Fast Matrix Factorization with Non-Uniform Weights on Missing Data

Xiangnan He, Jinhui Tang, *Senior Member, IEEE,* Xiaoyu Du, Richang Hong, *Member, IEEE,* Tongwei Ren, *Member, IEEE,* and Tat-Seng Chua

***Abstract*—Matrix factorization (MF) has been widely used to discover the low-rank structure and to predict the missing entries of data matrix. In many real-world learning systems, the data matrix can be very high-dimensional but sparse. This poses an imbalanced learning problem, since the scale of missing entries is usually much larger than that of observed entries, but they cannot be ignored due to the valuable negative signal. For efficiency concern, existing work typically applies a uniform weight on missing entries to allow a fast learning algorithm. However, this simplification will decrease modeling fidelity, resulting in suboptimal performance for downstream applications.**

In this work, we weight the missing data non-uniformly, and more generically, we allow any weighting strategy on the missing data. To address the efficiency challenge, we propose a fast learning method, for which the time complexity is determined by the number of observed entries in the data matrix, rather than the matrix size. The key idea is two-fold: 1) we apply truncated SVD on the weight matrix to get a more compact representation of the weights, and 2) we learn MF parameters with element-wise alternating least squares (eALS) and memorize the key intermediate variables to avoid repeating computations that are unnecessary. We conduct extensive experiments on two recommendation benchmarks, demonstrating the correctness, efficiency, and effectiveness of our fast eALS method.

***Index Terms*—Matrix Factorization, Missing Data, Element-wise Alternating Least Squares (eALS), Recommendation System.**

I. Introduction

Matrices are a common data structure to represent the relation between two types of entities in learning systems [1]–[3]. In relational learning, matrix factorization (MF) is a popular approach for dimension reduction by representing the rows (entities of one type) and columns (entities of another type) as two low-rank matrices. The optimization of dimension reduction is usually achieved by minimizing the reconstruction error between the low-rank model and the original data. Since the low-rank model can encode certain patterns latent in the original data, MF has been recognized as an effective pattern recognition technique and been widely used in a variety of tasks, such as relation prediction [4], [5], data compression [6], clustering [7], feature learning [8], topic modeling [9], etc.

When the relation of interest is sparse, it is always desired to predict whether a relation exists between two entities, known as relation prediction. The relation prediction task plays a vital role in many real-world learning systems, such as recommender systems [4], [10], language understanding [11], [12], social network mining [13], [14], and so on. For relation prediction, which can be seen as a classification task, it is crucial to account for the missing entries, since they provide valuable signal about negative instances [15]. For example, in recommendation, early collaborative filtering approaches predict ratings by modeling the observed data only [16]; later on, researchers find that this way of ignoring missing data leads to poor performance in the real top-N recommendation system [17]. Another example is that in the learning of word embeddings [18], negative sampling is performed on missing data to add the negative signal about word-context co-occurrence, which is a crucial setup to ensure the semantics of learned word embeddings.

Nevertheless, it is non-trivial to leverage the missing data, since its scale can be several orders of magnitude larger than the observed entries [15]. For example, in video recommendation data, users may only watch hundreds of videos on average among millions of videos, making the scale of missing data three orders of magnitude larger than the observed data in the user-video matrix [19]. This large-scale missing data poses efficiency challenges for learning the MF model. Towards this end, existing works have resorted to either sampling partial missing data as negative signal (aka., negative sampling [20], [21]) or modeling all missing data in a simplified way – by assigning them a uniform weight to be negative [22], [23]. Both solutions have pros and cons: negative sampling has controllable efficiency, but its effectiveness may suffer from the low quality of negative examples and slow convergence [24], [25]; while modeling all missing data is costly, it can be more effective [1], [18]. To pursue high effectiveness, we focus on learning from all missing data in this work.

The Singular Value Decomposition (SVD) [26] is a representative method for whole data-based MF. It assigns the same weight to all entries in the data matrix, regardless of whether they are observed or not (by default, the unobserved entries are assigned with a value of zero). This assumption makes the optimization problem have a nice structure and yields an analytical closed-form solution [27]. However, considering that the number of missing data can be much larger than the

Xiangnan He and Tat-Seng Chua are with the School of Computing, National University of Singapore, Singapore, 117417. E-mail: xiangnanhe@gmail.com, dcscts@nus.edu.sg

Jinhui Tang is with the School of Computer Science and Engineering, Nanjing University of Science and Technology, Nanjing, Jiangsu, China, 210094. E-mail: jinhuitang@njust.edu.cn. Corresponding author.

Xiaoyu Du is with the School of Computer Science and Engineering, University of Electronic Science and Technology of China, Chengdu, Sichuan, China, 610054. E-mail: duxy.me@gmail.com

Richang Hong is with the Hefei University of Technology, Hefei, Anhui, China, 230000. E-mail: hongrc.hfut@gmail.com

Tongwei Ren is with the State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing, Jiangsu, China, 210093. E-mail: rentw@nju.edu.cn

observed data in real applications, it is more desirable to assign the missing data a lower weight to address the class imbalance issue. To this end, the Weighted Alternating Least Square (WALS) [22] assigns a lower weight c_0 to all missing data, which is more flexible than the default setting of 1. However, we argue that WALS implicitly admits all missing data have the same likelihood to be negative, which may not be true in real applications. For example, in recommendation, we know that popular items are more likely to be known by users, and thus a missing on popular items is more likely to be a true negative. Lastly, it is worth mentioning that the uniform weight design in WALS is mainly due to the efficiency concern, since it allows for a clever speedup on ALS learning, which can avoid the high complexity brought by modeling all missing data. If we were to use non-uniform weights on missing data, the speedup trick of WALS is not applicable anymore, and the optimization complexity becomes unaffordable.

In this work, we enhance 1) the flexibility of MF by allowing the use of non-uniform weights on missing data, and 2) the practicability of weighted MF by developing an efficient optimization algorithm. In short, we allow each missing entry to be assigned with an individualized weight, which encodes its prior to be a negative instance; the learning task takes the whole data matrix into account, but its time complexity is dependent on the number of observed entries only, rather than the matrix size (which is row# $\times$ column#). The two significant enhancements of our method make it easy to address large-scale relation prediction issue with a more expressive modeling on missing data, which has not been possible by the traditional MF methods like SVD and ALS. Our solution is achieved in three steps: 1) we perform truncated SVD on the weight matrix of missing data, using a more compact low-rank model to represent (or approximate) the weights of missing entries; 2) we perform ALS optimization on each element of user and item latent vectors, rather than the traditional vector-wise manner [22], [27]; 3) we leverage the low-rank weights to design memoization strategies to reduce the time complexity significantly. Through comprehensive experiments on two real-world recommendation benchmarks, we verify the correctness and efficiency of our fast eALS method, and the effectiveness of using non-uniform weights on missing data for the recommendation task.

A preliminary version of this work has been published as a conference paper in SIGIR 2016 [1]. This paper is significantly different from its preliminary version in the methodology. Specifically, this work approaches a generic problem setting where any weighting strategy can be applied on missing data, but our previous work [1] can only deal with a simpler case where the missing entries of a column have the same weight; moreover, the recent work [28] can also be seen as a simpler case of this work where the missing entries of a row have the same weight. As such, the Preliminaries (Section II), Proposed Methods (Section III), and Experiments (Section IV) have been re-written to support our solution to the new generic problem. The key contributions of this paper are summarized as follows:

- We highlight the problem of optimizing MF with non-uniform weights on missing data and present an element-wise ALS algorithm to solve it.
- We propose a fast eALS algorithm that solves the weighted MF problem with low-rank weights on missing data. The algorithm has a low time complexity in proportion to the number of observed entries and is independent of the number of missing entries.
- We perform extensive experiments on two real-world datasets and demonstrate its correctness, efficiency, and effectiveness. The codes of our experiments can be found in: https://github.com/duxy-me/ext-als.

II. Preliminaries

This section provides some preliminaries about MF and formalizes the problem to solve in this paper. Moreover, we discuss the efficiency challenge in solving the problem. Note that part A of this section has been presented in the preliminary version [1] (cf. Section 3.1) and other parts are new. Before starting the section, we first introduce some notations.

We denote the original data matrix as $\mathbf{R} \in \mathbb{R}^{M \times N}$, where M and N denote the number of rows and columns in the data matrix, respectively. We use the set $\mathcal{R}$ to denote the set of observed entries in $\mathbf{R}$, i.e., for which the values are non-zero. Matrices $\mathbf{P} \in \mathbb{R}^{M \times K}$ and $\mathbf{Q} \in \mathbb{R}^{N \times K}$ denote the latent factor matrix for rows and columns respectively; that is, they are the results or model parameters of MF. We use the vector $\mathbf{p}_u$ to denote the u-th row of matrix $\mathbf{P}$, and we use the set $\mathcal{R}_u$ to denote the column indices with a nonzero value on row u, i.e., $\mathcal{R}_u = \{i | r_{ui} \neq 0\}$. We use the symbols $\mathbf{q}_i$ and $\mathcal{R}_j$ to denote the similar meanings for the column side. Throughout the paper, we use the uppercase bold font to denote a matrix, lowercase bold font to denote a vector, and lowercase italic font to denote a scalar; for example, $\mathbf{P}$ denotes a matrix, $\mathbf{p}_u$ denotes the u-th row vector in $\mathbf{P}$, and p_{ui} denotes the (u, i)-th entry in $\mathbf{P}$.

A. The MF model

MF maps both rows and columns into a low-dimension latent space (the dimension is K) such that their interactions are modeled as an inner product in that space [16]. Mathematically, each element r_{ui} of $\mathbf{R}$ is estimated as:

$$\hat{r}_{ui} =< \mathbf{p}_u, \mathbf{q}_i >= \mathbf{p}_u^T \mathbf{q}_i, \tag{1}$$

where $\mathbf{p}_u$ and $\mathbf{q}_i$ are model parameters, which can be understood as the latent feature vector for row u and column i, respectively. The model estimation $\hat{r}_{ui}$ can be seen as reconstruction for an observed entry, or prediction for an unobserved one. For example, in recommendation, r_{ui} denotes a user's rating on an item (the larger the better), and ranking all items by $\hat{r}_{ui}$ can be used to select top-N recommendations for u. In matrix-wise representation, the model can be expressed as $\mathbf{R} \approx \mathbf{P}\mathbf{Q}^T$, which implies the low-rank assumption of the data matrix [29].

