$ (where $\mathbb{1}$ is a row vector of n 1s). This approximation can be used, for example, to predict missing items of A.

3. SYSTEM MODEL AND DATA DISTRIBUTION

Having defined the computational problem, we will now state our assumptions on the network environment and the distribution of the data. Our network consists of nodes that are individual computational units (e.g., personal computers or smartphones) and have unique addresses. The number of nodes is potentially extremely large.

We assume that there is a membership service in our system. This service provides unique identities to the participants that include public and private keys for public key cryptography that are tied to the network address of the node. The membership service also offers peer sampling. That is, all nodes are assumed to have access to addresses of live nodes from the network. In practice, peer sampling can have a decentralized implementation that can be dynamic [Tölgyesi and Jelasity 2009] or it can be based on a static network with random neighbors [Roverso et al. 2013] that is able to deal with NAT devices as well. It can also be implemented as a centralized service.

Ideally, the neighbors returned by the peer sampling service should be uniform random samples of the live nodes, but in our application we may relax this assumption. As we shall see, SGD convergence only requires that the probability of visiting a node during a long random walk should not depend on its value, and all the nodes should have a roughly equal chance to be visited.

Every node can pass messages to other nodes, provided the address and the public key of the target node are known locally. Messages can be encrypted with the private key of the target, so wiretapping is not possible. In addition, messages can be lost or delayed and the nodes can leave the network and join again without prior notice. When a node rejoins the network, it retains the state it had when leaving the network.

As for the data distribution, we assume that each node has exactly one row (but note that our algorithms can be applied—and in fact profit from it—if a node has several

Algorithm 1 P2P low–rank factorization at node i

```
1: a_i                                                    ▷ row i of A
2: initialize Y
3: receivedY.add(Y)                                       ▷ initialize receivedY
4: initialize x_i                                         ▷ row i of X
5: quiet ← 0                                              ▷ rounds without receiving models
6: loop
7:     wait(Δ)
8:     if receivedY.isEmpty() then
9:         quiet ← quiet+1
10:        if quiet ≥ Δ_q then
11:            p ← selectPeer()
12:            send Y to p
13:        end if
14:    else
15:        quiet ← 0
16:        repeat                                         ▷ send all the received models
17:            Ỹ ← receivedY.remove()
18:            p ← selectPeer()
19:            send Ỹ to p
20:        until receivedY.isEmpty()
21:    end if
22: end loop

23: procedure ONRECEIVEY(Ỹ)
24:    (Y, x_i) ← update(Ỹ, x_i, a_i)
25:    receivedY.add(Y)
26: end procedure
```

rows). Our data distribution model is motivated by application scenarios, in which potentially sensitive data is available locally, such as private documents or ratings that naturally define rows of a matrix A, but where a decomposition needs to be found collectively without blatantly exposing this private data.

4. ALGORITHM

Our algorithm has its roots in the GOLF framework [Ormándi et al. 2013]. Algorithm 1 contains a version of the GOLF algorithm adapted to our problem. Each node i has its own approximation of the full matrix Y^* and an approximation x_i of row i of X^*. Thus, the nodes collectively store one version of the matrix X (the approximation of X^*) distributed just like matrix A with each node storing one row. Note that at every point in time, every node has its local approximation of the full matrix Y^* that may differ across the nodes. As we will see, these approximations interact through the single approximation of X^* and should converge to the same matrix Y^*. The output of the algorithm at any point in time is readily available as the local approximation at each node, so there is no need to explicitly combine the different approximations centrally.

All local approximate versions of Y^* perform a random walk in the network and get updated by the local data (a row of A and X) when visiting a node. First, each node i in the network initializes Y and x_i uniformly at random from the interval $[0, 1]$. The nodes then periodically send all the approximations of Y they received in the previous round to a randomly selected peer from the network. To select a random peer, we rely on a peer sampling service, as mentioned in Section 3. When receiving an approximation $\tilde{Y}$ (see procedure ONRECEIVEY) the node updates both Y and x_i using a stochastic

Algorithm 2 rank-k update at node i

1: η ▷ learning rate
2: **procedure** UPDATE(Y, x_i, a_i)
3: $\mathbf{err} \leftarrow a_i - x_i Y^T$
4: $x_i' \leftarrow x_i + \eta \cdot \mathbf{err} \cdot Y$
5: $Y' \leftarrow Y + \eta \cdot \mathbf{err}^T \cdot x_i$
6: **return** (Y', x_i')
7: **end procedure**

gradient rule and it subsequently stores this approximation in a list to be forwarded in the next round.

The algorithm involves periodic message sending from every node with a period of Δ. Later on, when we refer to one *iteration* or *round* of the algorithm, we simply mean a time interval of length Δ. Note that we do not require any synchronization of rounds over the network. Messages are sent independently, and they can be delayed or dropped as well. We require only that the delays are bounded and that the message drop probability is less than one. This allows the random walks to progress.

To avoid the termination of random walks due to failures, the algorithm includes a restarting mechanism based on the length of time during which no models are received. The timeout for this restart is $\Delta_q \cdot \Delta$. Unless otherwise stated, we use $\Delta_q = 10$. Note that if peer sampling is uniform, the distribution of the number of received models in a round is Poisson(1), which means that not receiving models for 10 rounds has a probability of $e^{-10} \approx 4.5 \cdot 10^{-5}$.

Algorithm 1 requires an implementation of the procedure UPDATE that computes the new approximations of Y^* and x_i. We will now elaborate on three different versions that implement different stochastic gradient update rules.

4.1. Update Rules for General Rank–k Factorization

Let us first consider the error function given in Equation (1) and derive an update rule to optimize this error function. The partial derivatives of $J(X, Y)$ by X and Y based on Equation (1) are

$$\frac{\partial J}{\partial X} = (XY^T - A)Y, \quad \frac{\partial J}{\partial Y} = (YX^T - A^T)X. \tag{6}$$

Since only x_i is available at node i, the gradient is calculated only w.r.t. x_i instead of X. Accordingly, the stochastic gradient update rule with a learning rate η can be derived by substituting x_i in the way shown in Algorithm 2. Although function J is not convex, it has been shown that all the local optima of J are also global [Srebro and Jaakkola 2003]. Thus, for a small enough η, any stable fix point of the dynamical system implemented by Algorithm 1 with the update rule in Algorithm 2 is guaranteed to be a global optimum.

In the case of real world applications such as recommender systems, where matrix A contains several undefined elements, the update rule of Algorithm 2 is insufficient. First of all, the gradient can be calculated only w.r.t. the elements of A that are known. Furthermore, to avoid overfitting the model to the known elements, we need to add regularization terms, as seen in Equation (4), to penalize extreme values in the model. We may also add bias to the model as shown in Equation (5). The partial derivatives based on Equation (5) are

$$\frac{\partial J}{\partial X} = (XY^T + b\mathbb{1} - A)Y + \alpha X, \quad \frac{\partial J}{\partial Y} = (YX^T + b\mathbb{1}^T - A^T)X + \alpha Y, \quad \frac{\partial J}{\partial b} = XY^T + b\mathbb{1} - A, \tag{7}$$

Algorithm 3 rank-k update at node i with regularization [and bias]

η ▷ learning rate
α ▷ regularization rate
▷ If bias is handled, the optional parts in [] should be added
procedure UPDATE($Y, x_i, a_i[, b_i]$)
  for all j, where a_{ij} is defined **do**
    $\text{err} \leftarrow a_{ij} - x_i Y_j^T \; [- b_i]$
    $x_i' \leftarrow (1 - \eta\alpha)x_i + \eta \cdot \text{err} \cdot Y_j$
    $Y_j' \leftarrow (1 - \eta\alpha)Y_j + \eta \cdot \text{err} \cdot x_i$
    $[b_i' \leftarrow b_i + \eta \cdot \text{err}]$
  end for
  return $(Y', x_i'[, b_i'])$
end procedure

where—for notational simplicity—we assume that all the values are defined. In practice, only those values that are defined are updated, so the gradient is evaluated only w.r.t. defined values. The regularized update rule (with optional bias) is shown in Algorithm 3.

4.2. Update Rule for Rank–k SVD

Apart from minimizing the error function given in Equation (1), let us now also set the additional goal that the algorithm converges to the SVD in the form of X^* and Y^*, as defined by Equation (3). This is indeed a harder problem: while (X^*, Y^*) minimizes (1), any other pair of matrices (X^*R^{-1}, Y^*R^T) will also minimize for any invertible matrix $R \in \mathbb{R}^{k\times k}$, and Algorithm 2 is free to converge to any of them.

