An α-β-Divergence-Generalized Recommender for Highly Accurate Predictions of Missing User Preferences

Mingsheng Shang, Ye Yuan, *Student Member, IEEE*, Xin Luo, *Senior Member, IEEE*, and MengChu Zhou, *Fellow, IEEE*

***Abstract*—To quantify user–item preferences, a recommender system (RS) commonly adopts a high-dimensional and sparse (HiDS) matrix. Such a matrix can be represented by a non-negative latent factor analysis model relying on a single latent factor (LF)-dependent, non-negative, and multiplicative update algorithm. However, existing models' representative abilities are limited due to their specialized learning objective. To address this issue, this study proposes an α-β-divergence-generalized model that enjoys fast convergence. Its ideas are three-fold: 1) generalizing its learning objective with α-β-divergence to achieve highly accurate representation of HiDS data; 2) incorporating a generalized momentum method into parameter learning for fast convergence; and 3) implementing self-adaptation of controllable hyperparameters for excellent practicability. Empirical studies on six HiDS matrices from real RSs demonstrate that compared with state-of-the-art LF models, the proposed one achieves significant accuracy and efficiency gain to estimate huge missing data in an HiDS matrix.**

Manuscript received April 2, 2020; revised July 22, 2020; accepted September 20, 2020. This work was supported in part by the National Natural Science Foundation of China under Grant 62002337, Grant 61772493, and Grant 61802360; in part by the Natural Science Foundation of Chongqing (China) under Grant cstc2019jcyjjqX0013; and in part by the Pioneer Hundred Talents Program of Chinese Academy of Sciences. This article was recommended by Associate Editor S. Ventura. *(Corresponding author: Xin Luo.)*

Mingsheng Shang is with the Chongqing Engineering Research Center of Big Data Application for Smart Cities, Chongqing Key Laboratory of Big Data and Intelligent Computing, and the Chongqing Institute of Green and Intelligent Technology, Chinese Academy of Sciences, Chongqing 400714, China (e-mail: msshang@cigit.ac.cn).

Ye Yuan is with the Chongqing Engineering Research Center of Big Data Application for Smart Cities, Chongqing Key Laboratory of Big Data and Intelligent Computing, and the Chongqing Institute of Green and Intelligent Technology, Chinese Academy of Sciences, Chongqing 400714, China, and also with the Chongqing College, University of Chinese Academy of Sciences, Beijing 100049, China (e-mail: yuanye@cigit.ac.cn).

Xin Luo is with the Chongqing Engineering Research Center of Big Data Application for Smart Cities, Chongqing Key Laboratory of Big Data and Intelligent Computing, and the Chongqing Institute of Green and Intelligent Technology, Chinese Academy of Sciences, Chongqing 400714, China, and also with the Hengrui (Chongqing) Artificial Intelligence Research Center, Department of Big Data Analyses Techniques, Cloudwalk, Chongqing 401331, China (e-mail: luoxin21@cigit.ac.cn).

MengChu Zhou is with the Institute of Systems Engineering and Collaborative Laboratory for Intelligent Science and Systems, Macau University of Science and Technology, Macau, China, and also with the Department of Electrical and Computer Engineering, New Jersey Institute of Technology, Newark, NJ 07102 USA (e-mail: zhou@njit.edu).

This article has supplementary downloadable material available at https://ieeexplore.ieee.org, provided by the authors.

Digital Object Identifier 10.1109/TCYB.2020.3026425

***Index Terms*—α-β-divergence, big data, convergence analysis, high-dimensional and sparse (HiDS) data, momentum, machine learning, missing data estimation, non-negative latent factor analysis (NLFA), recommender system (RS).**

I. Introduction

PEOPLE are suffering from a serious problem of extracting desired information from enormous data scattering in the ever-exploding worldwide web. Recommender systems (RSs) that are able to find their favorites out of massive data are becoming increasingly important for various applications [1]–[10]. The fundamental data source of an RS is a user–item rating matrix [2], [3], [8], where each user's preference on each item such as movies, music, and another user, is modeled according to his/her user–item usage history. With rapidly growing user and item counts, a user touches a tiny subset of items only. Hence, a rating matrix is inevitably high-dimensional and sparse (HiDS) [8], [13], [16], [50] with numerous missing entries. For instance, the Douban matrix [26] consists of 16 830 839 known ratings by 129 490 users on 58 541 items, with 99.78% of its entries missing.

Although an HiDS matrix can be extremely sparse, it possesses rich knowledge regarding desired patterns, such as user communities [14], item clusters [46], topological and temporal neighbors [20], and user/item latent features [18]–[20]. According to prior research, a latent factor (LF) model enables highly efficient and accurate extraction of essential features and knowledge from such an HiDS matrix [3], [8], [13]–[18]. However, it mostly fails to fulfill non-negativity constraints [2], [14], [19]–[23]. Note that an HiDS matrix from an RS [2], [3], [16] is commonly defined as non-negative. Therefore, a non-negative model can better represent its data characteristics and patterns [2]–[4].

Given a complete matrix, a non-negative matrix factorization (NMF) model can extract non-negative LFs from it efficiently for various analysis tasks [62]–[64]. Li *et al.* [62] proposed a novel framework by integrating multimatrix factorization seamlessly and simultaneously, which can significantly improve the data analysis performance. Yu *et al.* [63]

predicted the comprehensive drug–drug interactions via semi-NMF. Pan *et al.* [64] embedded an NMF model to boost the performance of mobile tracking. However, these models target a full matrix and suffer from unnecessarily high computational and storage costs when handling an HiDS matrix as caused by its extremely low data density.

To implement non-negative LF analysis (NLFA) on HiDS matrices efficiently [2], [3], [8], [13]–[15], a single LF-dependent, non-negative, and multiplicative update (SLF-NMU) algorithm is proposed [8]. It greatly alleviates the computational and storage burden of performing NLFA on an HiDS matrix, due to its dependence on the known part of an HiDS matrix only [8]. With it, an NLFA model is proposed [8]. Given an HiDS matrix, an NLFA model's computational cost is linear with its known entry count only, and storage cost is linear with the sum of its user and item counts only [8], [13], [52]. In other words, as a target matrix gets sparser, it outperforms traditional matrix-dependent models in terms of computational and storage efficiency more significantly. To better illustrate the workflow of an NLFA model, we give its prediction and processing flow diagrams in the supplementary file of this article.

Although an NLFA model can represent an HiDS matrix with low cost in both computation and storage, its objective function relies on the Euclidean distance. Note that such distance is only a special case of α-β-divergence [27], [37]. As discussed in [27], α-β-divergence is far more flexible and robust than the Euclidean distance in modeling various objective functions. Note that an objective function has vital effects on an NLFA model's performance like prediction accuracy for missing values. Hence, it is expected to improve an NLFA model's performance by modeling its objective function with α-β-divergence instead of the commonly adopted Euclidean distance.

Moreover, although an SLF-NMU algorithm is specifically designed for performing NLFA on an HiDS matrix efficiently, it leads to slow convergence [2], [13], [52]. In other words, it takes many iterations to converge. Although the time cost per iteration of an SLF-NMU algorithm is low due to its specific design for an HiDS matrix, its total time cost can be considerable due to its many iterations. Hence, how to accelerate its convergence becomes a throny issue.

Aiming at addressing the above issues, this study innovatively proposes an α-β-divergence-generalized model (α-βM). Its learning objective is modeled with α-β-divergence between an HiDS matrix and its low-rank approximation from a data density-oriented perspective, thereby significantly enhancing its representation learning ability to an HiDS matrix. Its training process is carried out with a newly proposed generalized-momentum-incorporated self-adaptive non-negative-multiplicative-update (GSN) algorithm, which enables its fast convergence on an HiDS matrix. This study makes the following contributions.

1) A generalized learning objective for performing NLFA on an HiDS matrix, which is based on α-β-divergence from a data density-oriented perspective. With the highly flexible and robust objective function relying on α-β-divergence and the consideration of data density, a resulting model better represents the sparse data of an HiDS matrix with high efficiency.
2) A GSN algorithm whose convergence is evidently accelerated via incorporating the momentum effect into a training process. It adopts a generalized momentum method to correctly compute its update velocity.
3) A self-adaptation scheme of controllable hyperparameters based on the principle of particle swarm optimization (PSO). It builds each particle on controllable hyperparameters and the fitness function on decision parameters, thereby making the entire swarm discover optimal controllable hyperparameters.
4) α-βM that combines the above essential components to achieve a highly accurate and efficient solution for performing NLFA on an HiDS matrix.

For validating the performance of the proposed α-βM, we conduct experiments on six HiDS matrices emerging from real RSs. Results reveal that α-βM enjoys its: 1) high representative ability, thanks to its generalized objective; 2) fast convergence via the incorporation of a generalized momentum method into its parameter training; and 3) convenience of model training due to its self-adaptation of hyperparameter.

Section II states the preliminaries. Section III presents our methods. Section IV provides experimental results and analysis. Section V discusses the related work. Finally, Section VI concludes this article.

II. Preliminaries

Notations adopted in this article are summarized in Table I.

A. Euclidean Distance-Based Learning Objective

Note that an HiDS matrix R is commonly adopted to describe user–item interactions in an RS [3], [14]–[16], which is defined as follows.

Definition 1: $R^{|U|\times|I|}$ is an HiDS matrix if $|\Lambda| \ll |\Gamma|$.

An NLFA model builds a low-rank approximation to an HiDS matrix, which is defined next.

Definition 2: Given R and Λ, building a rank-f approximation $\tilde{R}$ on Λ to make $\tilde{R} = XY^T$ as $X^{|U|\times f}$, $Y^{|I|\times f} = 0$.

Based on Euclidean distance, a learning objective for an NLFA model is formulated as [13]–[18]:

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

where the regularization terms prevent a resultant model from overfitting. Note that in (1), we adopt the principle of an instance-frequency-weighted regularization proposed in [47] on an NLFA model to diversify the regularization effects on involved LFs.

TABLE I
SYMBOL APPOINTMENT

Parameter	Description
U, I	Large entity sets corresponding to users and items.
U_i, I_u	Subsets of U and I related to item i and user u.
$R^{\lvert U\rvert\times\lvert I\rvert}$	A target HiDS matrix.
Λ, Γ, Ψ	Known, unknown and testing sets with $\lvert\Lambda\rvert \ll \lvert\Gamma\rvert$ and $\Lambda\cap\Psi=\phi$.
$\tilde{R}^{\lvert U\rvert\times\lvert I\rvert}$	A low-rank approximation to R built on Λ.
f	LF dimension/Rank of $\tilde{R}$ and $f \ll \min\{\lvert U\rvert,\lvert I\rvert\}$.
$r_{u,i}$	A single element in R.
$\tilde{r}_{u,i}$	A single element in $\tilde{R}$.
$X^{\lvert U\rvert\times f}$, $Y^{\lvert I\rvert\times f}$	LF matrices corresponding to U and I.
$x_{u,m}$, $y_{i,m}$	Single elements in X and Y.
λ	Regularization coefficients.
α, β	Parameters of α-jβ-divergence.
θ	Decision parameter in a generalized momentum method.
κ	Momentum velocity coefficient.
a_0	Initial state of the update velocity.
$\lvert\cdot\rvert$	Cardinality of an enclosed set.
$\lVert\cdot\rVert_F$	Frobenius norm of an enclosed matrix.
$\boldsymbol{tr}(\cdot)$	Trace of an enclosed matrix.
q	Number of particles in the whole swarm.
s_j, v_j	Position and velocity of the jth particle.
$\hat{s}$, $\check{s}$	Upper and lower bounds for a particle's position.
$\hat{v}$, $\check{v}$	Upper and lower bounds for the particle's velocity.
b_j	Best position of the jth particle.
$\hat{b}$	Best position in the whole swarm.
w	Inertia weight in the range of [0, 1].
c_1, c_2	Acceleration coefficients.
r_1, r_2	Uniformly distributed random numbers in the range of [0, 1].
$F(j)$	Fitness function corresponding to the jth particle.