B. Problem Formulation

Since MF performs dimension reduction on the original data matrix (K is typically set to be much smaller than M and N),

the objective function for model learning is usually formed as an error-based regression loss [16], [27]. In this work, we learn MF parameters by solving the minimization problem on the objective function as follows:

$$\begin{aligned} L &= \sum_{u=1}^{M}\sum_{i=1}^{N} w_{ui}(r_{ui} - \mathbf{p}_u^T\mathbf{q}_j)^2 + \lambda(\sum_{u=1}^{M}||\mathbf{p}_u||^2 + \sum_{i=1}^{N}||\mathbf{q}_i||^2) \\ &= ||\mathbf{W}\odot(\mathbf{R} - \mathbf{P}\mathbf{Q}^T)||^2 + \lambda(||\mathbf{P}||^2 + ||\mathbf{Q}||^2), \end{aligned} \tag{2}$$

where w_{ui} denotes the weight of the training instance r_{ui}, $\mathbf{W} \in \mathbb{R}^{M\times N}$ is the matrix form for all weights w_{ui}, and λ is a hyper-parameter to control the regularization strength to prevent overfitting. In this problem formulation, we consider all data entries in $\mathbf{R}$ and assign each data entry with an individualized weight w_{ui}, which is a generic setting that gives practitioners the flexibility to design the weighting strategy. Many previous efforts on MF do not deal with this generic problem setting, but instead use a specific weighting strategy. Here we discuss three most common strategies:

Strategy 1. Zero weight on missing entries. This strategy applies a zero weight on missing entries, i.e., $w_{ui} = 0$ if $(u, i) \notin \mathcal{R}$. Since only observed entries are used as training instances, the learning time complexity is low, which depends on the number of observed entries. This is a typical setting for the rating prediction task [16], [30], which aims to predict the values of missing entries in user-item rating matrix. When the data follows the missing at random (MAR) assumption, such a setting can provide unbiased estimation. However, the MAR assumption does not hold in many real-world applications, for example, a user is are more likely to rate movies of her interest, rather than a random set of movies [31]. In this case, the missing entries contain valuable signal about negative instances, and thus ignoring them will lead to suboptimal performance, especially for predicting whether a relation exists between two entities [17].

Strategy 2. Uniform weight on all entries. This strategy applies a uniform weight of 1 on all data entries, i.e., $w_{ui} = 1$ for all (u, i). The SVD method [26] can be directly applied to find the optimal solution for this problem. When the number of missing entries are of the same scale as the number of observed entries, such a setting may yield good performance. However, many real-world applications need to deal with sparse matrix that is highly imbalanced, for example, the observed ratings only take 1.2% of the rating matrix in the Netflix challenge data[1]. For such highly imbalanced learning scenarios, a uniform weighting strategy will make the parameter estimation process dominated by the missing entries, resulting in suboptimal performance.

Strategy 3. Uniform weight on missing entries. This strategy assigns all missing entries with the same weight c_0, which can be different as the weight for observed entries [22]:

$$w_{ui} = \begin{cases} c_{ui} & if (u, i) \in \mathcal{R}, \\ c_0 & if (u, i) \notin \mathcal{R}, \end{cases} \tag{3}$$

where c_{ui} denotes the weight of observed entry (u, i), and c_0 denotes the uniform weight for all missing entries. When dealing with sparse data, c_0 can be set as a smaller number than c_{ui} to alleviate the imbalanced learning issue. Hu et al. [22] demonstrated that this strategy yields better performance than a uniform weight on all entries in recommendation task. However, the deficiency is that it assumes all missing entries provide the same level of negative signal, which severely limits the fidelity for modeling real-world scenarios. For example, in recommendation systems, we know that the exposed but unclicked items (e.g., display ads) are more likely to be true negatives [25], which should be assigned with a higher weight than others. Another reasonable intuition is that the missing entries of active users (who have consumed many items) are more likely to be true negatives [28].

In this work, we do not assume any weighting strategy on the data entries, and provide a solution for solving the generic problem of Eq. (2). In other words, our solution subsumes the above-mentioned works that define various weighting strategies.

C. Efficiency Discussion

One key reason that the previous work assumes a specific weighting strategy for missing data is due to efficiency concern. Here we analyze the learning time complexity for the three strategies:

- For Strategy 1, the training set contains only $|\mathcal{R}|$ observed entries, thus standard optimization method like stochastic gradient descent (SGD) can be applied, which has the time complexity of $O(|R|K)$. This level of complexity is rather low, only requiring a traversal on all training instances and updating latent vectors $\mathbf{p}_u$ and $\mathbf{q}_i$ at each visit of instance (u, i).

- For Strategy 2, since SVD can be directly applied to find the global optimum solution, the learning complexity depends on the solver for SVD. The commonly used solver Lanczos Bidiagonalization (LBD) method [32] has a complexity linear with respect to the number of observed entries. As such, its actual running time is in the same magnitude as the SGD solver for Strategy 1, and we can denote the analytical time complexity of SVD as $O(|R|K)$.

- For Strategy 3, the training set contains all $M \times N$ data entries, for which standard optimization methods like SGD have the time complexity of $O(MNK)$. This level of complexity is rather high, being unaffordable for real-world large-scale applications that may have over millions of rows and columns (e.g., the user-item matrix in recommendation). Fortunately, the uniform weight constraint brings opportunities for speedup by memorizing some intermediate variables. Hu et al. [22] leveraged on memoization tricks and proposed an ALS-based algorithm (named as WALS), reducing the time complexity to $O(|\mathcal{R}|K^2 + (M + N)K^3)$. Note that the $O(K^3)$ term is brought by the matrix inversion operation, which is inevitable when optimizing a latent vector (i.e., $\mathbf{p}_u$ or $\mathbf{q}_i$) as a whole in ALS [1]. Even so, the $O(|\mathcal{R}|K^2)$ part is still more costly than SGD, which only requires $O(|\mathcal{R}|K)$ time. As a result, even with speedup, WALS may still be prohibitive for running on large data, where large K is crucial as it can lead to better representation ability and better performance. Lastly, it is worth mentioning that the speedup design in WALS is only

[1] https://en.wikipedia.org/wiki/Netflix_Prize

applicable when the missing entries have the same weight. When such a nice structure is broken, WALS degrades to ALS with a complexity of $O(MNK)$.

In this paper, we propose a new solution to efficiently solve the weighted MF problem. Distinct from the above-mentioned efforts that performed vector-wise (i.e., $\mathbf{p}_u$) or matrix-wise (i.e., $\mathbf{P}$) optimization, we perform optimization on the element level (i.e., p_{uk}), named as *element-wise* ALS (or eALS for short). Furthermore, we apply a low-rank model to represent the weights of missing entries. Unifying the two designs, our solution achieves a time complexity of the $O(|R|K)$ level, but is more flexible on the weights of missing entries.

III. Proposed Methods

We first present a vanilla element-wise ALS learner, which differs from the conventional vector-wise ALS [22], [27]. By performing optimization on each element of the parameter matrix $\mathbf{P}$ and $\mathbf{Q}$, not only we can avoid the expensive matrix inversion operation in optimization, but also allow for more flexible design of memoization strategies for further speedup. Next, we propose to represent the weights for missing entries with a low-rank model, which not only reduces the space to store the weights for missing data, but also opens the door for speeding up the eALS learner. Lastly, based on the low-rank weights, we elaborate the fast eALS algorithm and discuss its several properties. Note that part A of this section has presented in the preliminary version [1] (cf. Section 3.3) and other parts are different.

A. Vanilla Element-wise ALS Learner

One bottleneck of the previous WALS solution lies in the matrix inversion operation, which is due to the updating of the latent vector for a row (column) as a whole (more explanations see Section 3.2 of [1]). As such, it is a natural thought to avoid this operation by optimizing parameters at the element level. Specifically, we follow the coordinate descent setting [33], [34] that optimizes each element of the latent vector while leaving the others fixed.

First, we differentiate the objective function Eq. (2) with respect to p_{uf}:

$$\frac{\partial J}{\partial p_{uf}} = -2\sum_{i=1}^{N}(r_{ui}-\hat{r}_{ui}^{f})w_{ui}q_{if}+2p_{uf}\sum_{i=1}^{N}w_{ui}q_{if}^{2}+2\lambda p_{uf}, \quad (4)$$

where $\hat{r}_{ui}^{f} = \hat{r}_{ui} - p_{uf}q_{if}$, which can be understood as the model prediction in the absence of latent factor f. Given other variables fixed, the optimal solution of p_{uf} can be obtained at the point of $\frac{\partial J}{\partial p_{uf}} = 0$. Solving this equation, we can have:

$$p_{uf} = \frac{\sum_{i=1}^{N}(r_{ui} - \hat{r}_{ui}^{f})w_{ui}q_{if}}{\sum_{i=1}^{N} w_{ui}q_{if}^{2} + \lambda}. \quad (5)$$

Following the similar way, we can get the solver for item latent factor q_{if}:

$$q_{if} = \frac{\sum_{u=1}^{M}(r_{ui} - \hat{r}_{ui}^{f})w_{ui}p_{uf}}{\sum_{i=1}^{M} w_{ui}p_{uf}^{2} + \lambda}. \quad (6)$$

With the above solution that solves on variable with others fixed, we can get a learning algorithm by iteratively executing on all parameters until convergence. It is worth noting that since the objective function is non-convex in terms of all parameters together. As such, this element-wise ALS solver can only find local minima (where the critical points where gradients vanish). This is the same for the conventional ALS [22], [35] and other gradient descent methods in optimizing the objective function. As a consequence, the initialization of model parameters will affect the results. According to our experiment experience, eALS's performance is relatively stable with a Gaussian random initialization.

Algorithm 1 Vanilla eALS algorithm for weighted MF.