From now on, we will assume that the non-zero singular values of A are all unique, and that the rank of A is at least k, that is, $\sigma_1 > \cdots > \sigma_k > 0$. This makes the discussion simpler, but these assumptions can be relaxed, and the algorithm is applicable even if these assumptions do not hold.

Our key observation is that any optimal rank-1 approximation $X_1 Y_1^T$ of A is such that $X_1 \in \mathbb{R}^{m\times 1}$ contains the (unnormalized) left singular vector of A that belongs to σ_1, the largest singular value of A. Similarly, Y_1 contains the corresponding right singular vector. This is because for any optimal rank-k approximation XY^T, there is an invertible matrix $R \in \mathbb{R}^{k\times k}$ such that $X = X^*R$ and $Y^T = R^{-1}Y^{*T}$[Srebro and Jaakkola 2003]. For $k = 1$ this proves our observation because, as defined in Section 2, $X^* \sim u_1$ and $Y^* \sim v_1$. Furthermore, for $k = 1$,

$$X_1 Y_1^T = X^* {Y^*}^T = \sigma_1 u_1 v_1^T, \tag{8}$$

which means that (using Equation (2)) we have

$$A - X_1 Y_1^T = \sum_{i=2}^{r} \sigma_i u_i v_i^T. \tag{9}$$

Thus, a rank-1 approximation of the matrix $A - X_1 Y_1^T$ will reveal the direction of the singular vectors corresponding to the second largest singular value σ_2. A simple approach based on these observations is to first compute $X_1 Y_1^T$, a rank-1 approximation of A. Subsequently, we compute a rank-1 approximation $X_2 Y_2^T$ of $A - X_1 Y_1^T$. The rank-2 approximation of A containing the first two left and right singular vectors is obtained as $[X_1, X_2][Y_1, Y_2]^T$ according to the above observations. We then repeat this procedure k times to get the desired decomposition X^* and Y^* by filling in one column at a time sequentially in both matrices.

Algorithm 4 rank-k SVD update at node i

η ▷ learning rate
procedure UPDATE(Y, x_i, a_i)
  $a'_i \leftarrow a_i$
  for ℓ = 1 to k **do** ▷ y_ℓ : column ℓ of Y
    $\text{err} \leftarrow a'_i - x_{i\ell} \cdot y_\ell^T$
    $x'_{i\ell} \leftarrow x_{i\ell} + \eta \cdot \text{err} \cdot y_\ell$
    $y'_\ell \leftarrow y_\ell + \eta \cdot \text{err}^T \cdot x_{i\ell}$
    $a'_i = a'_i - x_{i\ell} \cdot y_\ell^T$
  end for
  return (Y', x'_i)
end procedure

Algorithm 5 Iterative synchronized rank–k SVD

A ▷ The matrix to be factorized
η ▷ learning rate
initialize Y
initialize X
while not converged **do**
  $A' = A$
  for ℓ = 1 to k **do**
    $\text{err} \leftarrow A' - x_\ell \cdot y_\ell^T$ ▷ x_ℓ : column ℓ of X
    ▷ y_ℓ : column ℓ of Y
    $x'_\ell \leftarrow x_\ell + \eta \cdot \text{err} \cdot y_\ell$
    $y'_\ell \leftarrow y_\ell + \eta \cdot \text{err}^T \cdot x_\ell$
    $A' = A' - x_\ell \cdot y_\ell^T$
  end for
  $X = X'; \quad Y = Y'$
end while

A more efficient and robust approach is to let all rank-1 approximations in this sequential naive approach evolve at the same time. Intuitively, when there is a reasonable estimate of the singular vector corresponding to the largest singular value, the next vector can already start progressing in the right direction, and so on. This idea is implemented in Algorithm 4.

4.3. Synchronized Rank–k SVD

As a baseline method in our experimental evaluation, we will use a synchronized version of Algorithm 1 with the update rule in Algorithm 4. In this version (shown in Algorithm 5), the rank-1 updates are done over the entire matrix A at once, and there is only one central version of the approximation of Y as opposed to the several independent versions we had previously. As in Algorithm 1, the matrices X and Y are initialized with uniform random values from $[0, 1]$. Note that in this algorithm, we require the columns of X and hence use a different notation: x_ℓ denotes the ℓ-th column, not the ℓ-th row.

Although formulated as a centralized sequential algorithm, this algorithm yields a simple MapReduce implementation, where the mappers compute parts of the gradients (e.g., the gradients of the rows of A as in Algorithm 4), while a single reducer sums up the components of the gradient and executes the update steps.

5. EXPERIMENTS

In our first set of experiments, we demonstrate various properties of our algorithm including convergence speed, scalability and robustness. Our test matrices include stan-

Table I. The main properties of the real data sets

	Iris	Pendigits	Segmentation
Number of instances (m)	150	10992	2310
Number of features (n)	4	16	19
Minimal k such that $\sum_{i=1}^{k} \sigma_i^2 / \sum_{i=1}^{n} \sigma_i^2 > 0.9$	2	10	5

Table II. Overview of the algorithm variants

	Singular Value Decomposition (SVD)	Low-Rank Decomposition (LRD)	LRD with Regularization (RLRD)
Decentralized Stochastic Gradient descent (DSG)	DSG-SVD Algorithms 1 and 4	DSG-LRD Algorithms 1 and 2	DSG-RLRD Algorithms 1 and 3
Stochastic Gradient descent (SG)	SG-SVD one rand. walk + Alg. 4	—	SG-RLRD one rand. walk + Alg. 3
batch Gradient descent (G)	G-SVD Algorithm 5	—	—

dard machine learning data sets as well as synthetic matrices with controllable properties. The simulated network environment has been simplified so that we can focus on the algorithmic properties under different parameter settings. A realistic trace-based case study is presented in Section 6.

In the case of the distributed algorithms, the number of nodes in the network is the same as the number of rows of the matrix to be factorized, since every node has exactly one row of the matrix. We used the PeerSim [Montresor and Jelasity 2009] simulator with the event-based engine, and we implemented the peer sampling service by the NEWSCAST [Tölgyesi and Jelasity 2009] protocol. Our implementation is publicly available.[1]

5.1. Algorithms

First we provide names for the algorithms we include in our experiments. Table II gives an overview of the algorithms and how they are combined from the different components.

Algorithm 1 with the update rule of Algorithm 4 (our main contribution) will be referred to as Decentralized Stochastic Gradient SVD (DSG-SVD). Replacing Algorithm 4 with Algorithm 2 we get Decentralized Stochastic Gradient Low Rank Decomposition (DSG-LRD). Recall that this algorithm will converge to a rank-k decomposition that is not necessarily the SVD of A. Algorithm 5 will be called Gradient SVD (G-SVD). Recall that this batch algorithm can be parallelized: for example, the gradient can be calculated row by row and then summed up to get the full gradient.

Note that the algorithm variant that applies the update rule in Algorithm 3 is not tested here; in this section we assume that all the matrices are fully defined. The regularized version of the method as tested in our recommender system case study and is described in Section 6.

Finally, we introduce a baseline algorithm, Stochastic Gradient SVD (SG-SVD). This algorithm uses the update rule of Algorithm 4, but we have only a single approximation Y at any point in time, and there is only one process, which repeatedly gets random rows of A and then applies the update rule in Algorithm 4 to the current approximation Y and the corresponding row of X.

[1] http://rgai.inf.u-szeged.hu/download/p2p14/svdsrc.zip,
https://github.com/RobertOrmandi/Gossip-Learning-Framework/tree/multicore

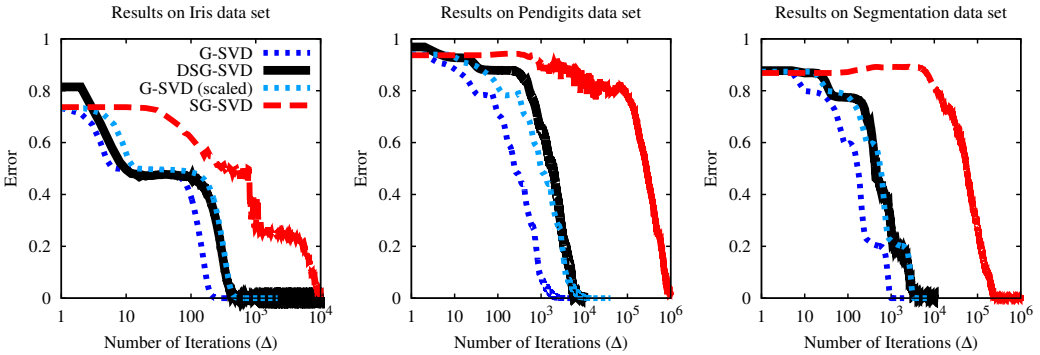

Fig. 1. Convergence on the real data sets. Error is based on cosine similarity. In the scaled version of G-SVD, the number of iterations is multiplied by $\log_{10} m$ (see text).

