A Generalized and Fast-converging Non-negative Latent Factor Model for Predicting User Preferences in Recommender Systems

Ye Yuan
Chongqing Engineering Research Center of Big Data Application for Smart Cities
Chongqing Key Laboratory of Big Data and Intelligent Computing, Chongqing Institute of Green and Intelligent Technology, Chinese Academy of Sciences, Chongqing, China, yuanye@cigit.ac.cn

Xin Luo†
Chongqing Engineering Research Center of Big Data Application for Smart Cities
Chongqing Key Laboratory of Big Data and Intelligent Computing, Chongqing Institute of Green and Intelligent Technology, Chinese Academy of Sciences, Chongqing, China, luoxin21@cigit.ac.cn

Mingsheng Shang
Chongqing Engineering Research Center of Big Data Application for Smart Cities
Chongqing Key Laboratory of Big Data and Intelligent Computing, Chongqing Institute of Green and Intelligent Technology, Chinese Academy of Sciences, Chongqing, China, msshang@cigit.ac.cn

Di Wu
Chongqing Engineering Research Center of Big Data Application for Smart Cities
Chongqing Key Laboratory of Big Data and Intelligent Computing, Chongqing Institute of Green and Intelligent Technology, Chinese Academy of Sciences, Chongqing, China, wudi@cigit.ac.cn

ABSTRACT

Recommender systems (RSs) commonly describe its user-item preferences with a high-dimensional and sparse (HiDS) matrix filled with non-negative data. A non-negative latent factor (NLF) model relying on a single latent factor-dependent, non-negative and multiplicative update (SLF-NMU) algorithm is frequently adopted to process such an HiDS matrix. However, an NLF model mostly adopts Euclidean distance for its objective function, which is naturally a special case of α-β-divergence. Moreover, it frequently suffers slow convergence. For addressing these issues, this study proposes a generalized and fast-converging non-negative latent factor (GFNLF) model. Its main idea is two-fold: a) adopting α-β-divergence for its objective function, thereby enhancing its representation ability for HiDS data; b) deducing its momentum-incorporated non-negative multiplicative update (MNMU) algorithm, thereby achieving its fast convergence. Empirical studies on two HiDS matrices emerging from real RSs demonstrate that with carefully-tuned hyperparameters, a GFNLF model outperforms state-of-the-art models in both computational efficiency and prediction accuracy for missing data of an HiDS matrix.

CCS CONCEPTS

• Computing methodologies • Machine learning • Machine learning approaches • Factorization methods

KEYWORDS

Non-negative latent factor, α-β-divergence, Momentum, High-dimensional and sparse, Recommender system, User preference

ACM Reference format:

Ye Yuan, Xin Luo, Mingsheng Shang and Di Wu. 2020. A Generalized and Fast-converging Non-negative Latent Factor Model for Predicting User Preferences in Recommender Systems. In *Proceedings of WWW '20: The Web Conference 2020 (WWW '20), April 20-24, 2020, Taipei, Taiwan.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3366423.3380133

1 Introduction

The rapid expansion world-wide-web has caused the problem of information overload. How to develop an intelligent system to filter the useful information out of massive data has become a heated issue. In such context, RSs have been proven highly efficient in addressing information overload by connecting valuable information to people actively, rather than passively, according to their information usage history [1-6, 39, 40].

In RSs, a user-item rating matrix [7-9] is usually the fundamental data source, where each entry is modeled according to the corresponding user-item usage history. High values in a rating matrix commonly denote strong user-item preferences [10-12]. With exponentially growing quantities of users and items in RSs, only a few items can be observed by each user. This phenomenon leads to the rating matrix high-dimensional and

† X. Luo (*Corresponding author*) is also with the Hengrui (Chongqing) Artificial Intelligence Research Center, Department of Big Data Analyses Techniques, Cloudwalk, Chongqing, China. Y. Yuan is also with the University of Chinese Academy of Sciences, Beijing, China.

WWW '20, April 20-24, 2020, Taipei, Taiwan

ACM ISBN 978-1-4503-7023-3/20/04
https://doi.org/10.1145/3366423.3380133

sparse (HiDS) [7, 13, 14], which contains numerous missing values. For instance, the Douban matrix [15] collected by the Chinese largest online book, movie and music database includes 58,541 items and 129,490 users. However, it only contains 16,830,839 known ratings and the density is 0.22%. Although a rating matrix generated by RSs can be extremely sparse, it contains rich knowledge regarding various desired patterns like item clustering [16] and users' potential favorites [17].

According to the previous work [18-21, 41], a latent factor (LF) model has been proven to address such an HiDS matrix efficiently. However, it fails to fulfill the non-negativity constraints. Note that an HiDS matrix in RSs is commonly defined to be non-negative [7, 22-24]. Therefore, non-negative features are able to better represent an HiDS matrix filled with non-negative data and describe hidden patterns like user profile features more precisely.

For this purpose, great efforts have been made for performing non-negative LF analysis on HiDS matrices. The weighted non-negative matrix factorization (WNMF) model by [25] represents the missing values through a binary weight matrix. The non-negative and multiplicative update (NMU) [26] algorithm is applied to WNMF for extracting desired non-negative LFs. A non-negative matrix completion (NMC) model by [27] adopts a full approximation to the target matrix and then extract non-negative LFs with the projected alternating least squares (ALS) [28]. Hernando et al. [29] propose a non-negative matrix factorization based on a Bayesian probabilistic model, which associates components to each user and each item with an understandable probabilistic meaning. Although these models can address such an HiDS matrix, they suffer from high computational and storage complexity since they rely on full matrices.

For performing non-negative LF analysis on HiDS matrices more efficiently, a single latent factor-dependent, non-negative and multiplicative update (SLF-NMU) algorithm is proposed [7] to extract non-negative LFs. SLF-NMU greatly alleviates the computational and storage burden since it only relies on the known values of an HiDS matrix rather than on the full matrix. With the high efficient update algorithm, a non-negative latent factor (NLF) model is further proposed [7]. Given an HiDS matrix, NLF's computational complexity is only linearly related to the known values. Meanwhile, its storage complexity is linear with the sum of its user and item counts only.

Motivations. Although an NLF model is able to represent an HiDS matrix with low cost in both computation and storage, there are still some bottlenecks to be addressed urgently:

a) NLF mostly relies on an objective function defined via Euclidean distance, which is naturally a special case of α-β-divergence [30]. Note that the objective function has vital effects on prediction accuracy for missing values. Can an NLF model with α-β-divergence in its objective function enhance its representation ability for HiDS data?

b) SLF-NMU is specifically designed for an HiDS matrix with low computational and storage complexity. However, it leads to slow convergence [7]. In other words, NLF models usually consume many iterations to converge, thereby resulting in considerable time cost on large-scale datasets. Hence, how to further achieve its fast convergence becomes highly significant.

Contributions. Aiming at addressing the above issues, this paper innovatively proposes a generalized and fast-converging non-negative latent factor (GFNLF) model. The main contributions of this paper include:

a) Adopting α-β-divergence in objective function for making NLF a generalized form, thereby enhancing its representation ability for HiDS data;

b) Incorporating a generalized momentum method in SLF-NMU to obtain a momentum-incorporated non-negative multiplicative update (MNMU) algorithm, therefore greatly improving its convergence rate;

c) Presenting algorithm design and analysis for a GFNLF model.

We conduct extensive experiments on two HiDS matrices emerging from real RSs to justify GFNLF's excellent performance on predicting missing values in HiDS matrices.

The rest of this paper is organized as follows: Section 2 gives the preliminaries. Section 3 presents our methods. Section 4 gives the experimental results. Finally, Section 5 concludes this paper.

2 Preliminaries

2.1 Problem Statement

An HiDS matrix is commonly used for describing the certain interactions between users and items in RSs [31], which is defined as follows:

***Definition* 1:** Let U and I be two large entity sets; $R^{|U|\times|I|}$ is a matrix where each value $r_{u,i}$ is connected with user u's preference on item i. Let R_K and R_U denote the known and unknown entity sets. R is an HiDS matrix if $|R_K| \ll |R_U|$.