Input: Data matrix $\mathbf{R}$, weight matrix $\mathbf{W}$, number of latent factors K, regularizer λ
Output: MF parameters $\mathbf{P}$ and $\mathbf{Q}$

Randomly initialize $\mathbf{P}$ and $\mathbf{Q}$;
for $(u, i) \in \mathcal{R}$ **do**
  $\hat{r}_{ui} \leftarrow$ Eq. (1);
end for
while Stopping criteria is not met **do**
  // Update $\mathbf{P}$ ▷ $O(NMK)$
  for $u \leftarrow 1$ **to** M **do**
    for $f \leftarrow 1$ **to** K **do**
      for $i \leftarrow 1$ **to** N **do** $\hat{r}_{ui}^{f} \leftarrow \hat{r}_{ui} - p_{uf}q_{if}$;
      $p_{uf} \leftarrow$ Eq. (5) ▷ $O(N)$
      for $i \leftarrow 1$ **to** N **do** $\hat{r}_{ui} \leftarrow \hat{r}_{ui}^{f} + p_{uf}q_{if}$;
    end for
  end for
  // Update $\mathbf{Q}$ ▷ $O(NMK)$
  for $i \leftarrow 1$ **to** N **do**
    for $f \leftarrow 1$ **to** K **do**
      for $u \leftarrow 1$ **to** M **do** $\hat{r}_{ui}^{f} \leftarrow \hat{r}_{ui} - p_{uf}q_{if}$
      $q_{if} \leftarrow$ Eq. (6) ▷ $O(M)$
      for $u \leftarrow 1$ **to** M **do** $\hat{r}_{ui} \leftarrow \hat{r}_{ui}^{f} + p_{uf}q_{if}$
    end for
  end for
end while

Time Complexity. We can see that by updating each element p_{uf} and q_{if} at a time, we can avoid the expensive matrix inversion operation, which is compulsory in traditional ALS. Through this way, we can eliminate the $O((M + N)K^3)$ term in the time complexity. Furthermore, we can follow the caching strategy as stated in [33], further reducing the time complexity from $O(MNK^2)$ (i.e., the time complexity of directly implementing the update rules) to $O(MNK)$. This time complexity is in the same level as evaluating all points in the data matrix $\mathbf{R}$. Algorithm 1 shows the vanilla eALS algorithm with the cache on $\hat{r}_{ui}$, which has a time complexity of $O(MNK)$.

B. Low-Rank Representation on Weights of Missing Data

The original problem of weighted MF assigns an individualized weight for each data point, which requires $O(MN)$ space to store all weights (denoted as the weight matrix $\mathbf{W}$).

This is very costly and unrealistic for large-scale applications. To be more specific, let us consider an intuitive example in recommendation that needs to deal with 1 million users and 1 million items. Assuming we use the 4-Byte float type to express a weight, then the space to store all weights is $1e^6 \times 1e^6 \times 4B = 4000GB = 4TB$. Such a large consumption of space will pose a great challenge for the infrastructure, not to mention that the space cost will increase quadratically with respect to the number of users and items.

Now that storing an individualized weight for each data point is practically infeasible, we consider using a more compact way to represent $\mathbf{W}$. Specifically, we perform truncated SVD on $\mathbf{W}$, obtaining two low-rank matrices which can reconstruct $\mathbf{W}$ without any error:

$$\mathbf{W} = \mathbf{A}\mathbf{B}^T \tag{7}$$

where $\mathbf{A} \in \mathbb{R}^{M \times Z'}, \mathbf{B} \in \mathbb{R}^{N \times Z'}$, and Z' denotes the rank size of weight matrix $\mathbf{W}$. If Z' is much smaller than M and N, using $\mathbf{A}$ and $\mathbf{B}$ to reconstruct $\mathbf{W}$ takes fewer space $O((M+N)Z'))$ than directly storing $\mathbf{W}$. If Z' is large such that the space complexity of $O((M+N)Z'))$ is still unaffordable, one can use truncated SVD with a smaller number of predefined rank size, but at the cost of approximating $\mathbf{W}$ with some errors — the smaller the number, the larger the error of the approximation. This is a tradeoff between the space cost and the precision of low-rank representation. To be precise, let Z be the predefined rank size of truncated SVD, then the space cost to store the weights is $O((M+N)Z))$. Since the common solver of SVD like the Lanczos Bidiagonalization (LBD) method [32] has a complexity linear with respect to the number of observed entries, the analytical time complexity of truncated SVD is $O(|\mathcal{R}|Z)$. Given Z is usually a small number (e.g., 1 for column-oriented [1] or row-oriented [28] weighting schemes), this time complexity is much lower than matrix factorization algorithms (which are shown in Table 1). As such, using truncated SVD on the weights will not significantly increase the actual runtime of our method.

In a sparse matrix, the number of missing entries is usually several magnitudes than the number of observed entries. As such, we apply truncated SVD on the weights of missing entries only, and use the original weights of observed entries as they are. Such a weighting strategy can be expressed as follows:

$$w_{ui} = \begin{cases} c_{ui} & \text{if}(u,i) \in \mathcal{R}, \\ \mathbf{a}_u^T \mathbf{b}_i & \text{if}(u,i) \notin \mathcal{R}, \end{cases} \tag{8}$$

where c_{ui} denotes the weight of observed entry (u,i), $\mathbf{a}_u \in \mathbb{R}^Z$ and $\mathbf{b}_i \in \mathbb{R}^Z$ denote the u-th row vector of $\mathbf{A}$ and i-th row vector of $\mathbf{B}$, respectively. In the next subsection, we present a fast algorithm to accelerate eALS by leveraging the low-rank structure of the weights of missing data.

C. Fast eALS Algorithm

The fast algorithm to be presented in this part reduces the time complexity from $O(MN)$-related to $O(|\mathcal{R}|)$-related, successfully avoiding the heavy burden brought by optimizing the missing data.

First, we reformulate the objective function by separating the terms on observed data and missing data:

$$\begin{aligned} J = &\sum_{(u,i)\in\mathcal{R}} c_{ui}(r_{ui} - \hat{r}_{ui})^2 + \sum_{(u,i)\notin\mathcal{R}} \mathbf{a}_u^T \mathbf{b}_i (r_{ui} - \hat{r}_{ui})^2 \\ &+ \lambda(\sum_{u=1}^{M} ||\mathbf{p}_u||^2 + \sum_{i=1}^{N} ||\mathbf{q}_i||^2) \end{aligned} \tag{9}$$

As we can see, the first term focuses on the observed data only and leads to a low complexity in optimization. The major cost comes from the second term that operates on all missing data. Next, we elaborate the derivation process of optimizing user latent factor p_{uf}, and its counterpart of optimizing item latent factor q_{if} can be achieved similarly.

First, we compute the derivative of J with respect to p_{uf} and set the derivative to zero, we can obtain the update rule of p_{uf}:

$$p_{uf} = \frac{\sum_{i\in\mathcal{R}_u}(r_{ui} - \hat{r}_{ui}^f)c_{ui}q_{if} - \sum_{i\notin\mathcal{R}_u} \mathbf{a}_u^T \mathbf{b}_i \hat{r}_{ui}^f q_{if}}{\sum_{i\in\mathcal{R}_u} c_{ui}q_{if}^2 + \sum_{i\notin\mathcal{R}_u} \mathbf{a}_u^T \mathbf{b}_i q_{if}^2 + \lambda}. \tag{10}$$

where $\hat{r}_{ui}^f = \hat{r}_{ui} - p_{uf}q_{if}$, i.e., the prediction without the component of latent factor f. With careful inspection, we can find that the major cost of executing the update rule comes from the two sum operations on missing data, i.e., $\sum_{i\notin\mathcal{R}_u} \mathbf{a}_u^T \mathbf{b}_i \hat{r}_{ui}^f q_{if}$ (in the numerator) and $\sum_{i\notin\mathcal{R}_u} \mathbf{a}_u^T \mathbf{b}_i q_{if}^2$ (in the denominator). We term the evaluation of the two costly terms as the rq problem and the q^2 problem, respectively, and show how to solve the two problems in an efficient way.

1. Solving the rq problem. First, we expand the term $\mathbf{a}_q^T \mathbf{b}_i$ using element-wise operations and obtain:

$$\sum_{i\notin\mathcal{R}_u} \mathbf{a}_u^T \mathbf{b}_i \hat{r}_{ui}^f q_{if} = \sum_{i\notin\mathcal{R}_u} \sum_{t=1}^{Z} a_{ut} b_{it} \sum_{k\neq f} p_{uk} q_{ik} q_{if}. \tag{11}$$

Then, we re-arrange the sum operations and obtain its equivalent form:

$$\begin{aligned} \sum_{i\notin\mathcal{R}_u} \mathbf{a}_u^T \mathbf{b}_i \hat{r}_{ui}^f q_{if} &= \sum_{k\neq f} p_{uk} \sum_{t=1}^{Z} a_{ut} \sum_{i\notin\mathcal{R}_u} b_{it} q_{ik} q_{if} \\ &= \sum_{k\neq f} p_{uk} \sum_{t=1}^{Z} a_{ut} (\sum_{i=1}^{N} b_{it} q_{ik} q_{if} - \sum_{i\in\mathcal{R}_u} b_{it} q_{ik} q_{if}) \end{aligned} \tag{12}$$

As we can see, the main computational bottleneck is in the term $\sum_{i=1}^{N} b_{it} q_{ik} q_{if}$, which needs to scan over all items. Nevertheless, a nice property is that this term is independent of u — which means if we sequentially update all elements in $\mathbf{P}$, and then $\mathbf{Q}$, this term can be pre-computed and used for the updating of all elements in $\mathbf{P}$ without computing it on-the-fly (the reverse way also applies). To achieve this, we define a 3-dimensional tensor $\mathbf{S}^q \in \mathbb{R}^{T \times F \times K}$, in which each element is defined as $S_{tfk}^q = \sum_{i=1}^{N} b_{it} q_{ik} q_{if}$; the $\mathbf{S}^q$ tensor is computed after updating $\mathbf{Q}$ and is cached in the updates of $\mathbf{P}$. With this cache, the rq problem can be approached as:

$$\sum_{i\notin\mathcal{R}_u} \mathbf{a}_u^T \mathbf{b}_i \hat{r}_{ui}^f q_{if} = \sum_{k\neq f} p_{uk} \sum_{t=1}^{Z} a_{ut} (S_{tkf}^q - \sum_{i\in\mathcal{R}_u} b_{it} q_{ik} q_{if}), \tag{13}$$

which can be computed in $O(KZ|\mathcal{R}_u|)$ time, rather than the raw time complexity of $O(KZMN)$.

2. Solving the q^2 problem. We can apply the similar cache strategy to address the costly q^2 problem:

$$\begin{aligned}\sum_{i\notin\mathcal{R}_u}\mathbf{a}_u^T\mathbf{b}_i q_{if}^2 &= \sum_{i\notin\mathcal{R}_u}\sum_{t=1}^{Z} a_{ut}b_{it}q_{if}^2 \\ &= \sum_{t=1}^{Z} a_{ut}(\sum_{i=1}^{N} b_{it}q_{if}^2 - \sum_{i\in\mathcal{R}_u} b_{it}q_{if}^2) \\ &= \sum_{t=1}^{Z} a_{ut}S_{tff}^q - \sum_{i\in\mathcal{R}_u}\mathbf{a}_u^T\mathbf{b}_i q_{if}^2,\end{aligned} \tag{14}$$

where S_{tff}^q denotes the (t,f,f)-th element of the $\mathbf{S}^q$ cache.