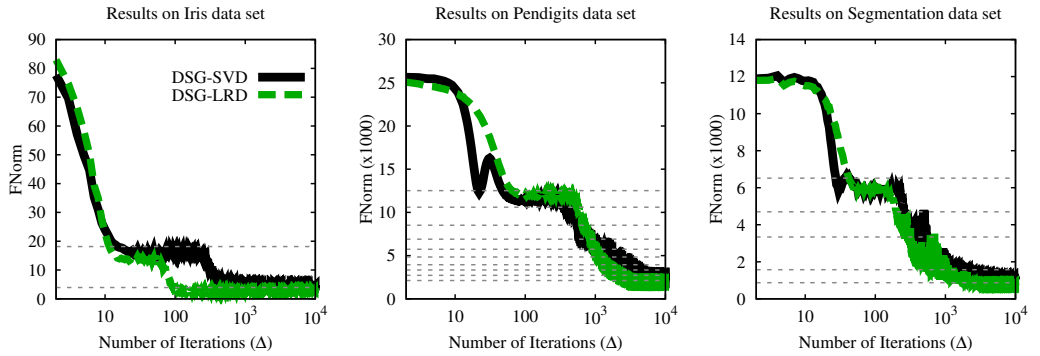

Fig. 2. Convergence on the real data sets. Error is based on the Frobenius norm. Horizontal dashed lines in top-down order show the FNORM value for the optimal rank-i approximations for $i = 1, \ldots, k$.

5.2. Error measures

Cosine similarity. To measure the difference between the correct and the approximated singular vectors, we used cosine similarity, because it is not sensitive to the scaling of the vectors (recall that our algorithm does not guarantee unit-length columns in X and Y). The formula for measuring the error of a rank-k decomposition XY^T is

$$\text{Error}(X, Y) = \frac{1}{2k} \sum_{i=1}^{k} 1 - \left| \frac{y_i^T v_i}{\|y_i\|} \right| + 1 - \left| \frac{x_i^T u_i}{\|x_i\|} \right|, \tag{10}$$

where x_i, y_i, u_i and v_i are column vectors of X, Y, U_k and V_k, respectively. Matrices U_k and V_k are the orthogonal matrices defined in Section 2.

Frobenius norm. Another measure of error is given by function $J(X, Y)$ defined in Equation (1). The advantage of this measure is that it can be applied to Algorithm 2 as well. However, this error measure obviously does not tell us whether the calculated matrices X and Y contain scaled singular vectors or not; it simply measures the quality of the rank-k approximation. We call this error measure FNORM.

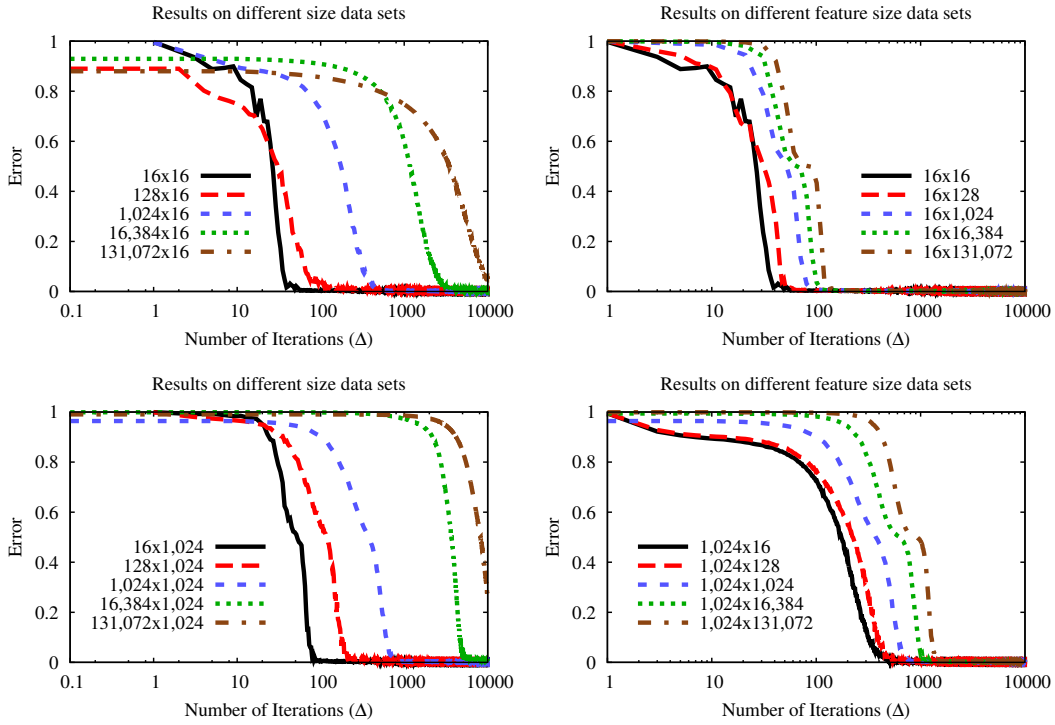

Fig. 3. Results on synthetic data sets using networks of different dimensions. We set $k = 1$, and all the matrices had a rank of 16.

5.3. Data sets

Synthetic data. We included experiments on synthetic data so that we could evaluate the scalability and the fault tolerance of our method in a fully controlled way. We first generated random singular vectors producing matrices U and V with the help of the butterfly technique [Genz 1999]. Since matrices to be decomposed often originate from graphs, and since the node degrees and the spectrum of real graphs usually follow a power law distribution [Mihail and Papadimitriou 2002; Chung et al. 2003], the expected singular values in the diagonal of Σ were generated from a Pareto distribution with parameters $x_m = 1$ and $\alpha = 1$. This way, we construct a matrix $A = U\Sigma V^T$ where we know and can control the singular vectors and values.

Real data. These matrices were constructed from data sets taken from the well-known UCI [Bache and Lichman 2013] machine learning repository. In these applications, the role of SVD is dimensionality reduction and feature extraction. The selected data sets are the Iris, the Pendigits (Pen-Based Recognition of Handwritten Digits) and the Segmentation (Statlog Image Segmentation) data sets. Parameter k was set so that the approximation has at least 90% of the information in the data set. That is, we set the minimal k such that $\sum_{i=1}^{k} \sigma_i^2 / \sum_{i=1}^{n} \sigma_i^2 > 0.9$ [Alpaydin 2010]. Table I illustrates the main properties of the selected data sets. In order to be able to compute the error over these matrices, we computed the exact singular value decomposition using the Jama [Nis 1999] library.

5.4. Convergence

The experimental results are shown in Figures 1 and 2. Here, we tested the convergence speed of the algorithms over the real data sets with the parameter k set according to Table I.

Figure 1 illustrates the deviation from the exact singular vectors. G-SVD is the fastest to converge, but it either needs a central database to store A or it requires a synchronized master-slave communication model when parallelized. We have also included a reference curve that is calculated by scaling the number of iterations by $\log_{10} m$ for G-SVD. The motivation is a more scalable potential implementation of G-SVD in which there is a hierarchical communication structure in place where nodes propagate partial sums of the gradient up a tree (with a branching factor of 10) instead of each node communicating with a central server as in [Benson et al. 2013]. This curve almost completely overlaps with that of DSG-SVD, our fully distributed robust algorithm.

We also illustrate the speedup w.r.t. SG-SVD. The major difference between SG-SVD and DSG-SVD is that in DSG-SVD there are m different approximations of Y, all of which keep updating the rows of X simultaneously, while in SG-SVD there is only one version of Y. Other than that, both algorithms apply exactly the same gradient update rule. In other words, in DSG-SVD any row of X experiences m times as many updates per unit time as in SG-SVD.

Figure 2 illustrates the difference between DSG-LRD and DSG-SVD. Both optimize the same error function (which is shown on the plots), however, DSG-SVD introduces the extra constraint of looking for the singular vectors of A. Fortunately this extra constraint does not slow down the convergence significantly in the case of the data sets we examined, although it does introduce a bumpier convergence pattern. The reason is that the singular vectors converge in a sequential order, and the vectors that belong to smaller singular values might have to be completely re-oriented when the singular vectors that precede them in the order of convergence have converged.