An NLF model builds a low-rank approximation to an HiDS matrix. Given R, it is defined as follows:

***Definition* 2.** Given R and R_K, $\hat{R}$ is R's rank-f approximation built on R_K only and $\hat{R}=XY^{\mathrm{T}}$. Note that f denotes the rank of $\hat{R}$ and $f \ll \min\{|U|,|I|\}$. $X^{|U|\times f}$ and $Y^{|I|\times f}$ denote the LF matrices corresponding to U and I.

Note that R is commonly defined to be non-negative. For better representing the target matrix and describing its hidden patterns more precisely, X and Y are expected to be non-negative to achieve an NLF model. Hence, for such purposes, an objective measuring R_K and the corresponding entry set in $\hat{R}$ is desired. With Euclidean distance, such an objective function is formulated by:

$$\begin{aligned}\varepsilon(X,Y) &= \sum_{(u,i)\in R_K}\left(\left(r_{u,i}-\hat{r}_{u,i}\right)^2+\lambda_X\|X\|_F^2+\lambda_Y\|Y\|_F^2\right)\\ &= \sum_{(u,i)\in R_K}\left(\left(r_{u,i}-\sum_{m=1}^{f}x_{u,m}y_{m,i}\right)^2+\lambda_X\sum_{m=1}^{f}x_{u,m}^2+\lambda_Y\sum_{m=1}^{f}y_{m,i}^2\right),\\ s.t.\quad &\forall u\in U, i\in I, m\in\{1,2,\cdots,f\}: x_{u,m}\geq 0, y_{m,i}\geq 0.\end{aligned} \tag{1}$$

where $\hat{r}_{u,i}=\sum_{m=1}^{f} x_{u,m}y_{m,i}$ and denotes the values in $\hat{R}$. $||\cdot||_F$ computes the Frobenius norm of a matrix. The regularization coefficients λ_X and λ_Y control the regularization effect, and the terms behind them are the regularization terms to avoid overfitting.

2.2 An NLF Model

An NLF model adopts the SLF-NMU algorithm to minimize objective (1) for extracting non-negative LFs from the known entity set R_K. This algorithm firstly applies the additive gradient descent (AGD) to each desired LF, resulting in the following update rules:

$$\begin{aligned}&\arg\min_{X,Y}\varepsilon(X,Y)\overset{AGD}{\Rightarrow}\\&\begin{cases}x_{u,m}\leftarrow x_{u,m}+\eta_{u,m}\sum_{i\in I_u}y_{m,i}r_{u,i}-\eta_{u,m}\sum_{i\in I_u}\left(y_{m,i}\hat{r}_{u,i}+\lambda_X x_{u,m}\right),\\ y_{m,i}\leftarrow y_{m,i}+\eta_{m,i}\sum_{u\in U_i}x_{u,m}r_{u,i}-\eta_{m,i}\sum_{u\in U_i}\left(x_{u,m}\hat{r}_{u,i}+\lambda_Y y_{m,i}\right).\end{cases}\end{aligned}\tag{2}$$

where $\eta_{u,m}$ and $\eta_{m,i}$ denote the learning rates corresponding to $x_{u,m}$ and $y_{m,i}$. I_u denotes the subset of I related to user u and U_i denotes the subset of U related to item i. For ensuring $x_{u,m}$ and $y_{m,i}$ non-negative during the updating process, SLF-NMU cancels the negative terms by manipulating $\eta_{u,m}$ and $\eta_{m,i}$. Consequently, the update rules for $x_{u,m}$ and $y_{m,i}$ are as follows:

$$\begin{aligned}&\arg\min_{X,Y}\varepsilon(X,Y)\overset{SLF-NMU}{\Rightarrow}\\&\begin{cases}x_{u,m}\leftarrow x_{u,m}\dfrac{\sum_{i\in I_u}y_{m,i}r_{u,i}}{\sum_{i\in I_u}y_{m,i}\hat{r}_{u,i}+\lambda_X|I_u|x_{u,m}},\\ y_{m,i}\leftarrow y_{m,i}\dfrac{\sum_{u\in U_i}x_{u,m}r_{u,i}}{\sum_{u\in U_i}x_{u,m}\hat{r}_{u,i}+\lambda_Y|U_i|y_{m,i}}.\end{cases}\end{aligned}\tag{3}$$

With (3), X and Y are achieved non-negative.

3 Methodology

3.1 An NLF Model with α-β-divergence

The commonly adopted Euclidean distance in NLF models is naturally a special case of α-β-divergence [30], which is parameterized by the two tuning parameters, α and β. Given U, I, R and R_K, the α-β-divergence measuring the dissimilarity between R_K and the corresponding entry set in $\hat{R}$ for an NLF model is defined as follows:

$$\begin{aligned}&d_{\alpha-\beta}=-\frac{1}{\alpha\beta}\sum_{r_{u,i}\in R_K}\left(r_{u,i}^{\alpha}\hat{r}_{u,i}^{\beta}-\frac{\alpha}{\alpha+\beta}r_{u,i}^{\alpha+\beta}-\frac{\beta}{\alpha+\beta}\hat{r}_{u,i}^{\alpha+\beta}\right)\\&for\ \alpha\neq 0,\beta\neq 0,\alpha+\beta\neq 0.\end{aligned}\tag{4}$$

By integrating the regularization terms and non-negativity constraints into $d_{\alpha-\beta}$, a generalized objective function is:

$$\begin{aligned}&\varepsilon=-\frac{1}{\alpha\beta}\sum_{r_{u,i}\in R_K}\left(\begin{array}{l}r_{u,i}^{\alpha}\hat{r}_{u,i}^{\beta}-\frac{\alpha}{\alpha+\beta}r_{u,i}^{\alpha+\beta}-\frac{\beta}{\alpha+\beta}\hat{r}_{u,i}^{\alpha+\beta}\\+\lambda_X\sum_{m=1}^{f}x_{u,m}^2+\lambda_Y\sum_{m=1}^{f}y_{m,i}^2\end{array}\right)\\&for\ \alpha\neq 0,\beta\neq 0,\alpha+\beta\neq 0\\&s.t.\ \ \forall u\in U,i\in I,m\in\{1,2,\cdots,f\}:x_{u,m}\geq 0,y_{m,i}\geq 0.\end{aligned}\tag{5}$$

The aforementioned analyses show that SLF-NMU algorithm is efficient to build an NLF model on HiDS matrices. To achieve a generalized form of SLF-NMU, this work analyzes the Karush-Kuhn-Tucker (KKT) conditions of the objective function with a more solid deduction.

Let $\Gamma^{|U|\times f}$ and $K^{f\times|I|}$ be the Lagrangian multipliers corresponding to the constraints $x_{u,m}\geq 0$ and $y_{m,i}\geq 0$, respectively. Then we obtain the Lagrangian function corresponding to (5):

$$\begin{aligned}L&=\varepsilon+tr\left(\Gamma X^{\mathrm{T}}\right)+tr\left(\mathrm{K}^{\mathrm{T}}Y\right)\\&=\varepsilon+\sum_{u}\sum_{m}\gamma_{u,m}x_{u,m}+\sum_{i}\sum_{m}\kappa_{m,i}y_{m,i},\end{aligned}\tag{6}$$

where the operator $tr(\cdot)$ computes the trace of matrix. Considering the partial derivatives of L with respect to $x_{u,m}$ and $y_{m,i}$, we have the following deduction based on (6):

$$\begin{cases}\dfrac{\partial L}{\partial x_{u,m}}=\dfrac{\partial\varepsilon}{\partial x_{u,m}}+\gamma_{u,m}=\sum_{i\in I_u}\left(-\frac{1}{\alpha}r_{u,i}^{\alpha}\hat{r}_{u,i}^{\beta-1}y_{m,i}+\frac{1}{\alpha}\hat{r}_{u,i}^{\alpha+\beta-1}y_{m,i}+2\lambda_X x_{u,m}\right)+\gamma_{u,m},\\ \dfrac{\partial L}{\partial y_{m,i}}=\dfrac{\partial\varepsilon}{\partial y_{m,i}}+\kappa_{m,i}=\sum_{u\in U_i}\left(-\frac{1}{\alpha}r_{u,i}^{\alpha}\hat{r}_{u,i}^{\beta-1}x_{u,m}+\frac{1}{\alpha}\hat{r}_{u,i}^{\alpha+\beta-1}x_{u,m}+2\lambda_Y y_{m,i}\right)+\kappa_{m,i}.\end{cases}\tag{7}$$