We can apply the similar derivation process on the item latent factor q_{if} to obtain its update rule:

$$q_{if} = \frac{\sum_{u\in\mathcal{R}_i} c_{ui}(r_{ui} - \hat{r}_{ui}^f)p_{uf} - \sum_{u\notin\mathcal{R}_i}\mathbf{a}_u^T\mathbf{b}_i\hat{r}_{ui}^f p_{uf}}{\sum_{u\in\mathcal{R}_i} c_{ui}p_{uf}^2 + \sum_{u\notin\mathcal{R}_i}\mathbf{a}_u^T\mathbf{b}_i p_{uf}^2 + \lambda}, \tag{15}$$

To speed up the computation of $\sum_{u\notin\mathcal{R}_i}\mathbf{a}_u^T\mathbf{b}_i\hat{r}_{ui}^f p_{uf}$ and $\sum_{u\notin\mathcal{R}_i}\mathbf{a}_u^T\mathbf{b}_i p_{uf}^2$, we similarly define the 3-dimensional $\mathbf{S}^p$ cache, in which each element is $S_{tfk}^p = \sum_{u=1}^{M} a_{ut}p_{uk}p_{uf}$. With this cache, the two costly terms can be efficiently computed as:

$$\begin{aligned}\sum_{u\notin\mathcal{R}_i}\mathbf{a}_u^T\mathbf{b}_i\hat{r}_{ui}^f p_{uf} &= \sum_{k\neq f} q_{ik}\sum_{t=1}^{Z} b_{it}\sum_{u\notin\mathcal{R}_i} a_{ut}p_{uk}p_{uf} \\ &= \sum_{k\neq f} q_{ik}\sum_{t=1}^{Z} b_{it}(S_{tfk}^p - \sum_{u\in\mathcal{R}_i} a_{ut}p_{uk}p_{uf})\end{aligned} \tag{16}$$

$$\begin{aligned}\sum_{u\notin\mathcal{R}_i}\mathbf{a}_u^T\mathbf{b}_i p_{uf}^2 &= \sum_{u\notin\mathcal{R}_i}\sum_{t=1}^{Z} a_{ut}b_{it}p_{uf}^2 \\ &= \sum_{t=1}^{Z} b_{it}S_{tff}^p - \sum_{u\in\mathcal{R}_i}\mathbf{a}_u^T\mathbf{b}_i p_{uf}^2\end{aligned} \tag{17}$$

Algorithm 2 summarizes the accelerated algorithm for our eALS method. Since each parameter update of eALS finds the optimal value for the parameter given the current status of other parameters, the training objective function is guaranteed to decrease with the training[2]. As such, for the stopping criteria, one can either check the objective function value, or rely on a hold-out validation data to investigate the metrics of interest.

D. Discussions

In this subsection, we discuss several properties of our fast eALS algorithm, including analyzing its time complexity, the fast computation of objective function, and how to do parallel learning.

[2]We omit rigorous proof here since it is obvious.

Algorithm 2 Fast eALS algorithm for weighted MF.

Input: Data matrix $\mathbf{R}$, number of latent factors K, regularizer λ, weights of observed entries $\{c_{ui}\}$, low-rank model for weights of missing entries $\mathbf{A}$ and $\mathbf{B}$
Output: MF parameters $\mathbf{P}$ and $\mathbf{Q}$

Randomly initialize $\mathbf{P}$ and $\mathbf{Q}$;
for $(u,i)\in\mathcal{R}$ **do**
  $\hat{r}_{ui} \leftarrow$ Eq. (1);
end for
while Stopping criteria is not met **do**
  // Update $\mathbf{S}^q$ cache ▷ $O(NK^2Z)$
  for all $t\leftarrow 1$ **to** Z, $f\leftarrow 1$ **to** K, $k\leftarrow 1$ **to** K **do**
    $S_{tfk}^q \leftarrow \sum_{i=1}^{N} b_{it}q_{ik}q_{if}$;
  end for
  // Update $\mathbf{P}$ ▷ $O(MK^2Z + |\mathcal{R}|KZ)$
  for $u\leftarrow 1$ **to** M **do**
    for $f\leftarrow 1$ **to** K **do**
      for $i\in\mathcal{R}_u$ **do** $\hat{r}_{ui}^f \leftarrow \hat{r}_{ui} - p_{uf}q_{if}$;
      $p_{uf} \leftarrow$ Eq. (10) ▷ $O(|\mathcal{R}_u|Z + KZ)$
      for $i\in\mathcal{R}_u$ **do** $\hat{r}_{ui} \leftarrow \hat{r}_{ui}^f + p_{uf}q_{if}$;
    end for
  end for
  // Update $\mathbf{S}^p$ cache ▷ $O(MK^2Z)$
  for all $t\leftarrow 1$ **to** Z, $f\leftarrow 1$ **to** K, $k\leftarrow 1$ **to** K **do**
    $S_{tfk}^p \leftarrow \sum_{u=1}^{M} a_{ut}p_{uk}p_{uf}$
  end for
  // Update $\mathbf{Q}$ ▷ $O(NK^2Z + |\mathcal{R}|KZ))$
  for $i\leftarrow 1$ **to** N **do**
    for $f\leftarrow 1$ **to** K **do**
      for $u\in\mathcal{R}_i$ **do** $\hat{r}_{ui}^f \leftarrow \hat{r}_{ui} - p_{uf}q_{if}$
      $q_{if} \leftarrow$ Eq. (15) ▷ $O(|\mathcal{R}_i|Z + KZ)$
      for $u\in\mathcal{R}_i$ **do** $\hat{r}_{ui} \leftarrow \hat{r}_{ui}^f + p_{uf}q_{if}$
    end for
  end for
end while

1) Time Complexity Analysis: The time complexities of key steps have been annotated in Algorithm 2. Summarily, the complexity of one eALS iteration is $O((M+N)K^2Z + |\mathcal{R}|KZ)$, which includes the complexity of updating a row latent factor $O(KZ+|\mathcal{R}_u|Z)$ and the complexity of updating a column latent factor $O(KZ + |\mathcal{R}_i|Z)$. As we can see, even eALS models all missing data, the essential time complexity is controlled by the number of observed data $|\mathcal{R}|$, rather than the matrix size $M\times N$. Thus the overall time complexity is in proportion to $|\mathcal{R}|$ and Z, which makes eALS extremely efficient for large-scale applications.

There are some other MF methods that model all the missing data. Their time complexities (of one iteration) are shown in Table I. Note that besides our proposed eALS, other MF methods shown in the table only support uniform weights on missing entries. As such, there is an additional term Z in our method, which denotes the rank size of the weights of missing data. For a fair comparison with other methods in time complexity, we assume Z to be 1 and use eALS to optimize the same objective function. First, our model is K times faster

TABLE I
TIME COMPLEXITY OF WHOLE DATA-BASED MF METHODS.

Method	Time Complexity
WALS (Hu *et al.* [22])	$O((M+N)K^3 + \|\mathcal{R}\|K^2)$
IALS1 (Pilászy *et al.* [36])	$O(K^3 + (M+N)K^2 + \|\mathcal{R}\|K)$
ii-SVD (Volkovs *et al.* [37])	$O((M+N)K^2 + MN\log K)$
RCD (Devooght *et al.* [23])	$O((M+N)K^2 + \|\mathcal{R}\|K)$
eALS (Algorithm 2)	$O((M+N)K^2Z + \|\mathcal{R}\|KZ)$

$|\mathcal{R}|$ denotes the number of non-zeros in the data matrix $\mathbf{R}$. M and N denote the number of rows and columns of data matrix $\mathbf{R}$, respectively. K denotes the latent dimension of MF. Z denotes the rank size of the weights of missing data.

than the vector-wise ALS [22], [35], and it has the same time complexity with RCD [23]. Moreover, it is faster than ii-SVD [37], another recent solution for item recommendation with implicit feedback. It is remarkable that RCD [23] leverages the gradient descent on a randomly chosen latent vector to learn a whole-data based MF. To find out a good learning rate for faster convergence adaptively, RCD runs a line search in each gradient step. Therefore, the major advantages of eALS over RCD are the high efficiency and simplicity.

2) Fast Computation of the Objective Function: The value of objective function is an important indicator on the training process. A direct calculation requires evaluating every entry in the $\mathbf{R}$ matrix, which takes $O(MNKZ)$ time and is very time-consuming. To address this problem, we leverage the low-rank weighting scheme and intermediate variables cached in Algorithm 2, devising a set of similar element-wise computations on $\mathbf{R}$ for acceleration. Here, we reformulate the major cost of objective function (i.e. the loss of the missing data):

$$\sum_{u=1}^{M}\sum_{i\notin\mathcal{R}_u}\mathbf{a}_u^T\mathbf{b}_i\hat{r}_{ui}^2 = \sum_{u=1}^{M}\sum_{i=1}^{N}\mathbf{a}_u^T\mathbf{b}_i\hat{r}_{ui}^2 - \sum_{u=1}^{M}\sum_{i\in\mathcal{R}_u}\mathbf{a}_u^T\mathbf{b}_i\hat{r}_{ui}^2 \quad (18)$$

It is obvious that the major computation comes from the first term, due to the iterations over all rows and columns. Thus we accelerate it with the transformation in Eq. (19):

$$\begin{aligned}\sum_{u=1}^{M}\sum_{i=1}^{N}\mathbf{a}_u^T\mathbf{b}_i\hat{r}_{ui}^2 &= \sum_{u=1}^{M}\sum_{i=1}^{N}\sum_{t=1}^{Z}a_{ut}b_{it}\sum_{k=1}^{K}p_{uk}q_{ik}\sum_{f=1}^{K}q_{if}p_{uf}\\ &= \sum_{u=1}^{M}\sum_{k=1}^{K}p_{uk}\sum_{f=1}^{K}p_{uf}\sum_{t=1}^{Z}a_{ut}\sum_{i=1}^{N}(b_{it}q_{ik}q_{if}) \quad (19)\\ &= \sum_{u=1}^{M}\sum_{k=1}^{K}p_{uk}\sum_{f=1}^{K}p_{uf}\sum_{t=1}^{Z}a_{ut}S_{tfk}^{q}.\end{aligned}$$

As can be seen, with the help of the $\mathbf{S}^q$ cache, we can reduce the time complexity to $O(MK^2Z)$, which is several orders of magnitude smaller than the direct computation of $O(MNKZ)$.

