Neurocomputing 421 (2021) 316–328

Contents lists available at ScienceDirect

Neurocomputing

journal homepage: www.elsevier.com/locate/neucom

# Hyper-parameter-evolutionary latent factor analysis for high-dimensional and sparse data from recommender systems

Jiufang Chen [a,f], Ye Yuan [b,c], Tao Ruan [d], Jia Chen [e], Xin Luo [a,g,*]

[a] School of Computer Science and Technology, Dongguan University of Technology, Dongguan, Guangdong 523808, China
[b] Chongqing Engineering Research Center of Big Data Application for Smart Cities, and Chongqing Key Laboratory of Big Data and Intelligent Computing, Chongqing Institute of Green and Intelligent Technology, Chinese Academy of Sciences, Chongqing 400714, China
[c] University of Chinese Academy of Sciences, Beijing 100049, China
[d] China Patent Information Center, Beijing 100088, China
[e] School of Cyber Science and Technology, Beihang University, Beijing, 100191 China
[f] School of Computer Science, China West Normal University, Nanchong, Sichuan 637002, China
[g] Chongqing Institute of Green and Intelligent Technology, Chinese Academy of Sciences, Chongqing 400714, China

ARTICLE INFO

*Article history:*
Received 11 August 2020
Revised 30 September 2020
Accepted 13 October 2020
Available online 22 October 2020
Communicated by Z. Wang

*Keywords:*
Big Data
Intelligent Computation
Latent Factor Analysis
Evolutionary Computing
Learning Algorithm
High-dimensional and Sparse Data
Parameter Free

ABSTRACT

High-dimensional and Sparse (HiDS) data generated by recommender systems (RSs) contain rich knowledge regarding users' potential preferences. A Latent factor analysis (LFA) model enables efficient extraction of essential features from such data. However, an LFA model relies heavily on its hyper-parameters like learning rate and regularization coefficient, which must be chosen with care. However, traditional grid-search-based manual tuning is extremely time-consuming and computationally expensive. To address this issue, this study proposes a hyper-parameter-evolutionary latent factor analysis (HLFA) model. Its main idea is to build a swarm by taking the hyper-parameters of every single LFA-based model as particles, and then apply particle swarm optimization (PSO) to make its both hyper-parameters, i.e., the learning rate and regularization coefficient, self-adaptive according to a pre-defined fitness function. Experimental results on six HiDS matrices from real RSs indicate that an HLFA model outperforms several state-of-the-art LF models in terms of computational efficiency, and most importantly, without loss of prediction accuracy for missing data of an HiDS matrix.



## 1. Introduction

The constant explosion of the World-Wide-Web has triggered the serious problem of information overload. It is hard for people to filter out desired information from mountains of data. Therefore, how to develop an intelligent system to filter the useful information out of massive data has become a common issue. In such a context, a Recommendation system (RS) emerges. Its main idea is to extract potential information from users' historical behaviors [1–3] like users' preferences.

In an RS, a user-item rating matrix is commonly the fundamental data source [4–6]. Because of explosively growing quantities of users and items, such a rating matrix is High-dimensional and sparse (HiDS) [40,41]. In an HiDS matrix, only a tiny subset of items can be observed by each user. For instance, the Epinions Matrix [7] is a typical HiDS matrix. It contains 631,064 ratings by 51,670 users on 83,509 different entries, with a data density of 0.015% only. Despite its sparsity, an HiDS matrix contains rich information like the user community [42] and users' potential favorites [8].

According to prior research [1,4,9,10], a Latent factor analysis (LFA) model can efficiently address an HiDS matrix. It maps the row and column entities of a target HiDS matrix into a unique and low-dimensional latent factor (LF) space to build its low-rank approximation based on its known data only. To date, great efforts on LFA have been made to achieve several sophisticated models, e.g., a biased regularized incremental simultaneous LF model [11], probabilistic LF model [12], and non-negative LF model [9,10,13].

To build an LFA-based model, an efficient and commonly-adopted learning algorithm is the Stochastic Gradient Descent (SGD) algorithm [1,14–18], which enjoys its low complexity and fast convergence [6,7]. However, the success of such an LFA-based model depends largely on the careful selection of the hyper-parameters, i.e., the learning rate and regularization

* Corresponding author.
*E-mail addresses:* chenjurl@outlook.com (J. Chen), yeyuan@cigit.ac.cn (Y. Yuan), ruantao0223@163.com (T. Ruan), chenjia@buaa.edu.cn (J. Chen), luoxin21@gmail.com (X. Luo).

https://doi.org/10.1016/j.neucom.2020.10.030
0925-2312/

coefficient. For instance, a small learning rate leads to slow convergence, and a large learning rate might result in overshooting [43]. Moreover, an inappropriate regularization coefficient makes an LF model suffer from overfitting or underfitting, both lead to severe performance loss [9]. Due to their critical effects on an LFA model's performance, they must be chosen with care [19–24]. Current approaches to hyper-parameter selection in an LFA model can be divided into the following two categories:

a) **Grid search.** It is a benchmark approach and easy to implement. However, it is extremely time-consuming and computationally expensive, and not suitable for big data-based industrial applications. For instance, considering the case that there are 10 candidate values for the learning rate and 10 candidate values for the regularization coefficient. Then a grid search process can result in 100 independent runs of an LFA model, making the time cost considerable.
b) **Self-adaptation.** Representative methods of this kind include Adagrad [19], AdaDelta [20], and Adam [21] algorithms. They adjust the learning rate in a self-adaptive way [22–24]. However, as unveiled in [28], they all consume most time in computing the statistics of the past gradients, thereby cost much more time in a single iteration than a standard SGD algorithm does. Meanwhile, they cannot make the regularization coefficient self-adaptive either.

Ideally, we want to make the hyper-parameters of an SGD-dependent LFA model, i.e., the learning rate and regularization coefficient, self-adaptive simultaneously and efficiently. To address this issue, this study proposes a hyper-parameter-evolutionary latent factor analysis (HLFA)-based model. Its main idea is to build a swarm taking the hyper-parameters of every single LFA-based model as particles, and then apply Particle Swarm Optimization (PSO) [25–27] to it for making its hyper-parameters adaptive with the predefined fitness functions. Moreover, the linearly decreasing inertia weight is introduced to avoid PSO being stacked by local solutions. The main contributions of this paper include:

a) An HLFA model that can automatically detect the appropriate learning rate and regularization coefficient simultaneously and efficiently; and
b) Algorithm design and analysis for an HLFA model.

For validating the performance of the proposed HLFA model, we conduct experiments on six HiDS matrices generated by industrial applications currently in use. The positive results indicate the efficiency of an HLFA model.

Section 2 gives the preliminaries. Section 3 presents the methods. Section 4 provides the experimental results. Finally, Section 5 draws the conclusion and shows our plan.

## 2. Preliminaries

### 2.1. Problem statement

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

**Definition 1.** Let $U$ and $I$ represent two large entity sets, and $R^{|U|\times|I|}$ quantize certain interactions among entities in $U$ and $I$. Let $R_K$ and $R_U$ denote the known and unknown entity sets of $R$. Then $R$ is an HiDS matrix if $|R_K| \ll |R_U|$.

**Definition 2.** An LFA-based model builds a low-rank approximation to an HiDS matrix. We have $\hat{R} = XY^T$ where $\hat{R}$ denotes $R$'s rank-f approximation built on $R_K$ only. $X^{|U|\times f}$ and $Y^{f\times|I|}$ denote the LF matrices corresponding to $U$ and $I$. Note that $f$ denotes the rank of $\hat{R}$ and $f \ll min\{|U|, |I|\}$.

To obtain $X$ and $Y$, an objective function is designed to measure the difference between each value in $R_K$ and corresponding entries in $\hat{R}$. With Euclidean distance, such an objective function is formulated by:

$$\varepsilon(X,Y) = \sum_{(u,i)\in R_K} \left( \left( r_{u,i} - \sum_{m=1}^{f} x_{u,m} y_{m,i} \right)^2 \right) \tag{1}$$

where $r_{u,i}$, $x_{u,m}$, $y_{m,i}$ denote single elements in $R$, $X$, $Y$, respectively.

To regularize an LFA-based model, L2-norm-based regularization is efficient [9]. With it, (1) is reformulated into:

$$\varepsilon(X,Y) = \sum_{(u,i)\in R_K} \left( \left( r_{u,i} - \sum_{m=1}^{f} x_{u,m} y_{m,i} \right)^2 + \lambda \left( \sum_{m=1}^{f} x_{u,m}^2 + \sum_{m=1}^{f} y_{m,i}^2 \right) \right), \tag{2}$$

where the coefficient $\lambda$ tunes the regularization effect.

### 2.2. An SGD-based LF model

As indicated by prior work [1,28,29], SGD is an efficient optimizer when performing an LFA-based model of HiDS matrices. With it, the objective function (1) is minimized as follows:

$$\begin{aligned} &\underset{X,Y}{\operatorname{argmin}}\, \varepsilon(X,Y) \overset{SGD}{\Rightarrow} \forall r_{u,i} \in R_K, \quad m \in \{1,2,\cdots,f\}: \\ &\qquad \begin{cases} x_{u,m}^t \leftarrow x_{u,m}^{t-1} - \eta \frac{\partial \varepsilon_{u,i}^{t-1}}{\partial x_{u,m}^{t-1}}, \\ y_{m,i}^t \leftarrow y_{m,i}^{t-1} - \eta \frac{\partial \varepsilon_{u,i}^{t-1}}{\partial y_{m,i}^{t-1}}; \end{cases} \end{aligned} \tag{3}$$

where $\varepsilon_{u,i} = \left( r_{u,i} - \sum_{m=1}^{f} x_{u,m} y_{m,i} \right)^2 + \lambda \left( \sum_{m=1}^{f} x_{u,m}^2 + \sum_{m=1}^{f} y_{m,i}^2 \right)$, $\eta$ denotes the learning rate and $t$ represents the $t$th iteration. By making $err_{u,i} = r_{u,i} - \sum_{m=1}^{f} x_{u,m} y_{m,i}$ and substituting (2) into (3), we achieve:

$$\begin{cases} x_{u,m}^t \leftarrow x_{u,m}^{t-1} - \eta \left( \lambda x_{u,m}^{t-1} - y_{m,i}^{t-1} err_{u,i}^{t-1} \right), \\ y_{m,i}^t \leftarrow y_{m,i}^{t-1} - \eta \left( \lambda y_{m,i}^{t-1} - x_{u,m}^{t-1} err_{u,i}^{t-1} \right). \end{cases} \tag{4}$$

As supported by previous research [1,13,30], an SGD-based LFA model's performance largely depends on the appropriate selection of learning rate $\eta$ and regularization coefficient $\lambda$.