We have the following observations from (6) and (7): a) to achieve the local optima of L with respect to $x_{u,m}$ and $y_{m,i}$, the partial derivatives in (7) should be set at zero simultaneously; b) the KKT condition of the Lagrangian function (6) is $\forall u\in U$, $i\in I$, $m\in\{1,\ldots,f\}$: $\gamma_{u,m}x_{u,m}=0$, $\kappa_{m,i}y_{m,i}=0$;

With (7) and the KKT condition (6), we obtain:

$$\begin{cases}x_{u,m}\sum_{i\in I_u}\left(-\frac{1}{\alpha}r_{u,i}^{\alpha}\hat{r}_{u,i}^{\beta-1}y_{m,i}+\frac{1}{\alpha}\hat{r}_{u,i}^{\alpha+\beta-1}y_{m,i}+2\lambda_X x_{u,m}\right)=0,\\ y_{m,i}\sum_{u\in U_i}\left(-\frac{1}{\alpha}r_{u,i}^{\alpha}\hat{r}_{u,i}^{\beta-1}x_{u,m}+\frac{1}{\alpha}\hat{r}_{u,i}^{\alpha+\beta-1}x_{u,m}+2\lambda_Y y_{m,i}\right)=0.\end{cases}\tag{8}$$

Note that (8) can be rearranged as following equation:

$$\begin{cases}x_{u,m}\sum_{i\in I_u}\left(\frac{1}{\alpha}\hat{r}_{u,i}^{\alpha+\beta-1}y_{m,i}+2\lambda_X x_{u,m}\right)=x_{u,m}\sum_{i\in I_u}\frac{1}{\alpha}r_{u,i}^{\alpha}\hat{r}_{u,i}^{\beta-1}y_{m,i},\\ y_{m,i}\sum_{u\in U_i}\left(\frac{1}{\alpha}\hat{r}_{u,i}^{\alpha+\beta-1}x_{u,m}+2\lambda_Y y_{m,i}\right)=y_{m,i}\sum_{u\in U_i}\frac{1}{\alpha}r_{u,i}^{\alpha}\hat{r}_{u,i}^{\beta-1}x_{u,m}.\end{cases}\tag{9}$$

With (9), the iterative expressions of $x_{u,m}$ and $y_{m,i}$ are formulated as follows:

$$\arg\min_{X,Y} \varepsilon(X,Y) \overset{SLF-NMU}{\Rightarrow} \begin{cases} x_{u,m} \leftarrow x_{u,m} \dfrac{\sum_{i\in I_u} r_{u,i}^{\alpha} \hat{r}_{u,i}^{\beta-1} y_{m,i}}{\sum_{i\in I_u} \hat{r}_{u,i}^{\alpha+\beta-1} y_{m,i} + \alpha\lambda_X |I_u| x_{u,m}}, \\ y_{m,i} \leftarrow y_{m,i} \dfrac{\sum_{u\in U_i} r_{u,i}^{\alpha} \hat{r}_{u,i}^{\beta-1} x_{u,m}}{\sum_{u\in U_i} \hat{r}_{u,i}^{\alpha+\beta-1} x_{u,m} + \alpha\lambda_Y |U_i| y_{m,i}}. \end{cases} \tag{10}$$

which is a generalized form of SLF-NMU algorithm. It can enhance representation ability for HiDS data by carefully choosing the values of α and β. Note that by substituting α=β=1 into (10), we have the update rule (3) for an NLF model with Euclidean distance [7].

3.2 A Generalized Momentum Method

As demonstrated by prior research [32], a standard momentum method can be used to accelerate a gradient descent (GD) algorithm significantly. The principle of a standard momentum method determines the current update state based on the current gradient and the latest update state. Given the decision parameter θ and objective function $\varepsilon(\theta)$, a standard momentum-based GD algorithm is defined as follows:

$$\begin{aligned} a_0 &= 0, \\ a_n &= \kappa a_{n-1} + \eta \nabla_{\theta} \varepsilon(\theta_{n-1}), \\ \theta_n &= \theta_{n-1} - a_n; \end{aligned} \tag{11}$$

where a_0 denotes the initial state of the update velocity and is always set at 0. The update velocity of the nth and (n-1)th iteration is a_n and a_{n-1}, respectively, κ denotes the momentum control parameter, and η denotes the learning rate. From (11), we can see that a standard momentum-based GD algorithm depends on gradients explicitly.

As shown in (10), SLF-NMU trains non-negative LFs multiplicatively for keeping their non-negativity and It depends on gradients implicitly. Therefore, a standard momentum method is incompatible with SLF-NMU. It means we need to adjust the standard momentum method to compatible with an algorithm implicitly depending on gradients.

Let θ'_n denote the predicted state of the decision parameter relying on the adopted algorithm (SLF-NMU in this paper) after the nth iteration, then we can calculate the update increment instead of explicit gradients as follows:

$$\Delta_n = \theta'_n - \theta_{n-1}. \tag{12}$$

Subsequently, we can formulate the update velocity in the nth iteration based on (11):

$$a_n = \kappa a_{n-1} - \Delta_n = \kappa a_{n-1} - (\theta'_n - \theta_{n-1}). \tag{13}$$

Based on (11) and (13), a generalized momentum method can be obtained by reformulating a standard momentum method:

$$\begin{aligned} a_0 &= 0, \\ a_n &= \kappa a_{n-1} - (\theta'_n - \theta_{n-1}), \\ \theta_n &= \theta_{n-1} - a_n. \end{aligned} \tag{14}$$

According to (14), we see that a generalized momentum method relies on the update increment, which is obtained by the latest state of the decision parameter and the predicted state of the decision parameter relying on SLF-NMU. Although SLF-NMU algorithm depends on gradients implicitly, the update increment can still be achieved. Hence, a generalized momentum method is compatible with SLF-NMU.

3.3 A GFNLF Model

By combining Sections 3.1 and 3.2, we further propose the MNMU algorithm to improve the convergence rate, thereby achieving a GFNLF model.

Let X_{n-1} and Y_{n-1} denote the state of X and Y after the (n-1)th iteration, and X'_n and Y'_n denote the predicted state after the nth iteration with SLF-NMU, respectively. Therefore, the predicted state of X and Y of the nth iteration is given by:

$$\theta'_n = \arg\min_{\theta}^{SLF-NMU} \theta_{n-1} \Rightarrow (X'_n, Y'_n) = \arg\min_{X,Y}^{SLF-NMU} \varepsilon(X_{n-1}, Y_{n-1}). \tag{15}$$

Subsequently, the update increment caused by SLF-NMU is computed as:

$$\Delta_n = \theta'_n - \theta_{n-1} = \begin{bmatrix} X'_n \\ Y'_n \end{bmatrix} - \begin{bmatrix} X_{n-1} \\ Y_{n-1} \end{bmatrix}. \tag{16}$$

Based on (12) and (13), a_1 is calculated as follows:

$$a_1 = \kappa a_0 - \Delta_1 = -\begin{bmatrix} X'_1 \\ Y'_1 \end{bmatrix} + \begin{bmatrix} X_0 \\ Y_0 \end{bmatrix}, \tag{17}$$

where X_0 and Y_0 denote the initial state of X and Y, which are randomly generated and non-negative initial defined [7]. Thus, the update rule for X_1 and Y_1 is achieved based on (14):

$$\begin{bmatrix} X_1 \\ Y_1 \end{bmatrix} = \begin{bmatrix} X_0 \\ Y_0 \end{bmatrix} - a_1 = \begin{bmatrix} X_0 \\ Y_0 \end{bmatrix} + \begin{bmatrix} X'_1 \\ Y'_1 \end{bmatrix} - \begin{bmatrix} X_0 \\ Y_0 \end{bmatrix} = \begin{bmatrix} X'_1 \\ Y'_1 \end{bmatrix}. \tag{18}$$