3) Parallel Learning: The key operations of eALS are easily parallelizable. First, the computations of the two caches $\mathbf{S}^q$ and $\mathbf{S}^p$ are based on the standard matrix multiplication operations (line 8 and 20), which are straightforward to be parallelized. Second, in updating the latent vectors $\mathbf{P}$ (line 10-17), the cache $\mathbf{S}^q$ is temporarily fixed, and the shared parameters $\hat{r}_{ui}$ are independent with each other. Therefore, it is practicable to update rows in parallel due to the nice independent property. Specifically, eALS can leverage multiple workers to update the model parameters for disjoint sets of rows concurrently. Similarly, this parallel strategy can also be applied in updating latent vectors $\mathbf{Q}$ (line 22-29).

It is notable that since the operations in SGD are strictly ordered and they are hard to be separated, controlling the possible losses is significant and difficult to leverage sophisticated strategies in parallel [38]. Meanwhile, the SGD loss always constrains the paralleling magnitude. Thus the ease of parallelization is an important advantage of our proposed eALS over the commonly used SGD learner. With coordinate descent, by parallelizing the key operations, our proposed eALS is embarrassingly parallel without any approximation loss.

TABLE II
STATISTICS OF THE EVALUATION DATASETS.

Dataset	Review#	Item#	User#	Sparsity
Yelp	731,671	25,815	25,677	99.89%
Amazon	5,020,705	75,389	117,176	99.94 %

IV. EXPERIMENTS

In this section, we perform experiments to verify the correctness, efficiency, and effectiveness of our fast eALS algorithm. All experiments are conducted on two real-world rating datasets, which are commonly used in recommendation systems. We first introduce the experimental settings, followed by the verification of correctness and efficiency, and the effectiveness of eALS in recommendation by modeling missing data with non-uniform weights. Note that part A, D and Table IV of this section have been presented in the preliminary version [1], and other parts are new.

A. Experimental Settings

1) Datasets: Two publicly accessible rating datasets are selected to evaluate the methods: Yelp[3] and Amazon Movies[4]. The Yelp dataset is about users' ratings on businesses (most of which are restaurants) and the Amazon dataset is about users' ratings on movies, where each rating is in the range of 1 to 5. We construct the data matrix $\mathbf{R}\in\mathbb{R}^{M\times N}$ by defining each row as a user, each column as an item, and each entry as the user's rating score on the item; if a user did not rate an item before, the corresponding entry in $\mathbf{R}$ will be defined as missing data. Table II summarizes the statistics of our experimented datasets[5]. We can see that both datasets are extremely sparse with the sparsity ratio over 99.8%. This provides empirical evidence on the necessity of using low-rank weights rather than storing the whole weight matrix as it is. Take the Amazon dataset as an example, storing the whole dataset matrix takes the space of $35GB$ ($75,389 * 117,176 * 4B \approx 35GB$), which is very space-consuming.

2) Evaluation Protocols: We split the data into training set and testing set by using the leave-one-out protocol, a widely used method in recommendation papers [20], [39].

[3]We used the Yelp Challenge dataset downloaded on October 2015 that contained 1.6 million reviews.

[4]http://snap.stanford.edu/data/web-Amazon-links.html

[5]The experimented datasets can be downloaded from: https://github.com/hexiangnan/sigir16-eals/tree/master/data

TABLE III
COMPARISON BETWEEN SVD AND OUR EALS ALGORITHM IN OPTIMIZING THE SAME OBJECTIVE FUNCTION.

Dataset	Approach	HR	NDCG	Training Loss
Yelp	SVD	0.1816	0.0439	$5.9222 * 10^5$
	eALS	0.1814	0.0438	$5.9223 * 10^5$
Amazon	SVD	0.2945	0.0690	$3.0172 * 10^6$
	eALS	0.2945	0.0690	$3.0172 * 10^6$

Specifically, the last rating of each user is hold out for testing, and the remaining data are used for training.

Besides validating the model training with the loss value, we also evaluate the performance of *Hit Ratio* (HR) and *Normalized Discounted Cumulative Gain* (NDCG), which judge the ranking quality of top-N recommendation; here we set the N to 100 without special mention. More specifically, for a user, we rank all unrated items by its prediction scores, and take out the top-100 items as the *recommended list*. If the testing item appears in the recommended list, it is treated as a hit, and the HR is set as 1. We can see that HR does not account for the position of a hit — as long as the testing item appears in the recommended list, it is treated as a success. NDCG addresses this deficiency by assigning higher rewords to hits at top positions and scoring successively lower-position hits with marginal fractional utility. More details about the two metrics can be found in [1].

Since the evaluation on each user produces a ranking list, we calculate HR and NDCG for each user and report the average score of all users. Clearly, a higher score denotes a better performance, and both measures are in the range of 0 to 1.

B. Correctness Verification

Here we verify the correctness of our proposed eALS algorithm. Since SVD is known to optimize the unweighted squared loss and can find the global minimum, we setup eALS to optimize the same objective function as SVD to verify its correctness. Specifically, we set c_{ui} for all observed entries as 1, the regularization parameter λ to 0, and $\mathbf{A}$ and $\mathbf{B}$ to a vector in which all entries are 1; the number of latent factors K is set to 64 for both methods. For SVD, we use the Python toolkit sparsesvd[6], which solves the SVD problem with the LBD method. For eALS, we run it for 100 iterations.

Table III shows the loss achieved by the two methods on the training set and their HR and NDCG scores on the testing set. We can see that eALS achieves almost the identical performance as SVD. On Yelp, the training loss of eALS is slightly higher than that of SVD, which is caused by the insufficient training of eALS, since eALS iteratively updates model parameters while SVD finds the global optima with a closed form solution; and the t-tests show that the two methods are in the same significance level. On Amazon, eALS sufficiently converges in 100 iterations, and both training loss and testing scores show that eALS achieves the same performance as SVD. To verify that eALS finds the exactly same solution as SVD, we further employ the point-wise measure *mean absolute error* (MAE) to evaluate the difference of the two methods' prediction on observed entries: on Yelp, the MAE is $3.1 * 10^{-4}$; and on Amazon, the MAE is $9.7 * 10^{-6}$. Such tiny error rate is acceptable, considering that eALS and SVD are implemented with different programming languages with different float precision settings. Overall, these results verify the correctness of our fast eALS algorithm.

[6]https://pypi.python.org/pypi/sparsesvd/

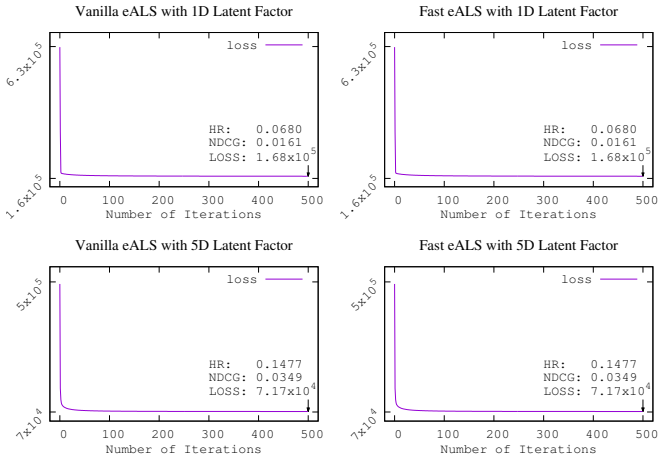


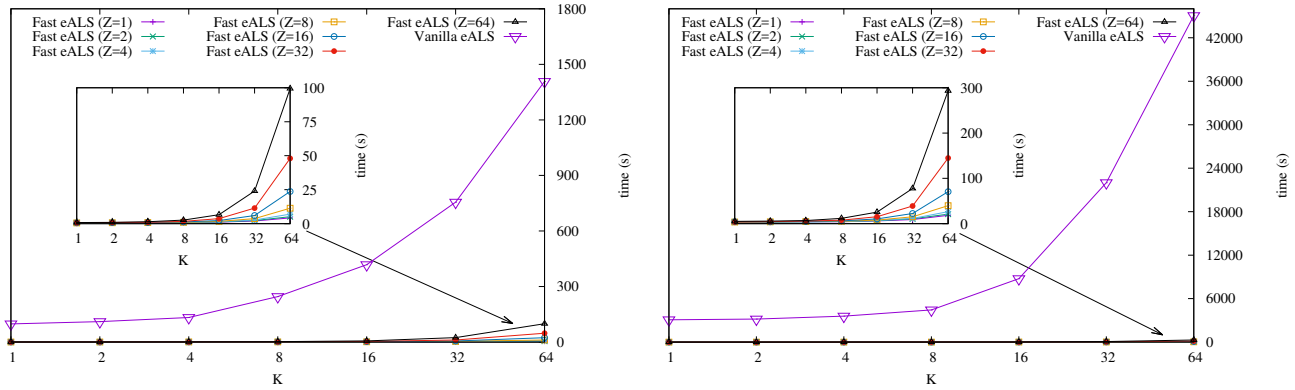
Fig. 1. The comparison between the vanilla eALS (Algorithm 1) and fast eALS (Algorithm 2) in terms of training loss and testing scores. The left two subfigures show vanilla eALS with K equals to 1 and 5, respectively, and the right two subfigures show fast eALS with the same setting.

Furthermore, we empirically test whether our fast eALS method (Algorithm 2) speeds up the vanilla eALS (Algorithm 1) without any sacrifice on the accuracy. To avoid the possible randomness that affects the results, we apply the same initialization on their model parameters . Figure 1 shows the training process of the two algorithms with the number of latent factors K setting to 1 and 5 on Yelp. We can see that the fast eALS algorithm obtains exactly the same result as the vanilla eALS, including the training loss of each epoch. This is as expected, since we derive the fast eALS based on rigorous mathematical operations without any approximation. This further justifies the correctness of the fast eALS algorithm, which is actually non-trivial to implement since it has several caches for speed-up purpose that need to be carefully updated. Interested readers can check out our implementation at: https://github.com/duxy-me/ext-als.