## 3. Methods

### 3.1. Particle swarm optimization

PSO is an evolutionary computation algorithm [31–34]. In it, $q$ particles of a swarm fly in $D$-dimensional search space at a certain speed. Each particle serves as a potential solution to the current optimization task. Their movement depends on two factors, i.e., velocity and position. More specifically, the velocity $v$ and position $s$ of the $j$th particle at the $t$th iteration are denoted by two vectors, i.e., $v_j = (v_{j,1}(t), v_{j,2}(t), \ldots, v_{j,D}(t))$ and $s_j(t) = (s_{j,1}(t), s_{j,2}(t), s_{j,D}(t))$. $v_{j\max} = (v_{j\max,1}, v_{j\max,2}, \ldots, v_{j\max,D})$, $s_{j,d}(t) \in [s_{\min,d}, s_{\max,d}]$, and $1 \le j \le q$, $1 \le d \le D$, where $v_{j\max}$ is the maximum velocity of the

whole dimension, $s_{\min,d}$ and $s_{\max,d}$ are lower and upper bounds for the $d$th dimension, respectively.

During the evolution process, each particle determines its next flying position through its own flying experience and the best flying experience among its peers [33,34]. One is named *pbest*, i.e., $pbest_j = (pbest_{j,1}, pbest_{j,2}, \ldots, pbest_{j,D})$, which is the best position discovered by itself, and the other is named *gbest*, i.e., $gbest = (gbest_1, gbest_2, \ldots, gbest_D)$, which is the best position in the whole swarm. The evolution rule of the $j$th particle at the $t$th iteration is:

$$\begin{aligned} v_j(t) &= w v_j(t-1) + c_1 r_1 \left(pbest_j(t-1) - s_j(t-1)\right) \\ &\quad + c_2 r_2 \left(gbest(t-1) - s_j(t-1)\right), \\ s_j(t) &= s_j(t-1) + v_j(t); \end{aligned} \tag{5}$$

where $w$ is non-negative inertia constant, which is used to balance the local and global search ability, and $w = 0.729$ is set by following [35], $c_1$ and $c_2$ are cognitive and social coefficients fixed to 2 following [35], $r_1$ and $r_2$ are two uniform random numbers in [0, 1], respectively [31–35].

### 3.2. An HLFA model

As mentioned above, we take a two-dimensional vector as one particle, whose first dimension denotes the learning rate and the second dimension represents the regularization coefficient. Hence, we update the velocities and positions as:

$$\begin{cases} v_j(t) = \left[\eta v_j(t), \lambda v_j(t)\right]^{\mathrm{T}} = \begin{bmatrix} \eta v_j(t) \\ \lambda v_j(t) \end{bmatrix}, \\ s_j(t) = \left[\eta_j(t), \lambda_j(t)\right]^{\mathrm{T}} = \begin{bmatrix} \eta_j(t) \\ \lambda_j(t) \end{bmatrix}; \end{cases} \tag{6}$$

where $\eta v$ and $\lambda v$ denotes the velocity of the dimensional learning rate and dimensional regularization coefficient, respectively. By combining (4) and (6), the update scheme of the SGD-based LFA model in the $t$th iteration is given as:

$$\begin{aligned} &\begin{cases} x_{u,m}(t) \leftarrow x_{u,m}(t) - s_j^1(t)\left(s_j^2(t) x_{u,m}(t) - y_{m,i}(t) err_{u,i}(t)\right), \\ y_{m,i}(t) \leftarrow y_{m,i}(t) - s_j^1(t)\left(s_j^2(t) y_{m,i}(t) - x_{u,m}(t) err_{u,i}(t)\right); \end{cases} \\ \Rightarrow &\begin{cases} x_{u,m}(t) \leftarrow x_{u,m}(t) - \eta_j(t)\left(\lambda_j(t) x_{u,m}(t) - y_{m,i}(t) err_{u,i}(t)\right), \\ y_{m,i}(t) \leftarrow y_{m,i}(t) - \eta_j(t)\left(\lambda_j(t) y_{m,i}(t) - x_{u,m}(t) err_{u,i}(t)\right); \end{cases} \end{aligned} \tag{7}$$

After taking the SGD-based update for $q$ times in the $t$th iteration, we have the LF for the first particle in the $(t+1)$th iteration:

$$\begin{cases} x_{u,m}(t+1) = x_{u,m}(t), \\ y_{m,i}(t+1) = y_{m,i}(t); \end{cases} \tag{8}$$

As described by (7) and (8), each particle (the learning rate and regularization coefficient) is updated by the previous particle, we can regard the learning rate and regularization coefficient of the SGD-based LF model as a particle.

Thus, we consider the problem as a two-dimensional vector optimization problem. Therefore, the velocity and position of the particles in the $(t+1)$th iteration are updated according to:

$$\begin{cases} v_j(t+1) = \begin{bmatrix} \eta v_j(t+1) \\ \lambda v_j(t+1) \end{bmatrix} = w \begin{bmatrix} \eta v_j(t) \\ \lambda v_j(t) \end{bmatrix} \\ \qquad + c_1 r_1 \left( pbest_j(t) - \begin{bmatrix} \eta_j(t) \\ \lambda_j(t) \end{bmatrix} \right) \\ \qquad + c_2 r_2 \left( gbest(t) - \begin{bmatrix} \eta_j(t) \\ \lambda_j(t) \end{bmatrix} \right), \\ s_j(t+1) = \begin{bmatrix} \eta_j(t+1) \\ \lambda_j(t+1) \end{bmatrix} = \begin{bmatrix} \eta_j(t) \\ \lambda_j(t) \end{bmatrix} + \begin{bmatrix} \eta v_j(t+1) \\ \lambda v_j(t+1) \end{bmatrix}. \end{cases} \tag{9}$$

To ensure that all particles can learn the best position, all the $pbest_j(t)$ can use the information of a particle historically best position to generate a new position in the search space, where $pbest_j(t)$ is based on fitness function $F$ according to:

$$pbest_j(t) = \begin{cases} \begin{bmatrix} \eta pbest_j(t-1) \\ \lambda pbest_j(t-1) \end{bmatrix}, F\left(\begin{bmatrix} \eta pbest_j(t-1) \\ \lambda pbest_j(t-1) \end{bmatrix}\right) \leqslant F\left(\begin{bmatrix} \eta_j(t) \\ \lambda_j(t) \end{bmatrix}\right); \\ \begin{bmatrix} \eta_j(t) \\ \lambda_j(t) \end{bmatrix}, F\left(\begin{bmatrix} \eta pbest_j(t-1) \\ \lambda pbest_j(t-1) \end{bmatrix}\right) > F\left(\begin{bmatrix} \eta_j(t) \\ \lambda_j(t) \end{bmatrix}\right). \end{cases} \tag{10}$$

Based on (10), we get the global optimal position as follow:

$$gbest(t) = \begin{bmatrix} \eta gbest(t) \\ \lambda gbest(t) \end{bmatrix} = \underset{\eta pbest(t), \lambda pbest(t)}{\operatorname{argmin}} F\left(\begin{bmatrix} \eta pbest(t) \\ \lambda pbest(t) \end{bmatrix}\right); \tag{11}$$

where $\eta pbest(t) = \{\eta pbest_1(t), \eta pbest_2(t), \ldots, \eta pbest_q(t)\}$, and $\lambda pbest(t) = \{\lambda pbest_1(t), \lambda pbest_2(t), \ldots, \lambda pbest_q(t)\}$. Moreover, each particle position should be bounded to shrink searching space, the candidate space of learning rate is $[\eta_{\min}, \eta_{\max}]$, the candidate space of the regularization coefficient is $[\lambda_{\min}, \lambda_{\max}]$.

$$s_j(t+1) = \begin{bmatrix} \eta_j(t+1) \\ \lambda_j(t+1) \end{bmatrix} = \begin{cases} \eta_j(t+1) = \begin{cases} \eta_{\min}, \eta_j(t+1) \leqslant \eta_{\min}, \\ \eta_{\max}, \eta_j(t+1) > \eta_{\max}; \end{cases} \\ \lambda_j(t+1) = \begin{cases} \lambda_{\min}, \lambda_j(t+1) \leqslant \lambda_{\min}, \\ \lambda_{\max}, \lambda_j(t+1) > \lambda_{\max}. \end{cases} \end{cases} \tag{13}$$

Meanwhile, the corresponding velocity of each particle needs to be restricted to reduce the possibility of skipping the global optimal solution or falling into the local optimal solution.

$$v_j(t+1) = \begin{bmatrix} \eta v_j(t+1) \\ \lambda v_j(t+1) \end{bmatrix} = \begin{cases} \eta v_j(t+1) = \begin{cases} v_{\min}, \eta v_j(t+1) \leqslant v_{\min}, \\ v_{\max}, \eta v_j(t+1) > v_{\max}; \end{cases} \\ \lambda v_j(t+1) = \begin{cases} v_{\min}, \lambda v_j(t+1) \leqslant v_{\min}, \\ v_{\max}, \lambda v_j(t+1) > v_{\max}; \end{cases} \end{cases} \tag{14}$$

where $v_{\max}$ and $v_{\min}$ mean the upper and lower bounds of $v$. In our study, we set the boundary of velocity as $v_{\max} = 0.2 \times \min[(\eta_{\max} - \eta_{\min}), (\lambda_{\max} - \lambda_{\min})]$ [32] and $v_{\min} = -v_{\max}$ [32], where $[\eta_{\min}, \eta_{\max}] = [2\text{–}12, 2\text{–}8]$ represents the boundary of the learning rate, and $[\lambda_{\min}, \lambda_{\max}] = [2\text{–}7, 2\text{–}3]$ denotes the boundary of the regularization coefficient, respectively.