By combining (18) with (17), we can obtain:

$$a_1 = -\begin{bmatrix} X'_1 \\ Y'_1 \end{bmatrix} + \begin{bmatrix} X_0 \\ Y_0 \end{bmatrix} = -\begin{bmatrix} X_1 \\ Y_1 \end{bmatrix} + \begin{bmatrix} X_0 \\ Y_0 \end{bmatrix} \Rightarrow \begin{bmatrix} X_1 \\ Y_1 \end{bmatrix} = \begin{bmatrix} X_0 \\ Y_0 \end{bmatrix} - a_1, \tag{19}$$

which keeps the same with the third equation in (16). With it, the update increment of the 2nd iteration is given by:

$$(X'_2, Y'_2) = \arg\min_{X,Y}^{SLF-NMU} \varepsilon(X_1, Y_1) \Rightarrow \Delta_2 = \begin{bmatrix} X'_2 \\ Y'_2 \end{bmatrix} - \begin{bmatrix} X_1 \\ Y_1 \end{bmatrix}, \tag{20}$$

where X'_2 and Y'_2 denote the predicted states of X and Y after the 2nd iteration with SLF-NMU. According to (14), (18) and (20), a_2 can be achieved by:

$$a_2 = \kappa a_1 - \Delta_2 = -\kappa\left(\begin{bmatrix} X_1 \\ Y_1 \end{bmatrix} - \begin{bmatrix} X_0 \\ Y_0 \end{bmatrix}\right) - \left(\begin{bmatrix} X_2' \\ Y_2' \end{bmatrix} - \begin{bmatrix} X_1 \\ Y_1 \end{bmatrix}\right)$$
$$\Rightarrow \begin{bmatrix} X_2 \\ Y_2 \end{bmatrix} = \begin{bmatrix} X_1 \\ Y_1 \end{bmatrix} - a_2 = \begin{bmatrix} X_2' \\ Y_2' \end{bmatrix} + \kappa\left(\begin{bmatrix} X_1 \\ Y_1 \end{bmatrix} - \begin{bmatrix} X_0 \\ Y_0 \end{bmatrix}\right). \tag{21}$$

In this manner, the update rules for 3rd to nth iteration are similar to (21). Therefore, we achieve the update algorithm based on (17) and (21) as follows:

$$\begin{cases} n=1: \begin{bmatrix} X_1 \\ Y_1 \end{bmatrix} = \begin{bmatrix} X_1' \\ Y_1' \end{bmatrix}, \\ n \geq 2: \begin{bmatrix} X_n \\ Y_n \end{bmatrix} = \begin{bmatrix} X_n' \\ Y_n' \end{bmatrix} + \kappa\left(\begin{bmatrix} X_{t-1} \\ Y_{t-1} \end{bmatrix} - \begin{bmatrix} X_{t-2} \\ Y_{t-2} \end{bmatrix}\right). \end{cases} \tag{22}$$

Note that the parameters update should be taken with respect to each LF with SLF-NMU. Hence, by combining (10) with (22), we design an MNMU algorithm:

$$\arg\min_{X,Y} \varepsilon(X,Y) \overset{MNMU}{\Rightarrow}$$
$$\begin{cases} n=1: \begin{cases} x_{u,m} \leftarrow x_{u,m} \dfrac{\sum_{i\in I_u} r_{u,i}^{\alpha}\, \hat{r}_{u,i}^{\beta-1}\, y_{m,i}}{\sum_{i\in I_u} \hat{r}_{u,i}^{\alpha+\beta-1}\, y_{m,i} + \alpha\lambda_X |I_u| x_{u,m}}, \\ y_{m,i} \leftarrow y_{m,i} \dfrac{\sum_{u\in U_i} r_{u,i}^{\alpha}\, \hat{r}_{u,i}^{\beta-1}\, x_{u,m}}{\sum_{u\in U_i} \hat{r}_{u,i}^{\alpha+\beta-1}\, x_{u,m} + \alpha\lambda_Y |U_i| y_{m,i}}; \end{cases} \\ n \geq 2: \begin{cases} x_{u,m} = x_{u,m} \dfrac{\sum_{i\in I_u} r_{u,i}^{\alpha}\, \hat{r}_{u,i}^{\beta-1}\, y_{m,i}}{\sum_{i\in I_u} \hat{r}_{u,i}^{\alpha+\beta-1}\, y_{m,i} + \alpha\lambda_X |I_u| x_{u,m}} + \kappa\left(x_{u,m} - x_{u,m}\right), \\ y_{m,i} = y_{m,i} \dfrac{\sum_{u\in U_i} r_{u,i}^{\alpha}\, \hat{r}_{u,i}^{\beta-1}\, x_{u,m}}{\sum_{u\in U_i} \hat{r}_{u,i}^{\alpha+\beta-1}\, x_{u,m} + \alpha\lambda_Y |U_i| y_{m,i}} + \kappa\left(y_{m,i} - y_{m,i}\right). \end{cases} \end{cases} \tag{23}$$

Note that the teams $\kappa\left(x_{u,m} - x_{u,m}\right)$ and $\kappa\left(y_{m,i} - y_{m,i}\right)$ in (23) are probably negative since there is no guarantee that each single LF is non-decreasing during the whole training process. Hence, we truncate these two terms to zeroes if they turn into be negative, resulting in the following update algorithm:

$$\arg\min_{X,Y} \varepsilon(X,Y) \overset{MNMU}{\Rightarrow}$$
$$\begin{cases} n=1: \begin{cases} x_{u,m} \leftarrow x_{u,m} \dfrac{\sum_{i\in I_u} r_{u,i}^{\alpha}\, \hat{r}_{u,i}^{\beta-1}\, y_{m,i}}{\sum_{i\in I_u} \hat{r}_{u,i}^{\alpha+\beta-1}\, y_{m,i} + \alpha\lambda_X |I_u| x_{u,m}}, \\ y_{m,i} \leftarrow y_{m,i} \dfrac{\sum_{u\in U_i} r_{u,i}^{\alpha}\, \hat{r}_{u,i}^{\beta-1}\, x_{u,m}}{\sum_{u\in U_i} \hat{r}_{u,i}^{\alpha+\beta-1}\, x_{u,m} + \alpha\lambda_Y |U_i| y_{m,i}}; \end{cases} \\ n \geq 2: \begin{cases} x_{u,m} = x_{u,m} \dfrac{\sum_{i\in I_u} r_{u,i}^{\alpha}\, \hat{r}_{u,i}^{\beta-1}\, y_{m,i}}{\sum_{i\in I_u} \hat{r}_{u,i}^{\alpha+\beta-1}\, y_{m,i} + \alpha\lambda_X |I_u| x_{u,m}} + \max\left\{0, \kappa\left(x_{u,m} - x_{u,m}\right)\right\}, \\ y_{m,i} = y_{m,i} \dfrac{\sum_{u\in U_i} r_{u,i}^{\alpha}\, \hat{r}_{u,i}^{\beta-1}\, x_{u,m}}{\sum_{u\in U_i} \hat{r}_{u,i}^{\alpha+\beta-1}\, x_{u,m} + \alpha\lambda_Y |U_i| y_{m,i}} + \max\left\{0, \kappa\left(y_{m,i} - y_{m,i}\right)\right\}. \end{cases} \end{cases} \tag{24}$$

With (24), we design the MNMU algorithm for a generalized and fast-converging non-negative latent factor (GFNLF) model. Note that when α=β=1 and κ=0, GFNLF is equal to NLF [7].

3.4 Algorithm Design and Analysis

According to the above analyses, we develop the Algorithm GFNLF. As shown in Algorithm GFNLF, some auxiliary matrices are employed. For instance, four auxiliary matrices are employed for X, i.e., XA, XB, XC and XD. Among them, XA and XB are used for caching the training increment on each instance $r_{u,i} \in R_K$, and XC and XD are used for caching the intermediate results of the (n-1)th and (n-2)th iterations. Similar settings are applied to Y.

