BIG DATA MINING AND ANALYTICS
ISSN 2096-0654 03/06 pp211–221
Volume 1, Number 3, September 2018
DOI: 10.26599/BDMA.2018.9020019

A Novel Deep Hybrid Recommender System Based on Auto-encoder with Neural Collaborative Filtering

Yu Liu, Shuai Wang, M. Shahrukh Khan, and Jieyu He*

Abstract: Due to the widespread availability of implicit feedback (e.g., clicks and purchases), some researchers have endeavored to design recommender systems based on implicit feedback. However, unlike explicit feedback, implicit feedback cannot directly reflect user preferences. Therefore, although more challenging, it is also more practical to use implicit feedback for recommender systems. Traditional collaborative filtering methods such as matrix factorization, which regards user preferences as a linear combination of user and item latent vectors, have limited learning capacities and suffer from data sparsity and the cold-start problem. To tackle these problems, some authors have considered the integration of a deep neural network to learn user and item features with traditional collaborative filtering. However, there is as yet no research combining collaborative filtering and content-based recommendation with deep learning. In this paper, we propose a novel deep hybrid recommender system framework based on auto-encoders (DHA-RS) by integrating user and item side information to construct a hybrid recommender system and enhance performance. DHA-RS combines stacked denoising auto-encoders with neural collaborative filtering, which corresponds to the process of learning user and item features from auxiliary information to predict user preferences. Experiments performed on the real-world dataset reveal that DHA-RS performs better than state-of-the-art methods.

Key words: hybrid recommender system; neural collaborative filtering; auto-encoder; implicit feedback

1 Introduction

The Recommender System (RS) is a tool and technique that helps people to attain content on the basis of their interest and thereby save a lot of time[1]. Many websites, like Amazon[2], Netflix[3], and other social networks, have adopted recommender systems. Collaborative filtering is one of the key techniques used in personalized recommender systems[1,4–9]. The essence of collaborative filtering is to reveal the relationship between users and items based on available user-item information, such as user ratings of items. Matrix Factorization (MF), one of the most popular collaborative filtering techniques, projects users and items into a shared latent space, and uses a latent feature vector to represent either a user or an item. Subsequently, the interaction of the user with the item is modeled as the inner product of the latent vectors. The final goal of a recommender system is to provide a list of items that approximate user preferences. However, collaborative filtering faces two problems. The first is that recommender systems mainly rely on implicit feedback, which is sparse and cannot truly reflect user preferences[7,10]. This limits the performance of recommender systems. The second problem is that traditional collaborative filtering uses linear models to learn user-item relations, which restricts the recommendation performance. For example, in

• Yu Liu, Shuai Wang, M. Shahrukh Khan, and Jieyue He are with School of Computer Science and Engineering, and also with MOE Key Laboratory of Computer Network and Information Integration, Southeast University, Nanjing 211189, China. E-mail: 220153311@seu.edu.cn; 53263887@qq.com; mushahrukhkhan@outlook.com; jieyuehe@seu.edu.cn.

∗ To whom correspondence should be addressed.

Manuscript received: 2018-01-31; accepted: 2018-03-04

Refs. [11–13], to generate a recommendation, the authors used MF to learn user and item latent vectors by decomposing a user rating matrix into user and item latent vectors that have high relevance. MF is a process of reducing dimension, which inevitably results in loss of user-item interactions. Hence, traditional collaborative filtering methods have difficulty in improving the precision of recommendations.