In this paper, since we mainly care about the convergence rate and the prediction accuracy of the SGD-based LFA model. The fitness functions link with the optimization objective to make a resultant swarm finely fit the target data. Therefore, we choose root mean squared error (RMSE) as the fitness function $F_1$ and mean absolute error (MAE) as the fitness function $F_2$, which are the standard of the performance of the PSO, and it is a key performance indicator for an HLFA model which is commonly used in RSs to prediction accuracy for an HiDS matrix's missing values. It is given below:

$$\begin{aligned} F_1(s) &= \sqrt{\left(\sum_{r_{u,i} \in \Gamma} \left(r_{u,i} - \hat{r}_{u,i}\right)^2\right) / |\Gamma|}, \\ F_2(s) &= \left(\sum_{r_{u,i} \in \Gamma} \left|r_{u,i} - \hat{r}_{u,i}\right|_{abs}\right) / |\Gamma|. \end{aligned} \tag{15}$$

where $\Gamma$ denotes the testing set and is disjoint with $R_K(u)$, $|\cdot|_{abs}$ calculate the absolute value of given value, $\hat{r}_{u,i}$ denotes the prediction value corresponding to the instance $r_{u,i} \in \Gamma$. Naturally, the fitness function should be reconsidered when the optimization objective of the whole swarm alters.

In each iteration, we adopt (7) to update the LF matrices $X$ and $Y$, and use (9) to update the learning rate and regularization coefficient. In this way, an HLFA model achieves. Next, we introduce a variant form of PSO, which can improve the performance of PSO.

### 3.3. Linearly decreasing inertia weight incorporation

A standard PSO has the advantages of high efficiency and good compatibility [25–27]. Nevertheless, it is easy to suffer from premature convergence and fall into local solutions. Therefore, linearly decreasing inertia weight [32] is introduced into the evolutionary process to further improve the accuracy of HLFA. The inertia weight $w$ varies linearly with the generations, which is given by:

$$w^t = w_{\max} - (w_{\max} - w_{\min}) \times \frac{t_{\text{round}}}{t_{\max}}; \tag{16}$$

where $w_{\max}$ and $w_{\min}$ are the maximal and minimal value of the inertia weight, which are 0.9 and 0.4 respectively according to [32]. $t_{\text{round}}$ represents the current iteration number and $t_{\max}$ represents the maximum number of training iterations.

Generally, reasonable control of the two acceleration coefficients helps improve the performance of PSO. Therefore, we make two acceleration coefficients change with time to reduce the local optima component and increase the global component as follows [39]:

$$\begin{aligned} c_1^t &= c_{\max} - (c_{\max} - c_{\min}) \times \tfrac{t_{\text{round}}}{t_{\max}}, \\ c_2^t &= c_{\min} + (c_{\max} - c_{\min}) \times \tfrac{t_{\text{round}}}{t_{\max}}. \end{aligned} \tag{17}$$

where $c_{\max}$ and $c_{\min}$ represent the maximal and minimal values of two acceleration coefficients, where their values are set as 2.5 and 0.5 according to [39], respectively.

### 3.4. Algorithm design and analysis

Based on the above analysis, we develop an HLFA Algorithm. In each iteration, HLFA updates LF matrices $X$ and $Y$ with (7), and updates the hyper-parameters with (9). As illustrated in Algorithm HLFA, $v$ and $s$ represent the velocity and position vector of the particle, *pbest* and *gbest* represent the local optimal position and the global optimal position, and $z$ denotes the value of the fitness function $F$, respectively.

**Procedure WeightAdjust**

| **Input: *E*, *n*** | |
|---|---|
| **Operation** | **Cost** |
| $w = w_{\max} - (w_{\max} - w_{\min}) \times (n/E)$ | $\Theta(1)$ |
| $c_1 = c_{\max} - (c_{\max} - c_{\min}) \times (n/E)$ | $\Theta(1)$ |
| $c_2 = c_{\min} + (c_{\max} - c_{\min}) \times (n/E)$ | $\Theta(1)$ |
| **Output: *w*, *c*₁, *c*₂** | |

As shown in Algorithm HLFA, we calculate its time complexity as follows:

$$\begin{aligned} T_{HLFA} &\approx \Theta((|U| + |I|) \times f + 3q \times 2 + 2 + q) \\ &\quad + \Theta(n \times q \times (|R_K| \times 3f + |\Gamma| \times f)) \\ &\approx \Theta(n \times q \times |R_K| \times f) \end{aligned} \tag{18}$$

Note that in (18) with $|R_k| \ll \max\{|U|,|I|\}$, we can omit the lower-order-terms to ensure the result reasonably. There are two main factors when computing the computational complexity of HLFA: 1) An SGD-based LF model's matrices $X$ and $Y$, whose storage cost is $\Theta((|U| + |I|) \times f)$; 2) the particle of PSO's auxiliary matrices $S$, $V$, *pbest* whose storage cost is:

$$S_{HLFA} = (|U| + |I|) \times f + 6q \tag{19}$$

Based on the above inferences, we see that an HLFA algorithm is highly efficient in both computation and storage. Note that the update of the inertia weight and acceleration coefficients according to Procedure WeightAdjust will affect the time complexity and storage capacity of HLFA.

**Algorithm. HLFA**

| **Input: *U*, *I*, $R_K$, *f*, $\Gamma$** | |
|---|---|
| **Operation** | **Cost** |
| **Initialize** $X^{\lvert U\rvert\times f}$ and $Y^{\lvert I\rvert\times f}$ randomly | $\Theta((\lvert U\rvert+\lvert I\rvert) \times f)$ |
| **Initialize** $q$, $r_1$, $r_2$, $c_1$, $c_2$, $w$ | $\Theta(1)$ |
| **Initialize** $\eta_{min}$, $\eta_{max}$, $\lambda_{min}$, $\lambda_{max}$, $v_{initmin}$, $v_{initmax}$ | $\Theta(1)$ |
| **Initialize** $w_{max}$, $w_{min}$, $c_{max}$, $c_{min}$ | $\Theta(1)$ |
| **Initialize** $S^{q\times 2}$, $V^{q\times 2}$ and $pbest^{q\times 2}$ randomly | $\Theta(3q \times 2)$ |
| **Initialize** $gbest_2$ and $z_q$ randomly | $\Theta(2 + q)$ |
| **Initialize** $n = 0$, *Max-training-round* = $E$ | $\Theta(1)$ |
| **while not** converge **and** $n \le E$ **do** | $\times n$ |
| **for** $j = 1$ **to** $q$ | $\times q$ |
| **for each** $r_{u,i} \in R_K$ | $\times \lvert R_K\rvert$ |
| $err_{u,i} = r_{u,i} - \sum_{m=1}^{f} x_{u,m} y_{m,i}$ | $\Theta(f)$ |
| **for** $m = 1$ **to** $f$ | $\times f$ |
| $x_{u,m} = x_{u,m} - \eta_i(\lambda_i x_{u,m} - y_{m,i} err_{u,i})$ | $\Theta(1)$ |
| $y_{m,i} = y_{m,i} - \eta_i(\lambda_i y_{m,i} - x_{u,m} err_{u,i})$ | $\Theta(1)$ |
| **end for** | – |
| **end for** | – |
| $z_j = F(s_j)$ according to (15) | $\Theta(\lvert\Gamma\rvert \times f)$ |
| **end for** | – |
| **for** $j = 1$ **to** $q$ | $\times q$ |
| **if** $z_j < F(pbest_j)$ **then** $pbest_j = s_j$ | $\Theta(1)$ |
| **if** $z_j < F(gbest)$ **then** $gbest = s_j$ | $\Theta(1)$ |
| **end for** | – |
| **for** $j = 1$ **to** $q$ | $\times q$ |
| $(w, c_1, c_2)$ **= WeightAdjust**$(E, n)$ | $\Theta(3)$ |
| $v_j = w v_j + c_1 r_1 (pbest_j - s_j) + c_2 r_2 (gbest - s_j)$ | $\Theta(1)$ |
| **if** $v_j = v_{min}$ **then** $v_j = v_{min}$ | $\Theta(1)$ |
| **else if** $v_j > v_{max}$ **then** $v_j = v_{max}$ | $\Theta(1)$ |
| $s_j = s_j + v_j$ | $\Theta(2)$ |
| **if** $\eta_j \le \eta_{min}$ **then** $\eta_j = \eta_{min}$ | $\Theta(1)$ |
| **else if** $\eta_j > \eta_{max}$ **then** $\eta_j = \eta_{max}$ | $\Theta(1)$ |
| **if** $\lambda_j \le \lambda_{min}$ **then** $\lambda_j = \lambda_{min}$ | $\Theta(1)$ |
| **else if** $\lambda_j > \lambda_{max}$ **then** $\lambda_j = \lambda_{max}$ | $\Theta(1)$ |
| **end for** | – |
| $n = n + 1$ | $\Theta(1)$ |
| **end while** | – |
| **Output: *X*, *Y*** | |

## 4. Experimental results and analysis

### 4.1. General settings

#### 4.1.1. Evaluation metrics

In industrial application, accuracy and efficiency of prediction for missing values of an HiDS matrix are what we mainly care about. To evaluate the prediction accuracy of an algorithm, root mean squared error (RMSE) and mean absolute error (MAE)

[17,36,37,44] are frequently chosen as the evaluation metric. Formally,

$$RMSE = \sqrt{\left(\sum_{r_{u,i}\in\Gamma}\left(r_{u,i}-\hat{r}_{u,i}\right)^2\right)/|\Gamma|},$$
$$MAE = \left(\sum_{r_{u,i}\in\Gamma}\left|r_{u,i}-\hat{r}_{u,i}\right|_{abs}\right)/|\Gamma|. \quad (20)$$

where $\hat{r}_{u,i}$ denotes the generated prediction for the testing instance $r_{u,i} \in \Gamma$, which represents the missing value at the $u$th row and $i$th column of a target HiDS matrix. $|\cdot|$ represents the cardinality of a given set. $|\Gamma|$ denotes the size of the validation dataset $\Gamma$. All experiments are conducted on a Tablet with a 2.4 GHz Xeon E5-2680V4 CPU and 16 GB RAM and implemented in JAVA SE 8U172.

#### 4.1.2. Datasets

Six datasets adopt in our experiment, which are collected from the actual industrial application. All datasets are a) high-dimensional, b) extremely sparse. Hence, our experiment results are representative.