Algorithm: GFNLF

Input: U, I, R_K, f, κ, λ_X, λ_Y	
Operation	**Cost**
initialize $X^{|U|\times f}$, $XA^{|U|\times f}$, $XB^{|U|\times f}$ non-negatively	$\Theta(|U|\times f)$
initialize $XC^{|U|\times f}$=$XD^{|U|\times f}$=X	$\Theta(|U|\times f)$
initialize $Y^{|I|\times f}$, $YA^{|I|\times f}$, $YB^{|I|\times f}$ non-negatively	$\Theta(|I|\times f)$
initialize $YC^{|I|\times f}$=$YD^{|I|\times f}$=Y	$\Theta(|I|\times f)$
initialize λ_X, λ_Y, $n=0$, *Max-training-round* $=N$	$\Theta(1)$
while not converge **and** $n \leq N$ **do**	$\times n$
reset XA=0, XB=0	$\Theta(|U|\times f)$
reset YA=0, YB=0	$\Theta(|I|\times f)$
for each $r_{u,i}$ **in** R_K	$\times|R_K|$
$\hat{r}_{u,i} = \sum_{m=1}^{f} x_{u,m} y_{m,i}$	$\Theta(f)$
for m=1 **to** f	$\times f$
$xa_{u,m} = xa_{u,m} + r_{u,i}^{\alpha} \hat{r}_{u,i}^{\beta-1} y_{m,i}$	$\Theta(1)$
$xb_{u,m} = xb_{u,m} + \hat{r}_{u,i}^{\alpha+\beta-1} y_{m,i} + \alpha\lambda_X x_{u,m}$	$\Theta(1)$
$ya_{m,i} = ya_{m,i} + r_{u,i}^{\alpha} \hat{r}_{u,i}^{\beta-1} x_{u,m}$	$\Theta(1)$
$yb_{m,i} = yb_{m,i} + \hat{r}_{u,i}^{\alpha+\beta-1} x_{u,m} + \alpha\lambda_Y y_{m,i}$	$\Theta(1)$
end for	-
end for	-
for $u \in U$	$\times|U|$
for m=1 **to** f	$\times f$
$x_{u,m}$=$x_{u,m}$($xa_{u,m}$/$xb_{u,m}$)+max{0, κ($xc_{u,m}$-$xd_{u,m}$)}	$\Theta(1)$
end for	-
end for	-
for $i \in I$	$\times|I|$
for m=1 **to** f	$\times f$
$y_{m,i}$= $y_{m,i}$($ya_{m,i}$/$yb_{m,i}$)+max{0, κ($yc_{m,i}$- $yd_{m,i}$)}	$\Theta(1)$
end for	-
end for	-
set XD=XC, XC=X	$\Theta(|U|\times f)$
set YD=YC, YC=Y	$\Theta(|I|\times f)$
n=n+1	$\Theta(1)$
end while	-
Output: X, Y	

As given in Algorithm GFNLF, we analyze the computational cost. Then, we can formulate the computational complexity as follows:

$$T_{GFNLF} = n\cdot\Theta\left(|U|+|I|\right)\times f + |R_K|\times f \approx \Theta\left(n\times|R_K|\times f\right). \tag{25}$$

Note that in (25) we adopt the condition $|R_K| \gg \max\{|U|,|I|\}$ to omit the lower-order-terms to achieve the final result reasonably. GFNLF's computational complexity is linear with $|R_K|$ since n and f are both positive constants. Meanwhile, GFNLF only uses ten matrices, i.e., X, XA, XB, XC, XD, Y, YA, YB, YC and YD, along with U, I and R_K. The storage cost only takes:

$$S_{GFNLF} = 5\times|U|\times f + 5\times|I|\times f + |R_K| + 6\times|U| + 6\times|I| = 5\times(|U|+|I|)\times f + |R_K| + 6\times(|U|+|I|). \quad (26)$$

Based on the above inferences, GFNLF is highly efficient in both computation and storage.

4 Experimental Results

4.1 General Settings

Evaluation Protocol. For industrial applications, one major motivation for processing an HiDS matrix is to predict its missing values for implementing the full relationship mapping among involved entities. Owing to its popularity and usefulness, we adopt it as the evaluation protocol in our experiments. It is commonly measured by root mean squared error (RMSE) [33]:

$$RMSE = \sqrt{\left(\sum_{r_{u,i}\in R_T} r_{u,i} - \hat{r}_{u,i}\right)^2 \Big/ |R_T|}, \quad (27)$$

where R_T denotes the test set and is disjoint with R_K, $\hat{r}_{u,i}$ denotes the generated prediction for the testing instance $r_{u,i}\in R_T$, which simulates the missing value at the uth row and ith column of a target HiDS matrix, and $|\cdot|$ calculates the cardinality of a given set, respectively. All experiments are conducted on a Tablet with a 2..4GHz Xeon E5-2680 V4 CPU and 128GB RAM, and implemented in JAVA SE 8U172.

Datasets. Two public HiDS matrices collected by real RSs currently used are included in our experiments. Hence, results on them are highly representative and useful. Their details are as follows:

a) D1: **MovieLens 20M**. It is collected by the MovieLens system [34] maintained by the GroupLens research team. It has 20,000,263 entries in (0.5, 5), from 138,493 users on 26,744 movies. Its data density is 0.54% only.

b) D2: **Dating Agency**. It is collected by an online dating website LibimSeTi [35], with 17,359,346 known entries in the range of (1, 10), from 135,359 users on 168,791 profiles. Its data density is 0.076% only

Note that we adopt the 80%-20% train-test settings and five-fold cross-validation on all datasets. The training process terminates if: a) the iteration count reaches a threshold, i.e. 1000; and b) the gap of RMSE in two consecutive iterations becomes smaller than 5×10^{-6}.

Model Settings. For obtaining objective and precise results, we adopt the following settings: 1) Setting the regularization coefficients $\lambda=\lambda_X=\lambda_Y$ and fixing the LF dimension f=20; 2) Initializing X and Y with the same randomly-generated and non-negative arrays to eliminate the impact caused by random initial guesses; 3) Repeating each set of experiments for 5 times and taking the average of the experimental outputs as their final results.

4.2 Effects of α and β

As discussed in Section 3, GFNLF's objective function relies on α-β-divergence. Hence, α and β are the key parameters for deciding GFNLF's performance. In this set of experiments, we validate the effects of α and β without momentum (κ=0) first. We fix λ=0.05 and the tested scales for α and β are both (0.2, 3.0). Figure 1 depicts the lowest RMSE of GFNLF for each value of α. Table 1 shows the optimal RMSE of GFNLF. Figure 2 shows GFNLF's time cost per iteration as α and β vary. From them, we have the following finding:

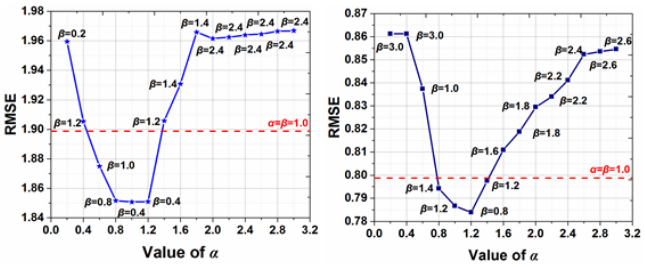

Figure 1: Lowest RMSE of GFNLF for each value of α. For each value of α, we record the lowest RMSE as β varies.

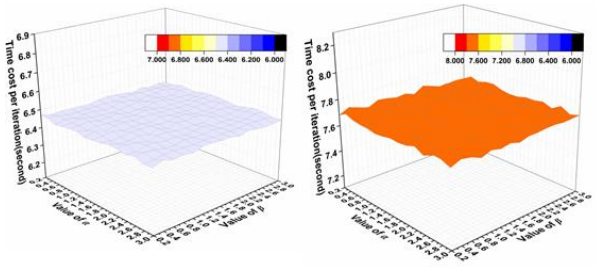

Figure 2: Time cost per iteration as α and β vary. Color indicates the magnitude.

Table 1: Optimal RMSE relying on α and β.