Note that the objective function has vital effects on an NLFA model's performance like prediction accuracy for missing values. However, the Euclidean distance is only a special case of α-β-divergence, which is able to serve as a more flexible and robust objective function [27]. Consequently, a highly interesting question arises: with the α-β-divergence-based objective function, will an NLFA model achieve significant performance gain?

B. Generalized Momentum Method

A generalized momentum method [13] works compatibly with an algorithm implicitly depending on gradients. Given decision parameter θ and objective function $\varepsilon(\theta)$, it updates θ at the nth iteration as follows:

$$\begin{cases} a_0 = 0, \\ a_n = \kappa a_{n-1} - \left(\theta'_n - \theta_{n-1}\right), \\ \theta_n = \theta_{n-1} - a_n \end{cases} \tag{2}$$

where θ'_n denotes the predicted state of θ relying on the adopted algorithm after the nth iteration, and a_n denotes the state of a_0 after the nth iteration.

Note that an SLF-NMU algorithm makes an NLFA model converge slow [2], [13], [52]. As unveiled by prior research [31], a gradient descent (GD)-based algorithm can be accelerated by a standard momentum method. However, an SLF-NMU algorithm is incompatible with a standard momentum method since it depends on gradients implicitly. Hence, it is highly desired to accelerate an SLF-NMU algorithm through a generalized momentum method.

C. Particle Swarm Optimization

As unveiled by [35] and [71]–[73], PSO can be used to implement hyperparameter adaptation for a general learning algorithm [48]. Given a swarm consisting of q particles, its evolution rule regarding the jth particle at the nth iteration is formulated as

$$\forall j \in \{1, \ldots, q\}: \begin{cases} v_{j_n} = w v_{j_{n-1}} + c_1 r_1 \left(b_{j_{n-1}} - s_{j_{n-1}}\right) \\ + \ c_2 r_2 \left(\hat{b}_{n-1} - s_{j_{n-1}}\right) \\ s_{j_n} = s_{j_{n-1}} + v_{j_n} \end{cases} \tag{3}$$

where s_{j_n}, v_{j_n}, b_{j_n}, and $\hat{b}_n$, respectively, denote the states of s_j, v_j, b_j, and $\hat{b}$ after the nth iteration.

III. PROPOSED METHODS

A. α-β-Divergence-Generalized Learning Objective

From [27], [28], [32], and [37], we see that the Euclidean distance is only a special case of α-β-divergence. Therefore, we can generalize (1) with a common form of α-β-divergence, thereby achieving the following problem formulation:

$$\begin{aligned} \varepsilon = & \sum_{r_{u,i}\in\Lambda} \left(\begin{array}{l} -\frac{1}{\alpha\beta}\left(r_{u,i}^{\alpha}\tilde{r}_{u,i}^{\beta} - \frac{\alpha}{\alpha+\beta} r_{u,i}^{\alpha+\beta} - \frac{\beta}{\alpha+\beta}\tilde{r}_{u,i}^{\alpha+\beta}\right) \\ + \ \lambda \sum_{m=1}^{f}\left(x_{u,m}^2 + y_{i,m}^2\right) \end{array} \right) \\ \text{s.t. } & \alpha > 0, \beta > 0 \quad \forall u \in U, i \in I \\ & m \in \{1, 2, \ldots, f\} : x_{u,m} \geq 0, y_{i,m} \geq 0. \end{aligned} \tag{4}$$

Note that in (4), we do not consider the border conditions of $\alpha = 0$ and/or $\beta = 0$ to prevent the divided-by-zero error. As shown in [27] and [37], we arrive at the Kullback–Leibler, Itakura–Saito, or Log divergences under such border conditions, which are rarely adopted in building an NLFA mode for an RS. Nonetheless, we should point out that the following methodology is compatible with these border cases. On the other hand, we also adopt a data-density-oriented regularization strategy in (4) for accurately diversifying the regularization effects on each LF. With it, we achieve an α-β-divergence-generalized learning objective for an NLFA model. Note that by substituting $\alpha = \beta = 1$ into (4), we obtain a Euclidean distance-based learning objective (1).

B. SLF-NMU-Based Learning Rule

An SLF-NMU algorithm is highly efficient in building an NLFA model on HiDS matrices [13]–[18]. Its learning rules for LFs from X and Y in (4) can be achieved by analyzing its Karush–Kuhn–Tucker (KKT) conditions [40]. Let $T^{\lvert U\rvert\times f}$ and $K^{\lvert I\rvert\times f}$ be the Lagrangian multipliers corresponding to the constraints $X \geq 0$ and $Y \geq 0$, respectively. Then, we achieve the Lagrangian function corresponding to (4) as

$$\begin{aligned} L &= \varepsilon + \boldsymbol{tr}\left(TX^T\right) + \boldsymbol{tr}\left(K^T Y\right) \\ &= \varepsilon + \sum_{u}\sum_{m} t_{u,m} x_{u,m} + \sum_{i}\sum_{m} \kappa_{i,m} y_{i,m}. \end{aligned} \tag{5}$$

Considering the partial derivatives of L in (5) with respect to $x_{u,m}$ and $y_{i,m}$, we have the following deduction:

$$\begin{cases} \frac{\partial L}{\partial x_{u,m}} = \frac{\partial \varepsilon}{\partial x_{u,m}} + t_{u,m} \\ = \sum_{i\in I_u}\left(-\frac{1}{\alpha} r_{u,i}^{\alpha} \tilde{r}_{u,i}^{\beta-1} y_{i,m} + \frac{1}{\alpha}\tilde{r}_{u,i}^{\alpha+\beta-1} y_{i,m} + 2\lambda x_{u,m}\right) \\ \quad + t_{u,m} \\ \frac{\partial L}{\partial y_{i,m}} = \frac{\partial \varepsilon}{\partial y_{i,m}} + \kappa_{i,m} \\ = \sum_{u\in U_i}\left(-\frac{1}{\alpha} r_{u,i}^{\alpha} \tilde{r}_{u,i}^{\beta-1} x_{u,m} + \frac{1}{\alpha}\tilde{r}_{u,i}^{\alpha+\beta-1} x_{u,m} + 2\lambda y_{i,m}\right) \\ \quad + \kappa_{i,m}. \end{cases} \tag{6}$$

By combining (5) and (6), we have the following findings.

1) The partial derivatives in (6) should be set at 0 to achieve the local optima of L.
2) The KKT condition of (5) is: $\forall u \in U, i \in I, m \in \{1, \ldots, f\} : t_{u,m} x_{u,m} = 0$ and $\kappa_{i,m} y_{i,m} = 0$.

By substituting the above findings into (5) and (6), we achieve the following inferences:

$$\begin{cases} x_{u,m}\sum_{i\in I_u}\left(-\frac{1}{\alpha} r_{u,i}^{\alpha} \tilde{r}_{u,i}^{\beta-1} y_{i,m} + \frac{1}{\alpha}\tilde{r}_{u,i}^{\alpha+\beta-1} y_{i,m} + 2\lambda x_{u,m}\right) = 0 \\ y_{i,m}\sum_{u\in U_i}\left(-\frac{1}{\alpha} r_{u,i}^{\alpha} \tilde{r}_{u,i}^{\beta-1} x_{u,m} + \frac{1}{\alpha}\tilde{r}_{u,i}^{\alpha+\beta-1} x_{u,m} + 2\lambda y_{i,m}\right) = 0. \end{cases} \tag{7}$$

Note that (7) can be rearranged into the following equation:

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

From (8), we have the iterative updates of $x_{u,m}$ and $y_{i,m}$:

$$\arg\min_{X,Y} \varepsilon(X, Y) \overset{\text{SLF-NMU}}{\Rightarrow} \begin{cases} x_{u,m} \leftarrow x_{u,m}\frac{\sum_{i\in I_u} r_{u,i}^{\alpha}\tilde{r}_{u,i}^{\beta-1} y_{i,m}}{\sum_{i\in I_u}\tilde{r}_{u,i}^{\alpha+\beta-1} y_{i,m} + \alpha\lambda|I_u|x_{u,m}} \\ y_{i,m} \leftarrow y_{i,m}\frac{\sum_{u\in U_i} r_{u,i}^{\alpha}\tilde{r}_{u,i}^{\beta-1} x_{u,m}}{\sum_{u\in U_i}\tilde{r}_{u,i}^{\alpha+\beta-1} x_{u,m} + \alpha\lambda|U_i|y_{i,m}} \end{cases} \tag{9}$$

which is the SLF-NMU-based learning rule in an α-β-divergence-generalized learning objective. Note that by setting $\alpha = \beta = 1$ into (9), we obtain the SLF-NMU-based learning rules for a Euclidean distance-based NLFA model in [8] and [13].

C. Generalized-Momentum Incorporation

From (9), we see that SLF-NMU depends on gradients implicitly. Note that a standard momentum method [31] accelerates a learning algorithm by fusing its gradient information in the current and last iterations, which depends on gradients explicitly. Hence, SLF-NMU is incompatible with a standard momentum algorithm. However, to be shown later, its update increment in each iteration can be achieved conveniently. Thus, a generalized momentum method relying on (2) is compatible with it.

Let X_{n-1} and Y_{n-1} be the states of X and Y after the $(n-1)$th iteration, and X'_n and Y'_n be the expected states obtained by (9), respectively. Thus, we have:

$$\left(X'_n, Y'_n\right) = \overset{\text{SLF-NMU}}{\arg\min_{X,Y}}\ \varepsilon(X_{n-1}, Y_{n-1}). \tag{10}$$

Then, the update increment by SLF-NMU is given as

$$\Delta_n = \theta'_n - \theta_{n-1} = \begin{bmatrix} X'_n \\ Y'_n \end{bmatrix} - \begin{bmatrix} X_{n-1} \\ Y_{n-1} \end{bmatrix}. \tag{11}$$

Based on (2) and (11), the state of the update velocity after the first training iteration is formulated as

$$a_1 = \kappa a_0 - \Delta_1 = -\begin{bmatrix} X'_1 \\ Y'_1 \end{bmatrix} + \begin{bmatrix} X_0 \\ Y_0 \end{bmatrix} \tag{12}$$

where X_0 and Y_0 denote the initial states of X and Y, that is, randomly generated non-negative matrices as discussed in [8], [13], and [16]. Then, X_1 and Y_1, that is, the states of X and Y after the first training iteration, are achieved based on (2) and (12) as

$$\begin{bmatrix} X_1 \\ Y_1 \end{bmatrix} = \begin{bmatrix} X_0 \\ Y_0 \end{bmatrix} - a_1 = \begin{bmatrix} X_0 \\ Y_0 \end{bmatrix} + \begin{bmatrix} X'_1 \\ Y'_1 \end{bmatrix} - \begin{bmatrix} X_0 \\ Y_0 \end{bmatrix} = \begin{bmatrix} X'_1 \\ Y'_1 \end{bmatrix}. \tag{13}$$