- **D1: MovieLens 10 M**. It is collected from the MovieLens system maintained by the GroupLens [36] research team. It has a rating density of 1.31% and a rating scale of [0.5, 5], which contains 10,000,054 ratings of 10,681 movies by 71,567 users.
- **D2: Douban**. It is collected by the Chinese largest online book, movie, and music database Douban [8]. It has a rating scale of [1, 5] and a density of 0.22% only, which contains 16,830,839 ratings of 58,541 items by 129,490 users.
- **D3: EachMovie**. It is collected from the EachMovie system by the DEC Systems Research Center [30]. It has a rating scale of [0, 1] and the data density is 2.37%, which contains 2,811,718 ratings of 1,628 different movies by 72,916 users.
- **D4: Flixter**. It is collected from the Flixter commercial website [17]. It has a rating scale of [0.5, 5] and the data density is 0.11% only, which includes 8,196,077 ratings applied to 48,794 movies on 147,612 users.
- **D5: Epinion**. It is collected by the Trustlet website [38]. It has a range of [1, 5] and the data density is 0.015% only, which includes 13,668,320 ratings by 120,492 users on 775,760 articles.
- **D6: MovieLens 20 M**. It is collected by the MovieLens system [36] maintained by the GroupLens research team. It has a range of [0.5, 5] and the data density is 0.54% only, which contains 20,000,263 entries from 138,493 users on 26,744 movies.

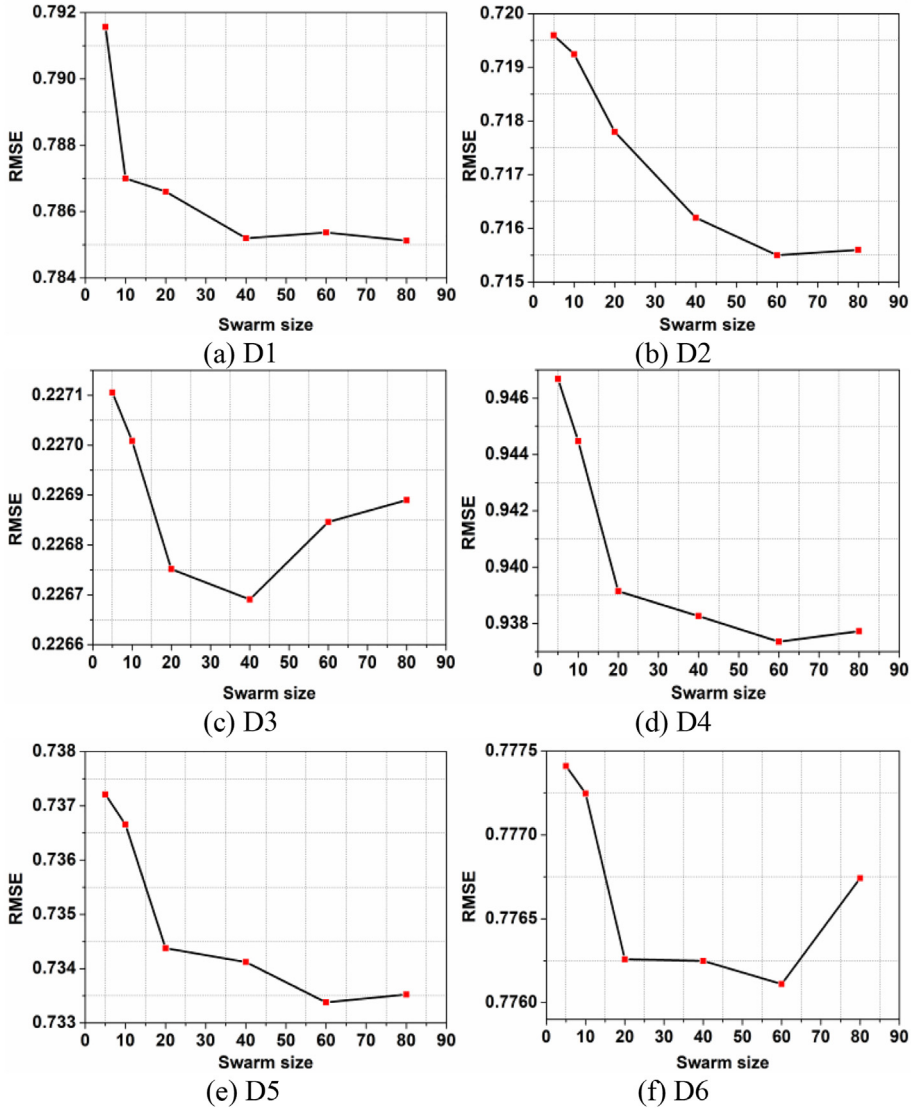



**Fig. 1.** M5′s RMSE as swarm size varies.

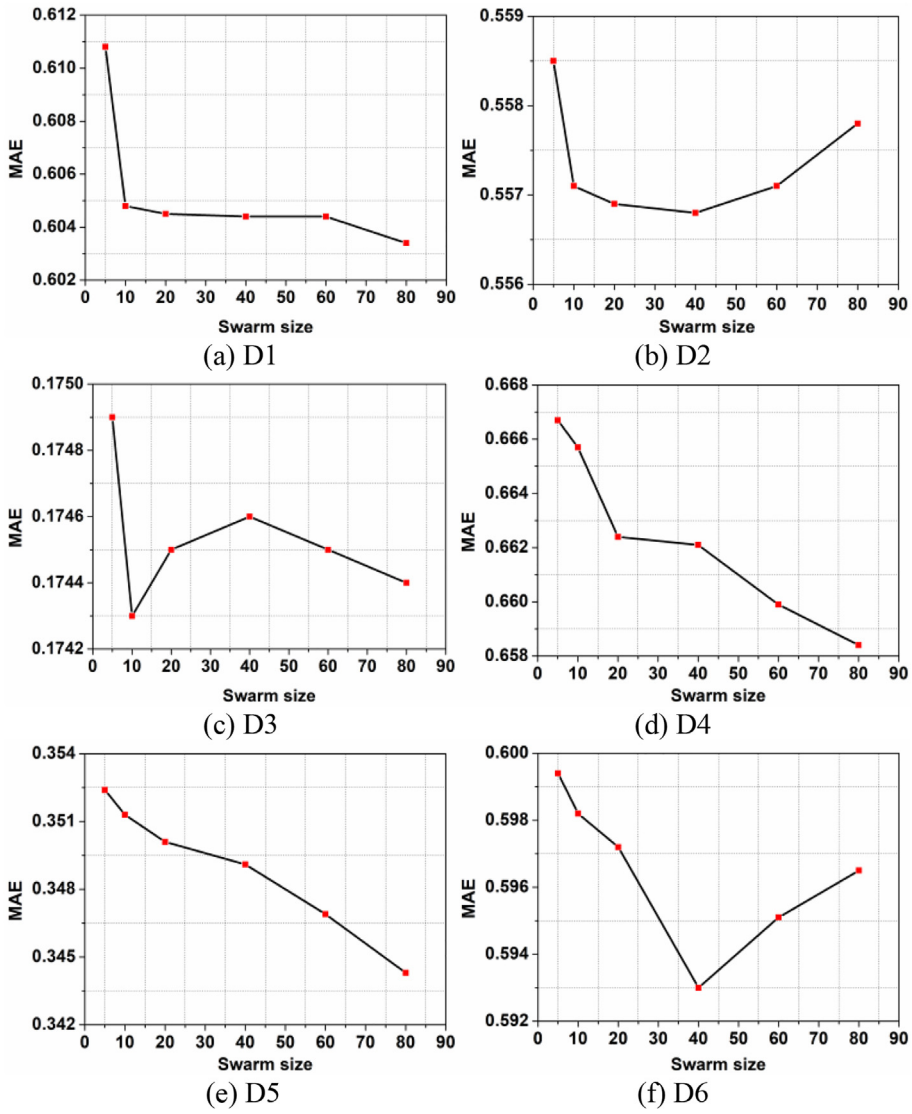



**Fig. 2.** M5′s MAE as swarm size varies.

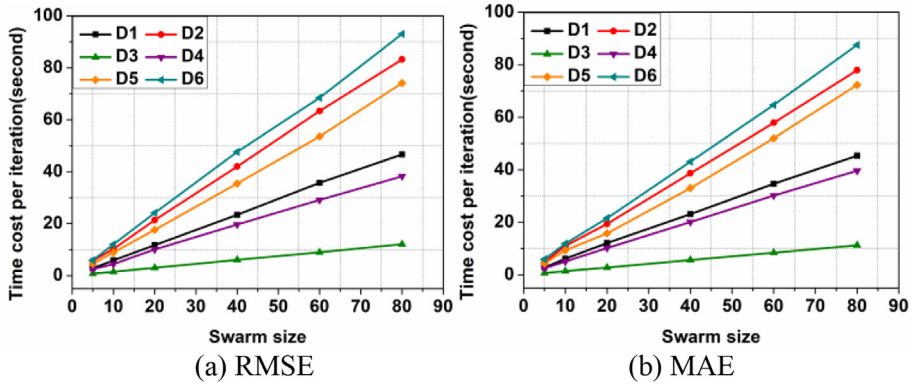



**Fig. 3.** M5′s time cost per iteration as swarm size varies.

Each dataset is randomly split into ten disjoint subsets for implementing 70%–10%–20% train-validation-test settings. More specifically, on each dataset, we adopt seven subsets as a training set to train a model, one as a validating set to monitor the training process for making a model achieve its optimal outputs, and the last two as the test set for test the performance of each achieved model. This process is sequentially repeated ten times to obtain the final results. The termination condition is uniform for all

**Table 1**
Final settings of learning rate and regularization coefficient for M1.

| Dataset | $\eta$ | $\lambda$ |
|---|---|---|
| D1 | $2^{-12}$ | $2^{-6}$ |
| D2 | $2^{-11}$ | $2^{-5}$ |
| D3 | $2^{-9}$ | $2^{-6}$ |
| D4 | $2^{-10}$ | $2^{-4}$ |
| D5 | $2^{-9}$ | $2^{-4}$ |
| D6 | $2^{-12}$ | $2^{-6}$ |