Researchers have proposed various approaches to the development of collaborative-filtering-based recommender systems. Some researchers have explored the application of Deep Neural Networks (DNN) to recommender systems by designing collaborative filtering based on neural networks. DNNs have demonstrated their learning capacities in many tasks[14], such as computer vision and speech recognition. Salakautdinov et al.[4] used Restricted Boltzmann Machines (RBMs) in recommender systems and designed two-layer RBMs to model explicit item ratings given by users. Georgiev and Nakov[15] extended the RBMs model to deal with real value ratings and, further, combined original training data with data generated by bootstrapping to improve model performance. Ouyang et al.[5] attempted to explore high-level user-item relations using an Auto-Encoder (AE) and proposed a three-layer auto-encoder to model users explicit ratings on items. Considering that collaborative filtering is a process of filling the rating matrix, Wu et al.[16] proposed the Collaborative Denoising Auto-Encoder (CDAE), which utilizes a Denoising Auto-Encoder (DAE)[17] to perform collaborative filtering. CDAE assumes user ratings to be corrupt and the recommendation process involves reconstructing original users ratings. It learns latent user-item relations by minimizing reconstruction error. He et al.[6] proposed a Neural Collaborative Filtering (NCF) framework for implicit feedback recommendations. The authors utilized a neural network to learn the latent vectors of users and items in a way similar to MF. NCF is flexible and can be easily extended to learn the non-linear relations between users and items and it also demonstrates that MF is a special model case. To be clear, we use NCF to refer to the framework proposed in Ref. [6], and the term neural collaborative filtering to refer to collaborative filtering based on neural networks. All these works have attempted to improve the performance of recommender systems by modeling user-item relations using different neural networks that have more powerful learning capabilities.

Most studies have focused on user ratings, but rating information alone cannot fully reveal user-item relations. Additionally, the sparsity of user ratings lowers the performance of most CF-based methods. To improve performance, some researchers have used traditional collaborative filtering to integrate user and item side information to then construct hybrid recommender systems. Popular neural networks such as the auto-encoder and its variants, like the DAE, can effectively learn useful structures and extract representative features from input data. Furthermore, some authors have employed neural networks to extract user and item features from side information to improve the performance of traditional collaborative filtering. Wang et al.[10] proposed Collaborative Deep Learning (CDL), in which a DAE learns item features that act as item latent vectors for MF-CDL, which decomposes a rating matrix and learns latent factors in one model. As MF involves learning both user and item latent vectors, Li et al.[7] proposed mDA-CF, which extends the CDL model by adding the user latent vectors learned from user side information by a DAE. mDA-CF combines extracted user and item features and recommendations in a joint framework. He and McAuley[18] noted that visual information about items plays an important role in real-world recommendations. So, the authors developed the visual Bayesian personalized ranking model, which employs a Convolutional Neural Network (CNN)[19,20] to extract features from item images. These methods take advantage of neural networks to improve the performances of collaborative filtering, but the core of collaborative filtering is still MF.

The authors of the above mentioned studies attempted to improve recommendation performance either by introducing side information via neural networks or designing novel collaborative filtering models based on neural networks. However, there has been no work that uses neural networks for both tasks. In this paper, we present a top-K deep hybrid recommender system framework based on auto-encoders (DHA-RS). DHA-RS uses Stacked Denoising Auto-Encoder (SDAE), which is an extension of the stacked auto-encoder[21] that incorporates noise and SDAEs to form a deep network. This network extracts user and item features from auxiliary information as user and item latent vectors for neural collaborative filtering. In addition, we propose two models based on the DHA-RS framework: GMF++ (Generalized Matrix Factorization++) and MLP++ (Multi-Layer

Perceptron++), both of which integrate user and item side information. These models mainly differ in their settings of neural collaborative filtering: in GMF++, the latent factors generate the element-wise product and then fully connects to the neural network, whereas MLP++ concatenates the latent factors and then connects to the neural network. The experimental results on the real dataset MovieLens-1M reveal that DHA-RS has better performance and more extensibility than the existing frameworks with respect to top-K recommendation problems. Although the performance of MLP++ is not as good as GMF++, subsequent results have shown that with an increasing number of hidden layers, MLP++ performs better, which suggests that using deep neural networks is effective for collaborative recommendations and that MLP++ has room for more improvement.

The rest of the paper is organized as follows. We introduce our recommendation framework, DHA-RS, in detail in Section 2 and discuss our experiments in Section 3. In the last section, we summarize our work and briefly introduce our plans for future work.

2 Method

In this section, we first present the general DHA-RS framework and demonstrate how the SDAEs and neural collaborative filtering are integrated into one model. Then, we explain how the SDAEs extract user and item features from auxiliary information. Lastly, we present the two models, GMF++ and MLP++, which use different settings in the neural collaborative filtering module. Table 1 lists the symbols used in our approach. Next, we describe the general structure of the DHA-RS.

Table 1 Frequently used symbols.