By substituting (13) into (12), we have

$$\begin{aligned} a_1 &= -\begin{bmatrix} X'_1 \\ Y'_1 \end{bmatrix} + \begin{bmatrix} X_0 \\ Y_0 \end{bmatrix} = -\begin{bmatrix} X_1 \\ Y_1 \end{bmatrix} + \begin{bmatrix} X_0 \\ Y_0 \end{bmatrix} \\ &\Rightarrow \begin{bmatrix} X_1 \\ Y_1 \end{bmatrix} = \begin{bmatrix} X_0 \\ Y_0 \end{bmatrix} - a_1 \end{aligned} \tag{14}$$

which is consistent with the third equation of (2). Analogously, Δ_2, the update increment of the second iteration, is:

$$\left(X'_2, Y'_2\right) = \overset{\text{SLF-NMU}}{\arg\min_{X,Y}}\ \varepsilon(X_1, Y_1) \Rightarrow \Delta_2 = \begin{bmatrix} X'_2 \\ Y'_2 \end{bmatrix} - \begin{bmatrix} X_1 \\ Y_1 \end{bmatrix} \tag{15}$$

where X'_2 and Y'_2 denote X and Y's expected state by (9) after the second training iteration based on X_1 and Y_1 in (14). By combining (2), (13), and (15), the update velocity a_2 is computed as:

$$\begin{aligned} a_2 &= \kappa a_1 - \Delta_2 = -\kappa\left(\begin{bmatrix} X_1 \\ Y_1 \end{bmatrix} - \begin{bmatrix} X_0 \\ Y_0 \end{bmatrix}\right) \\ &\quad - \left(\begin{bmatrix} X'_2 \\ Y'_2 \end{bmatrix} - \begin{bmatrix} X_1 \\ Y_1 \end{bmatrix}\right) \\ &\Rightarrow \begin{bmatrix} X_2 \\ Y_2 \end{bmatrix} = \begin{bmatrix} X_1 \\ Y_1 \end{bmatrix} - a_2 = \begin{bmatrix} X'_2 \\ Y'_2 \end{bmatrix} + \kappa\left(\begin{bmatrix} X_1 \\ Y_1 \end{bmatrix} - \begin{bmatrix} X_0 \\ Y_0 \end{bmatrix}\right). \end{aligned} \tag{16}$$

Note that situations in the third to nth iterations are the same as that in the second iteration. Hence, we achieve the accelerated learning scheme as

$$\begin{cases} n = 1 : \begin{bmatrix} X_1 \\ Y_1 \end{bmatrix} = \begin{bmatrix} X'_1 \\ Y'_1 \end{bmatrix} \\ n \geq 2 : \begin{bmatrix} X_n \\ Y_n \end{bmatrix} = \begin{bmatrix} X'_n \\ Y'_n \end{bmatrix} + \kappa\left(\begin{bmatrix} X_{n-1} \\ Y_{n-1} \end{bmatrix} - \begin{bmatrix} X_{n-2} \\ Y_{n-2} \end{bmatrix}\right). \end{cases} \tag{17}$$

Finally, by substituting (9) into (17), we obtain

$$\arg\min_{X,Y} \varepsilon(X, Y) \overset{\text{SLF-NMU with Generalized Momentum}}{\Rightarrow}$$

$$\begin{cases} n=1: \begin{cases} x_{u,m}^{1} \leftarrow x_{u,m}^{0} \dfrac{\sum_{i\in I_u} r_{u,i}^{\alpha}\, \tilde{r}_{u,i}^{\beta-1}\, y_{i,m}^{0}}{\sum_{i\in I_u} \tilde{r}_{u,i}^{\alpha+\beta-1}\, y_{i,m}^{0} + \alpha\lambda |I_u|\, x_{u,m}^{0}} \\ y_{i,m}^{1} \leftarrow y_{i,m}^{0} \dfrac{\sum_{u\in U_i} r_{u,i}^{\alpha}\, \tilde{r}_{u,i}^{\beta-1}\, x_{u,m}^{0}}{\sum_{u\in U_i} \tilde{r}_{u,i}^{\alpha+\beta-1}\, x_{u,m}^{0} + \alpha\lambda |U_i|\, y_{i,m}^{0}}; \end{cases} \\ n\geq 2: \begin{cases} x_{u,m}^{n} = x_{u,m}^{n-1} \dfrac{\sum_{i\in I_u} r_{u,i}^{\alpha}\, \tilde{r}_{u,i}^{\beta-1}\, y_{i,m}^{n-1}}{\sum_{i\in I_u} \tilde{r}_{u,i}^{\alpha+\beta-1}\, y_{i,m}^{n-1} + \alpha\lambda |I_u|\, x_{u,m}^{n-1}} \\ \quad + \max\left\{0, \kappa\left(x_{u,m}^{n-1} - x_{u,m}^{n-2}\right)\right\} \\ y_{i,m}^{n} = y_{i,m}^{n-1} \dfrac{\sum_{u\in U_i} r_{u,i}^{\alpha}\, \tilde{r}_{u,i}^{\beta-1}\, x_{u,m}^{n-1}}{\sum_{u\in U_i} \tilde{r}_{u,i}^{\alpha+\beta-1}\, x_{u,m}^{n-1} + \alpha\lambda |U_i|\, y_{i,m}^{n-1}} \\ \quad + \max\left\{0, \kappa\left(y_{i,m}^{n-1} - y_{i,m}^{n-2}\right)\right\}. \end{cases} \end{cases} \tag{18}$$

Note that (18) conducts the non-negative projection to the teams $\kappa(x_{u,m}^{n-1} - x_{u,m}^{n-2})$ and $\kappa(y_{i,m}^{n-1} - y_{i,m}^{n-2})$ to prevent LFs in X and Y from becoming negative.

D. Self-Adaptation of κ and λ

The controllable hyperparameters κ and λ in (18) decides the performance of a resultant model. Their tuning requires a two-fold grid-search [74] that costs much time. Hence, we propose to make them self-adaptation with PSO [53], [71]–[73]. We build q particles in a two-dimension space for a swarm, where the jth particle denotes the controllable hyperparameters for the same group of LFs:

$$\begin{cases} s_j = \left[\lambda_j, \kappa_j\right] \\ v_j = \left[v_{j\lambda}, v_{j\kappa}\right]. \end{cases} \tag{19}$$

Then, by substituting (19) into (3), we have the evolution process of $\kappa_j$and λ_j at the nth iteration:

$$\forall j \in \{1, \ldots, q\}: \begin{cases} \begin{bmatrix} v_{j\lambda} \\ v_{j\kappa} \end{bmatrix}_{n} = w \begin{bmatrix} v_{j\lambda} \\ v_{j\kappa} \end{bmatrix}_{n-1} + c_1 r_1 \left(\begin{bmatrix} b_j \\ \end{bmatrix}_{n-1} - \begin{bmatrix} \lambda_j \\ \kappa_j \end{bmatrix}_{n-1} \right) \\ \quad + c_2 r_2 \left(\hat{b}_{n-1} - \begin{bmatrix} \lambda_j \\ \kappa_j \end{bmatrix}_{n-1} \right) \\ \begin{bmatrix} \lambda_j \\ \kappa_j \end{bmatrix}_{n} = \begin{bmatrix} \lambda_j \\ \kappa_j \end{bmatrix}_{n-1} + \begin{bmatrix} v_{j\lambda} \\ v_{j\kappa} \end{bmatrix}_{n}. \end{cases} \tag{20}$$

Note that in (20), PSO makes its particles evolve based on b_j and $\hat{b}$. Therefore, we need to update them as follows:

$$\forall j \in \{1, \ldots, q\}: \begin{cases} b_j = \begin{cases} b_{j\,n-1}, F\left(b_{j\,n-1}\right) \leq F\left(s_{j_n}\right) \\ s_{j_n}, F\left(b_{j\,n-1}\right) > F\left(s_{j_n}\right); \end{cases} \\ \hat{b}_n = \begin{cases} \hat{b}_{n-1}, F\left(\hat{b}_{n-1}\right) \leq F\left(s_{j_n}\right) \\ s_{j_n}, F\left(\hat{b}_{n-1}\right) > F\left(s_{j_n}\right). \end{cases} \end{cases} \tag{21}$$

For making the entire swarm better fit the known set Λ, we adopt the following fitness function for the jth particle:

$$F(j) = \sqrt{\left(\sum_{r_{u,i}\in\Psi} \left(r_{u,i} - \hat{r}_{u,i}\right)^2\right) / |\Psi|}. \tag{22}$$

To be shown later in Section IV-A, α-βM adopts $F(j)$ according to the performance evaluation metrics.

As unveiled by [48], the position and velocity of each particle must be constrained in a certain range

$$\begin{cases} s_{j_n} = \left[\lambda_{j_n}, \kappa_{j_n}\right] = \begin{cases} \lambda_{j_n} = \min\left(\hat{\lambda}, \max\left(\check{\lambda}, \lambda_{j_n}\right)\right) \\ \kappa_{j_n} = \min\left(\hat{\kappa}, \max\left(\check{\kappa}, \kappa_{j_n}\right)\right); \end{cases} \\ v_{j\,n} = \begin{bmatrix} v_{j\lambda}, v_{j\kappa} \\ n \quad n \end{bmatrix} = \begin{cases} v_{j\lambda\,n} = \min\left(\hat{v}_{\lambda}, \max\left(\check{v}_{\lambda}, v_{j\lambda\,n}\right)\right) \\ v_{j\kappa\,n} = \min\left(\hat{v}_{\kappa}, \max\left(\check{v}_{\kappa}, v_{j\kappa\,n}\right)\right) \end{cases} \end{cases} \tag{23}$$

where $\hat{s} = [\hat{\lambda}, \hat{\kappa}]$,$\check{s} = [\check{\lambda}, \check{\kappa}]$, $\hat{v} = [\hat{v}_{\lambda}, \hat{v}_{\kappa}]$, and $\check{v} = [\check{v}_{\lambda}, \check{v}_{\kappa}]$. We commonly have $\hat{v} = 0.2 \times (\hat{s} - \check{s})$ and $\check{v} = -\hat{v}$ according to [48].

Note that $\forall j \in \{1, \ldots, q\}$, s_j is linked with the same group of LF matrices, that is, X and Y. Thus, each iteration actually consists of q subiterations. In the jth subiteration of the nth evolving iteration, X and Y are trained as

$$\arg\min_{X,Y} \varepsilon(X, Y) \overset{GSN}{\Rightarrow}$$

$$\begin{cases} n=1: \begin{cases} x_{(1)u,m}^{1} \leftarrow x_{(1)u,m}^{0} \dfrac{\sum_{i\in I_u} r_{u,i}^{\alpha}\, \tilde{r}_{(1)u,i}^{\beta-1}\, y_{(1)i,m}^{0}}{\sum_{i\in I_u} \tilde{r}_{(1)u,i}^{\alpha+\beta-1}\, y_{(1)i,m}^{0} + \alpha\, \lambda_1 |I_u|\, x_{(1)u,m}^{0}} \\ y_{(1)i,m}^{1} \leftarrow y_{(1)i,m}^{0} \dfrac{\sum_{u\in U_i} r_{u,i}^{\alpha}\, \tilde{r}_{(1)u,i}^{\beta-1}\, x_{(1)u,m}^{0}}{\sum_{u\in U_i} \tilde{r}_{(1)u,i}^{\alpha+\beta-1}\, x_{(1)u,m}^{0} + \alpha\, \lambda_1 |U_i|\, y_{(1)i,m}^{0}} \end{cases} \\ n\geq 2: \begin{cases} x_{(j)u,m}^{n} = x_{(j)u,m}^{n-1} \dfrac{\sum_{i\in I_u} r_{u,i}^{\alpha}\, \tilde{r}_{(j)u,i}^{\beta-1}\, y_{(j)i,m}^{n-1}}{\sum_{i\in I_u} \tilde{r}_{(j)u,i}^{\alpha+\beta-1}\, y_{(j)i,m}^{n-1} + \alpha\lambda_{j_{n-1}} |I_u|\, x_{(j)u,m}^{n-1}} \\ \quad + \max\left\{0, \kappa_{j_{n-1}}\left(x_{(j)u,m}^{n-1} - x_{(j)u,m}^{n-2}\right)\right\}, \\ y_{(j)i,m}^{n} = y_{(j)i,m}^{n-1} \dfrac{\sum_{u\in U_i} r_{u,i}^{\alpha}\, \tilde{r}_{(j)u,i}^{\beta-1}\, x_{(j)u,m}^{n-1}}{\sum_{u\in U_i} \tilde{r}_{(j)u,i}^{\alpha+\beta-1}\, x_{(j)u,m}^{n-1} + \alpha\lambda_{j_{n-1}} |U_i|\, y_{(j)i,m}^{n-1}} \\ \quad + \max\left\{0, \kappa_{j_{n-1}}\left(y_{(j)i,m}^{n-1} - y_{(j)i,m}^{n-2}\right)\right\} \end{cases} \end{cases} \tag{24}$$