**Table 2**
Time cost per iteration and iteration count of RMSE (SECONDS).

| Data set | Time cost per iteration/Iteration count | | | | | |
|---|---|---|---|---|---|---|
| | M1 | M2 | M3 | M4 | M5 | M6 |
| D1 | 0.576/798 | 1.781/235 | 3.176/998 | 3.397/228 | 6.667/**18** | 6.359/92 |
| D2 | 1.076/877 | 3.206/365 | 5.631/1000 | 5.523/291 | 10.268/**41** | 10.488/82 |
| D3 | 0.153/1000 | 0.528/423 | 0.846/1000 | 0.879/167 | 1.510/**51** | 1.922/51 |
| D4 | 0.460/1000 | 1.449/562 | 2.666/1000 | 2.774/466 | 4.947/**19** | 4.682/44 |
| D5 | 0.913/1000 | 2.964/945 | 4.855/1000 | 4.627/438 | 8.902/**41** | 8.745/47 |
| D6 | 1.266/799 | 3.919/276 | 6.163/1000 | 7.010/271 | 12.04/**50** | 12.328/61 |

**Table 3**
Time Cost Per Iteration and Iteration Count Of MAE (SECONDS).

| Data set | Time cost per iteration/Iteration count | | | | | |
|---|---|---|---|---|---|---|
| | M1 | M2 | M3 | M4 | M5 | M6 |
| D1 | 0.640/841 | 1.988/248 | 3.208/1000 | 2.977/399 | 6.164/56 | 6.513/**51** |
| D2 | 1.135/892 | 3.648/324 | 5.519/1000 | 5.242/479 | 10.971/**57** | 10.022/61 |
| D3 | 0.178/998 | 0.544/367 | 0.819/1000 | 0.798/1000 | 1.487/**37** | 1.49/55 |
| D4 | 0.543/936 | 1.637/539 | 2.609/1000 | 3.033/453 | 5.109/68 | 4.767/**58** |
| D5 | 1.032/998 | 3.007/1000 | 4.612/1000 | 4.197/1000 | 9.353/**23** | 8.985/29 |
| D6 | 1.395/810 | 4.074/289 | 6.211/1000 | 6.035/453 | 11.868/**16** | 11.341/37 |

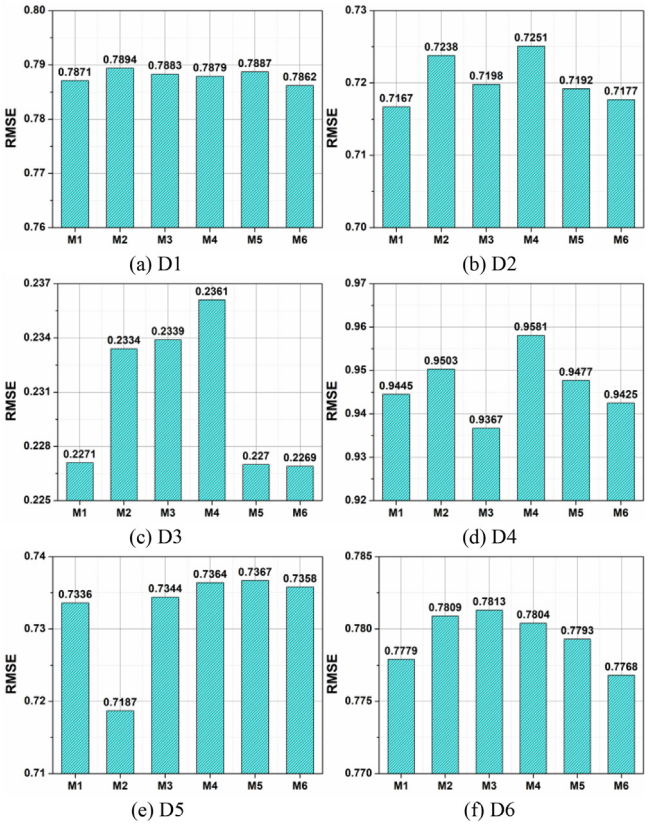



**Fig. 4.** Lowest RMSE of M1-M6.

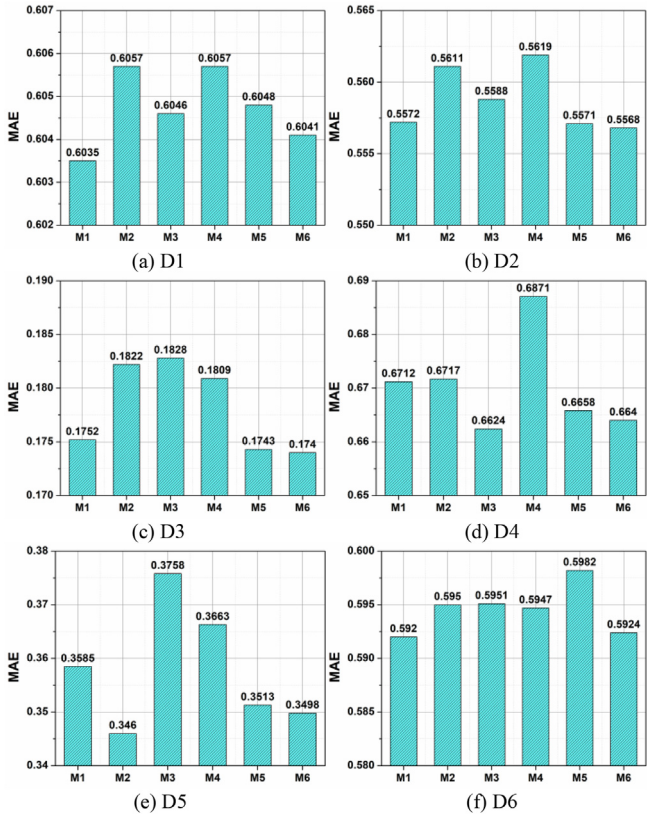



**Fig. 5.** Lowest MAE of M1-M6.

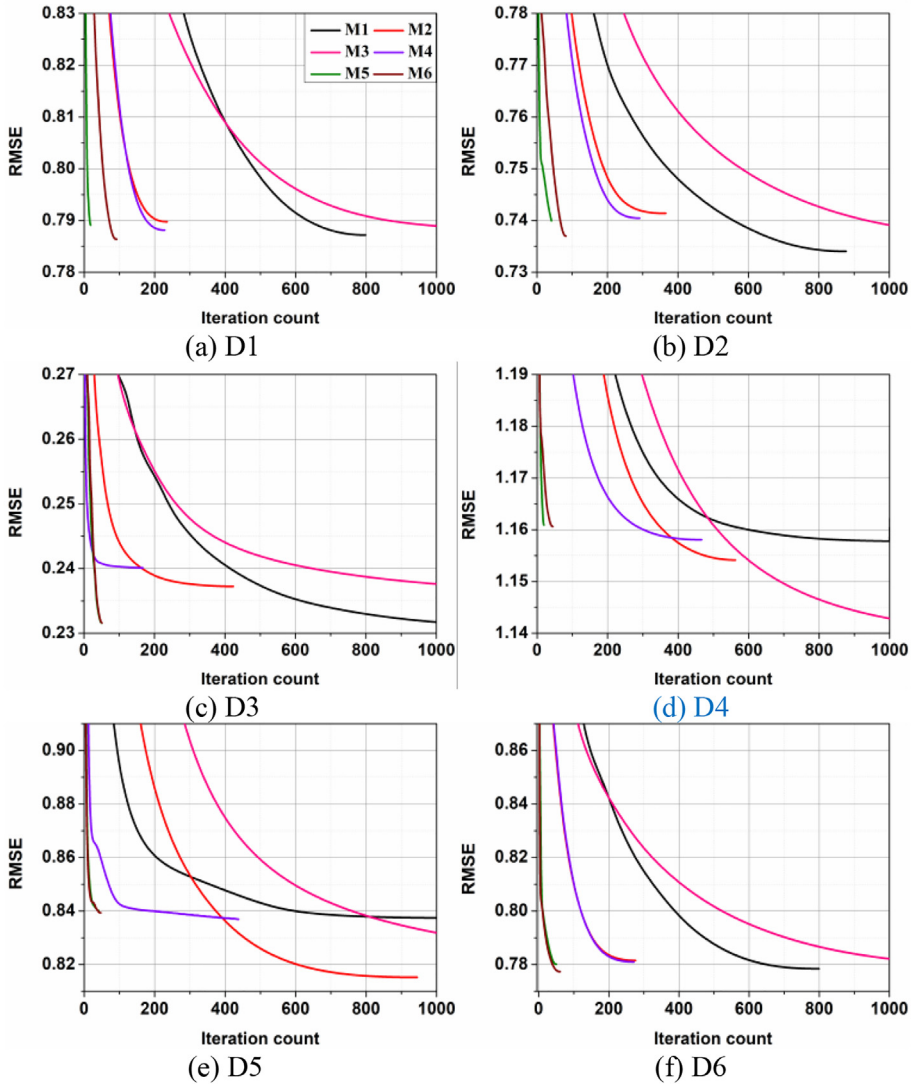



**Fig. 6.** The validation training process of RMSE on M1-M6. All panels share the legend in panel (a).

involved models, i.e., the preset iteration threshold is 1000, and the training process is terminated when the error difference between two iterations is less than $10^{-5}$.

**Model Settings.** To obtain a fair and objective comparison, we adopt the following measures settings:

- All the experiences base on $l_2$ norm regularization to prevent overfitting as the LFs of $X$ and $Y$ are initialized randomly generated arrays for eliminating the performance bias.
- We set the LF dimension $f$ = 20 for M1-M6 according to prior researches [28,45] to balance the representative learning ability and computational efficiency of each model.
- The dimension of velocity and position are set D = 2, the search space of the learning rate is [2–12, 2–8], and the search space of the regularization coefficient is [2–7, 2–3]. $r_1$ and $r_2$ are two uniformly distributed random numbers in a range of [0, 1] according to [32].