5.5. Scalability

To evaluate the scalability of DSG-SVD, we generated a range of synthetic matrices of various sizes using the method described earlier. Figure 3 shows the results of our experiments. Clearly, the method is somewhat more sensitive to varying the number of nodes (that is, to varying the first dimension m) than to varying the second dimension n (the length of a row). This is not surprising as the full row of A is always used in a single update step irrespective of its length, whereas a larger m requires visiting more nodes, which requires more iterations.

However, when m is large, we can apply *sampling techniques* [Drineas et al. 2006]. That is, we can consider only a small sample of the network drawn uniformly or from an appropriately biased distribution and calculate a high quality SVD based on that subset. To illustrate such a sampling technique, we implemented uniform sampling. When we wish to select a given proportion p of the nodes, each node decides locally about joining the sample with a probability p. The size of the resulting sample will follow the binomial distribution $B(N, p)$.

Figure 4 shows our experimental results with $p = 1/2$ and $p = 1/3$. Clearly, while communication costs overall are decreased proportionally to the sample size, on our benchmark both precision and convergence speed remain almost unchanged. The only exception is the Iris data set. However, this is a relatively small data set over which the variance of our uniform sampling method is relatively high (note, for example, that the run with $p = 1/3$ resulted in a better performance than with $p = 1/2$).

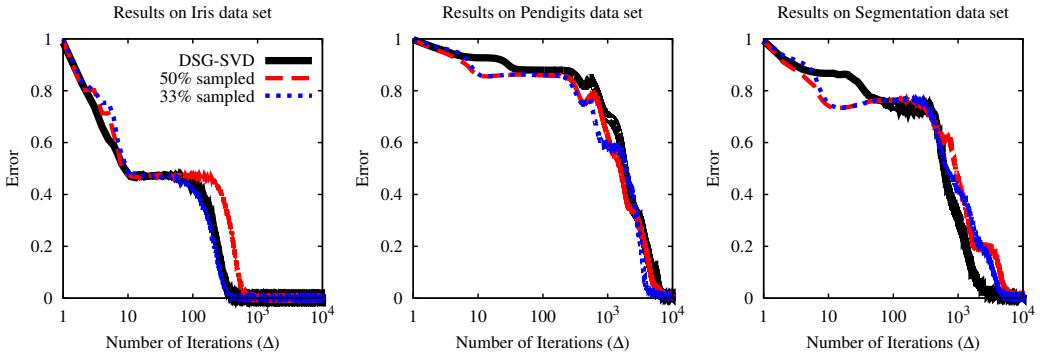

Fig. 4. Results when only the 50/33% randomly sampled instances were used from the data set.

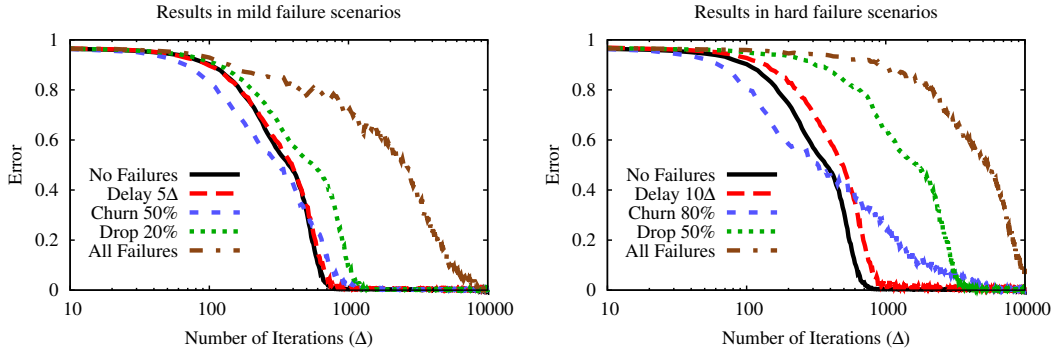

Fig. 5. Results in different failure scenarios using a 1024×1024 synthetic matrix with a rank of 16. We set $k = 1$.

5.6. Failure Scenarios

We used two different failure scenarios: a mild and a hard one. In the two scenarios, message delay was drawn uniformly at random from between Δ and 5Δ or 10Δ time units, respectively. Messages were dropped with a probability of 20% or 50%, respectively.

Node churn was modeled based on statistics from a BitTorrent trace [Roozenburg 2006] as well as known empirical findings [Stutzbach and Rejaie 2006]. We draw the online session length for each node independently from a lognormal probability distribution with parameters $\mu = 5\Delta$ and $\sigma = 0.5\Delta$. Offline session lengths are determined implicitly by fixing the number of nodes that are offline at the same time. For the two scenarios, 50% or 80% of the nodes were offline at the same time, respectively.

The results of the algorithms in these extreme failure scenarios can be seen in Figure 5. As we observe, the different types of failures (whether examined separately or taken together) only delay, but not prevent, convergence. This is in sharp contrast with competing approaches that require synchronization, such as G-SVD.

An interesting observation is that when only churn is modeled, at the beginning of the simulation convergence is actually faster. This effect is due to the fact that—since most of the nodes are offline—the effective network is smaller, but this small sample still allows for an approximation of the SVD. However, convergence eventually

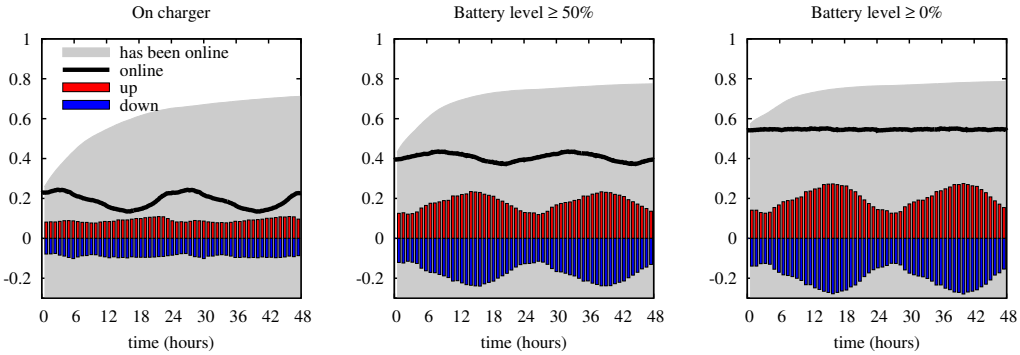

Fig. 6. Proportion of users online, and proportion of users that have been online, as a function of time. The bars indicate the proportion of the simulated users that log in and log out (shown as a negative proportion), respectively, in the given period. For a comparison, battery level thresholds of 50% and 0% are also shown.

slows down as full precision cannot be achieved without taking most of the nodes into account in the particular matrix we experimented with.

6. TRACE-BASED EXPERIMENTS WITH A RECOMMENDER SYSTEM

Here, our goal is to demonstrate that the proposed method can be applied to real-world collaborative movie recommendation data. In order to achieve our goal, we simulate node churn based on a real trace of smartphone user behavior. We also ensure that we use the phone only when it is connected to a charger, so as to save the battery. Based on the well-known MovieLens datasets we calculate the message size (the size of matrix Y) and we take into account the bandwidth limitations of smartphone connections as well. As we will show, even on the largest available dataset of about 70,000 users, our best algorithm variant achieves a competitive performance in 1-2 hours without excessive bandwidth utilization, and reasonable prediction performance is attained much sooner. To sum up, our setup is realistic, battery friendly, and the performance is completely acceptable for a recommender system.

6.1. Trace Properties

The trace we used was collected by a locally developed openly available smartphone app called STUNner [Berta et al. 2014]. In a nutshell, the app monitors and collects information about charging status, battery level, bandwidth, and NAT type.

We have traces of varying lengths taken from 1191 different users. We divided these traces into 2-day segments (with a one-day overlap), resulting in 40,658 segments altogether. With the help of these segments, we were able to simulate a virtual 2-day period by assigning a different segment to each simulated node. When we needed more users than segments, we re-sampled the segments to inflate the number of users artificially.

To ensure our algorithm is phone and user friendly, we defined a user to be online (available) when the user has a network connection and the phone is connected to a charger, hence we never use battery power at all. In addition, we also treated those users as offline who had a bandwidth of less than 1 Mbit/s, and we filtered out the online sessions that lasted less than a minute as well.