Algorithm 1 α-βM

Input: $\boldsymbol{U}, \boldsymbol{I}, \boldsymbol{\Lambda}, \boldsymbol{f}, \boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{q}, \boldsymbol{w}, \boldsymbol{c}_1, \boldsymbol{c}_2$

Operation	Cost
initialize $X^{\lvert U\rvert\times f}, X_A^{\lvert U\rvert\times f}, X_B^{\lvert U\rvert\times f}$ $non-negatively$	$\Theta(\lvert U\rvert \times f)$
initialize $Y^{\lvert I\rvert\times f}, Y_A^{\lvert I\rvert\times f}, Y_B^{\lvert I\rvert\times f}$ $non-negatively$	$\Theta(\lvert I\rvert \times f)$
initialize $X_C^{\lvert U\rvert\times f} = X_D^{\lvert U\rvert\times f} = X$, $Y_C^{\lvert I\rvert\times f} = Y_D^{\lvert I\rvert\times f} = Y$	$\Theta((\lvert U\rvert + \lvert I\rvert) \times f)$
initialize $Y_{r1}, Y_{r2}, n = 0$, $Max-training-round = N$	$\Theta(1)$
Initialize $\hat{b}^{1\times 2}, \hat{s}^{1\times 2}, \check{s}^{1\times 2}, \hat{v}^{1\times 2}, \check{v}^{1\times 2}$	$\Theta(2)$
initialize $S^{q\times 2}, V^{q\times 2}, b^{q\times 2}$	$\Theta(q \times 2)$
initialize F^q	$\Theta(q)$
while not *converge* **and** $n \le N$ **do**	$\times n$
for $j = 1$ **to** Y_q	$\times q$
reset $X_A = 0, X_B = 0, Y_A = 0, Y_B = 0$	$\Theta((\lvert U\rvert + \lvert I\rvert) \times f)$
$(Y_X, Y_Y) =$ **UPDATE**$(U, I, \Lambda, f, \alpha, \beta, X, Y, X_A, Y_A, X_B, Y_B, X_C, Y_C, X_D, Y_D, S)$	T_{UPDATE}
set $X_D = X_C, X_C = X$	$\Theta(\lvert U\rvert \times f)$
set $Y_D = Y_C, Y_C = Y$	$\Theta(\lvert I\rvert \times f)$
$F_j = fitness(s_{j,.})$	$\Theta(1)$
end for	$-$
$(Y_S, Y_V, Y_b, \hat{b}) =$ **SELF-ADAPTATION** $(Y_q, \hat{b}, Y_F, Y_b, Y_S, Y_V, \hat{s}, \check{s}, \hat{v}, \check{v})$	$T_{\text{SELF-ADAPTATION}}$
$n = n + 1$	$\Theta(1)$
end while	$-$

Output: $\boldsymbol{X}, \boldsymbol{Y}$

Procedure 1 UPDATE

Input: $\boldsymbol{U}, \boldsymbol{I}, \boldsymbol{\Lambda}, \boldsymbol{f}, \boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{X_A}, \boldsymbol{Y_A}, \boldsymbol{X_B}, \boldsymbol{Y_B}, \boldsymbol{X_C}, \boldsymbol{Y_C}, \boldsymbol{X_D}, \boldsymbol{Y_D}, \boldsymbol{S}$

Operation	Cost
for each $r_{u,i} \in \Lambda$	$\times \lvert\Lambda\rvert$
$\tilde{r}_{u,i} = \sum_{m=1}^{f} x_{u,m} y_{i,m}$	$\Theta(f)$
for $m = 1$ **to** f	$\times f$
$xa_{u,m} = xa_{u,m} + r_{u,i}^{\alpha} \tilde{r}_{u,i}^{\beta-1} y_{i,m}$	$\Theta(1)$
$xb_{u,m} = xb_{u,m} + \tilde{r}_{u,i}^{\alpha+\beta-1} y_{i,m} + \alpha s_{j,1} x_{u,m}$	$\Theta(1)$
$ya_{i,m} = ya_{i,m} + r_{u,i}^{\alpha} \tilde{r}_{u,i}^{\beta-1} x_{u,m}$	$\Theta(1)$
$yb_{i,m} = yb_{i,m} + \tilde{r}_{u,i}^{\alpha+\beta-1} x_{u,m} + \alpha s_{j,1} y_{i,m}$	$\Theta(1)$
end for	$-$
end for	$-$
for $u \in U$	$\times \lvert U\rvert$
for $m = 1$ **to** f	$\times f$
$x_{u,m} = x_{u,m}(xa_{u,m}/xb_{u,m}) + \max\{0, s_{j,2}(xc_{u,m} - xd_{u,m})\}$	$\Theta(1)$
end for	$-$
end for	$-$
for $i \in I$	$\times \lvert I\rvert$
for $m = 1$ **to** f	$\times f$
$y_{i,m} = y_{i,m}(ya_{i,m}/yb_{i,m}) + \max\{0, s_{j,2}(yc_{i,m} - yd_{i,m})\}$	$\Theta(1)$
end for	$-$
end for	$-$

Output: update $\boldsymbol{X}, \boldsymbol{Y}$

where the subscript (j) on X and Y denotes that their current states are linked with the jth particle, that is, κ_j and λ_j. By combining (20)–(24), we obtain a GSN algorithm to build α-βM.

Note that it is ideal to make α and β's self-adaptation to further improve the practicability of α-βM. Nonetheless, α and β play the role of modeling hyperparameters in α-βM, which is very different from the role of κ and λ. The self-adaptation rules of κ and λ discussed in this section are not applicable to them based on our study. This is because α and β affect the solution space of α-βM. Once they vary, the learning objective of α-βM changes, and its stationary points also move. Thus, the performance of α-βM is impaired. Yet their self-adaptation is desired. We plan to make further efforts to do so.

E. Algorithm Design and Analysis

According to the above analyses, we develop Algorithm (Algorithm 1) α-βM, which uses several auxiliary matrices.

1) For X, X_A and X_B are adopted to cache the training increment on each instance $r_{u,i} \in \Lambda$, and X_C and X_D to cache the intermediate results of the $(n-1)$th and $(n-2)$th iterations. Similar design is also applied to Y.
2) To implement self-adaptation of λ and κ, S, V, P_B, and F are adopted to cache the updated position, updated velocity, best position, andfitness function value of particles, respectively.

Note that Algorithm α-βM relies on two procedures, that is, UPDATE (Procedure 1) for LF matrices X and Y, and SELF-ADAPTATION (Procedure 2) for controllable hyperparameters λ and κ. According to them, we see that their costs are $T_{\text{UPDATE}} \approx \Theta((\lvert U\rvert + \lvert I\rvert) \times f + \lvert\Lambda\rvert \times f)$ and $T_{\text{SELF-ADAPTATION}} \approx q$, respectively. So Algorithm α-βM's cost is

$$\begin{aligned} T &\approx n \cdot \Theta(q \times ((\lvert U\rvert + \lvert I\rvert) \times f + \lvert\Lambda\rvert \times f) + q) \\ &\approx \Theta(n \times q \times \lvert\Lambda\rvert \times f). \end{aligned} \tag{25}$$

Note that in (25), we adopt the condition of $\lvert\Lambda \gg \max\{\lvert U\rvert, \lvert I\rvert\}$ to reasonably omit the lower order terms. Thus, Algorithm α-β's computational complexity is linear with $\lvert\Lambda\rvert$ since n, q, and f are all positive constants.

Meanwhile, we see that Algorithm α-βM uses auxiliary matrices, that is, X, Y, X_A, Y_A, X_B, Y_B, X_C, Y_C, X_D, Y_D, S, V, P_B, and F. Therefore, we have its storage cost as

$$\begin{aligned} S &= 5 \times (\lvert U\rvert + \lvert I\rvert) \times f + 7 \times q + \lvert\Lambda\rvert \\ &\approx 5 \times (\lvert U\rvert + \lvert I\rvert) \times f + \lvert\Lambda\rvert. \end{aligned} \tag{26}$$

Based on (25) and (26), Algorithm α-βM has high efficiency in both storage and computation.

Based on the above inferences, we have depicted the flow diagram of α-βM in Fig. S2(b) in the supplementary file of this article. Moreover, we have also depicted the flow diagram of a Euclidean-based NLFA model in Fig. S2(a). From them, we clearly see the differences between them as follows.

1) α-βM's learning objective is based on α-β-divergence, which is more robust and flexible [27] than a Euclidean distance-based one.
2) α-βM adopts a GSN algorithm to achieve fast convergence, while a Euclidean distance-based one adopts SLF-NMU, thus having slow convergence.
3) α-βM's controllable hyperparameters are self-adaptation, while a Euclidean distance-based NLFA model's hyperparameters require manual tuning.

Next, we validate α-βM's performance by using industrial datasets.

Procedure 2 SELF-ADAPTATION

Input: $\boldsymbol{q}, \hat{\boldsymbol{b}}, \boldsymbol{F}, \boldsymbol{b}, \boldsymbol{S}, \boldsymbol{V}, \hat{s}, \check{s}, \hat{v}, \check{v}$

Operation & **Cost**	
for $j = 1$**to**q	$\times q$
if $F_j < \mathit{fitness}(b_{j,\cdot})$	$\Theta(1)$
$\mathit{fitness}(b_{j,\cdot}) = F_j$	$\Theta(1)$
for $k = 1$ **to** 2	$\Theta(2)$
$b_{j,k} = s_{j,k}$	$\Theta(1)$
end for	$-$
end if	$-$
if $F_j < \mathit{fitness}(\hat{b})$	$\Theta(1)$
$\mathit{fitness}(\hat{b}) = F_j$	$\Theta(1)$
for $k = 1$ **to** 2	$\Theta(2)$
$\hat{b}_k = s_{j,k}$	$\Theta(1)$
end for	$-$
end if	$-$
end for	$-$
for $i = 1$**to** q	$\times q$
for $k = 1$ **to** 2	$\times 2$
$v_{j,k} = wv_{j,k} + c_1 r_1 (b_{j,k} - s_{j,k}) + c_1 r_1 (\hat{b}_k - s_{j,k})$	$\Theta(1)$
if $v_{j,k} > \hat{v}_k$	$\Theta(1)$
$v_{j,k} = \hat{v}_k$	$\Theta(1)$
else if $v_{j,k} < \check{v}_k$	$\Theta(1)$
$v_{j,k} = \check{v}_k$	$\Theta(1)$
end if	$-$
$s_{j,k} = s_{j,k} + v_{j,k}$	$\Theta(1)$
if $s_{j,k} > \hat{s}_k$	$\Theta(1)$
$s_{j,k} = \hat{s}_k$	$\Theta(1)$
else if $s_{j,k} < \check{s}_k$	$\Theta(1)$
$s_{j,k} = \check{s}_k$	$\Theta(1)$
end if	$-$
end for	$-$
end for	$-$

Output: update $\boldsymbol{S}, \boldsymbol{V}, \boldsymbol{b}, \hat{\boldsymbol{b}}$

IV. Experimental Results and Analysis

A. General Settings

Evaluation Protocol: For industrial applications [3], [4], [8], [13], [16], [57], one major motivation to analyze an HiDS matrix is to recover the full connections among involved entities. We adopt it as the evaluation protocol for the experiments due to its popularity. It is commonly measured by root mean-squared error (RMSE) [8], [13], [14], that is

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

where $|\cdot|$ calculates the cardinality of an enclosed set. All experiments are conducted on a Tablet with a 2.4-GHz Intel Xeon E5-2680 V4 CPU and 128-GB RAM.