### 4.2. Effect of swarm size

Naturally, the number of swarm size determines the performance of the model. Hence, it appears necessary to validate its effects. Figs. 1–3 respectively show the swarm size's effects in the prediction accuracy and time cost of M5. From these results, we have the following findings:

- **HLFA's prediction accuracy is affected by the swarm size**. For instance, as depicted in Figs. 1 and 2, with the particle swarm size increasing, M5 achieves a higher prediction accuracy, e.g., M5′s RMSE on D1 are 0.7916, 0.787, 0.7866, 0.7852, 0.7854 and 0.7851 when $q$ increase 5 to 80, respectively. The lowest RMSE is 0.7851 with $q$ = 80 and the highest RMSE is 0.7916 with $q$ = 5, the gap reaches 0.82%. Although the RMSE diagrams of some datasets, especially on D3 and D6, do not seem to fit this pattern, the variation interval is subtle, which may cause by the randomness of PSO. M5′s MAE on D1 are 0.6108, 0.6048, 0.6045, 0.6044, 0.6044 and 0.6034 when increase 5 to 80, respectively. The lowest MAE is 0.6034 with $q$ = 80 and the highest MAE is 0.6108 with $q$ = 5, the gap reaches 1.21%. Therefore, the authenticity of conclusions from other datasets are not affected by it.
- **HLFA's time cost per iteration increases linearly as swarm size increases.** For instance, as depicted in Fig. 3(a), on D1 of RMSE, HLFA's time cost per iteration is 2.90, 5.89, 11.73, 23.40, 35.69 and 46.65 s as $q$ = 5, 10, 20, 40, 60 and 80,

respectively; on D1 of MAE in Fig. 3(b), HLFA's time cost per iteration is 2.90, 6.16, 12.12, 23.08, 34.67 and 45.41 s as $q$ = 5, 10, 20, 40, 60 and 80, respectively. Similar outcomes are found in other testing cases, which is consistent with HLFA's theoretical computational complexity.

Consequently, with the swarm size $q$ increases from 5 to 80, HLFA achieves a significant gain in prediction accuracy while further increase can gain minor gain. However, its computational cost keeps increasing linear with $q$. To well balance the prediction accuracy for missing data and computational cost, we set $q$ = 10 in the following experiments.

### 4.3. Performance comparison

In this part of the experiments, we compare the proposed HLFA with several state-of-the-art LFA models. We compare the prediction accuracy and time consumption of six models on six datasets. The details of compared models summarize as follows:

- M1: an SGD-based LFA model [1]. For this original LF model, we select the appropriate learning rate and regularization coefficient using a grid-search-method. For each learning rate and regularization coefficient, we run a full training process and check its performance, choose the best one, and compare it with other models.
- M2: An RMSprop-based LFA model [28]. It replaces the sum squares of the past stochastic gradients with their decaying average to make its learning rate self-adaptive.
- M3: An Ada-Delta-based LFA model [28]. Similar to M2, it also considers the past stochastic gradients at previous update points. However, it controls the historical learning information more carefully by recording its decaying square average of the past update increments.
- M4: an Adam-based LFA model [28], which adjusts the learning rate based on both the exponentially decaying square average and the exponentially decaying average of past stochastic gradients.
- M5: An HLFA model proposed in this study.
- M6: An HLFA model incorporating linearly decreasing inertia weight.

The detailed selection of learning rate $\eta$ and regularization coefficients $\lambda$ is shown in Table 1. The time cost per iteration of validation on RMSE and MAE and converging iteration counts of M1-M6 summarize in Tables 2 and 3, respectively. From them, we have the following finding:

- **HLFA can adaptively predict the missing data of an HiDS matrix with maintaining competitive prediction accuracy**. For instance, as shown in Fig. 4, on D1, D3, D4 and D6, HLFA's lowest RMSE are 0.7862, 0.2269, 0.9425 and 0.7768, about 0.11%, 0.09%, 0.2%, and 0.14% lower than that of M1, respectively. However, on D2 and D5, the RMSE is a little higher than that of M1′s about 0.14% and 0.3%. HLFA's lowest MAE in Fig. 5 on D2-D5 are 0.5568, 0.174, 0.664 and 0.3498, which are lower than M1′s MAE at 0.07%, 0.68%, 1.07%, and 2.43%, respectively. However, on D1 and D6, the MAE is a little higher than that of M1 about 0.1% and 0.007%. Notably, the RMSE and MAE of in D4 on M2, it shows excellent prediction accuracy ability. In conclusion, HLFA can maintain competitive prediction accuracy. Similar conclusions can be made on other datasets. This phenomenon means that effectively controlling the inertial weight and acceleration coefficients of particles in the search space can balance the global and local optimization capabilities of particles in the search space.
- **HLFA's convergence performance is very fast**. For instance, as shown in Table 2 and Fig 6, M1 takes 798, 877, 1000, 1000, 1000, 799 iterations to get the lowest RMSE on D1-D6, respectively. While M5 we proposed in this paper, it takes about 18, 41, 51, 19, 41, and 50 iterations to reach the lowest RMSE on D1-D6, respectively. This denotes that M5 reduces 97.74%, 95.32%, 94.9%, 98.1%, 95.9% and 93.74% converging iteration count than M1′s. As shown in Table 3 and Fig 6 M1 takes 841, 892, 998, 936, 998, 810 iterations to get the lowest MAE n Table 3 and Fig. 7 on D1-D6, while M5 takes 56, 57, 37, 68, 23 and 16 iterations to reach the lowest MAE. It reduces 93.34%, 93.61%, 96.29%, 92.73%, 97.69%, 98.02% converging iteration count than M1′s. Similar situations occur when compared with other models.
- **HLFA shows a good advantage in computational efficiency**. As depicted in Fig. 8, on D1, M5 consumes 120 s to achieve its lowest RMSE. This is 73.91% of 460 s by M1, 71.29% of 418 s by M2, 96.21% of 3169 s by M3, 84.50% of 774 s by M4, and 79.49% of 585 s by M6. Similar situations are found in other datasets according to Fig. 8. However, on MAE in Fig. 9 of D1, D2 and D4, M6′s computational efficiency is higher than that of M5′s, while it is still higher than other models. Remarkably, M6′s total time cost is slightly higher than that of M5. We have analyzed the causes of this phenomenon above.
- An HLFA model's time cost per iteration is much higher thn its peers. Note that an HLFA model maintains a swarm consisting of q particles describing the learning rate and regularization coefficient. In its single iteration, all these q particles are adopted to update desired LFs once to make the swarm evolve. Thus, an HLFA model's single iteration actually consists of q sub-iterations, where each one traverses on RK to update involved LFs. Hence, the proposed HLFA model's time cost per iteration is q times that of an SGD-based LFA model's. For