Dataset	O. α and β*	RMSE. O**	RMSE. α=β=1***	Gap
D1	α=1.2, β=0.8	**0.7839**	0.7876	0.47%
D2	α=1.0, β=0.4	**1.8507**	1.8982	2.50%

*Optimal value of α and β.
**GFNLF's RMSE with the optimal value of α and β
***GFNLF's RMSE with α=β=1 (GFNLF is equal to NLF).

a) As depicted in Figure 1, GFNLF's RMSE for missing data in HiDS matrices is closely connected with the value of α and β. For each value of α, the lowest RMSE is different respecting to β. Of cause, the situation is similar if we take the value of β as the horizontal axis. There is always an optimal case of α and β to enable GFNLF to achieve the optimal RMSE on each dataset. For instance, on D1, GFNLF achieves the optimal RMSE with α=1.2

and β=0.8. On D2, the optimal case is α=1.0 and β=0.8. Moreover, the RMSE increases significantly as α is relatively small or large.
b) As discussed in Section 3, we turn GFNLF into NLF by making α=β=1. As shown in Table 1, α=β=1 is not the optimal case, which corresponds to the frequently adopted Euclidean distance. For instance, on D1, GFNLF achieves the optimal RMSE at 0.7839 with α=1.2 and β=0.8. However, the lowest RMSE is 0.7876 with α=β=1. The gaps of RMSE are 0.47%. Furthermore, the gap even reaches 2.50% on D2. These results indicate that the frequently adopted Euclidean distance may not be the best choice for NLF to achieve the highest prediction accuracy for missing data.
c) As shown in Figure 2, the time cost per iteration is stable as α and β vary. From (24), the update rules of LFs depend on the same operation, which results in nearly the same constant time. However, we also observe slight fluctuations of time cost per iteration as α and β vary. The main reason is raising an arbitrary real number to the power of different α and β.

4.3 Effects of λ

As shown in (24), GFNLF's performance also relies on the regularization coefficients λ=λ_X=λ_Y. In this set of experiments, we fix α and β with the optimal case summarized in Table 1. Meanwhile, the tested scale for λ is (0.01, 0.15). Figure 3 depicts RMSE as λ varies. From them, we have the following findings:
a) As shown in Figure 3, it is necessary to choose regularizing coefficients carefully to ensure high prediction accuracy. With too small or too big values of λ, RMSE increases. For instance, as shown in Figure 3(a), on D1, the RMSE with λ=0.01, 0.03, 0.05, 0.07, 0.09, 0.11, 0.13 and 0.15 is 0.8113, 0.7898, 0.7839, 0.7901, 0.8009, 0.8136, 0.8253 and 0.8372, respectively. The gap between the highest RMSE and lowest RMSE is 6.36%. The difference in RMSE is quite significant.
b) The optimal value of λ is data-dependent. For instance, the optimal λ for D1 is 0.05. On D2, the lowest RMSE is achieved with λ=0.07. Note that the tuning process of λ is a tedious task, which can be addressed via offline tuning, or adopting empirical values. Of course, the ideal way is to pick up the optimal regularizing coefficients automatically. This clearly requires our future work.

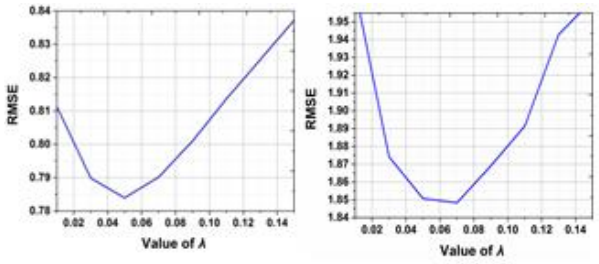

Figure 3: Lowest RMSE as λ varies.

4.4 Effects of κ

Although GFNLF has an outstanding performance on prediction accuracy according to Sections 4.2 and 4.3, it usually consumes many iterations to achieve the lowest RMSE without momentum (κ=0). Figure 4 shows the training process without momentum.

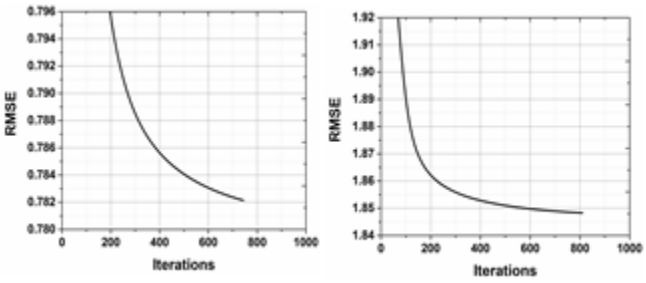

Figure 4: Training process without momentum.

As shown in Figure 4, it consumes 743 iterations on D1 and 810 iterations on D2 to achieve the lowest RMSE. As discussed in Section 1, the main reason is that SLF-NMU leads to slow convergence. It results in considerable time cost on large-scale datasets which is unacceptable for industrial applications like online RSs. Hence, we need to improve its convergence rate.

In this part, we mainly discuss the effects of momentum on convergence rate. Figure 5 depicts the training process as κ varies. Table 2 summarizes iterations and lowest RMSE as κ varies. The tested scale is (0, 1.4). From them, we have the following findings:

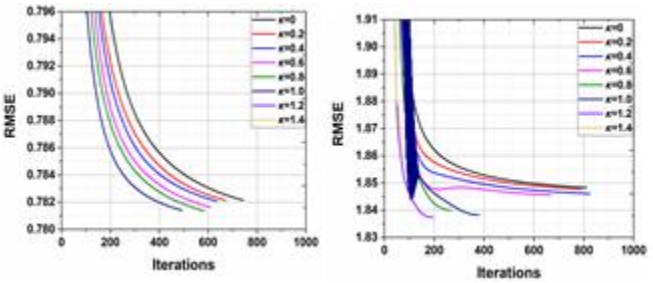

Figure 5: Training process as κ varies.

a) With appropriate κ, the generalized momentum method obviously improves its convergence rate without prediction accuracy loss, even a slight improvement. As recorded in Figure 5(a) and Table 2, GFNLF consumes 493 iterations with κ=1.0 to achieve the lowest RMSE at 0.7814 on D1. However, it consumes 743 iterations to achieve the lowest RMSE at 0.7839 with κ=0. The iterations decreases at 33.64% without prediction accuracy loss. The similar situation is more significant on D2. For instance, GFNLF consumes 195 iterations with κ=1.2, only 24.07% of 810 iterations with κ=0.
b) GFNLF's training process is sensitive to κ. It is very important to choose the appropriate κ. Small κ eliminates the momentum effects, while large κ makes a learning algorithm overshoot a local optimum. This assertion is supported by Figure 5 and Table 2. For instance, on D2, GFNLF consumes 790 iterations to achieve the RMSE at 1.8476 with κ=0.2. In contrast, with κ=1.2, GFNLF only consumes 195 iterations to achieve the RMSE at 1.8374. The

iterations decrease at 75.31% while the prediction accuracy increases at 0.55%. As κ grows too large, GFNLF suffers overshooting, which results in significant accuracy loss. For instance, on D2, GFNLF achieves the lowest RMSE at 1.8617 with κ=1.4, 1.30% higher than 1.8374 with κ=1.2. On D1, GFNLF is even impossible to converge with κ=1.2 and κ=1.4.

Table 2: Iterations and lowest RMSE as κ varies.

Dataset	κ	Iterations	Lowest RMSE
D1	0	743	0.7839
	0.2	671	0.7821
	0.4	635	0.7820
	0.6	609	0.7816
	0.8	585	**0.7813**
	1.0	**493**	0.7814
	1.2	non-convergence	******
	1.4	non-convergence	******
D2	0	810	1.8482
	0.2	790	1.8476
	0.4	824	1.8460
	0.6	669	1.8455
	0.8	268	1.8400
	1.0	380	1.8383
	1.2	195	**1.8374**
	1.4	77	1.8617

4.5 Comparison with State-of-the-art Models

In this part of experiments, we compare GFNLF with the most widely used state-of-the-art models in terms of computational efficiency and prediction accuracy. The details of compared models are summarized in Table 3.

To ensure a fair comparison, we adopt the following settings in terms of the hyperparameters in each model:

a) For M1 and M2, we select the optimal hyperparameters discussed in Section 4;

b) For M1-M3, we set LF dimension f=20. For M4 and M5, we set their hidden unit at the optimal value 500 following [36, 37];

c) The regularization coefficient is also vital for M3-M5 [7, 36, 37]. Hence, we first draw a grid search with respect to them to seek for their optimal values on each dataset.

Table 3: Details of compared models.

No.	Model	Description
M1	optimal α, β κ=0	GFNLF without momentum acceleration.
M2	optimal α, β optimal κ	GFNLF with momentum acceleration.
M3	α=β=1 κ=0	GFNLF is equal to Original NLF [7].
M4	I-AutoRec	Autoencoder-based approach to RSs [36].
M5	DCCR	Hybrid deep recommender model with autoencoder and multilayered perceptron [37].