Datasets: Six HiDS matrices adopted in our experiments are collected by real RSs.

1) *D1 (MovieLens 20M* [41]*):* It includes 20 000 263 known ratings in the range of [0.5, 5], from 138 493 users on 26 744 movies. Its known data density is 0.54%.
2) *D2 (Douban* [26]*):* It includes 16 830 839 known ratings by 129 490 users on 58 541 items. Its rating range is in [1, 5] and data density is 0.22%.
3) *D3 (Dating Agency* [42]*):* It contains 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.
4) *D4 (Flixter* [43]*):* It contains 8 196 077 ratings in [0.5, 5] from 147 612 users on 48 794 movies. Its data density is 0.11%.
5) *D5 (Yahoo-R2-9* [60]*):* It is collected by Yahoo, with 8 846 899 observed ratings from 1 823 178 users on 136 736 songs. Its rating scale is [1, 5]. Its density is 0.0035% only.
6) *D6 (Extended Epinion* [61]*):* It contains 13 668 320 known ratings from 120 492 users on 775 760 articles, which are collected by Trustlet. Its rating scale is [1, 5]. Its density is 0.015%.

Note that we divide 80% of the dataset into a training set and the rest into a testing set. Moreover, the five-fold cross-validation is adopted. Considering the hyperparameters of each model, we tune them to achieve the optimal outputs on one-fold of experiments and then adopt the same values on the remaining four folds. The above process is repeated ten times to achieve the final averaged results. Considering each model in a single run, the training process breaks if: 1) the iteration count reaches 1000 or 2) the RMSE difference in two consecutive iterations is within 5×10^{-6}.

B. Parameter Sensitivity

1) Effects of κ and λ's Adaptation: In this part, we aim at validating the performance of λ and κ's self-adaptation in α-βM. We manually tune them to achieve the optimal outputs. This process is depicted in Figs. S3 and S4 in the supplementary file of this article. From them, we clearly see that without self-adaptation, λ and κ should be tuned with care to enable the best performance of α-βM.

For instance, as depicted in Fig. S3(a), with $\kappa = 1$, the RMSE with $\lambda = 0.02$, 0.04, 0.06, 0.08, 0.10, and 0.12 is 0.7894, 0.7826, 0.7869, 0.7965, 0.8091, and 0.8213, respectively. The lowest RMSE is 0.7826 with $\lambda = 0.04$, and the highest is 0.8213 with $\lambda = 0.12$. The accuracy gain with the optimal $\lambda = 0.04$ and $\kappa = 1$ comes to 4.71%, which is significant. Similar outcomes are also found on $D2$–6, as shown in Figs. S3(b)–(f).

Moreover, according to Fig. S4, on $D1$–5, α-βM's convergence rate is generally improved by a generalized momentum method without prediction accuracy loss. We further see that a small κ results in no obvious momentum effects, while large κ makes a learning algorithm overshoot a local optimum. For instance, as shown in Fig. S4(c), α-βM consumes 798 iterations to achieve RMSE at 1.8544 with $\kappa = 0.2$. In contrast, with $\kappa = 1.2$, α-βM only consumes 275 iterations to achieve the RMSE at 1.8458. The number of iterations decreases at 65.54% and the prediction error also decreases at 0.46%. On the other hand, as κ grows too large, α-βM suffers overshooting. For instance, α-βM achieves the RMSE at 1.8943 with $\kappa = 1.4$, 2.56% higher than 1.8458 with $\kappa = 1.2$ on $D3$, and exhibits an irregular training curve, as depicted in Fig. S4(c). The main reason for this phenomenon is that the momentum term makes the learning algorithm no longer guarantee monotone descent of its objective function. Furthermore, from a numerical point of view, a momentum method is not an interpolation but an extrapolation method, which affects the

TABLE II
MODEL PERFORMANCE WITH SELF-ADAPTED AND MANUALLY TUNED λ AND κ

Dataset	Best λ and κ	RMSE	Iterations	*Per	**Total
D1	0.04, 1.0	**0.7826**	461	**6.485**	2989.6
	Self-adaptation	0.7839	**40**	65.23	**2609.2**
D2	0.06, 1.0	0.7103	411	**5.779**	2375.2
	Self-adaptation	**0.7100**	**32**	57.93	**1853.7**
D3	0.10, 1.2	1.8458	275	**7.861**	2161.8
	Self-adaptation	**1.8432**	**16**	79.22	**1267.5**
D4	0.04, 0.8	**0.8914**	644	**2.748**	1769.7
	Self-adaptation	0.8918	**27**	27.98	**755.5**
D5	0.10, 1.2	1.1364	414	**8.099**	3352.9
	Self-adaptation	**1.1356**	**34**	80.12	**2724.1**
D6	0.06, 0.0	**0.5309**	44	**5.818**	255.9
	Self-adaptation	0.5316	**3**	58.44	**175.3**

*Time cost per iteration (seconds). **Total time cost (seconds).

stability of a learning algorithm. Hence, α-βM cannot converge well with too large momentum effects. Similar results are found on other datasets.

Meanwhile, we have encountered a special case on $D6$. The prediction accuracy of α-βM keeps decreasing with the incorporation of momentum effects. Its lowest RMSE is achieved with $\lambda = 0.06$ and $\kappa = 0$. This phenomenon indicates that $D6$ is extremely sensitive with the learning algorithm. Any attempts to accelerate the training process via a momentum method will result in accuracy loss, as shown in Figs. S3 and S4 in the supplementary file.

Moreover, Table II records the model performance with self-adapted and manually tuned hyperparameters. As shown in Table II, optimal λ and κ vary on $D1$–6, indicating their data dependence. Thus, their tuning process needs to be repeated and is thus time-consuming.

The above results indicate the necessity of making κ and λ's self-adaptation. The training process of α-βM with adaptive κ and λ is depicted in Fig. S5 in the supplementary file. Note that we make $S = 10$, $w = 0.729$, and $c_1 = c_2 = 2$ in (20) for the self-adaptation of κ and λ according to [48].

1) *Self-Adaptation of κ and λ Affects no Prediction Accuracy of α-βM:* According to Table II and Fig. S5, α-βM with self-adaptation of κ and λ achieves lower RMSE than the one with manually tuned optimal κ and λ does on $D2$, $D3$, and $D5$. For instance, on $D3$, α-βM's RMSE is 1.8432 with self-adaptation of κ and λ, and 1.8458 with manually tuned optimal κ and λ. Thus, κ and λ's self-adaptation yields an accuracy gain at 0.14%. Considering $D1$, $D4$, and $D6$, self-adaptation of κ and λ leads to slight accuracy loss. For instance, on $D1$, α-βM's RMSE is 0.7839 with self-adaptation of κ and λ, and 0.7826 with manually tuned optimal κ and λ. The accuracy loss caused by self-adaptation of κ and λ is 0.16% only.
2) *Self-Adaptation of κ and λ Greatly Reduces the Time Cost to Train α-βM:* First of all, it should be pointed out that by making κ and λ's self-adaptation via the principle of PSO as in (20), α-βM's time cost per iteration increases significantly. The main reason is that the swarm is made up by q particles, making α-βM's each iteration actually consist of q subiterations as shown in Algorithm α-βM. As recorded in Table II, α-βM's time cost per iteration with self-adaptation of κ and λ is about ten times (as $q = 10$ in our experiment) that of a model with manually tuned κ and λ. Nonetheless, from Table II, we see that due to the self-adaptation of κ and λ, the converging iteration count of α-βM decreases drastically. Hence, compared with the model with manually tuned κ and λ, its total time cost decreases significantly. For instance, as recorded in Table II, with/without self-adaptation of κ and λ, α-βM consumes 1267.5 and 2161.8 s, respectively. Thus, its time cost reduces 58.63% due to the self-adaptation of κ and λ. The similar conclusions can be made on the other datasets. Moreover, considering the model with manually tuned κ and λ, Table II only records its time cost with the pretuned κ and λ, but does not record the time cost to tune them. Since such tuning requires a two-fold grid search, the cost is much more than the training cost. From this point of view, α-βM's time efficiency is greatly enhanced with self-adaptation of κ and λ.
3) *α-βM Implements the Self-Adaptation of κ and λ Efficiently:* To better illustrate the self-adaptation of λ and κ, we have depicted their adaptation curves during the training process of α-βM on $D1$ in Fig. S6 in the supplementary file of this article. Please note that κ and λ are randomly initialized in the range of [0, 1.4] and [0.02, 0.12] for each particle, respectively. According to Fig. S6(a), values of κ maintained by ten particles of α-βM initially fluctuate between 0 and 0.14, and then tend to converge at a stable value uniformly (about 0.815) for ensuring the prediction accuracy of α-βM. Considering λ, the phenomenon is highly similar: values maintained by ten particles of α-βM fluctuate initially and then converge at the stable value of 0.31 during the training process. Note that similar results are also encountered on $D2$–5.
4) To summarize, the self-adaptation of κ and λ is highly necessary. It greatly reduces the time cost with no accuracy loss (or even slight accuracy gain).

2) Effects of α and β: As discussed in Section III, α and β's self-adaptation rule remains unveiled at the current stage, making their manual tuning still necessary. Hence, in this part of experiments, we validate their effects. Note that as analyzed before, the α-β-divergence degenerates to the Euclidean distance when $\alpha = \beta = 1$. Table III records α-βM's RMSE with the best α and β versus $\alpha = \beta = 1$. Figs. S7 and S8 in the supplementary file depict the RMSE and time cost per iteration as α and β change. From them, we have the following important findings:

1) *α-βM's Prediction Accuracy for Missing Data of an HiDS Matrix Is Closely Connected With α and β; the Commonly Adopted Euclidean Distance Is Not the Best Choice on Any Experimental Dataset:* For instance, as recorded in Table III and Fig. S7(d), on $D4$, α-βM's RMSE is 0.8782 with the best $\alpha = 1.0$ and $\beta = 0.8$, while 0.8918 with $\alpha = \beta = 1$. Thus, the optimized α-β-divergence makes α-βM achieve an accuracy gain at 1.53% over the case with the Euclidean distance. Similar

TABLE III
α-βM'S RMSE WITH THE BEST α AND β VERSUS $\alpha = \beta = 1$ (I.E., THE EUCLIDEAN DISTANCE)

Dataset	Best α and β	RMSE	RMSE as α=β=1	Gap
D1	α=1.2, β=0.8	**0.7811**	0.7839	0.36%
D2	α=1.0, β=0.8	**0.7058**	0.7100	0.59%
D3	α=1.0, β=0.4	**1.8187**	1.8432	1.33%
D4	α=1.0, β=0.8	**0.8782**	0.8918	1.53%
D5	α=1.2, β=1.0	**1.1277**	1.1356	0.70%
D6	α=1.0, β=0.4	**0.5212**	0.5316	1.96%

outcomes also exist on the other datasets, as shown in Table III and Fig. S7.

2) *Variation of α and β can Hardly Affects α-βM's Time Cost per Iteration:* This phenomenon is reasonable, since from (24), we see that change of α and β can only affect the fundamental expression of α-βM's parameter update rules. However, since we need to raise an arbitrary number to the power of α and β in (24), α-βM's time cost per iteration can be slightly affected by α and β's variation, as recorded in Fig. S8 in the supplementary file.
3) *α and β's Self-Adaptation is Highly Desired:* According to Table III, we see that their optimal values are data-dependent. However, their adaptation cannot be implemented similarly with that of κ and λ. This is because κ and λ are controllable parameters in a single model, while α and β are modeling parameters affecting the fundamental distance metric of our learning objective.

C. Comparison With State-of-the-Art Models

In this part, we compare α-βM with several state-of-the-art LF models to validate its performance.

M1 (α-βM): The proposed model of this study.

M2 (NLFA): A widely adopted non-negative model that enables highly efficient LF analysis of an HiDS matrix [8], [52]. Its objective function relies on the Euclidean distance and its parameter learning is based on an SLF-NMU algorithm.