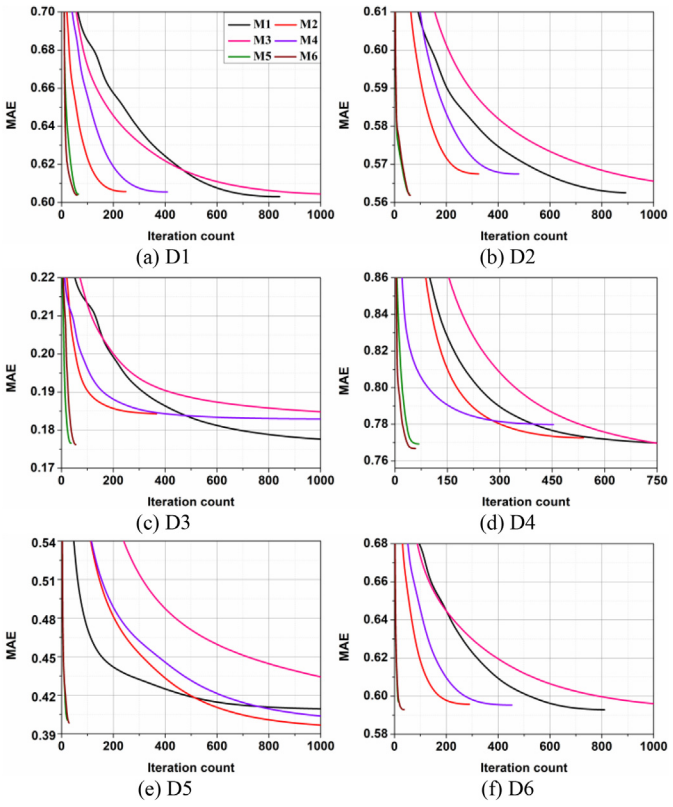



**Fig. 7.** The validation training process of MAE on M1-M6. All panels share the legend in panel (a).

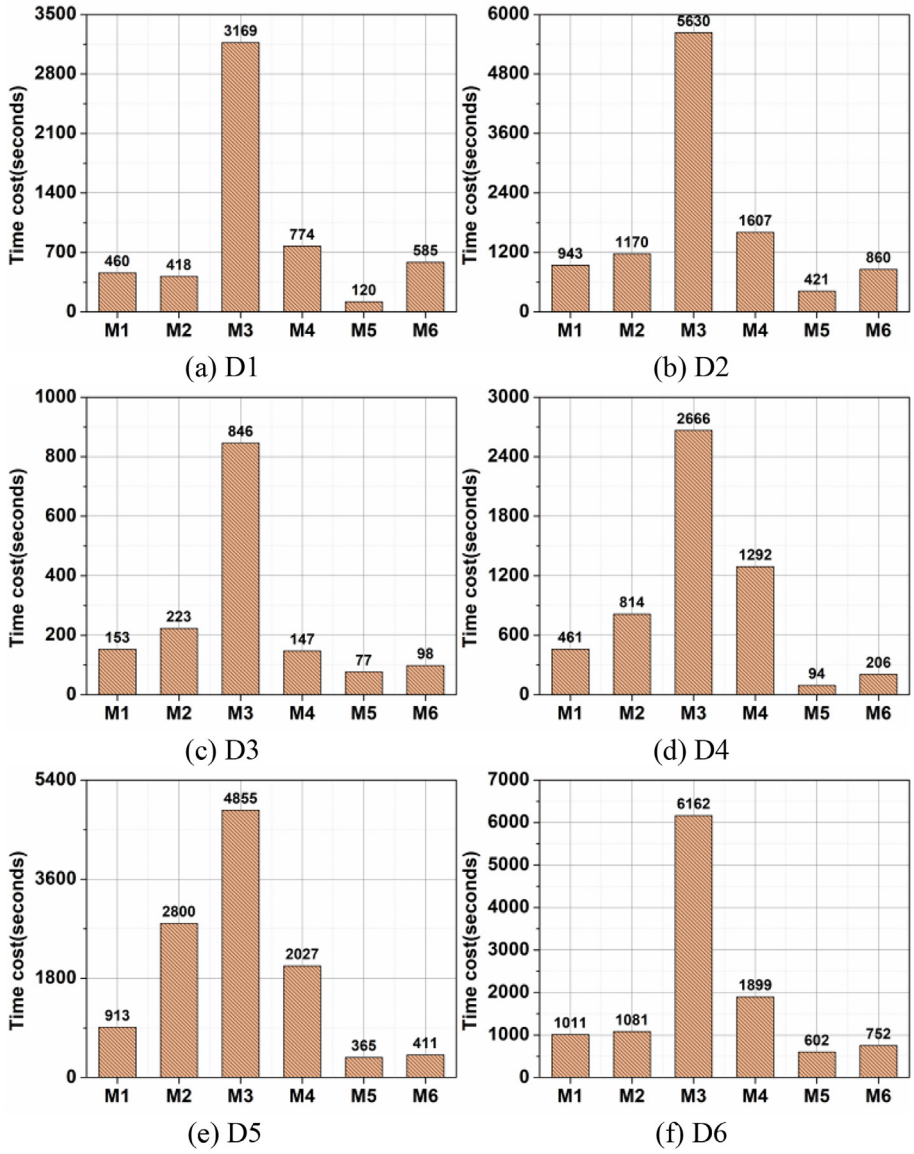



**Fig. 8.** Time cost of M1-M6 to converge in RMSE.

instance, on D2, M5 takes 10.268 s per iteration with q = 10, which is about 10 times that of 1.076 s by M1. Similar conclusions can be made on the other datasets. Nonetheless, from Table 2 and Fig. 7, we see that owing to the self-evolutionary $\eta$ and $\lambda$, the converging iteration count of HLFA decreases drastically. Hence, its computational efficiency is much higher than that of its peers, as depicted in Figs. 8 and 9.

- **Summary.** The prediction accuracy for missing data and computation efficiency of M5 are analyzed above. Based on the experimental results, we summarize that: 1) swarm size should be considered comprehensively from the aspects of prediction accuracy and computational efficiency; 2) HLFA greatly reduces the computational cost needed to search for the appropriate learning rate and regularization coefficient; 3) HLFA maintains a relatively competitive prediction accuracy.

## 5. Conclusions

This paper innovatively proposes an HLFA model that implements an efficient and self-adaptive selection of the optimal learning rate and regularization coefficient. Its efficiency is empirically proven on six HiDS matrices generated by industrial applications. With highly-efficient self-adaptation of its hyper-parameters, it also achieves highly-competitive prediction accuracy for the missing data of an HiDS matrix. Moreover, HLFA linearly decreases the inertia weight and changes two acceleration coefficients to avoid premature convergence, thereby further improving its prediction accuracy for missing data of an HiDS matrix.

This study adopts the standard PSO algorithm to implement the hyper-parameter evolution in an HLFA model. However, more efficient evolution algorithms like Genetic PSO [46], neighborhood fuzzy PSO [47], improved adaptive PSO [48], beetle antennae search [49], and artificial bee colony [50], are available. It is highly interesting to investigate their effects in an HLFA model. We plan to address this issue in the future.

## 6. Note

This research is supported in part by the National Natural Science Foundation of China under grants 61772493, in part by the Guangdong Province Universities and College Pearl River

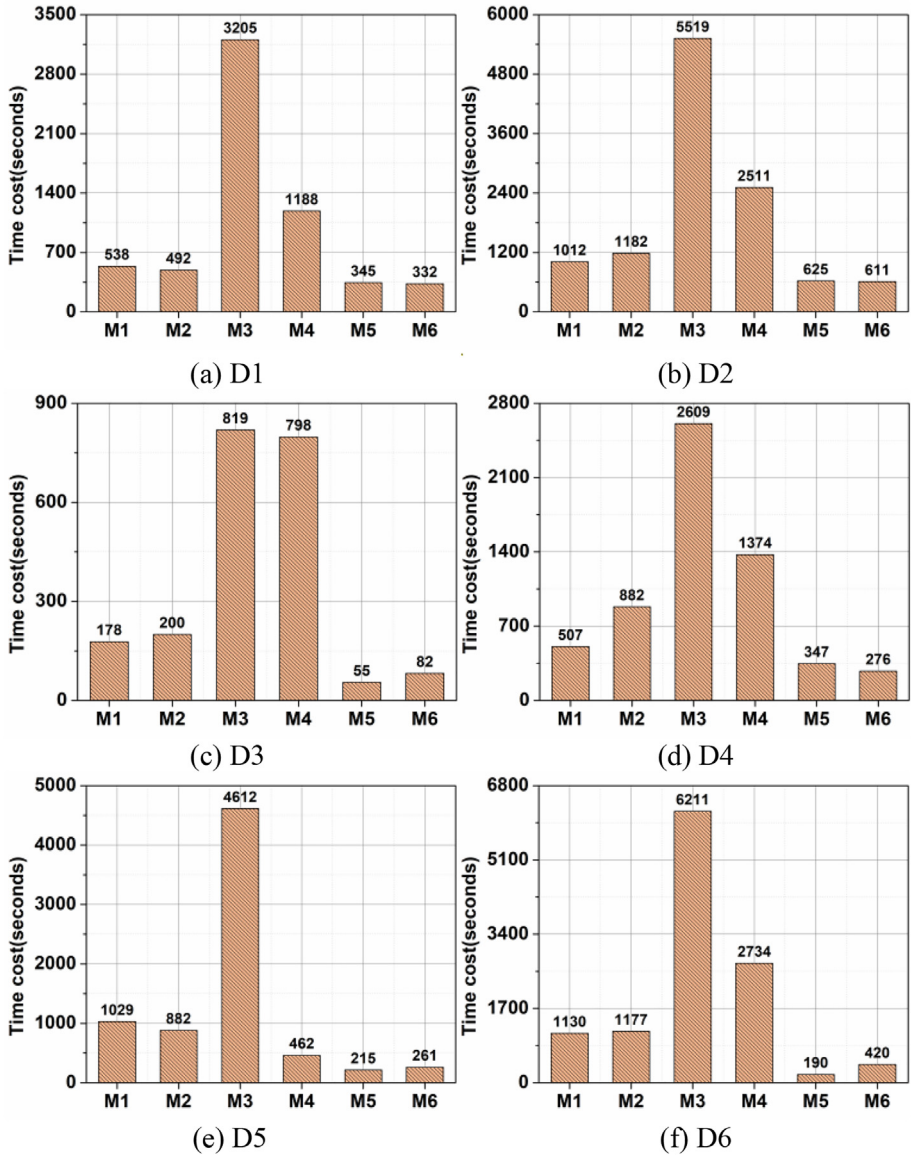



**Fig. 9.** Time cost of M1-M6 to converge in MAE.

Scholar Funded Scheme (2019), and in part by the Natural Science Foundation of Chongqing (China) under grant cstc2019j-cyjjqX0013. Jiufang Chen, Ye Yuan and Tao Ruan are co-first authors of this paper (*Corresponding author: Xin Luo*).

## CRediT authorship contribution statement

**Jiufang Chen:** Investigation, Methodology, Software. **Ye Yuan:** Investigation, Methodology, Software. **Tao Ruan:** Visualization. **Jia Chen:** Visualization. **Xin Luo:** Conceptualization, Methodology.

## Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

## References

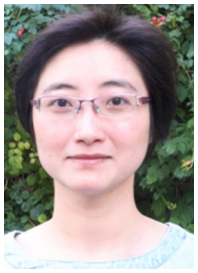
[1] Y. Koren, R. Bell, C. Volinsky, Matrix factorization techniques for recommender systems, IEEE Comput. 42 (8) (Aug. 2009) 30–37.
[2] A. Gatzioura, M. Sanchez-Marre, A case-based recommendation approach for market basket data, IEEE Intell. Syst. 30 (1) (Jan. 2015) 20–27.
[3] G. Adomavicius, A. Tuzhilin, Toward the next generation of recommender systems: a survey of the state-of-the-art and possible extensions, IEEE Trans. Knowl. Data Eng. 17 (6) (2005) 734–749.
[4] X. Luo, M.C. Zhou, S. Li, Y.N. Xia, Z.H. You, Q.S. Zhu, H. Leung, An efficient second-order approach to factorizing sparse matrices in recommender systems, IEEE Trans. Ind. Inf. 11 (4) (2015) 946–956.
[5] B. Sarwar, G. Karypis, J. Konstan, J. Reidl, Item-based collaborative-filtering recommendation algorithms, in: Proc. of the 10th Int. Conf. World Wide Web, 2001, pp. 285–295.
[6] J.B. Zhang, Z.Q. Lin, B. Xiao, C. Zhang, An optimized item-based collaborative filtering recommendation algorithm, in: IEEE Int. Conf. on Network Infrastructure and Digital Content, Nov. 2009, pp. 414-418.
[7] R. Andersen, C. Borgs, J. Chayes, U. Feige, A. Flaxman, A. Kalai, V. Mirrokni, M. Tennenholtz, Trust-based recommendation systems: an axiomatic approach, in: Proc. of the 17th Int. Conf. on World Wide Web ACM, Apr. 2008, pp. 199–208.
[8] H. Ma, I. King, M.R. Lyu, Learning to recommend with social trust ensemble, in: Proc. of the 32nd Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 2009, pp. 203–210