C. Efficiency Study

We investigate the actual speedup brought by our design of the fast eALS algorithm. All experiments in this subsection are run on the same machine (Intel Xeon 2.67GHz CPU and 24GB RAM) for fair comparison on the efficiency. Figure 2 shows the training time of the vanilla eALS and our fast eALS with different settings of Z and K. In the figure, the x-axis denotes the setting of K, the y-axis denotes the training time per iteration, and different lines indicate different settings of Z. We have the following key observations:

- The fast eALS is several magnitudes faster than the vanilla eALS algorithm. For example, in Yelp, the vanilla eALS takes 98 seconds to train a small model of $K = 1$, while the fast eALS takes only 0.8 seconds to train the same



(a) Training time per iteration in Yelp

(b) Training time per iteration in Amazon

Fig. 2. Comparison on the training time of the vanilla eALS algorithm and fast eALS (with different settings of Z).

model even with a large Z of 64. The speedup is more significant for the larger Amazon data, where the vanilla eALS takes over 42,000 seconds (i.e., half day) to train a model of $K = 64$, while the fast eALS takes only 300 seconds to train the same model with a large Z of 64. This acceleration is over 100 times, which is highly valuable in practice and is difficult to achieve with simply engineering efforts. Intuitively, one needs to have over 100 machines and implements an effective distributed system with a negligible network cost and linear scale-up on the number of machines, which is very difficult to achieve in practice [40].

- The running time of fast eALS exhibits a linear relationship with respect to Z, which can be seen clearly from the inside box of the figure. For example in Yelp, for $K = 64$, the y-axis of $Z = 64$ is twice of that of $Z = 32$, which is the same for the Amazon dataset. Moreover, the running time exhibits a quadratic relationship with respect to K. These results are as expected, verifying the analytical time complexity of the fast eALS algorithm — $O((M+N)K^2Z + |\mathcal{R}|KZ)$.

Furthermore, we compare the efficiency of the fast eALS algorithm with two whole data-based MF methods — the *Randomized block Coordinate Descent* (RCD) [23] method and the WALS method [22]. Since both methods only support uniform weighting on missing data, we set the Z of eALS to 1 for a fair comparison. Table IV shows the average training time per iteration of the three methods.

TABLE IV
TRAINING TIME PER ITERATION OF FAST EALS, RCD AND WALS.

	Yelp			Amazon		
K	**eALS**	**RCD**	**WALS**	**eALS**	**RCD**	**WALS**
32	1s	1s	10s	9s	10s	74s
64	4s	3s	46s	23s	17s	4.8m
128	13s	10s	221s	72s	42s	21m
256	1m	0.9m	23m	4m	2.8m	2h
512	2m	2m	2.5h	12m	9m	11.6h
1024	7m	11m	25.4h	54m	48m	74h

s, m, and h denote seconds, minutes and hours, respectively.

Analytically, WALS has the time complexity of $O((M + N)K^3 + |\mathcal{R}|K^2)$, while eALS and RCD have the same time complexity which is K times smaller than that of WALS. As can be seen from the table, with the increase of K, WALS takes much longer time than eALS and RCD. Specifically, when K is 512, WALS requires 11.6 hours for one iteration on Amazon, while eALS only takes 12 minutes. Although eALS does not empirically shown to be K times faster than ALS due to the more efficient matrix inversion implementation (we used the fastest known algorithm [41] with time complexity around $O(K^{2.376})$), the speed-up is already very significant. Moreover, as RCD and eALS have the same analytical time complexity, their actual running time are in the same magnitude; the minor differences can be caused by some implementation details, such as the data structures used.

D. Effectiveness in Item Recommendation

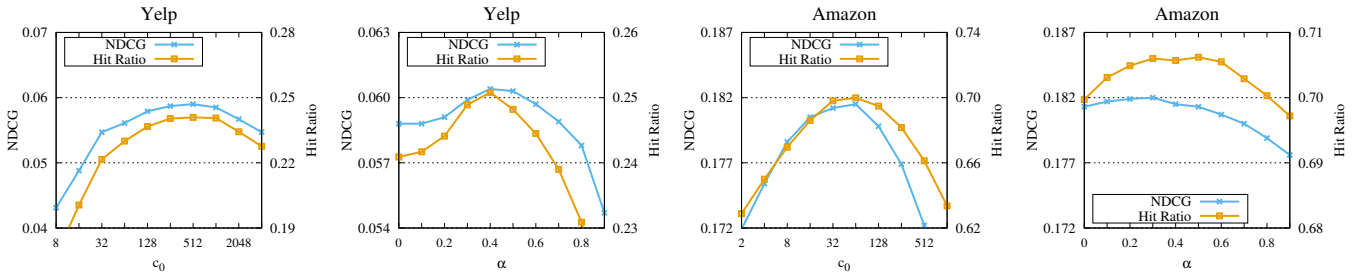
In this subsection, we explore the effectiveness eALS in the real-world task of item recommendation. As mentioned in Section IV-A, this is a personalized ranking task, and we employ the leave-out-one protocol to evaluate the performance with NDCG and HR. We aim to answer the following research questions:

RQ1 Is the non-uniform weighting strategy on missing data effective to offer better performance?

RQ2 How does eALS perform as compared with existing whole data-based MF methods for recommendation?

Next, we describe experimental results to answer the two questions. Note that our findings are consistent across the number of latent factors K, thus we show the results of $K = 128$ only, a relatively large number that retains good model capability.

RQ1: Non-Uniform Weights on Missing Data. Existing MF methods for recommendation assign a uniform weight on missing data for the ease of efficient optimization. This implies that all unrated items for a user have an equal probability to be negative, which may not be true. Since the visual interfaces of many Web systems tend to showcase popular items, when all other factors are equal, popular items are more likely to be known by users in general [42]. As such, it is reasonable to think that a miss on a popular item is more probable to be truly



(a) eALS vs. c_0 ($\alpha = 0$) (b) eALS vs. α ($c_0 = 512$) (c) eALS vs. c_0 ($\alpha = 0$) (d) eALS vs. α ($c_0 = 64$)

Fig. 3. Performance of eALS with respect to c_0 and α on Yelp and Amazon. Each subfigure shows the scores of varying one parameter with the other fixed.

irrelevant (as opposed to unknown) to the user. To account for this effect, we design the weights for missing entries based on item popularity:

$$w_{ui} = c_0 \frac{f_i^\alpha}{\sum_{j=1}^N f_j^\alpha}, \tag{20}$$

where f_i denotes the frequency of item i, in the training set: $|\mathcal{R}_i|/\sum_{j=1}^N |\mathcal{R}_j|$. The weight for each observed entry (i.e., c_{ui} in Eq. (9)) is set as 1, and c_0 is a hyper-parameter to determine the overall weight of missing data. The exponent α controls the significance level of popular items over unpopular ones — when $\alpha > 1$ the weights of popular items are promoted to strengthen the difference against unpopular ones; while setting α within the lower range of $(0, 1)$ suppresses the weight of popular items and has a smoothing effect. This weighting strategy basically assumes that the missing entries of popular items carry more negative signal. It is obvious that the rank size of such a weight matrix (on missing entries only) is 1, and with truncated SVD, we can get its low-rank representation as:

$$\begin{cases} \mathbf{a}_u = [c_0] \\ \mathbf{b}_i = [\frac{f_i^\alpha}{\sum_{j=1}^N f_j^\alpha}]. \end{cases} \tag{21}$$

Our fast eALS implementation is initialized with this setting on the weights of missing entries. It is worth noting that other non-uniform weighting strategies can also be applied here, such as the user-oriented scheme proposed in [28], or we can combine user-oriented and item-oriented schemes in Eq. (21). Since the aim of this experiment is to show the effectiveness of customizing weights in eALS, rather than demonstrating state-of-the-art recommendation performance, we leave this further exploration on the weighting scheme as future work. Note that this initialization leads to a recommendation method same as our preliminary work [1]. As such, the results presented below are also the same.

Results. Figure 3 shows the performance of eALS with respect to c_0 and α. Let us first focus on Figure 3a and 3c, where the weights of missing data follow a uniform distribution (controlled by $\alpha = 0$); we vary c_0 to study how does the overall weight of missing data affect the performance. The optimal c_0 on Yelp (Figure 3a) is around 512, and that on Amazon (Figure 3c) is around 64. Correspondingly, the weights of each zero entry are 0.02 and 0.0001 respectively ($w_0 = c_0/N$). However, both datasets exhibit similar patterns: when c_0 is smaller than the optimal value, the performance drops significantly. In other words, when the weights of zero entry are close to 0, the performance degrades. That reflects the importance of weighting the missing data. Moreover, too large c_0 also leads to bad performance. That is why the traditional SVD technique [17], which assigns the same weight to all the entries, is suboptimal here.

Then, we vary α with the optimal c_0 (in the case of $\alpha = 0$) to check the performance change. As demonstrated in Figure 3b and 3d, the optimal α is around 0.4 on both datasets. Below 0.4, with the increase of α, the performance of eALS is gradually improved. But when α increases above 0.5, the performance of eALS becomes worse. That reveals that weighting missing data according item popularity is important for recommendation. We further verify the improvement with the one-sample paired t-test. The results (p-value $< 10^{-4}$) for both metrics on the two datasets indicates the effectiveness of our method.

In the following experiments, we fix c_0 and α according to the best performance evaluated by HR, i.e., $c_0 = 512, \alpha = 0.4$ for Yelp and $c_0 = 64, \alpha = 0.5$ for Amazon.

RQ2: Performance Comparison. We compare eALS with two whole data-based MF methods which are originally designed for the item recommendation task:

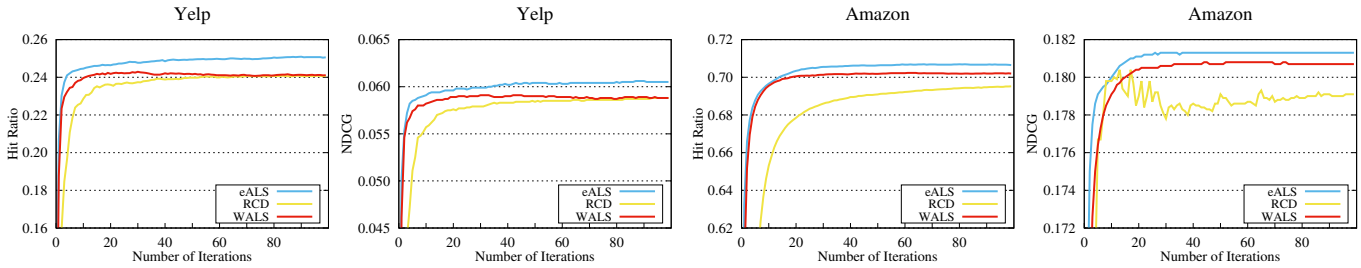
- **WALS** [22]. This is the weighted ALS method that optimizes the whole-data based MF. It assigns the same weight w_0 to all missing data. We tuned the w_0 carefully and reported the best performance.