The observed churn pattern is illustrated in Figure 6 based on all the 2-day periods we identified. Although our sample contains users from all over the world, they are mostly from Europe, and some are from the USA. The indicated time is GMT, thus we did not convert times to local time. It is interesting to note that if we consider only network availability (any battery level allowed), then the diurnal pattern becomes

Table III. The main properties of the MovieLens data sets and algorithm parameters

	100k	1m	10m
Number of users (m)	943	6,040	69878
Number of movies (n)	1682	3706	10677
Number of ratings	100,000	1,000,209	10,000,054
Density	0.059	0.042	0.014
Training / Test	90.57% / 9.43%	93.96% / 6.04%	93.01% / 6.99%
Time period	20.09.97 - 22.04.98	25.04.00 - 28.02.03	09.01.95 - 05.01.09
$\eta/\alpha/k$	$10^{-2}/0.1/5$	$10^{-2}/0.1/5$	$10^{-2}/0.1/5$
message size	0.5 Mbit	1.2 Mbit	3.4 Mbit
Δ for DSG-RLRD(/10)	60s (6s)	60s (6s)	60s (6s)
Δ_q for DSG-RLRD(/10)	10 (200)	10 (150)	10 (100)
$\Delta\Delta_q$ for HOTPOTATO	1200s	900s	600s

apparent in the login and logout frequencies due to short session lengths during the day, and long sessions during the night. The network availability itself is static. If we require the phone to be on a charger, then the diurnal pattern of availability becomes clear. During the night, more phones are available (as they tend to be on a charger), but the churn rate remains lower. Note that during the simulated 2-day period, about 30% of the users remain permanently offline based on our definition.

6.2. Data Sets and Algorithm Parameters

We used the well known and widely used MovieLens data sets [Resnick et al. 1994][2]. In these data sets, users rate movies from one to five stars that define the user-item matrix we wish to decompose. We divided the data sets into training and test sets with the script proposed by the authors of the data sets. The properties of the data sets can be seen in Table III. The dimensions of the user-item matrices to be factorized are determined by the number of users (the rows) and the number of movies (the columns).

Obviously, in this machine learning application we measured prediction error as opposed to the approximation error of the entire matrix (i.e., Equation (1)). In our experiments prediction error was defined as the root mean squared error (RMSE) over the matrix entries in the test set.

The learning rate (η) and the regularization rate (α) were set to standard values based on preliminary experiments for each data set. Figure 7 illustrates these preliminary experiments. The plots are based on the SG-RLRD algorithm, which is identical to SG-SVD except that we use Algorithm 3 as the update rule (no orthogonalization). The algorithm passed over the whole dataset 100 times. The center of each heatmap shows the values we selected to be used in the experiments (as shown in Table III). It is clear that increasing η improves the performance up to a point where the algorithm becomes unstable and we no longer have convergence. Increasing α stabilizes the convergence, but reduces the performance. Increasing the network size favors smaller learning rates more, since we have more updates during the 100 passes over the data.

An interesting question that remains is how to set parameter k for our low-rank decomposition algorithms. In Figure 7, the RMSE for various values of k is shown (right) by using a standard matrix decomposition algorithm to optimize Equation (4). As we observe, after a certain point, increasing k has only a small effect on the prediction performance. For this reason in our experiments we set $k = 5$ as a good compromise between prediction performance and communication costs.

Table III shows the message sizes associated with the three data sets. In all the experiments, we send each message at a rate of 1 Mbit/s. This sets an upper bound on bandwidth utilization, because only one message is sent at a time by a node ac-

[2]Data sets were downloaded from: http://grouplens.org/datasets/movielens/

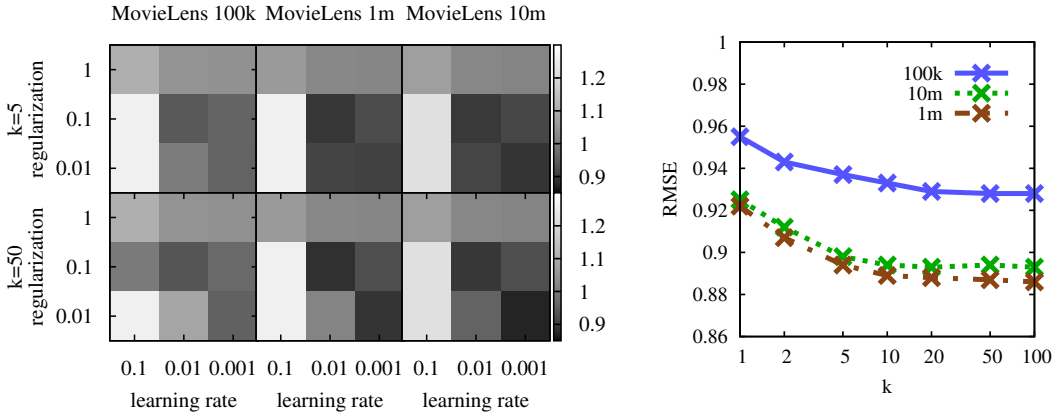

Fig. 7. RMSE with various hyperparameters (learning rate (η), regularization (α), rank (k)) on the MovieLens data sets (on the heatmaps a darker color represents a better performance).

cording to each algorithm variant. However, the actual bandwidth utilization in our experiments was set lower than 100 KBit/s. To control the bandwidth, a lower bound can be set using the parameters Δ and Δ_q. Recall that the effect of these parameters is that if no message arrives for $\Delta\Delta_q$ seconds then the restart mechanism is triggered (a new random walk is started). By limiting the maximum waiting time we control the average waiting time, which in turn determines the bandwidth utilization.

6.3. Algorithms

The first algorithm variant we experimented with is DSG-RLRD with bias. As shown in Table II, DSG-RLRD is Algorithm 1 with the update rule in Algorithm 3. Here, we set Δ to one minute. As we shall see, this variant performed rather poorly in our realistic simulation scenario. Hence, we introduce here a number of additional variants with the goal of speeding up convergence.

Our first idea is a slight modification of DSG-RLRD. In this variant, we initiate only $m/10$ models (random walks) in the network (the original version starts m models), but we reduce the period (round length) to $\Delta/10$. This results in the same average network traffic, while the fewer models perform faster walks. We will call this variant DSG-RLRD/10. The restarting technique in Algorithm 1 is of key importance for DSG-RLRD/10 because there are fewer models in the system. Here, we set the parameter Δ_q as shown in Table III. This parameter was set by manual optimization so that the actual overall amount of traffic in the system is the same as that of DSG-RLRD.

Our second modified version is called HOTPOTATO. This variant still uses Algorithm 3 as an update rule, but instead of running Algorithm 1 the nodes immediately forward each model they receive. To avoid saturating the bandwidth, and to recover failed random walks, in HOTPOTATO we also need to control the number of models that circulate in the network to get approximately the same average bandwidth—and thus fewer random walks, since here random walks have a maximum speed—as in the other two algorithm variants. We initiated $m/10$ models and adapted the same restarting idea as in Algorithm 1 with the same timeout as used by DSG-RLRD/10. Our preliminary results confirmed that this resulted in approximately the same amount of traffic.

Our third idea is to apply continuous *merging* inspired by [Ormándi et al. 2013]. That is, we modify Algorithm 1 so that, instead of a queue of received models, we now

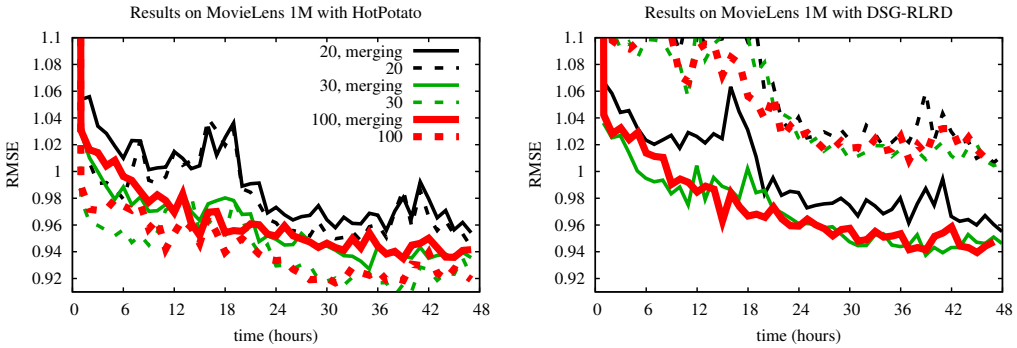

Fig. 8. The effect of the number of neighbors (20, 30 and 100 neighbors) per node.

have exactly one local model at all times. When a model is received, it is averaged with the local model, and the average will become the new local approximation. In addition, we also keep track of how many models are received (to model the size of the queue). This results in exactly the same communication pattern as that of Algorithm 1, yet the models become rather homogeneous over the network: the effect of each update is spread to each model quickly. Merging can be applied to any of the algorithm variants above, with identical parameters.