Figure 6 depicts the lowest RMSE of compared models. Their time cost per iteration is depicted in Figure 7. Table 4 records the total time cost of compared models to achieve the lowest RMSE. From them, we have the following findings:

1) Effectiveness evaluation

a) M2's prediction accuracy is competitive to its peers. On D2, M2 is able to achieve the best performance in prediction accuracy. For instance, as shown in Figure 6(b), M2 has the lowest RMSE at 1.8374, 0.58% lower than 1.8482 by M1, 1.44% lower than 1.8752 by M3, 0.39% lower than 1.8447 by M4, and 0.26% lower than 1.8493 by M5. The situation is different on D1. M2 still outperforms M1, M3 and M4. Only M5 achieves higher prediction accuracy than M2. The gap is only 0.03%. These results indicate that M2 shows excellent performance in prediction accuracy for missing values.

b) M1 and M2 obviously outperform M3 owing to its objective function being α-β-divergence. The phenomenon indicates that adopting α-β-divergence in NLF is able to enhance representation ability for HiDS data by carefully choosing the values of α and β. Moreover, compared M1 with M2, M2 outperforms M1. For instance, M1 achieves the lowest RMSE at 0.7839 on D1 and 1.8482 on D2. In contrast, M2 achieves the lowest RMSE at 0.7814 and 1.8374, respectively. This phenomenon indicates that the prediction accuracy is slightly improved with appropriate κ.

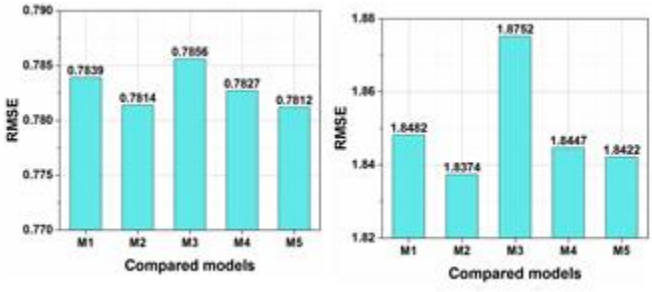

Figure 6: Lowest RMSE of compared model.

2) Efficiency evaluation

a) The time cost per iteration of M1, M2 and M3 is almost the same level. For instance, as shown in Figure 7(a), on D1, the time cost per iteration by M1, M2, and M3 is 6.4593, 6.4856, and 6.4601 seconds, respectively. By comparing M1 and M2, M2's time cost per iteration is slightly higher than that of M1. The main reason is that the latter consumes more constant time in addressing the momentum term. We can make the similar conclusion on D2. It demonstrates that MNUM can improve the convergence rate without significantly increasing the time cost per iteration.

b) M4 and M5's time cost per iteration are much higher than its peers do due to their deep neural network (DNN)-based learning strategy. They need to cost many operations about matrix manipulation and updating weights and biases by back-propagation. For instance, as shown in Figure 7(a), the time cost per iteration of M4 is 74.6492 seconds, about 11.5 times of M1-M3 do. Without GPU acceleration in our experiment, it is indeed expensive to train a DNN-based LF model on a large-scale HiDS matrix.

c) M2's computational efficiency is significantly higher than that of its peers. As recorded in Table 4, M2 obviously consumes less total time cost to achieve its lowest RMSE. For instance, on D1, M2 consumes 3197.3 seconds to achieve its lowest RMSE. This is 66.62% of 4799.3 seconds by M1, 64.44% of 4961.3 seconds by M3, 6.48% of 49343.1 seconds by M4, and 7.15% of 44712.8 seconds by M5. Note that although M2's time cost per iteration is slightly higher than that of M1 and M3 as shown in Figure 7, M2's iteration is obviously less due to adopting the generalized momentum method.

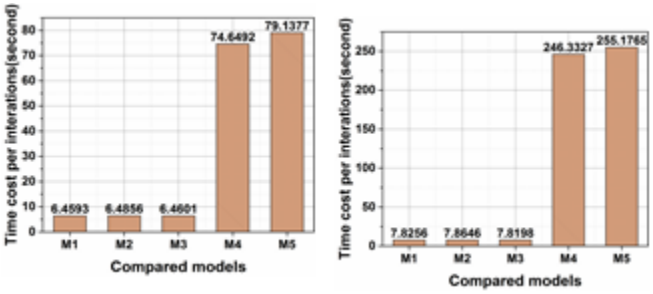

Figure 7: Time cost per iteration of compared models.

Table 4: Total time cost of compared models (second).

Dataset	M1	M2	M3	M4	M5
D1	4799.3	**3197.3**	4961.3	49343.1	44712.8
D2	6338.7	**1533.6**	6138.5	46064.2	44655.8

4.6 Summary

Based on the experimental results, we summarize that:

a) By carefully selecting appropriate α and β, GFNLF outperforms NLF with Euclidean distance on prediction accuracy;

b) The momentum-incorporated non-negative multiplicative update (MNMU) algorithm is able to significantly improve the convergence rate without accuracy loss;

c) When compared with state-of-the-art models, GFNLF can achieve obviously higher computational efficiency as well as competitive prediction accuracy for missing values in HiDS matrices.

5 Conclusion

High-dimensional and sparse matrices (HiDS) matrices with non-negativity constraints are commonly encountered in RSs. High values in an HiDS matrix usually denote strong user-item preferences. NLF can address such matrices efficiently. However, NLF mostly adopts Euclidean distance for its objective function, which is naturally a special case of α-β-divergence. Moreover, it frequently suffers slow convergence. To address this issue, this paper innovatively proposes a generalized and fast-converging non-negative latent factor (GFNLF) model. We first turn NLF into a generalized form by adopting α-β-divergence to enhance its representation ability for HiDS data. Subsequently, we design a momentum-based single latent factor-dependent, non-negative and multiplicative update (MNMU) to achieve its fast convergence. Empirical studies show that GFNLF outperforms state-of-the-art models in both computational efficiency and prediction accuracy for missing data of an HiDS matrix. Hence, it is useful for RSs to desire highly efficient, accurate and non-negative latent factor analysis.

However, the performance of GFNLF depends largely on carefully tuning hyperparameters (α, β and κ). In this work, we pre-tune their values, but the tuning process is an inefficient task. Consequently, making them self-adaptation is an essential issue. According to prior research [38], evolutionary computing-based frameworks may be useful for adaptive picking up the optimal hyperparameters. Of cause, great efforts are needed for redirecting such frameworks to GFNLF. We plan to address such issues in the future.

ACKNOWLEDGMENTS

This work is supported in part by the National Natural Science Foundation of China under grants 61772493, 61902370, 61602434 and 61702475, in part by the Natural Science Foundation of Chongqing (China) under grants cstc2019jcyj-msxmX0578 and cstc2019jcyjjqX0013, and in part by the Pioneer Hundred Talents Program of Chinese Academy of Sciences.

REFERENCES

[1] Paul Resnick and Hal R. Varian. 1997. Recommender systems. Communications of the ACM 40, 3 (1997), 56-59.

[2] Chao Wang, Qi Liu, Runze Wu, Enhong Chen, Chuanren Liu, Xunpeng Huang, and Zhenya Huang. 2018. Confidence-Aware Matrix Factorization for Recommender Systems. In Proceedings of the 32nd AAAI Conference on Artificial Intelligence (AAAI). 434-442.

[3] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. 2017. Neural collaborative filtering. In Proceedings Of the 26th International Conference on World Wide Web (WWW). 173-182.

[4] Gábor Takács, István Pilászy, Bottyán Németh, and Domonkos Tikk. 2009. Scalable Collaborative Filtering Approaches for Large Recommender Systems. Journal of Machine Learning Research 10, 623-656.

[5] Tianqiao Liu, Zhiwei Wang, Jiliang Tang, Songfan Yang, Gale Yan Huang, and Zitao Liu. 2019. Recommender Systems with Heterogeneous Side Information. In Proceedings Of the 28th International Conference on World Wide Web (WWW). 3027-3033.

[6] Cen Chen, Peilin Zhao, Longfei Li, Jun Zhou, Xiaolong Li, and Minghui Qiu. 2017. Locally connected deep learning framework for industrial-scale recommender systems. In Proceedings Of the 26th International Conference on World Wide Web (WWW). 769-770.