- **RCD** [23]. This is the state-of-the-art implicit MF method that optimizes the same objective function as ALS but with a faster coordinate descent learner. We similarly tuned the w_0; for the line search related parameters, we use the suggested values in the authors' implementation[7].

The recommendation accuracy of each training iteration is shown in Figure 4. All improvements are statistically significant as evidenced by the one-sample paired t-test ($p < 0.01$). First of all, the performance of eALS is best upon convergence. We believe that the improvements are mainly from the weighting strategy on the missing data. Our proposed eALS assigns adaptive weights to the missing data while both WALS and RCD apply uniform weights.

Second, RCD converges slower than eALS and WALS. This is caused by the difference between global optimization and local optimization. In each iteration, RCD updates to a

[7] https://github.com/rdevooght/MF-with-prior-and-updates



(a) Iterations vs. HR (b) Iterations vs. NDCG (c) Iterations vs. HR (d) Iterations vs. NDCG

Fig. 4. Recommendation accuracy in each training iteration of three whole-data based MF methods, eALS, RCD and WALS (K = 128).

suboptimal point, which may be a wrong direction to the global optimal point. The effect of local optimization is also verified in Figure 4d. RCD attains high NDCG in a short time, while its low HR and the later oscillation demonstrate that the high NDCG of RCD is unstable. Another reason for this situation may come from the RCD's adaptive strategy. In the early iterations, the optimizer tends to make rapid learning by using a large learning rate, that may lead to the unexpected (i.e. suboptimal) results. Nevertheless, WALS outperforms RCD in most situations, that demonstrates ALS is better than gradient descent learner in these tasks.

V. Related work

Matrix Factorization is a representative method that represent a data matrix as two low-dimension matrices. The decomposition process can distill co-occurrence patterns in data [43]. Moreover, the reconstructed low-rank model can be used to recover missing information, such as its application in recommendation that predicts users' ratings on unknown items [4], [44].

However, in many real-world applications, the data matrices are can be highly sparse. For example, in recommendation, handling missing data is particularly important for learning from implicit data, since they provide valuable negative signal. Along this line, we can categorize previous works into two types: *sample-based learning* and *whole-data based learning*:

- The first type samples negative instances from missing data [10], [20], [45], [46]. For example, the BPR method proposed by Rendle et al. [20] randomly samples negative instances from missing entries, maximizing the margin between the model prediction of observed entries and that of sampled negatives. Recently, He et al. [39] develops adversarial training methods for BPR to increase the robustness of the learned model. By negative sampling, the number of negative instances is greatly reduced, therefore the overall time complexity is controllable [10]. However, the downside is that they usually have a slower convergence rate and the performance is highly dependent of the design of the sampler [18], [24], [25].

- The second type treats all missing entries as negative instances [15], [22], [34], [47]. For example, the WALS method proposed by Hu et al. [22] models all missing entries as negative instances with a label of 0, assigning them with a lower weight in point-wise regression learning. Recently, Ding et al. [47] develops a pairwise learning framework to model the margin between observed entries (based on view histories) and all missing entries. These methods model negative instances with a higher coverage, but the downside is that the learning algorithm could be much slower.

To pursue model effectiveness, we focus on whole-data based learning in this work, aiming to develop an efficient solution to address the inefficiency issue. For this line of research, several previous efforts have been made, such as [22], [23], [34], [34], [36], [37]. We find that these methods have a common limitation — weighting missing entries with a same weight. This design is mainly for efficiency concern, since fast learning algorithms can be obtained with this constraint. However, it decreases the modeling flexibility and may result in suboptimal performance. The works that are closest to ours are [15], [28], [35], which consider applying non-uniform weights on missing entries. However, these methods only supports simple weighting scheme, either row-based or column-based, and cannot be extended to other more complex schemes. This work addresses the research gap by developing efficient learning algorithms for any weighting scheme on missing data. Lastly, it is worth noting that the algorithm proposed in the recent work [28] is a special case of our fast eALS method, since it can be exactly recovered by setting the weights on missing entries to be user-oriented.

VI. Conclusion

In this paper, we studied the problem of learning MF with non-uniform weights on missing data. Targeting at the L_2 square loss, we first proposed to apply ALS optimization at each element level, namely, eALS. To address the efficiency challenge in solving the weighted MF problem, we then proposed a low-rank weighting strategy on missing data, which not only saves the space in storing weights but also allows us to further speedup the eALS method. To this end, we developed a fast eALS algorithm by a clever use of memoization caches, for which the time complexity is determined by the number of observed entries only rather than the whole data matrix. We conducted extensive experiments on two public rating datasets, verifying the correctness, efficiency, and effectiveness of our proposed fast eALS method.

We believe that optimizing MF with missing data is a fundamental problem in learning on sparse matrices. While most existing works assign a uniform weight on missing data, this work opens the door for designing complex weighting

schemes for missing data. This will benefit a wide variety of tasks that can be solved with MF. In future, we plan to extend eALS to MF with side information, such as spatial contexts [48], user reviews [49], visual content [50], and knowledge graphs [51]. Moreover, we will consider applying non-uniform weights for missing data on the more generic embedding models, such as collective factorization [7] and neural factorization machines [52]. Lastly, we are interested in applying our method on other tasks, such as the knowledge graph completion and word representation learning.

ACKNOWLEDGMENT

The authors thank the anonymous reviewers for their reviewing efforts. This research is supported by the National Natural Science Foundation of China (Grant No. 61772275, 61732007, 61321491, 61202320, 61501063), the Outstanding Youth Science Foundation (No. 61722204), the Scientific Research Foundation of Science and Technology Department of Sichuan Province(Grant No.2016JY0240), and the Collaborative Innovation Center of Novel Software Technology and Industrialization. This research is also part of NExT++, supported by the National Research Foundation, Prime Ministers Office, Singapore under its IRC@Singapore Funding Initiative. This work is a significant extension of [1], which appeared in the *Proceedings of SIGIR 2016*.

REFERENCES

[1] X. He, H. Zhang, M.-Y. Kan, and T.-S. Chua, "Fast matrix factorization for online recommendation with implicit feedback," in *SIGIR 2016*, pp. 549–558.

[2] Z. Ma, A. E. Teschendorff, A. Leijon, Y. Qiao, H. Zhang, and J. Guo, "Variational bayesian matrix factorization for bounded support data," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 37, no. 4, pp. 876–889, 2015.

[3] C. Li, J. Xing, A. Sun, and Z. Ma, "Effective document labeling with very few seed words: A topic model approach," in *CIKM*, 2016, pp. 85–94.

[4] X. Luo, M. Zhou, S. Li, Z. You, Y. Xia, and Q. Zhu, "A nonnegative latent factor model for large-scale sparse matrices in recommender systems via alternating direction method," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 27, no. 3, pp. 579–592, 2016.

[5] J. Tang, X. Shu, G. Qi, Z. Li, M. Wang, S. Yan, and R. Jain, "Tri-clustered tensor completion for social-aware image tag refinement," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 8, pp. 1662–1674, 2017.

[6] J. C. S. de Souza, T. M. L. Assis, and B. C. Pal, "Data compression in smart distribution systems via singular value decomposition," *IEEE Transactions on Smart Grid*, vol. 8, no. 1, pp. 275–284, 2017.

[7] X. He, M.-Y. Kan, P. Xie, and X. Chen, "Comment-based multi-view clustering of web 2.0 items," in *Proc. of WWW '14*, 2014, pp. 771–782.

[8] Z. Ma, Y. Lai, W. B. Kleijn, Y.-Z. Song, L. Wang, and J. Guo, "Variational bayesian learning for dirichlet process mixture of inverted dirichlet distributions in non-gaussian image feature modeling," *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1–15, 2018.

[9] C. Li, Y. Duan, H. Wang, Z. Zhang, A. Sun, and Z. Ma, "Enhancing topic modeling for short texts with auxiliary word embeddings," *ACM Transactions on Information Systems*, vol. 36, no. 2, p. 11, 2017.

[10] X. He, L. Liao, H. Zhang, L. Nie, X. Hu, and T.-S. Chua, "Neural collaborative filtering," in *WWW 2017*, pp. 173–182.

[11] W. Lei, X. Wang, M. Liu, I. Ilievski, X. He, and M. Kan, "SWIM: A simple word interaction model for implicit discourse relation recognition," in *IJCAI*, 2017, pp. 4026–4032.

[12] J. Tang, X. Shu, Z. Li, G.-J. Qi, and J. Wang, "Generalized deep transfer networks for knowledge propagation in heterogeneous domains," *ACM Transactions on Multimedia Computing, Communications, and Applications*, vol. 12, no. 4s, p. 68, 2016.

[13] Z. Zhao, H. Lu, D. Cai, X. He, and Y. Zhuang, "User preference learning for online social recommendation," *IEEE Transactions on Knowledge and Data Engineering*, vol. 28, no. 9, pp. 2522–2534, 2016.

[14] L. Liao, X. He, H. Zhang, and T.-S. Chua, "Attributed social network embedding," *IEEE Transactions on Knowledge and Data Engineering*, 2018.

[15] F. Yuan, X. Xin, X. He, G. Guo, W. Zhang, C. Tat-Seng, and J. M. Jose, "fBGD: Learning embeddings from positive unlabeled data with BGD," in *UAI*, 2018.

[16] Y. Koren, "Factorization meets the neighborhood: A multifaceted collaborative filtering model," in *KDD 2008*, pp. 426–434.

[17] P. Cremonesi, Y. Koren, and R. Turrin, "Performance of recommender algorithms on top-n recommendation tasks," in *RecSys 2010*, pp. 39–46.

[18] X. Xin, F. Yuan, X. He, and J. Jose, "Allvec: Learning word representations without negative sampling," in *ACL 2018*, pp. 1853–1862.

[19] J. Chen, H. Zhang, X. He, L. Nie, W. Liu, and T.-S. Chua, "Attentive collaborative filtering: Multimedia recommendation with item- and component-level attention," in *SIGIR 2017*, 2017, pp. 335–344.

[20] S. Rendle, C. Freudenthaler, Z. Gantner, and L. Schmidt-Thieme, "Bpr: Bayesian personalized ranking from implicit feedback," in *UAI 2009*, pp. 452–461.

[21] C. Yang, C. Zhang, X. Chen, J. Ye, and J. Han, "Did you enjoy the ride: Understanding passenger experience via heterogeneous network embedding," in *ICDE*, 2018.