M3 (NeuMF): A state-of-the-art model combines the principle of LF analysis and deep neural networks [39]. It explores an HiDS matrix's linear latent representation via LF analysis and its nonlinear interaction through a multilayered perceptron. *M*3 is initially designed for item ranking test with cross entropy as the learning objective, which can impair its ability to estimate missing data estimation. So we replace it with the widely adopted Euclidean distance defined on the specific known entries of an HiDS matrix to adapt *M*3 to our evaluating protocol.

M4 (I-AutoRec): An LF analysis model based on an autoencoder paradigm [44]. It takes the target HiDS matrix as its input and output simultaneously and trains desired LFs via a backward propagation scheme. In general, it precisely represents nonlinear interactions hidden in the known entries of an HiDS matrix.

M5 Deep Collaborative Conjunctive Recommender (DCCR): A deep collaborative conjunctive model consists of two different types of neural-network models, that is, an autoencoder and a multilayered perceptron [45]. Its autoencoder-based component extracts user and item LFs, while its multilayered perceptron-based component fuses user and item LFs to model their nonlinear interactions.

M6 (β-NLFA): An NLFA model whose learning objective is built based on a β-divergence function trains its LFs via SLF-NMU [2]. Note that as discussed in [27], a β-divergence function is much more generalized than a Euclidean distance function, but still a special case of an α-β-divergence function.

M7 Neural Rating Regression (NRT): A deep model that can simultaneously predict precise ratings and generate abstractive tips with good linguistic quality that simulates user experience and feelings [69]. It translates user and item LFs into a concise sentence by gated recurrent neural networks for improving the recommendation accuracy.

M8 (FBNLFA): A fast NLFA model that incorporates linear biases and adopts a generalized momentum method to achieve a fast and stable training process.

Note that all tested models depend on hyperparameters. As mentioned before, for each model on each dataset, we tune their hyperparameters on one-fold to obtain their optimal values and apply the same values to the remaining four folds to achieve the averaged outputs. The five-fold cross-validations on each dataset are repeated ten times to achieve the final averaged results. In each single test on the same dataset, all compared models' LF dimension is set at 20 uniformly (which is also widely adopted in prior research [3], [8], [25] to well balance the model efficiency and representation ability), and initialized with the same randomly generated arrays. We aim to derive objective and unbiased results via the above settings.

Figs. S9 and S10 in the supplementary file depict the RMSE and time cost of *M*1–6, respectively. From them, we have the following observations.

1) *M1 Accurately Recovers Missing Data of an HiDS Matrix:* On *D*2 and *D*4–6, *M*1 achieves lower RMSE than its peers do. For instance, on *D*2, as depicted in Fig. S9(b), *M*1's RMSE is 0.7058, which is 0.70% lower than 0.7108 by *M*2, 0.75% lower than 0.7111 by *M*3, 0.51% lower than 0.7094 by *M*4, 0.32% lower than 0.7081 by *M*5, 0.54% lower than 0.7096 by *M*6, 0.68% lower than 0.7106 by *M*7, and 0.38% lower than 0.7084 by *M*8. The similar results are observed on *D*4–6, as depicted in Figs. S10(d)–(f). On *D*1, *M*1's RMSE is slightly higher than that of *M*3 and *M*7, that is, NeuMF and NRT. However, it outperforms the other compared models. On *D*3, *M*1 is only outperformed by *M*7. Note that *M*3 and *M*7 are state-of-the-art deep-learning-incorporated models with high representative learning ability, while its RMSE is higher than that of *M*1 on *D*2 and *D*4–6, and slightly lower than that of *M*1 on *D*1 and *D*3. Thus, *M*1's ability to estimate an HiDS matrix's missing data is impressive. When considering involved NLFA models, that is, *M*1, *M*2, *M*6, and *M*8, we see that *M*1's RMSE always beats the other three's. This is consistent with our expectation since their learning objectives are all special cases of *M*1's.
2) *When Addressing an HiDS Matrix, M1's Computational Cost Is the Lowest Among Its Peers:* For instance, as depicted in Fig. S10(a), on *D*1, *M*1 consumes 2703.6 s

TABLE IV
HYPERPARAMETERS CALLING FOR MANUALLY TUNING IN EACH MODEL

Model	Description
M1	α, modeling parameter of α-β-divergence β, modeling parameter of α-β-divergence
M2	λ, regularization coefficient
M3-5	λ, regularization coefficient N, layer count η, learning rate
M6	β, modeling parameter of β-divergence λ, regularization coefficient
M7	λ, regularization coefficient N, layer count η, learning rate d, beam size L, maximum sentence length
M8	κ, momentum velocity coefficient λ, regularization coefficient

TABLE V
AVERAGE RANKS OF ALL COMPARED MODELS

Rank	M1	M2	M3	M4	M5	M6	M7	M8
Accuracy	1.50	7.16	6.50	5.00	3.83	5.33	2.33	4.33
Efficiency	1.00	4.00	5.83	5.83	6.33	3.00	8.00	2.00

to converge, which is 46.97% of 5755.9 s by $M2$, 5.28% of 51163.6 s by $M3$, 5.48% of 49343.1 s by $M4$, 6.04% of 44712.8 s by $M5$, 48.82% of 5538.4 s by $M6$, 4.64% of 58298.3 s by $M7$, and 86.58% of 3122.7 s by $M8$. The similar outcomes are also seen on $D2$–6 as depicted in Figs. S10(b)–(f) in the supplementary file. First, it is reasonable that $M1$ consumes much less time to converge than $M3$–5 and $M7$ do. Note that $M30$-5 and $M7$ are all deep neural-network-based models, which make a compromise between computational efficiency and representation learning ability. As indicated by prior research, such a shortcoming can be alleviated with the help of GPU [39], [44], [45]. However, since an NLFA model like M1 can also be accelerated with parallelization as discussed in [8], it indeed better fits the needs of industrial applications that require frequently building new data models. Note that $M2$, $M6$, and $M8$ are all NLFA models relying on SLF-NMU. For $M2$ and $M6$, they do not have the acceleration design as that of $M1$. Hence, they consume much more iterations to converge than $M1$ does, which greatly increases their time cost. Although $M6$ is accelerated by a generalized momentum method, $M1$ makes hyperparameters self-adaptation to further reduce the time cost to train α-βM. However, we clearly see that they both are much faster than $M3$–5 and $M7$.

3) *M1's Parameter Tuning Is Highly Convenient:* When considering the time cost of a learning model, its parameter tuning should be taken into consideration. We have summarized the hyperparameters calling for manually tuning in each model in Table IV. From it, we clearly see that $M1$'s parameter tuning can be conducted conveniently with α and β only.

1) Significance Analysis: We adopt the Friedman test [51] to validate the statistical significance of the above results. Let r_{ij} be the rank of the jth of k compared models on the ith of N testing cases, we compare average rank of each compared models, that is, $A_j = \sum_{j=1}^{N} r_i^j / N$. We summarize each model's average rank in terms of efficiency and accuracy in Table V. From Table V, we compute the Friedman value of our comparison tests as follows:

$$\chi_F^2 = \frac{12N}{k(k+1)} \left[\sum_j A_j^2 - \frac{k(k+1)^2}{4} \right]. \tag{28}$$

With (28), the testing score is obtained as follows:

$$F_F = \frac{(N-1)\chi_F^2}{N(k-1) - \chi_F^2}. \tag{29}$$

Note that (29) follows the F-distribution with $k-1$ and $(k-1)(N-1)$ degrees of freedom [51]. Hence, we reject the null hypothesis that states the equivalence of compared models with the critical level α if F_F is greater than the critical value.

Note that we have eight models tested on six datasets. In terms of prediction accuracy, we have one testing case on each dataset, that is, RMSE. Hence, we have $k = 8$ and $N = 6$, and F_F with (7, 35) degrees of freedom. Similarly, we also have $k = 8$ and $N = 6$, and F_F with (7, 35) degrees of freedom in terms of efficiency. The critical value of $F(7, 35)$ for $\alpha = 0.05$ is 2.315. Therefore, we can reject the null hypothesis if the testing scores are higher than 2.315. Based on (28) and (29), the testing scores F_F for prediction accuracy and efficiency is 8.309 and 109.545, respectively. Both of the testing scores are much higher than 2.315. As a result, we assert that $M1$–8 are significantly different in performance with a confidence level at 95%.

For further identifying tested models' performance, a Nemenyi test [51] is adopted. In it, two models are significantly different if their performance rank difference is greater than the following critical difference value [51]:

$$\delta = q_\alpha \sqrt{k(k+1)/6N} \tag{30}$$

where q_α is based on the Studentized range statistic [51]. With six tested models in the experiment, the critical value $q_\alpha = 2.780$ with the critical level $\alpha = 0.1$ [51]. By setting $k = 8$, $N = 6$, and $q_\alpha = 2.780$ into (30), we obtain that $\delta = 3.424$. It indicates that any two models with a rank difference higher than 3.931 have significant difference with a confidence level at 90%.

The results of the Nemenyi analysis are depicted in Fig. S11 in the supplementary file of this article. As drawn in Fig. S11(a), we see that considering prediction accuracy, $M1$ outperforms $M2$–8. Especially, it significantly outperforms $M2$ and $M3$. Considering computational efficiency, $M1$ always steadily consumes the least total time cost among all tested models. Meanwhile, as illustrated in Fig. S11(b), the computational efficiency of $M1$ is significantly higher than $M3$–5 and $M7$.

D. Summary

According to the experimental results, we conclude that:

1) With carefully tuned α and β, it achieves high prediction accuracy for missing data of an HiDS matrix.

2) Its computational efficiency is significantly higher than state-of-the-art models due to its generalized momentum-incorporated parameter learning rules and self-adaptation of controllable hyperparameters.

V. Related Work

Predicting Missing User Preferences: High values in a rating matrix commonly denote strong user–item preferences. Consequently, an RS has the need of predicting user preferences via estimating the missing data in an HiDS rating matrix. Recently, one of the research hotspots in RS is deep learning-based recommenders [70] due to the high representation learning ability of a deep learning model. Li *et al.* [69] proposed an NRT model, which can translate user and item LFs into a concise sentence to improve the recommendation performance. Wang *et al.* [45] proposed a DCCR, which combines an autoencoder and a multilayered perceptron to consider the interaction effect between users and items. In spite of their high-prediction accuracy, a deep model requires large computation resources and consumes much time to train, which greatly reduces its practicability on an HiDS matrix.

Generalizing Learning Objective: As shown in Section IV, an objective function has vital effects on the performance of an NLFA model. Note that generalizing the learning objective for improving a model's representation learning ability has been explored in some work. Cichoki *et al.* [27] proposed a generalized NMF model based on α-β-divergence for extracting non-negative LFs from a complete matrix. However, it depends on a non-negative and multiplicative update (NMU) learning rule, which is not applicable to an HiDS matrix filled with numerous missing data. Luo *et al.* [2] adopted a β-divergence objective function [28] to build a β-NLFA model on an HiDS matrix. But β-divergence is still a special case of α-β-divergence. Moreover, β-NLFA relies on SLF-NMU, which suffers from slow model convergence.

Parameter Learning for High Convergence Rate: To further improve a model's convergence rate, researchers have put forward various parameter learning methods. Kingma and Ba [65] proposed the adaptive moment estimation (Adam) algorithm that adaptively adjusts the learning rate based on both the exponentially decaying square average and an exponentially decaying average of past stochastic gradients. Luo *et al.* [66] adopted dynamic bounds on learning rates to achieve a gradual and smooth transition from Adam to SGD, thereby proposed an Adabound algorithm. However, these learning algorithms depend on gradients explicitly. According to [8] and [13], for an NLFA model, SLF-NMU trains non-negative LFs multiplicatively for keeping their non-negativity and it depends on gradients implicitly. Therefore, the above self-adaptation schemes are not as compatible as a generalized momentum method is with SLF-NMU.