[7] Xin Luo, Mengchu Zhou, Shuai Li, Zhuhong You, Yunni Xia, and Qingsheng Zhu. 2016. 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 27, 3 (2016), 579-592.

[8] Trong Dinh Thac Do and Longbing Cao. 2018. Coupled poisson factorization integrated with user/item metadata for modeling popular and sparse ratings in scalable recommendation. In Proceedings of the 32nd AAAI Conference on Artificial Intelligence (AAAI). 2918-2925.

[9] Jing Lin, Weike Pan, and Zhong Ming. 2018. MF-DMPC: Matrix Factorization with Dual Multiclass Preference Context for Rating Prediction. In Proceedings of the International Conference on Web Services (ICWS). 337-349.

[10] James Chambua, Zhendong Niu, and Yifan Zhu. 2019. User preferences prediction approach based on embedded deep summaries. Expert Systems with Applications 132, 87-98.

[11] Yixin Cao, Xiang Wang, Xiangnan He, Zikun Hu, and Tat-Seng Chua. 2019. Unifying Knowledge Graph Learning and Recommendation: Towards a Better Understanding of User Preferences. In Proceedings of the 28th International Conference on World Wide Web (WWW). 151-161.

[12] Qiang Liu, Shu Wu, Liang Wang. 2017. DeepStyle: Learning user preferences for visual recommendation. In Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR). 841-844.

[13] Xin Luo, MengChu Zhou, Yunni Xia, Qingsheng Zhu, Ahmed Chiheb Ammari, and Ahmed Alabdulwahab. 2016. Generating Highly Accurate Predictions for Missing QoS Data via Aggregating Nonnegative Latent Factor Models. IEEE Transactions on Neural Networks and Learning Systems 27, 3 (2016), 524-537.
[14] Xin Dong, Lei Yu, Zhonghuo Wu, Yuxia Sun, Lingfeng Yuan, and Fangxi Zhang. 2017. A hybrid collaborative filtering model with deep structure for recommender systems. In Proceedings of the 31st AAAI Conference on Artificial Intelligence (AAAI). 1309-1315.
[15] Hao Ma, Irwin King, and Michael R. Lyu. 2009. Learning to recommend with social trust ensemble. in Proceedings of the 32nd International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR). 203-210.
[16] Tavi Nathanson, Ephrat Bitton, and Kenneth Y. Goldberg. 2007. Eigentaste 5.0: constant-time adaptability in a recommender system using item clustering. In Proceedings of the ACM conference on Recommender systems (RecSys). 149-152.
[17] Dimitrios Rafailidis and Alexandros Nanopoulos. 2015. Modeling users preference dynamics and side information in recommender systems. IEEE Transactions on Systems, Man, and Cybernetics 46, 6 (2015), 782-792.
[18] Guangneng Hu, Xinyu Dai, Fengyu Qiu, Rui Xia, Tao Li, Shujian Huang, and Jiajun Chen. 2018. Collaborative filtering with topic and social latent factors incorporating implicit feedback. ACM Transactions on Knowledge Discovery from Data 12, 2 (2018), 1-23.
[19] Yehuda Koren, Robert M. Bell, and Chris Volinsky. 2009. Matrix Factorization Techniques for Recommender Systems. IEEE Computer 42, 8 (2009), 30-37.
[20] Ye Yuan, Xin Luo, and Mingsheng Shang. 2018. Effects of preprocessing and training biases in latent factor models for recommender systems. Neurocomputing 275, 2019-2030.
[21] Zhiyong Cheng, Ying Ding, Lei Zhu, and Mohan S. Kankanhalli. 2018. Aspect-aware latent factor model: Rating prediction with ratings and reviews. In Proceedings Of the 27th International Conference on World Wide Web (WWW). 639-648.
[22] Guibing Guo, Jie Zhang, and Neil Yorke-Smith. 2015. TrustSVD: collaborative filtering with both the explicit and implicit influence of user trust and of item ratings. In Proceedings of the 29th AAAI Conference on Artificial Intelligence (AAAI). 123-129.
[23] Hao Li, Keqin Li, Jiyao An, Weihua Zheng, and Kenli Li. 2019. An efficient manifold regularized sparse non-negative matrix factorization model for large-scale recommender systems on GPUs. Information Sciences 496, 464-484.
[24] Jing Lin, Weike Pan, and Zhong Ming. 2018. MF-DMPC: Matrix Factorization with Dual Multiclass Preference Context for Rating Prediction. In Proceedings of International Conference on Web Services (ICWS). 337-349.
[25] Sheng Zhang, Weihong Wang, James Ford, and Fillia Makedon. 2006. Learning from Incomplete Ratings Using Non-negative Matrix Factorization. In Proceedings of the International Conference on Data Mining (ICDM). 549-553.
[26] Daniel D. Lee and H. Sebastian Seung. 1999. Learning the parts of objects by non-negative matrix factorization. Nature 401, 788-791.
[27] Yangyang Xu, Wotao Yin, Zaiwen Wen, and Yin Zhang. 2012. An alternating direction algorithm for matrix completion with nonnegative factors. Frontiers of Mathematics in China 7, 2 (2012), 365-384.
[28] Yifan Hu, Yehuda Koren, and Chris Volinsky. 2008. Collaborative filtering for implicit feedback datasets. In Proceedings of the 8th International Conference on Data Mining (ICDM). 263-272.
[29] Antonio Hernando, Jesús Bobadilla, and Fernando Ortega. 2016. A non negative matrix factorization for collaborative filtering recommender systems based on a Bayesian probabilistic model. Knowledge-Based Systems 97, 188-202.
[30] Andrzej Cichocki, Sergio Cruces, and Shun-ichi Amari. 2011. Generalized alpha-beta divergences and their application to robust nonnegative matrix factorization. Entropy 13, 1 (2011), 134-170.
[31] Jin Shang and Mingxuan Sun. 2018. Local Low-Rank Hawkes Processes for Temporal User-Item Interactions. In Proceedings of the International Conference on Data Mining (ICDM). 27-436.
[32] Stephen Boyd and Lieven Vandenberghe. 2009. Convex Optimization. Cambridge University Press.
[33] Jonathan L. Herlocker, Joseph A. Konstan, Loren G. Terveen, and John Riedl. 2004. Evaluating collaborative filtering recommender systems. ACM Transactions on Information Systems 22, 1 (2004), 5-53.
[34] Joseph A. Konstan, Bradley N. Miller, David Maltz, Jonathan L. Herlocker, Lee R. Gordon, and John Riedl. 1997. GroupLens: applying collaborative filtering to Usenet news. Communications of the ACM 40, 3 (1997), 77-87.
[35] Lukas Brozovsky and Vaclav Petricek. 2007. Recommender system for online dating service. eprint arXiv:cs/0703042.
[36] Suvash Sedhain, Aditya Krishna Menon, Scott Sanner, and Lexing Xie. 2015. AutoRec: Autoencoders Meet Collaborative Filtering. In Proceedings of the 24th International Conference on World Wide Web. 111-112.
[37] Qingxian Wang, Binbin Peng, Xiaoyu Shi, Tianqi Shang, and Mingsheng Shang. 2019. DCCR: Deep Collaborative Conjunctive Recommender for Rating Prediction. IEEE Access 7, 60186-60198.
[38] Pablo Ribalta Lorenzo, Jakub Nalepa, Michal Kawulok, Luciano Sánchez Ramos, and José Ranilla Pastor. 2017. Particle swarm optimization for hyper-parameter selection in deep neural networks. In Proceedings of the ACM International Conference on Genetic and Evolutionary Computation (GECCO). 481-488.
[39] Shuai Li, Alexandros Karatzoglou, and Claudio Gentile. 2016. Collaborative filtering bandits. In Proceedings of the 39th International conference on Research and Development in Information Retrieval (SIGIR). 539-548.
[40] Claudio Gentile, Shuai Li, Purushottam Kar, Alexandros Karatzoglou, Giovanni Zappella, and Evans Etrue. 2017. On Context-Dependent Clustering of Bandits. In Proceedings of the 34th International Conference on Machine Learning (ICML). 1253-1262.
[41] Antoine Bordes, Nicolas Usunier, Alberto García Durán, Jason Weston, and Oksana Yakhnenko. 2013. Translating embeddings for modeling multi-relational data. In Proceedings of advances in neural information processing systems (NIPS). 2787-2795.