[22] Y. Hu, Y. Koren, and C. Volinsky, "Collaborative filtering for implicit feedback datasets," in *ICDM 2008*, pp. 263–272.

[23] R. Devooght, N. Kourtellis, and A. Mantrach, "Dynamic matrix factorization with priors on unknown values," in *KDD 2015*, pp. 189–198.

[24] S. Rendle and C. Freudenthaler, "Improving pairwise learning for item recommendation from implicit feedback," in *WSDM 2014*, pp. 273–282.

[25] J. Ding, F. Feng, X. He, G. Yu, Y. Li, and D. Jin, "An improved sampler for bayesian personalized ranking by leveraging view data," in *WWW 2018*, pp. 13–14.

[26] V. Klema and A. Laub, "The singular value decomposition: Its computation and some applications," *IEEE Transactions on automatic control*, vol. 25, no. 2, pp. 164–176, 1980.

[27] N. Srebro and T. Jaakkola, "Weighted low-rank approximations," in *ICML 2003*, pp. 720–727.

[28] H. Li, X. Diao, J. Cao, and Q. Zheng, "Collaborative filtering recommendation based on all-weighted matrix factorization and fast optimization," *IEEE Access*, vol. 6, pp. 25 248–25 260, 2018.

[29] Y. Zhang, G. Lai, M. Zhang, Y. Zhang, Y. Liu, and S. Ma, "Explicit factor models for explainable recommendation based on phrase-level sentiment analysis," in *SIGIR*, 2014, pp. 83–92.

[30] S. Wang, J. Tang, Y. Wang, and H. Liu, "Exploring hierarchical structures for recommender systems," *IEEE Transactions on Knowledge and Data Engineering*, vol. 30, no. 6, pp. 1022–1035, 2018.

[31] B. M. Marlin, R. S. Zemel, S. Roweis, and M. Slaney, "Collaborative filtering and the missing at random assumption," in *UAI 2007*, pp. 267–276.

[32] L. Komzsik, *The Lanczos method: evolution and application*. SIAM, 2003.

[33] S. Rendle, Z. Gantner, C. Freudenthaler, and L. Schmidt-Thieme, "Fast context-aware recommendations with factorization machines," in *SIGIR 2011*, pp. 635–644.

[34] I. Bayer, X. He, B. Kanagal, and S. Rendle, "A generic coordinate descent framework for learning from implicit feedback," in *WWW 2017*, pp. 1341–1350.

[35] R. Pan, Y. Zhou, B. Cao, N. Liu, R. Lukose, M. Scholz, and Q. Yang, "One-class collaborative filtering," in *ICDM 2008*, pp. 502–511.

[36] I. Pilászy, D. Zibriczky, and D. Tikk, "Fast als-based matrix factorization for explicit and implicit feedback datasets," in *RecSys 2010*, pp. 71–78.

[37] M. Volkovs and G. W. Yu, "Effective latent models for binary feedback in recommender systems," in *SIGIR 2015*, pp. 313–322.

[38] R. Gemulla, E. Nijkamp, P. J. Haas, and Y. Sismanis, "Large-scale matrix factorization with distributed stochastic gradient descent," in *KDD 2011*, pp. 69–77.

[39] X. He, Z. He, X. Du, and T. Chua, "Adversarial personalized ranking for recommendation," in *SIGIR*, 2018, pp. 355–364.

[40] S. Rendle, D. Fetterly, E. J. Shekita, and B.-y. Su, "Robust large-scale machine learning in the cloud," in *KDD 2016*, pp. 1125–1134.

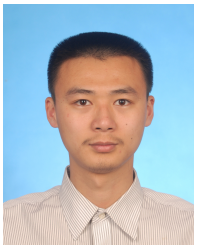
[41] D. Coppersmith and S. Winograd, "Matrix multiplication via arithmetic progressions," in *STOC 1987*, pp. 1–6.

[42] X. He, M. Gao, M.-Y. Kan, Y. Liu, and K. Sugiyama, "Predicting the popularity of web 2.0 items based on user comments," in *SIGIR 2014*, pp. 233–242.

[43] Z. Ma, J. Xue, A. Leijon, Z. Tan, Z. Yang, and J. Guo, "Decorrelation of neutral vector variables: Theory and applications," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 29, no. 1, pp. 129–143, 2018.
[44] H. Zhang, F. Shen, W. Liu, X. He, H. Luan, and T.-S. Chua, "Discrete collaborative filtering," in *SIGIR 2016*, pp. 325–334.
[45] X. He, X. Du, X. Wang, F. Tian, J. Tang, and T. Chua, "Outer product-based neural collaborative filtering," in *IJCAI*, 2018, pp. 2227–2233.
[46] Y. Zhang, Q. Ai, X. Chen, and W. B. Croft, "Joint representation learning for top-n recommendation with heterogeneous information sources," in *CIKM*, 2017, pp. 1449–1458.
[47] J. Ding, G. Yu, X. He, Y. Quan, Y. Li, T. Chua, D. Jin, and J. Yu, "Improving implicit recommender systems with view data," in *IJCAI*, 2018, pp. 3343–3349.
[48] C. Yang, L. Bai, C. Zhang, Q. Yuan, and J. Han, "Bridging collaborative filtering and semi-supervised learning: a neural approach for poi recommendation," in *KDD*, 2017, pp. 1245–1254.
[49] X. He, T. Chen, M.-Y. Kan, and X. Chen, "Trirank: Review-aware explainable recommendation by modeling aspects," in *CIKM 2015*, pp. 1661–1670.
[50] S. Wang, Y. Wang, J. Tang, K. Shu, S. Ranganath, and H. Liu, "What your images reveal: Exploiting visual contents for point-of-interest recommendation," in *WWW*, 2017, pp. 391–400.
[51] Q. Ai, V. Azizi, X. Chen, and Y. Zhang, "Learning heterogeneous knowledge base embeddings for explainable recommendation," *Algorithms*, no. 137, 2018.
[52] X. He and T. Chua, "Neural factorization machines for sparse predictive analytics," in *SIGIR*, 2017, pp. 355–364.

Xiangnan He is currently a senior research fellow with School of Computing, National University of Singapore (NUS). He received his Ph.D. in Computer Science from NUS. His research interests span information retrieval, data mining, and multi-media analytics. He has over 50 publications appeared in several top conferences such as SIGIR, WWW, and MM, and journals including TKDE, TOIS, and TMM. His work on recommender systems has received the Best Paper Award Honourable Mention in WWW 2018 and ACM SIGIR 2016. Moreover, he has served as the PC member for several top conferences including SIGIR, WWW, MM, KDD etc, and the regular reviewer for journals including TKDE, TOIS, TMM, TNNLS etc.

Jiuhui Tang is currently a Professor in School of Computer Science and Engineering, Nanjing University of Science and Technology, China. He received the B.Eng. and Ph.D. degrees from the University of Science and Technology of China, Hefei, China, in 2003 and 2008, respectively. From 2008 to 2010, he worked as a research fellow in School of Computing, National University of Singapore. His current research interests include multimedia content analysis and retrieval, social media mining and machine learning. He has authored over 100 papers in top-tier journals and conferences. Dr. Tang is a recipient of the inaugural ACM China Rising Star Award, the Best Paper Awards in ACM MM 2007, PCM 2011 and ICIMCS 2011, the Best Paper Runner-up in ACM MM 2015, and the Best Student Paper Awards in MMM 2016 and ICIMCS 2017.



Xiaoyu Du is currently a lecturer in the School of Software Engineering of Chengdu University of Information Technology, Chengdu, a visiting scholar in the NExT++ research center of National University of Singapore, and a Ph.D. candidate of University of Electronic Science and Technology of China, Chengdu. He received his M.E. degree in computer software and theory in 2011 and B.S. degree in computer science and technology in 2008, both from Beijing Normal University, Beijing. His research interests include information retrieval, computer vision, and machine learning.

Richang Hong Richang Hong received the PhD degree from the University of Science and Technology of China, Hefei, China, in 2008. He is a professor with the Hefei University of Technology, Hefei, China. He was a research fellow in the School of Computing, National University of Singapore, from Sep. 2008 to Dec. 2010. He has coauthored more than 70 publications in the areas of his research interests, which include multimedia content analysis and social media. He received the Best Paper Award in the ACM Multimedia 2010, Best Paper Award in the ACM ICMR 2015, and the Honorable Mention of the IEEE Trans. on Multimedia Best Paper Award 2015. He served as the associate editor of the IEEE Multimedia Magazine, Information Sciences and Signal Processing, Elsevier and the technical program chair of the MMM 2016 and ACM ICIMCS 2017. He is a member of the IEEE, the ACM and the executive committee member of the ACM SIGMM China Chapter.

Tongwei Ren Tongwei Ren is currently an associate professor with the State Key Laboratory for Novel Software Technology, Nanjing University. He received his Ph.D. in computer science and technology from Nanjing University. His research interest mainly includes multimedia computing and computer vision. He has over 50 publications appeared in international conferences such as MM, ICCV, and AAAI, and journals such as TIP and NEUCOM. His works on image analysis have received the Best Paper Award Honorable Mention of ICIMCS 2014 and the Best Paper Runner-up of PCM 2015. He has served as the workshop chair of ICIMCS 2015, the publication chair of PCM 2017 and the program chair of ICIMCS 2018. He has also served as the PC member for conferences including BigMM, ICIP, and PCM, and the regular reviewer for journals including TIP, TOMM and TMM.

Tat-Seng Chua is the KITHCT Chair Professor at the School of Computing, National University of Singapore. He was the Acting and Founding Dean of the School during 1998-2000. Dr Chuas main research interest is in multimedia information retrieval and social media analytics. In particular, his research focuses on the extraction, retrieval and question-answering (QA) of text and rich media arising from the Web and multiple social networks. He is the co-Director of NExT, a joint Center between NUS and Tsinghua University to develop technologies for live social media search. Dr Chua is the 2015 winner of the prestigious ACM SIGMM award for Outstanding Technical Contributions to Multimedia Computing, Communications and Applications. He is the Chair of steering committee of ACM International Conference on Multimedia Retrieval (ICMR) and Multimedia Modeling (MMM) conference series. Dr Chua is also the General Co-Chair of ACM Multimedia 2005, ACM CIVR (now ACM ICMR) 2005, ACM SIGIR 2008, and ACM Web Science 2015. He serves in the editorial boards of four international journals. Dr. Chua is the co-Founder of two technology startup companies in Singapore. He holds a PhD from the University of Leeds, UK.