Self-Adaptation of Controllable Hyperparameter: Self-adaption of hyperparameters avoids tedious parameter tuning to improve a resultant model's practicability. The Bayesian optimization [67], [68] is a commonly adopted method to address this issue. However, it consumes much time to compute the inverse of a covariance matrix, thus failing to address high-dimensional problems efficiently. A colony algorithm [58] and a genetic algorithm [59] can also address this issue, but they make compromise in computational efficiency. However, it is interesting to incorporate them into α-βM for making hyperparameters self-adaptation, thereby exploring the possibility of improving diversified hyperparameter adaptation. We plan to address this issue as future work.

VI. Conclusion

This study has proposed α-βM for performing NLFA on an HiDS matrix. Compared with the state-of-the-art models, it enjoys its: 1) high ability to represent an HiDS matrix due to its α-β-divergence-generalized learning objective and 2) fast convergence relying on its generalized momentum-incorporated learning scheme.

The proposed α-βM has two types of hyperparameters, controllable and model ones. We have incorporated the principle of PSO to make the former self-adapted. However, a standard PSO algorithm often suffers from premature convergence [48], [53], [54]. Therefore, by incorporating improved PSO, for example, the aging mechanism PSO [55], genetic learning PSO [56], quantum mechanism PSO [57], and hybrid PSO [75], [77], can help α-βM achieve further performance gain. On the other hand, model hyperparameters decide the detailed form of α-βM's α-β-divergence-generalized learning objective. They presently require manual tuning. Since they affect α-βM from a model's perspective, their adaptation strategies are not compatible with that of controllable hyperparameters. How to implement their self-adaptation remains open.

References

[1] H. Wu, Z. X. Zhang, K. Yue, B. B. Zhang, J. He, and L. C. Sun, "Dual-regularized matrix factorization with deep neural networks for recommender systems," *Knowl. Based Syst.*, vol. 145, pp. 46–58, Apr. 2018.

[2] X. Luo, Y. Yuan, M. C. Zhou, Z. G. Liu, and M. S. Shang, "Non-negative latent factor model based on β-divergence for recommender systems," *IEEE Trans. Syst., Man, Cybern., Syst.*, early access, Aug. 21, 2019, doi: 10.1109/TSMC.2019.2931468.

[3] Y. Koren, R. Bell, and C. Volinsky, "Matrix factorization techniques for recommender systems," *IEEE Comput.*, vol. 42, no. 8, pp. 30–37, Aug. 2009.

[4] J. Sha, Y. Y. Du, and L. Qi, "A user requirement oriented Web service discovery approach based on logic and threshold petri net," *IEEE/CAA J. Autom. Sinica*, vol. 6, no. 6, pp. 1528–1542, Nov. 2019.

[5] H. Zahid, T. Mahmood, A. Morshed, and T. Sellis, "Big data analytics in telecommunications: Literature review and architecture recommendations," *IEEE/CAA J. Autom. Sinica*, vol. 7, no. 1, pp. 18–38, Jan. 2020.

[6] J. Chen, D. Lian, and K. Zheng, "Improving one-class collaborative filtering via ranking-based implicit regularizer," in *Proc. 33rd AAAI Conf. Artif. Intell.*, 2019, pp. 37–44.

[7] H. Zhang, Y. Sun, M. Zhao, W. S. Chow, and Q. M. Wu, "Bridging user interest to item content for recommender systems: An optimization model," *IEEE Trans. Cybern.*, vol. 50, no. 10, pp. 4268–4280, Oct. 2020.

[8] X. Luo, M. C. Zhou, Z. D. Wang, Y. N. Xia, and Q. S. Zhu, "An effective scheme for QoS estimation via alternating direction method-based matrix factorization," *IEEE Trans. Services Comput.*, vol. 12, no. 4, pp. 503–518, Jul./Aug. 2019.

[9] V. Kumar, A. K. Pujari, S. K. Sahu, V. R. Kagita, and V. Padmanabhan, "Collaborative filtering using multiple binary maximum margin matrix factorizations," *Inf. Sci.*, vol. 380, pp. 1–11, Feb. 2017.

[10] C. C. Leng, H. Zhang, G. R. Cai, I. Cheng, and A. Basu, "Graph regularized Lp smooth non-negative matrix factorization for data representation," *IEEE/CAA J. Autom. Sinica*, vol. 6, no. 2, pp. 584–595, Mar. 2019.

[11] Y. Ren, G. Li, J. Zhang, and W. Zhou, "Lazy collaborative filtering for data sets with missing values," *IEEE Trans. Cybern.*, vol. 43, no. 6, pp. 1822–1834, Dec. 2013.

[12] C. Wang, Z. Deng, J. Lai, and P. Yu, "Serendipitous recommendation in E-commerce using innovator-based collaborative filtering," *IEEE Trans. Cybern.*, vol. 49, no. 7, pp. 2678–2692, Jul. 2019.

[13] X. Luo, Z. G. Liu, S. Li, M. S. Shang, and Z. D. Wang, "A fast non-negative latent factor model based on generalized momentum method," *IEEE Trans. Syst., Man, Cybern., Syst.*, early access, vol. 51, no. 1, pp. 610–620, Jan. 2021.

[14] Y. Yuan, X. Luo, and M. Shang, "Effects of preprocessing and training biases in latent factor models for recommender systems," *Neurocomputing*, vol. 275, pp. 2019–2030, Jan. 2018.

[15] H. Li, K. Li, J. An, and K. Li, "An online and scalable model for generalized sparse non-negative matrix factorization in industrial applications on multi-GPU," *IEEE Trans. Ind. Informat.*, early access, Jan. 31, 2019, doi: 10.1109/TII.2019.2896634.

[16] P. Zhang, Z. Zhang, T. Tian, and Y. G. Wang, "Collaborative filtering recommendation algorithm integrating time windows and rating predictions," *Appl. Intell.*, vol. 49, pp. 3146–3157, Mar. 2019.

[17] M. S. Shang, X. Luo, Z. G. Liu, J. Chen, Y. Yuan, and M. C. Zhou, "Randomized latent factor model for high-dimensional and sparse matrices from industrial applications," *IEEE/CAA J. Autom. Sinica*, vol. 6, no. 1, pp. 131–141, Jan. 2019.

[18] T. Himabindu, V. Padmanabhan, and A. K. Pujari, "Conformal matrix factorization based recommender system," *Inf. Sci.*, vol. 467, pp. 685–707, Oct. 2018.

[19] X. Luo, M. Zhou, S. Li, and M. Shang, "An inherently nonnegative latent factor model for high-dimensional and sparse matrices from industrial applications," *IEEE Trans. Ind. Informat.*, vol. 14, no. 5, pp. 2011–2022, May 2018.

[20] D. Rafailidis and A. Nanopoulos, "Modeling users preference dynamics and side information in recommender systems," *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 46, no. 6, pp. 782–792, Jun. 2016.

[21] Y. Li *et al.*, "An efficient recommendation method for improving business process modeling," *IEEE Trans. Ind. Informat.*, vol. 10, no. 1, pp. 502–513, Feb. 2014.

[22] B. O. Ayinde and J. M. Zurada, "Deep learning of constrained autoencoders for enhanced understanding of data," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 28, no. 9, pp. 3969–3979, Sep. 2018.

[23] L. Hao, K. Q. Li, J. Y. An, W. H. Zheng, and K. L. Li "An efficient manifold regularized sparse non-negative matrix factorization model for large-scale recommender systems on GPUs," *Inf. Sci.*, vol. 496, pp. 464–484, Sep. 2019.

[24] Y. Xu, W. Yin, Z. Wen, and Y. Zhang, "An alternating direction algorithm for matrix completion with nonnegative factors," *Front. Math. China*, vol. 7, no. 2, pp. 365–384, 2012.

[25] X. Wu, B. Cheng, and J. L. Chen, "Collaborative Filtering Service Recommendation Based on a Novel Similarity Computation Method," *IEEE Trans. Services Comput.*, vol. 10, no. 3, pp. 352–365, May/Jun. 2017.

[26] H. Ma, I. King, and M. R. Lyu, "Learning to recommend with social trust ensemble," in *Proc. 32nd Int. ACM SIGIR Conf. Res. Develop. Inf. Retrieval*, 2009, pp. 203–210.

[27] A. Cichocki, S. Cruces, and S. Amari, "Generalized alpha-beta divergences and their application to robust nonnegative matrix factorization," *Entropy*, vol. 13, no. 1, pp. 134–170, 2011.

[28] R. Hennequin, B. David, and R. Badeau, "Beta-divergence as a subclass of Bregman divergence," *IEEE Signal Process. Lett.*, vol. 18, no. 2, pp. 83–86, Feb. 2011.

[29] D. Sun and C. Fevotte, "Alternating direction method of multipliers for non-negative matrix factorization with the beta-divergence," in *Proc. IEEE Int. Conf. Acoust. Speech Signal Process.*, Florence, Italy, 2014, pp. 6201–6205.

[30] I. S. Dhillon and S. Sra, "Generalized nonnegative matrix approximations with Bregman divergences," in *Proc. 18th Int. Conf. Neural Inf. Process. Syst.*, 2005, pp. 283–290.

[31] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge, U.K.: Cambridge Univ. Press, 2009.

[32] O. Lehmann, E. Martín, R. J. Butler, and G. T. Lloyd, "Biases with the generalized euclidean distance measure in disparity analyses with high levels of missing data," *Palaeontology*, vol. 62, no. 5, pp. 837–849, 2019.

[33] A. Che, P. Wu, F. Chu, and M. Zhou, "Improved quantum-inspired evolutionary algorithm for large-size lane reservation," *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 45, no. 12, pp. 1535–1548, Dec. 2015.

[34] Q. Kang, J. Wang, M. Zhou, and A. C. Ammari, "Centralized charging strategy and scheduling algorithm for electric vehicles under a battery swapping scenario," *IEEE Trans. Intell. Transp. Syst.*, vol. 17, no. 3, pp. 659–669, Mar. 2016.

[35] S. Hsieh, T. Sun, C. Lin, and C. Liu, "Effective learning rate adjustment of blind source separation based on an improved particle swarm optimizer," *IEEE Trans. Evol. Comput.*, vol. 12, no. 2, pp. 242–251, Apr. 2008.

[36] L. Feng and B. Bhanu, "Semantic concept co-occurrence patterns for image annotation and retrieval," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 38, no. 4, pp. 785–799, Apr. 2016.

[37] J. Zhang, "Divergence function, duality, and convex analysis," *Neural Comput.*, vol. 16, no. 1, pp. 159–195, Jan. 2004.

[38] P. Zhou, L. Du, H. Wang, L. Shi, and Y. Shen, "Learning a robust consensus matrix for clustering ensemble via Kullback–Leibler divergence minimization," in *Proc. 24th Int. Joint Conf. Artif. Intell.*, 2015, pp. 4112–4118.

[39] X. He, L. Liao, and H. Zhang, "Neural collaborative filtering," in *Proc. 26th Int. Conf. World Wide Web*, 2017, pp. 173–182.

[40] H. Wu, "The Karush–Kuhn–Tucker optimality conditions in an optimization problem with interval-valued objective function," *Eur. J. Oper. Res.*, vol. 176, no. 1, pp. 46–59, 2007.

[41] J. A. Konstan, B. N. Miller, D. Maltz, J. L. Herlocker, L. R. Gordon, and J. Riedl, "GroupLens: Applying collaborative filtering to usenet news," *Commun. ACM*, vol. 40, no. 3, pp. 77–87, 1997.

[42] L. Brozovsky and V. Petricek, "Recommender system for online dating service," 2007. [Online]. Available: arXiv:cs/0703042.

[43] K. Goldberg, T. Roeder, D. Gupta, and C. Perkins, "Eigentaste: A constant time collaborative filtering algorithm," *Inf. Retrieval*, vol. 4, no. 2, pp. 133–151, 2001.