Notation	Description
U	User set
I	Item set
p	Dimension of user features
q	Dimension of item features
L	Number of layers of SDAE
$\boldsymbol{R} \in \mathbb{R}^{\lvert U\rvert \times \lvert I\rvert}$	Rating matrix
$\boldsymbol{X} \in \mathbb{R}^{\lvert U\rvert \times p}$	Auxiliary information of users
$\boldsymbol{Y} \in \mathbb{R}^{\lvert I\rvert \times q}$	Auxiliary information of items
$\boldsymbol{p}_u$	User latent vector
$\boldsymbol{q}_i$	Item latent vector
$\boldsymbol{W}_l$	Weight matrix of layer-l in SDAE
$\boldsymbol{b}_l$	Bias vector of layer-l in SDAE

2.1 DHA-RS: A general framework

Let U and I denote the sets of users and items and the users binary rating matrix $\boldsymbol{R} = [r_{ui}]^{\lvert U\rvert \times \lvert I\rvert}$, whose element r_{ui} represents whether user u rates item i, and if user u has a rating record for item i, then $r_{ui} = 1$, otherwise, $r_{ui} = 0$. The goal of recommendation for implicit feedback is to generate an item list that reflects the users preference.

$$r_{ui} = \begin{cases} 1, & u \text{ rates } i; \\ 0, & \text{otherwise} \end{cases} \tag{1}$$

Actually, it is easy to find subsidiary information, such as user age, occupation, gender, and item plot and genre. As discussed above, subsidiary information can help improve the performance of recommender systems. For a user side-information matrix $\boldsymbol{X}$ and an item side-information matrix $\boldsymbol{Y}$, we use SDAEs to learn the user and item features by minimizing the reconstruction errors of the output and the original user and item features. The learned features are stored in the middle hidden-layer vector. Unlike traditional collaborative filtering, DHA-RS uses neural collaborative filtering to explore user-item relations, which has a more powerful learning capability.

Figure 1 shows a schematic of the DHA-RS framework. The DHA-RS has two main components: (1) the feature-extraction and ID-embedding module and (2) the neural collaborative filtering module. For feature extraction and ID embedding, DHA-RS uses two SDAEs to learn the user and item features, respectively. User (item) input $\boldsymbol{X}$ ($\boldsymbol{Y}$) is input to the SDAE to extract user (item) latent features. The user (item) ID is initiated by one-hot encoding, which is sparse, and is then embedded into a dense vector. The

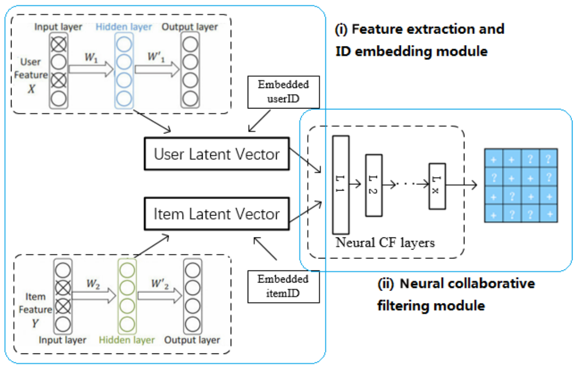


Fig. 1 Deep hybrid recommender system based on auto-encoders model.

extracted user (item) feature vector and embedded user (item) ID are then concatenated to form the user (item) latent vector. Later, the user and item latent vectors are fed into a neural network to learn the user-item relation and finally generate a predicted rating.

2.2 Feature extraction by stacked denoising auto-encoder

The AE is a neural network that learns an identity function, which then generates an output that approximates the input. Auto-encoders are often used to learn features from a data set. The DAE is a variant of the AE, which is designed to tackle the noise problem. To learn robust features, some of the input data is stochastically set to zero. In another words, the DAE tries to reconstruct the original input using a corrupt version of it. The SDAE is a deep network formed by stacking multiple DAEs. Figure 2 shows an SDAE with four layers. This model is structurally symmetric. The process of going from layer-0 to layer-2 can be regarded as encoding, whereby noisy input $\boldsymbol{X}_0$ is abstracted to a high-level presentation $\boldsymbol{X}_2$. The process of going from layer-2 to layer-4 can be treated as decoding, whereby $\boldsymbol{X}_2$ is reconstructed to its original clean presentation. The values of the neurons in the middle layer are the extracted features of the input data. SDAE is robust to noise and can learn more stable features due to the stochastic addition of noise to the original input.