Although in this study we needed to invest some effort into setting Δ_q so that all the variants generate similar traffic, in practice Δ_q is not a sensitive parameter. As mentioned above, it simply guarantees a lower bound on the average number of messages per round. In practice the actual average bandwidth per node used in the experiments was less than 100 KBit/s.

As a baseline, we also include the batch gradient algorithm that simulates a centralized solution. Here, in each round, each online node uploads its gradients based on its current approximation of x_i, Y, and b_i to a central server. The server then calculates the sum of the gradients and sends the sum back to all the nodes. The nodes then update their local model. We set the parameters of the batch gradient algorithm so that the same amount of data is transferred by the nodes as in the other algorithms (not counting the communication of the server, which would double the communication complexity). The learning rate here was $\eta = 10^{-4}$.

6.4. Peer Sampling

In these experiments, for simplicity, the peer sampling service is assumed to be based on a static network, in which every node has a fixed number of random neighbors out of the whole population. Most of these neighbors are offline at any point in time (see Figure 6). This is a realistic setup on the real Internet, since in the case of stable connections, it is more economical to perform NAT traversal potentially with the assistance of a server in the connection phase. A more sophisticated peer sampling service could also repair the network [Roverso et al. 2013]. Figure 8 shows our simulations performed to explore the effect of the number of neighbors. We can quite clearly see bumps in the performance for a neighborhood size of 20. These are caused by daily periodicity: overnight, fewer users are online, and many of them are newly joined nodes (see Figure 6). Higher neighborhood sizes smooth these periods out. Based on this finding, we opted for 100 random neighbors per node as with this setting there are enough online neighbors in our churn scenario to allow for the random walks to proceed.

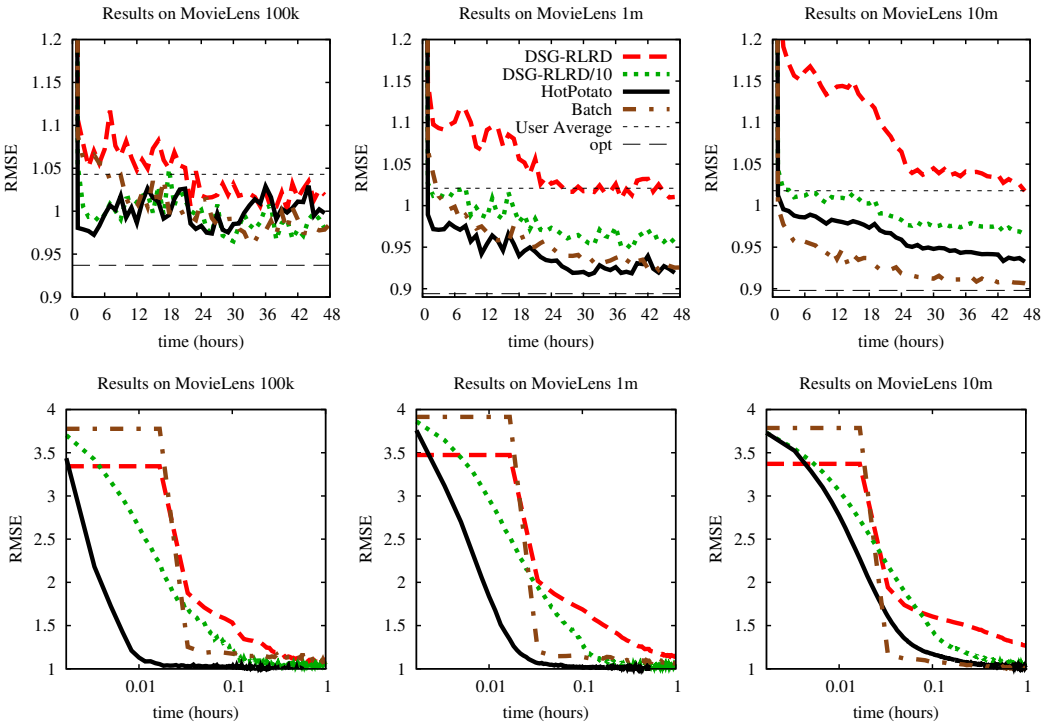

Fig. 9. Results on the MovieLens data sets without the merging technique. The bottom row shows the first hour of the same data on log-scale. User Average shows the error of always predicting the average rating for each user.

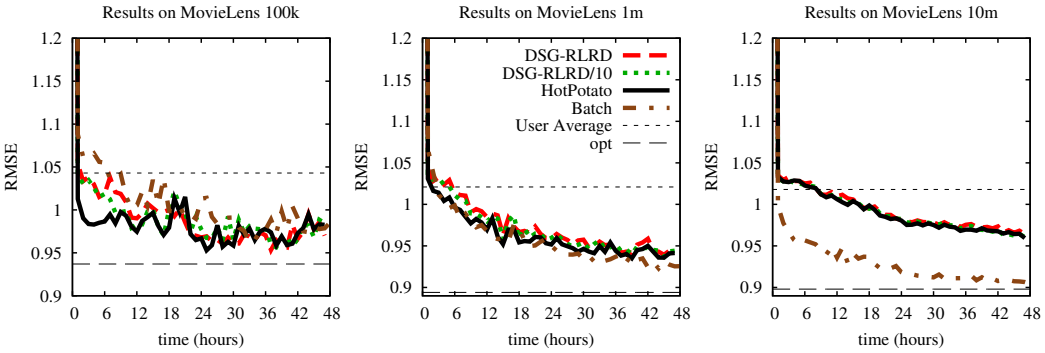

Fig. 10. Results on the MovieLens data sets with the merging technique. User Average shows the error of always predicting the average rating for each user.

6.5. Results

The results of our experiments are shown in Figures 9 and 10. The average of three runs is shown for all the parameter settings. Note that at any given point in time, the RMSE is calculated over the part of the testing set that resides at nodes online at that point in time. We calculate RMSE for online nodes by using the local model of the node. This is motivated by the fact that we wish to provide a service to online nodes, i.e., the nodes that participate in the learning process. Even this way, the fastest algorithms approximate the prediction performance of the model that was trained over the entire dataset offline (shown as the "opt" level) as well as that of the batch model.

In the case of the algorithm variants without merging, it is apparent that, with the same communication budget, it is preferable to have fewer models in the network that move faster. On a larger network DSG-RLRD becomes slower because its slow random walks are able to visit only a smaller part of the network. HOTPOTATO performs best in each scenario, which clearly indicates that, at least during the first two days, it is much better to try and visit as many nodes as possible and sacrifice the redundancy of the other variants that maintain more, albeit slower, random walks.

Figure 10 contains our results with the merging variants. The results are quite striking: due to the spreading of the effect of each update by the averaging technique, all the variants become almost identical. This is because each variant in effect now does the same thing: it calculates the update starting from a very similar model Y at each node, and spreads the effect of the update to every other node through a gossip algorithm. The variants differ in the details of the averaging process, but since all the three approaches are similarly effective due to the quick broadcast implemented by gossip, the difference is not significant in our experiments. The periodic variants clearly benefit from averaging, as can be seen also in Figure 8. Still, HOTPOTATO without merging slightly outperforms the variant with merging, which makes it the best choice over the scenarios and time frames (both short and long) we examined.

Even in the case of the largest message size, we utilize about 6% of the bandwidth on the lowest-bandwidth nodes. Increasing bandwidth utilization clearly results in a linear speedup; that is, we could afford to run the algorithm 10 times faster at the cost of using 10 times more bandwidth, which would still be within the affordable range.

Since the actual application scenario of a recommender system involves continuous model training and update, the actual bandwidth utilization could be reduced even further, given that user preferences do not change dramatically over short periods.

7. PRIVACY CONSIDERATIONS

In our computational model, private information never leaves the owner's device. We argued that this is a *necessary condition* of privacy in practice, otherwise we either have to trust a server, which we wish to avoid, or else we need to rely on cryptographic techniques that are not yet practical enough for an actual application of the scale and type we target [Nikolaenko et al. 2013].

Evidently, decentralization is not a *sufficient condition* of privacy. There are, however, promising results on decentralized privacy. Yan et al. have shown [Yan et al. 2013] that a similar peer-to-peer algorithm for online convex optimization has intrinsic privacy-preserving properties that are derived from the properties of the underlying communication topology. That is, the gradients of the local function cannot be reconstructed for various topologies of user communication graphs. In their case, the full model was transmitted, while for matrix factorization, the user vectors are kept locally. The inaccessibility of the user vectors and the non-convexity of the function to be optimized makes it even more difficult to reconstruct the gradients.