[44] S. Sedhain, A. K. Menon, S. Sanner, and L. Xie, "AutoRec: Autoencoders meet collaborative filtering," in *Proc. 24th Int. Conf. World Wide Web*, 2015, pp. 111–112.

[45] Q. Wang, B. Peng, X. Shi, T. Shang, and M. Shang, "DCCR: Deep collaborative conjunctive recommender for rating prediction," *IEEE Access*, vol. 7, pp. 60186–60198, 2019.

[46] A. Hernando, J. Bobadilla, and F. Ortega, "A non negative matrix factorization for collaborative filtering recommender systems based on a Bayesian probabilistic model," *Knowl. Based Syst.*, vol. 97, pp. 188–202, Apr. 2016.

[47] X. Luo, Z. D. Wang, and M. S. Shang, "An instance-frequency-weighted regularization scheme for non-negative latent factor analysis on high-dimensional and sparse data," *IEEE Trans. Syst., Man, Cybern., Syst.*, early access, Aug. 6, 2019, doi: 10.1109/TSMC.2019.2930525.

[48] R. C. Eberhart and Y. H. Shi, "Particle swarm optimization: Developments, applications and resources," in *Proc. Congr. Evol. Comput.*, Seoul, South Korea, 2001, pp. 81–86.

[49] X. Luo, M. C. Zhou, S. Li, L. Hu, and M. S. Shang, "Non-negativity constrained missing data estimation for high-dimensional and sparse matrices from industrial applications," *IEEE Trans. Cybern.*, vol. 50, no. 5, pp. 1844–1855, May 2020.

[50] Y. F. Ma, Z. Y. Wang, H. Yang, and L. Yang, "Artificial intelligence applications in the development of autonomous vehicles: A survey," *IEEE/CAA J. Autom. Sinica*, vol. 7, no. 2, pp. 315–329, Mar. 2020.

[51] J. Demšar, "Statistical comparisons of classifiers over multiple data sets," *J. Mach. Learn. Res.*, vol. 7, no. 1, pp. 1–30, 2006.

[52] X. Luo, M. Zhou, Y. Xia, and Q. Zhu, "An efficient non-negative matrix-factorization-based approach to collaborative filtering for recommender systems," *IEEE Trans. Ind. Informat.*, vol. 10 no. 2, pp. 1273–1284, May 2014.

[53] S. Gao, M. Z. Zhou, Y. Wang, J. Cheng, H. Yachi, and J. Wang, "Dendritic neuron model with effective learning algorithms for classification, approximation, and prediction," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 30, no. 2, pp. 601–614, Feb. 2019.

[54] Y. H. Shi and R. Eberhart, "Empirical study of particle swarm optimization," in *Proc. IEEE Congr. Evol. Comput.*, Washington, DC, USA, 1999, pp. 1945–1950.

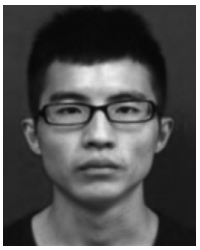
[55] W. N. Chen *et al.*, "Particle swarm optimization with an aging leader and challengers," *IEEE Trans. Evol. Comput.*, vol. 17, no. 2, pp. 241–258, Apr. 2013.

[56] Y. J. Gong *et al.*, "Genetic learning particle swarm optimization," *IEEE Trans. Cybern.*, vol. 46, no. 10, pp. 2277–2290, Oct. 2016.

[57] H. Wang and C. M. Qiao, "A nodes' evolution diversity inspired method to detect anomalies in dynamic social networks," *IEEE Trans. Knowl. Data Eng.*, vol. 32, no. 10, pp. 1868–1880, Oct. 2020.
[58] V. O. Costa and C. R. Rodrigues, "Hierarchical ant colony for simultaneous classifier selection and hyperparameter optimization," in *Proc. Congr. Evol. Comput.*, 2018, pp. 1–8.
[59] C. D. Francescomarino *et al.*, "Genetic algorithms for hyperparameter optimization in predictive business process monitoring," *Inf. Syst.*, vol. 74, pp. 67–83, May 2018.
[60] S. Schelter, S. Ewen, K. Tzoumas, and V. Markl, "'All roads lead to Rome': Optimistic recovery for distributed iterative data processing," in *Proc. 22nd Int. Conf. Inf. Knowl. Manag.*, 2013, pp. 1919–1928.
[61] P. Massa and P. Avesani, "Trust-aware recommender systems," in *Proc. 1st ACM Conf. Recommender Syst.*, 2007, pp. 17–24.
[62] Z. C. Li, J. H. Tang, and T. Mei, "Deep collaborative embedding for social image understanding," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 41, no. 9, pp. 2070–2083, Sep. 2019.
[63] H. Yu *et al.*, "Predicting and understanding comprehensive drug-drug interactions via semi-nonnegative matrix factorization," *BMC Syst. Biol.*, vol. 12, no. 1, pp. 101–110, 2018.
[64] J. J. Pan, S. J. Pan, Y. Jie, L. M. Ni, and Y. Qiang, "Tracking mobile users in wireless networks via semi-supervised colocalization," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 34, no. 3, pp. 587–600, Mar. 2012.
[65] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in*Proc. Int. Conf. Learn. Represent.*, 2015, pp. 303–309.
[66] L. Luo, Y. Xiong, Y. Liu, and X. Sun, "Adaptive gradient methods with dynamic bound of learning rate," in*Proc. Int. Conf. Learn. Represent.*, 2019.
[67] B. Shahriari, K. Swersky, Z. Y. Wang, R. P. Adams, and N. D. Freitas, "Taking the human out of the loop: A review of Bayesian optimization," *Proc. IEEE*, vol. 104, no. 1, pp. 148–175, Jan. 2016.
[68] A. H. Victoria and G. Maragatham, "Automatic tuning of hyperparameters using Bayesian optimization," *Evol. Syst.*, May 2020, doi: 10.1007/s12530-020-09345-2.
[69] P. Li, Z. Wang, Z. Ren, L. Bing, and W. Lam, "Neural rating regression with abstractive tips generation for recommendation," in *Proc. 40th Int. Conf. Res. Develop. Inf. Retrieval*, 2017, pp. 345–354.
[70] S. Zhang, L. Yao, A. Sun, and Y. Tay, "Deep learning based recommender system: A survey and new perspectives," *ACM Comput. Surveys*, vol. 52, no. 1, pp. 1–5, 2019.
[71] J. Li, J. Zhang, C. Jiang, and M. Zhou, "Composite particle swarm optimizer with historical memory for function optimization," *IEEE Trans. Cybern.*, vol. 45, no. 10, pp. 2350–2363, Oct. 2015.
[72] W. Dong and M. Zhou, "A supervised learning and control method to improve particle swarm optimization algorithms," *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 47, no. 7, pp. 1135–1148, Jul. 2017.
[73] Y. Cao, H. Zhang, W. Li, M. Zhou, Y. Zhang, and W. A. Chaovalitwongse, "Comprehensive learning particle swarm optimization algorithm with local search for multimodal functions," *IEEE Trans. Evol. Comput.*, vol. 23, no. 4, pp. 718–731, Aug. 2019.
[74] P. Zhang, S. Shu, and M. Zhou, "An online fault detection method based on SVM-grid for cloud computing systems," *IEEE/CAA J. Autom. Sinica*, vol. 5, no. 2, pp. 445–456, Mar. 2018.
[75] H. Yuan, J. Bi, W. Tan, M. Zhou, B. H. Li, and J. Li, "TTSA: An effective scheduling approach for delay bounded tasks in hybrid clouds," *IEEE Trans. Cybern.*, vol. 47, no. 11, pp. 3658–3668, Nov. 2017.
[76] G. Tian, Y. Ren, and M. Zhou, "Dual-objective scheduling of rescue vehicles to distinguish forest fires via differential evolution and particle swarm optimization combined algorithm," *IEEE Trans. Intell. Transp. Syst.*, vol. 17, no. 11, pp. 3009–3021, Nov. 2016.
[77] H. Yuan, J. Bi, and M. Zhou, "Multiqueue scheduling of heterogeneous tasks with bounded response time in hybrid green IaaS clouds," *IEEE Trans. Ind. Informat.*, vol. 15, no. 10, pp. 5404–5412, Oct. 2019.

Mingsheng Shang received the B.E. degree in management from Sichuan Teachers College, Nanchong, China, in 1995, and the Ph.D. degree in computer science from the University of Electronic Science and Technology of China, Chengdu, China, in 2007.

He is currently a Professor of Computer Science and Technology with the Chongqing Institute of Green and Intelligent Technology, Chinese Academy of Sciences, Chongqing, China. His interests are in complex network analysis and big data applications.



Ye Yuan (Student Member, IEEE) received the B.S. degree in electronic information engineering and the M.S. degree in signal processing from the University of Electronic Science and Technology of China, Chengdu, China, in 2010 and 2013, respectively. He is currently pursuing the Ph.D. degree in computer science with the Chongqing College, University of Chinese Academy of Sciences, Beijing, China.

In 2013, he joined the Chongqing Institute of Green and Intelligent Technology, Chinese Academy of Sciences, Chongqing, China. His research interests include data mining, recommender system, and intelligent computing.

Xin Luo (Senior Member, IEEE) received the B.S. degree in computer science from the University of Electronic Science and Technology of China, Chengdu, China, in 2005, and the Ph.D. degree in computer science from Beihang University, Beijing, China, in 2011.

In 2016, he joined the Chongqing Institute of Green and Intelligent Technology, Chinese Academy of Sciences, Chongqing, China, as a Professor of Computer Science and Engineering. He is currently also a Distinguished Professor of Computer Science with the Dongguan University of Technology, Dongguan, China. His current research interests include big data analysis and intelligent control. He has published over 120 papers (including over 50 IEEE Transactions papers) in the above areas.

Prof. Luo was a recipient of the Hong Kong Scholar Program jointly by the Society of Hong Kong Scholars and China Postdoctoral Science Foundation in 2014, the Pioneer Hundred Talents Program of Chinese Academy of Sciences in 2016, and the Advanced Support of the Pioneer Hundred Talents Program of Chinese Academy of Sciences in 2018. He is currently serving as an Associate Editor for the IEEE/CAA JOURNAL OF AUTOMATICA SINICA, IEEE ACCESS, and *Neurocomputing*.

MengChu Zhou (Fellow, IEEE) received the B.S. degree in control engineering from the Nanjing University of Science and Technology, Nanjing, China, in 1983, the M.S. degree in automatic control from the Beijing Institute of Technology, Beijing, China, in 1986, and the Ph.D. degree in computer and systems engineering from Rensselaer Polytechnic Institute, Troy, NY, USA, in 1990.

He joined the New Jersey Institute of Technology (NJIT), Newark, NJ, USA, in 1990, where he is currently a Distinguished Professor of Electrical and Computer Engineering. He has over 900 publications, including 12 books, over 600 journal papers (over 500 in IEEE transactions), 28 patents, and 29 book-chapters. His research interests are in Petri nets, intelligent automation, Internet of Things, big data, Web services, and intelligent transportation.

Dr. Zhou was a recipient of the Humboldt Research Award for U.S. Senior Scientists from Alexander von Humboldt Foundation, Franklin V. Taylor Memorial Award and the Norbert Wiener Award from IEEE Systems, Man and Cybernetics Society, the Excellence in Research Prize and Medal from NJIT, and Edison Patent Award from the Research and Development Council of New Jersey. He is the Founding Editor of IEEE Press Book Series on Systems Science and Engineering and Editor-in-Chief of IEEE/CAA JOURNAL OF AUTOMATICA SINICA. He is a Life Member of the Chinese Association for Science and Technology-USA and served as its President in 1999. He is a Fellow of the International Federation of Automatic Control, the American Association for the Advancement of Science, the Chinese Association of Automation, and National Academy of Inventors.