[9] X. Luo, M.C. Zhou, Y.N. Xia, Q.S. Zhu, An efficient non-negative matrix-factorization-based approach to collaborative filtering for recommender systems, IEEE Trans. Ind. Inf. 10 (2) (2014) 1273–1284.
[10] X. Luo, J.P. Sun, Z.D. Wang, S. Li, M.S. Shang, Symmetric and non-negative latent factor models for undirected, high dimensional and sparse networks in industrial applications, IEEE Trans. Ind. Inf. 13 (6) (2017) 3098–3107.
[11] G. Takács, I. Pilászy, B. Németh, D. Tikk, Scalable collaborative filtering approaches for large recommender systems, J. Mach. Learn. Res. 10 (2009) 623–656.
[12] N. Barbieri, G. Manco, E. Ritacco, Probabilistic approaches to recommendations, Synth. Lect. Data Min. Knowl. Discovery 5 (2) (2014) 1–197.
[13] F. Shang, L.C. Jiao, F. Wang, Graph dual regularization non-negative matrix factorization for co-clustering, Pattern Recogn. 45 (6) (2012) 2237–2250.
[14] Y. Nishioka, K. Taura, Scalable task-parallel SGD on matrix factorization in multicore architectures, IEEE International Parallel and Distributed Processing Symposium Workshop, 2015, pp. 1178–1184.
[15] N. Feng, R. Benjamin, R. Christopher, J.W. Stephen, Hogwild!: a lock-free approach to parallelizing stochastic gradient descent, Adv. Neural Inf. Process. Syst. 24 (2011) 693–701.
[16] Z. Lu, H. Bai, B. Sun, A gradient-based algorithm for optimizing sensing matrix with normalization constraint, in: Proc. of the IEEE 11th Int. Conf. on Industrial Electronics and Applications, 2016, pp. 2376–2380
[17] J. Mohsen, E. Martin, A matrix factorization technique with trust propagation for recommendation in social networks in: Proc. of the Fourth ACM Conf. on Recommender Systems, 2010, pp. 135–142.
[18] H. Li, K.L. Li, J.Y. An, K.Q. Li, MSGD: a novel matrix factorization approach for large-scale collaborative filtering recommender systems on GPUs, IEEE Trans. on Parallel and Distributed Systems, vol. 29, no. 7, 2018, pp. 1530–1544.
[19] J. Duchi, E. Hazan, Y. Singer, Adaptive subgradient methods for online learning and stochastic optimization, J. Mach. Learn. Res. 12 (7) (2011) 2121–2159.
[20] M.D. Zeiler, "ADADELTA: an adaptive learning rate method, Comput. Sci. (2012).
[21] D. Kingma, J. Ba, Adam: a method for stochastic optimization, in: Proc. of the 2015 Int. Conf. for Learning Representations, 2015.
[22] U. Andayani, E.B. Nababan, B. Siregar, M.A. Muchtar, T. Hamonangan, I. Siregar, Optimization backpropagation algorithm based on Nguyen-Widrom adaptive weight and adaptive learning rate, in: Proc. of the 2017 Int. Conf, on Industrial Engineering and Applications, Nagoya, Japan, 2017, pp. 363–367.
[23] M. Wang, C. Wang, Learning from adaptive neural dynamic surface control of strict-feedback systems, IEEE Trans. Neural Networks Learn. Syst. 26 (6) (2015) 1247–1259.
[24] S.L. Dai, C. Wang, M. Wang, Dynamic learning from adaptive neural network control of a class of non-affine nonlinear systems, IEEE Trans. on Neural Networks Learn. Syst. 25 (1) (2014) 111–123.
[25] Y. Shi, R.C. Eberhart, Empirical study of particle swarm optimization, in: Proc. of the 1999 IEEE Congress on Evolutionary Computation, 1999, pp. 1945–1950
[26] R. Chenga, Y.C Jin, A social learning particle swarm optimization algorithm for scalable optimization, Inf. Sci. 291 (2015) 43–60.
[27] Y.L. Cao, H. Zhang, W.F. Li, M.C. Zhou, Y. Zhang, W.A. Chaovalitwongse, Comprehensive learning particle swarm optimization algorithm with local search for multimodal functions, IEEE Trans. Evol. Comput. 23 (4) (2019).
[28] X. Luo, D.X. Wang, M.C. Zhou, H.Q. Yuan, Latent factor-based recommenders relying on extended stochastic gradient descent algorithms, IEEE Trans. Syst. Man Cybern. Syst. DOI 10.1109/TSMC.2018.2884191.
[29] X. Luo, M.C. Zhou, Y.N. Xia, Q.S. Zhu, A.C. Ammari, A. Alabdulwahab, Generating highly accurate predictions for missing QoS-data via aggregating non-negative latent factor models, IEEE Trans. Neural Networks Learn. Syst. 27 (3) (2016) 579–592.
[30] X. Luo, Z.G. Liu, S. Li, M.S. Shang, Z.D. Wang, A fast non-negative latent factor model based on generalized momentum method, IEEE Trans. System Man Cybern. Syst. (2018), https://doi.org/10.1109/TSMC.2018.2875452, Nov.
[31] J. Kennedy, Particle swarm optimization, Encyclopedia Mach. Learn. (2010) 760–766.
[32] R.C. Eberhart, Y.H. Shi, Particle swarm optimization: developments, applications and resources, in: Proc. of the 2001 IEEE Congress on Evolutionary Computation, 2001, pp. 81–86
[33] J. Liang, A. Qin, P. Suganthan, S. Baskar, Comprehensive learning particle swarm optimizer for global optimization of multimodal functions, IEEE Trans. Evol. Comput. 10 (3) (2006) 281–295.
[34] N. Zeng, Z. Wang, H. Zhang, F.E. Alsaadi, A novel switching delayed PSO algorithm for estimating unknown parameters of lateral flow immunoassay, Cogn. Comput. 8 (2) (2016) 143–152.
[35] R.C. Eberhart, Y. Shi, Comparing inertia weights and constriction factors in particle swarm optimization, in: Proc. of the 2000 IEEE Congress on Evolutionary Computing, La Jolla, CA, USA, vol. 1, 2000, pp. 84–88.
[36] J.A. Konstan, B.N. Miller, D. Maltz, J.L. Herlocker, L.R. Gordon, J. Riedl, GroupLens: applying collaborative filtering to Usenet news, Commun. ACM 40 (3) (1997) 77–87.
[37] X. Luo, H. Wu, H.Q. Yuan, M.C. Zhou, Temporal pattern-aware QoS prediction via biased non-negative latent factorization of tensors, IEEE Trans. Cybern. 50 (5) (2020) 1798–1809.
[38] P. Massa, P. Avesani, Trust-aware collaborative filtering for recommender systems, Lect. Notes Comput. Sci. 3290 (2004) 492–508.
[39] A. Ratnaweera, S.K. Halgamuge, H.C, Watson, Self-organizing hierarchical particle swarm optimizer with time-varying acceleration coefficients, IEEE Trans. Evol. Comput. 8 (3) (2004) 240–255.
[40] Y.F. Li, A. Ngom, Versatile sparse matrix factorization and its applications in high-dimensional biological data analysis, in: Proc of the 8th Int. Conf. on Pattern Recognition in Bioinformatics Springer, 2013, pp. 91–101
[41] Y. Yuan, Q. He, X. Luo, M.S. Shang, A multilayered-and- randomized latent factor model for high-dimensional and sparse matrices, IEEE Trans. Big Data, DOI: 10.1109/TBDATA.2020.2988778.
[42] Y.C. Jing, X.Z. Zhang, L.F. Wu, J.Q. Wang, Z.M. Feng, D. Wang, Recommendation on Flickr by combining community user ratings and item importance, in: Proc. of 2012 Int. Conf. on Multimedia and Expo, 2012, pp. 1–6
[43] T. Schaul, Y. Lecun, Adaptive learning rates and parallelization for stochastic, sparse, non-smooth gradients, Comput. Sci. 35 (8) (2013) 493–531.
[44] R. Salakhutdinov, A. Mnih, Probabilistic matrix factorization, Adv. Neural Inf. Process. Syst. 20 (2008) 1257–1264.
[45] X. Luo, M.C. Zhou, S. Li, Y. Xia, Z.H. You, Q.S. Zhu, H. Leung, Incorporation of efficient second-order solvers into latent factor models for accurate prediction of missing QoS data, IEEE Trans. Cybern. 48 (4) (2018) 1216–1228.
[46] Y.J. Gong, J.J. Li, Y.C. Zhou, Y. Li, H.S. Chung, Y.H. Shi, J. Zhang, Genetic learning particle swarm optimization, IEEE Trans. Syst. Man Cybern. 46 (10) (2016) 2277–2290.
[47] P. Roy, G.S. Mahapatra, K.N. Dey, Forecasting of software reliability using neighborhood fuzzy particle swarm optimization based novel neural network, IEEE/CAA J. Autom. Sin. 6 (6) (2019) 1365–1383.
[48] W. Deng, H.M. Zhao, X.H. Yang, J.X. Xiong, M. Sun, B. Li, Study on an improved adaptive PSO algorithm for solving multi-objective gate assignment, Appl. Soft Comput. 59 (2017) 288–302.
[49] X. Y Jiang, S. Li, Beetle antennae search without parameter tuning (BAS-WPT) for multi-objective optimization, arXiv:1711.02395v1, 2017.
[50] F. Kang, J. Li, Z. Ma, Rosenbrock artificial bee colony algorithm for accurate global optimization of numerical functions, Inf. Sci. 181 (16) (2011) 3508–3531.

**Jiufang Chen** received the B.S. degree in computer science and technology from China West Normal University, Nanchong, China, in 2017. She is currently pursuing her M.S. degree at China West Normal University, Nanchong, Sichuan, China. She is also a visiting student at Chongqing Institute of Green and Intelligent Technology, Chinese Academy of Sciences, Chongqing China. Her research interests focus on Incomplete Data Analysis and Evolutionary Computing.

**Ye Yuan** 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, Chengdu, China, in 2010 and 2013, respectively. He is currently working toward the Ph.D. degree in computer science from Chongqing Institute of Green and Intelligent Technology, Chinese Academy of Sciences, Chongqing, China. His research interests include data mining and intelligent computing.

**Tao Ruan** received the M.A. degree from the China University of Geosciences, Beijing, China, in 2013. Then she joined the China Patent Information Center, is now an assistant researcher mainly focusing on patent processing in the field of the computer, and is also responsible for the processing of PCT and PRI, processing of applicators and inventors and the like. Her research interests include big data analysis, network security and domestic and foreign patent translation in the fields of computer, electricity, machinery and the like.



**Jia Chen** received the B.S. degree in electronic circuit and system from Beihang University, Beijing, China, in 2004, and the Ph.D. degree in computer science from Beihang University, Beijing, China, in 2019. Since July 2020, she has been a faculty with the School of Cyber Science and Technology, Beihang University. Her research interests are in cyberspace security, recommender system and latent factor model.

**Xin Luo** 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 100 papers (including over 50 IEEE TRANSACTIONS papers) in the above areas. Dr. Luo was a recipient of the Hong Kong Scholar Program jointly by the Society of Hong Kong Scholars and China Post-Doctoral 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 and Neurocomputing.