Wang et al. show a stronger theoretical result [Wang et al. 2015] in a slightly different context, namely that stochastic gradient updates actually implement the theoretical definition of differential privacy [Dwork 2011], which is the state-of-the-art approach in privacy discussions.

However, if we use a peer sampling service that is not itself secure, then it might become possible to isolate any node and learn both its input and output in the same round. This problem is targeted in our parallel ongoing study [Birman et al. 2015] where peer sampling is de-randomized and the continuous observation of both the input and output of a node becomes very hard. For privacy considerations, we should rely on static networks set up by a peer sampling server, as was done in Section 6, because in that case only statistical attacks are possible: one needs to compromise a sufficient number of devices so as to have a reasonable chance of collecting the input and output of a given node in the same round. Such statistical attacks can be made prohibitively expensive in a very large network, unless joining the network is free and uncontrolled.

One can apply additional methods to harden privacy. Another idea we have developed [Danner and Jelasity 2015] involves mini-batch gradient search, in which we create a relatively small group of nodes in every update where a certain novel secure multiparty computation algorithm is executed to calculate the sum of the gradients that is also robust to failure. This ensures that the members of the group become indistinguishable. Similar algorithms have been proposed in [Ahmad and Khokhar 2006].

While in the distributed algorithm the input data is better protected than in a centralized approach, the recommendations based on the final model can be still used to infer something about the original data. For centralized algorithms the standard approach to ensure differential privacy is to add noise to the gradient (see e.g., [Berlioz et al. 2015] for differentially private matrix factorization). Noise can be added for the distributed algorithms as well, in a similar manner (see e.g., [Huang et al. 2015]). In this case the added noise protects not only the output model from deanonymization, but the communication as well. An important difference between centralized and distributed algorithms with added noise is that while the output model is protected in both cases, the original data may still have a single point of failure in the centralized approach.

Overall, we argue that keeping private information local is a practical necessity, and with additional measures privacy can be improved without sacrificing practicality.

8. CONCLUSIONS

In this paper, we proposed an SGD algorithm with an update rule that has stable fix points only in the SVD of a matrix A. The output of the algorithm for rank k are two matrices X and Y that contain scaled versions of the first k left and right singular vectors of A, respectively. Matrices X and Y are unique apart from the scaling of the columns. We also proposed regularized and non-regularized versions of low-rank matrix decomposition techniques that can cope with partially defined matrices as well, by calculating the gradient only over the matrix entries that are known.

The most important feature of the algorithms is that they simultaneously achieve practical levels of privacy, efficiency and robustness. We operate over fully distributed data where each network node stores one full row of A. The output matrix X is also fully distributed: node i that stores row i of A computes row i of X, the private model for the node. Matrices A and X are private in that only the node that stores a given row has access to it. A version of the matrix Y is available in full at all nodes. Through experimental evaluation we studied the convergence speed of the algorithm, and demonstrated that it is competitive with other gradient methods that require more freedom

for data access. We also demonstrated the remarkable robustness of the method in extreme failure scenarios.

We made clear the practicality of our approach using a case study where we simulated a recommender system over a trace collected from real smartphone users. We made sure that our application respected the bandwidth limitations (we used at most 6% of the available bandwidth, usually less) and the battery was also protected by just using devices that are plugged in to a charger. We demonstrated that the algorithm attains close to optimal prediction accuracy in a reasonable time in a realistic scenario.

Our ongoing work includes exploiting the potential sparsity of the matrix. In recommender systems, the user item matrix is typically very sparse, for example. One idea in this direction is to only communicate the parts of the local approximations that were changed during the update step. Exploiting sparsity would be a first step to improve the scalability of the algorithm in terms of the number of features (or items in recommender applications) as well.

REFERENCES

Dimitris Achlioptas and Frank McSherry. 2005. On Spectral Learning of Mixtures of Distributions. In *Proc. 18th Annual Conference on Learning Theory (COLT)*. 458–469.

Waseem Ahmad and Ashfaq Khokhar. 2006. Secure Aggregation in Large Scale Overlay Networks. In *IEEE Global Telecommunications Conference (GLOBECOM '06)*. DOI:http://dx.doi.org/10.1109/GLOCOM.2006.315

Ethem Alpaydin. 2010. *Introduction to Machine Learning* (2nd ed.). The MIT Press.

Yossi Azar, Amos Fiat, Anna R. Karlin, Frank McSherry, and Jared Saia. 2001. Spectral Analysis of Data. In *Proc. 33rd Symposium on Theory of Computing (STOC)*. 619–626.

K. Bache and M. Lichman. 2013. UCI Machine Learning Repository. (2013). http://archive.ics.uci.edu/ml

Austin R. Benson, David F. Gleich, and James Demmel. 2013. Direct QR factorizations for tall-and-skinny matrices in MapReduce architectures. *CoRR* (2013).

Arnaud Berlioz, Arik Friedman, Mohamed Ali Kaafar, Roksana Boreli, and Shlomo Berkovsky. 2015. Applying Differential Privacy to Matrix Factorization. In *Proceedings of the 9th ACM Conference on Recommender Systems*. ACM, 107–114.

Michael W. Berry, Susan T. Dumais, and Gavin W. O'Brien. 1995. Using Linear Algebra for Intelligent Information Retrieval. *SIAM Rev.* 37, 4 (1995), 573–595.

Árpád Berta, Vilmos Bilicki, and Márk Jelasity. 2014. Defining and Understanding Smartphone Churn over the Internet: a Measurement Study. In *Proceedings of the 14th IEEE International Conference on Peer-to-Peer Computing (P2P 2014)*. IEEE. DOI:http://dx.doi.org/10.1109/P2P.2014.6934317

Ken Birman, Márk Jelasity, Robert Kleinberg, and Edward Tremel. 2015. Building a Secure and Privacy-Preserving Smart Grid. *ACM SIGOPS Operating Systems Review* 49, 1 (Jan. 2015), 131–136. DOI:http://dx.doi.org/10.1145/2723872.2723891

Cheng-Tao Chu, Sang Kyun Kim, Yi-An Lin, YuanYuan Yu, Gary Bradski, Andrew Y. Ng, and Kunle Olukotun. 2007. Map-Reduce for Machine Learning on Multicore. In *Advances in Neural Information Processing Systems 19 (NIPS 2006)*, B. Schölkopf, J. Platt, and T. Hoffman (Eds.). MIT Press, 281–288.

Fan Chung, Linyuan Lu, and Van Vu. 2003. Eigenvalues of random power law graphs. *Annals of Combinatorics* 7, 1 (2003), 21–33.

Gábor Danner and Márk Jelasity. 2015. Fully Distributed Privacy Preserving Mini-batch Gradient Descent Learning. In *Proceedings of the 15th IFIP International Conference on Distributed Applications and Interoperable Systems (DAIS 2015) (Lecture Notes in Computer Science)*, Alysson Bessani and Sara Bouchenak (Eds.), Vol. 9038. Springer International Publishing, 30–44. DOI:http://dx.doi.org/10.1007/978-3-319-19129-4_3

Petros Drineas, Alan Frieze, Ravi Kannan, Santosh Vempala, and V. Vinay. 2004. Clustering large graphs via the Singular Value Decomposition. *Machine Learning* (2004), 9–33.

Petros Drineas, Ravi Kannan, and Michael W. Mahoney. 2006. Fast Monte Carlo Algorithms for Matrices II: Computing a Low-Rank Approximation to a Matrix. *SIAM J. Comput.* 36, 1 (2006), 158–183. DOI:http://dx.doi.org/10.1137/S0097539704442696

Petros Drineas, Iordanis Kerenidis, and Prabhakar Raghavan. 2002. Competitive Recommendation Systems. In *Proc. 34th Symposium on Theory of Computing (STOC)*. 82–90.

Cynthia Dwork. 2011. A firm foundation for private data analysis. *Commun. ACM* 54, 1 (Jan. 2011), 86–95. DOI:http://dx.doi.org/10.1145/1866739.1866758

Rainer Gemulla, Erik Nijkamp, Peter J. Haas, and Yannis Sismanis. 2011. Large-scale Matrix Factorization with Distributed Stochastic Gradient Descent. In *Proc. 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*. ACM, 69–77. DOI:http://dx.doi.org/10.1145/2020408.2020426

Alan Genz. 1999. Methods for generating random orthogonal matrices. In *Proc. Monte Carlo and Quasi-Monte Carlo Methods (MCQMC 1998)*, H. Niederreiter and J. Spanier (Eds.). Springer, 199–213.

Genevieve Gorrell. 2006. Generalized Hebbian Algorithm for Incremental Singular Value Decomposition in Natural Language Processing.. In *Proc. 11th Conference of the European Chapter of the Association for Computational Linguistics (EACL)*, Diana McCarthy and Shuly Wintner (Eds.). The Association for Computer Linguistics.

Naiyang Guan, Dacheng Tao, Zhigang Luo, and Bo Yuan. 2012a. NeNMF: An Optimal Gradient Method for Nonnegative Matrix Factorization. *IEEE Transactions on Signal Processing* 60, 6 (2012), 2882–2898. DOI:http://dx.doi.org/10.1109/TSP.2012.2190406

Naiyang Guan, Dacheng Tao, Zhigang Luo, and Bo Yuan. 2012b. Online Nonnegative Matrix Factorization With Robust Stochastic Approximation. *IEEE Trans. Neural Netw. Learning Syst.* 23, 7 (2012), 1087–1099. DOI:http://dx.doi.org/10.1109/TNNLS.2012.2197827

Zhenqi Huang, Sayan Mitra, and Nitin Vaidya. 2015. Differentially private distributed optimization. In *Proceedings of the 2015 International Conference on Distributed Computing and Networking*. ACM, 4.

Sibren Isaacman, Stratis Ioannidis, Augustin Chaintreau, and Margaret Martonosi. 2011. Distributed Rating Prediction in User Generated Content Streams. In *Proc. Fifth ACM Conf. on Rec. Sys.* ACM, 69–76. DOI:http://dx.doi.org/10.1145/2043932.2043948

Ravindran Kannan, Hadi Salmasian, and Santosh Vempala. 2005. The Spectral Method for General Mixture Models. In *Proc. 18th Annual Conference on Learning Theory (COLT)*. 444–457.

David Kempe and Frank McSherry. 2004. A decentralized algorithm for spectral analysis. In *Proc. 36th Symposium on Theory of Computing (STOC)*. ACM, 561–568.

Jon Kleinberg. 1999. Authoritative sources in a hyperlinked environment. *J. ACM* 46, 5 (1999), 604–632.

Satish Babu Korada, Andrea Montanari, and Sewoong Oh. 2011. Gossip pca. In *Proc. ACM SIGMETRICS joint int. conf. on Measurement and modeling of comp. sys.* ACM, 209–220.

Yehuda Koren, Robert Bell, and Chris Volinsky. 2009. Matrix Factorization Techniques for Recommender Systems. *Computer* 42, 8 (2009), 30–37. DOI:http://dx.doi.org/10.1109/MC.2009.263

Quoc Le, Marc'Aurelio Ranzato, Rajat Monga, Matthieu Devin, Kai Chen, Greg Corrado, Jeff Dean, and Andrew Ng. 2012. Building high-level features using large scale unsupervised learning. In *Proc. 29th International Conference on Machine Learning (ICML)*, John Langford and Joelle Pineau (Eds.). Omnipress, 81–88.

Yongjun Liao, Pierre Geurts, and Guy Leduc. 2010. Network Distance Prediction Based on Decentralized Matrix Factorization. In *Proc. 9th Int. IFIP TC 6 Netw. Conf. (LNCS)*, Mark Crovella, LauraMarie Feeney, Dan Rubenstein, and S.V. Raghavan (Eds.), Vol. 6091. Springer, 15–26. DOI:http://dx.doi.org/10.1007/978-3-642-12963-6_2

Qing Ling, Yangyang Xu, Wotao Yin, and Zaiwen Wen. 2012. Decentralized low-rank matrix completion. In *Proceedings of the 2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2925–2928. DOI:http://dx.doi.org/10.1109/ICASSP.2012.6288528

Yucheng Low, Joseph Gonzalez, Aapo Kyrola, Danny Bickson, Carlos Guestrin, and Joseph M. Hellerstein. 2010. GraphLab: A New Parallel Framework for Machine Learning. In *Conf. on Uncertainty in Artif. Intel.*

Frank McSherry. 2001. Spectral Partitioning of Random Graphs. In *Proc. 42nd Annual Symposium on Foundations of Computer Science (FOCS)*. 529–537.

Milena Mihail and Christos Papadimitriou. 2002. On the Eigenvalue Power Law. In *Rand. and Approx. Tech. in Comp. Sci.*, José D.P. Rolim and Salil Vadhan (Eds.). LNCS, Vol. 2483. Springer, 254–262. DOI:http://dx.doi.org/10.1007/3-540-45726-7_20

Alberto Montresor and Márk Jelasity. 2009. PeerSim: A Scalable P2P Simulator. In *Proc. 9th IEEE Int. Conf. on Peer-to-Peer Comp.* IEEE, 99–100. DOI:http://dx.doi.org/10.1109/P2P.2009.5284506 extended abstract.

Valeria Nikolaenko, Stratis Ioannidis, Udi Weinsberg, Marc Joye, Nina Taft, and Dan Boneh. 2013. Privacy-preserving Matrix Factorization. In *Proc. 2013 ACM SIGSAC Conf. on Comp. and Comm. Security (CCS'13)*. ACM, 801–812. DOI:http://dx.doi.org/10.1145/2508859.2516751

T. Nis. 1999. JAMA: A Java Matrix Package. (1999). http://math.nist.gov/javanumerics/jama

Róbert Ormándi, István Hegedűs, and Márk Jelasity. 2013. Gossip learning with linear models on fully distributed data. *Concurrency and Computation: Practice and Experience* 25, 4 (2013), 556–571. DOI:http://dx.doi.org/10.1002/cpe.2858

Christos H. Papadimitriou, Hisao Tamaki, Prabhakar Raghavan, and Santosh Vempala. 2000. Latent Semantic Indexing: A Probabilistic Analysis. *J. Comput. System Sci.* 61, 2 (2000), 217–235.

Fabio Petroni and Leonardo Querzoni. 2014. GASGD: Stochastic Gradient Descent for Distributed Asynchronous Matrix Completion via Graph Partitioning. In *Proceedings of the 8th ACM Conference on Recommender Systems (RecSys '14)*. ACM, New York, NY, USA, 241–248. DOI:http://dx.doi.org/10.1145/2645710.2645725

Paul Resnick, Neophytos Iacovou, Mitesh Suchak, Peter Bergstrom, and John Riedl. 1994. GroupLens: an open architecture for collaborative filtering of netnews. In *Proc. 1994 ACM Conf. on Computer supported cooperative work (CSCW '94)*. ACM, 175–186.

Jelle Roozenburg. 2006. *Secure Decentralized Swarm Discovery in Tribler*. Master's thesis. Parallel and Distributed Systems Group, Delft University of Technology. http://www.pds.ewi.tudelft.nl/~epema/MSc-theses/MSc-thesis-Roozenburg.pdf

Roberto Roverso, Jim Dowling, and Márk Jelasity. 2013. Through the Wormhole: Low Cost, Fresh Peer Sampling for the Internet. In *Proceedings of the 13th IEEE International Conference on Peer-to-Peer Computing (P2P 2013)*. IEEE. DOI:http://dx.doi.org/10.1109/P2P.2013.6688707

Nathan Srebro and Tommi Jaakkola. 2003. Weighted Low-Rank Approximations. In *Proc. 20th International Conference on Machine Learning (ICML)*. AAAI Press, 720–727.

Daniel Stutzbach and Reza Rejaie. 2006. Understanding churn in peer-to-peer networks. In *Proc. 6th ACM SIGCOMM conf. on Internet Measurement (IMC)*. ACM, 189–202. DOI:http://dx.doi.org/10.1145/1177080.1177105

Norbert Tölgyesi and Márk Jelasity. 2009. Adaptive Peer Sampling with Newscast. In *Euro-Par 2009 (LNCS)*, Vol. 5704. Springer, 523–534. DOI:http://dx.doi.org/10.1007/978-3-642-03869-3_50

Yu-Xiang Wang, Stephen Fienberg, and Alex Smola. 2015. Privacy for Free: Posterior Sampling and Stochastic Gradient Monte Carlo. In *Proceedings of The 32nd International Conference on Machine Learning (ICML'15)*. 2493—2502.

Feng Yan, Shreyas Sundaram, SVN Vishwanathan, and Yuan Qi. 2013. Distributed autonomous online learning: Regrets and intrinsic privacy-preserving properties. *IEEE Transactions on Knowledge and Data Engineering* 25, 11 (2013), 2483–2493.

Martin A. Zinkevich, Alex Smola, Markus Weimer, and Lihong Li. 2010. Parallelized stochastic gradient descent. In *Advances in Neural Information Processing Systems 23 (NIPS 2010)*. 2595–2603.