In an SDAE with L layers, $\boldsymbol{X}_l$ represents the output of layer-l. For the original input $\boldsymbol{X}_c$, the elements of $\boldsymbol{X}_c$ are stochastically set to zero, and the noisy data are fed into the SDAE. $\boldsymbol{W}_l$ and $\boldsymbol{b}_l$ represent the weight and bias of layer-l. The output of each layer is generated by the following steps:

- The weight parameter $\boldsymbol{W}_l$ is generated by $N(0, \lambda_W^{-1}\boldsymbol{I})$,
- The bias parameter $\boldsymbol{b}_l$ is generated by $N(0, \lambda_b^{-1}\boldsymbol{I})$,

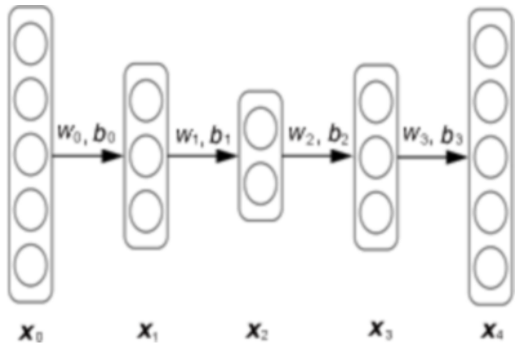


Fig. 2 A four-layer stacked denoising auto-encoder.

- The output $\boldsymbol{X}_l$ is generated by $N(\sigma(\boldsymbol{X}_{l-1}\boldsymbol{W}_l + \boldsymbol{b}_l), \lambda_X^{-1}\boldsymbol{I})$,

where λ_W and λ_b are hyper parameters used in parameter initialization. SDAE learns the user and item features based on auxiliary information by minimizing the reconstruction error and using the middle layer output as the extracted user and item feature. The reconstruction error can be described as follows:

$$L = \|X_c - X_L\|_{\mathrm{F}}^2 + \lambda_\Omega \|\boldsymbol{\Omega}\|_{\mathrm{F}}^2 \tag{2}$$

where $\boldsymbol{\Omega}$ denotes the parameter of the model, and λ_Ω is its regularization term.

Typically, auxiliary information consists of user and item attributes, such as age, occupation, plot, and genre. For example, the text information of a movie includes the title, genre, and plot. This information can be dealt with using some natural language processing methods to transform them into a bag of words that can be used as SDAE input. Similarly, DHA-RS uses SDAEs to learn user and item features from side information. By incorporating user and item side information into the collaborative filtering process, the relations between the user and item are enriched.

2.3 Generalized matrix factorization and multi-layer perceptron with side information

Within the DHA-RS framework and inspired by the NCF framework[6], in this paper, we propose two models: GMF++ and MLP++ with side information. These models differ in that the neural collaborative filtering process uses different computational methods and layers. The collaborative filtering module of GMF++ uses a computational method similar to the inner product of MF, whereas the collaborative filtering of MLP++ concatenates the embedded user and item latent vectors and then learns the user-item relations by a multi-layer neural network. Both models integrate side information to improve model performance.

2.3.1 GMF++

GMF++ first uses SDAEs to extract user and item features from auxiliary information, the details of which are described above in Section 2.2. Then, the extracted user and item features are concatenated with the embedded ID to obtain the respective latent vectors. In collaborative filtering, GMF++ computes the element-wise product of the user and item latent vectors and outputs the calculated vectors to a fully connected neural layer. The element-wise products of the user and item latent vectors can be defined as follows:

$$\phi_1(\boldsymbol{p}_u, \boldsymbol{q}_i) = \boldsymbol{p}_u \cdot \boldsymbol{q}_i \tag{3}$$

where $\cdot$, $\boldsymbol{p}_u$, and $\boldsymbol{q}_i$ denote the element-wise product of the vectors, user latent vector, and item latent vector, respectively. GMF++ then projects the vectors to the output layer:

$$\hat{r}_{ui} = a_{\text{out}}(\boldsymbol{h}^{\mathrm{T}}(\boldsymbol{p}_u \cdot \boldsymbol{q}_i)) \tag{4}$$

where a_{out} denotes the activation function, $\boldsymbol{h}^{\mathrm{T}}$ denotes the weights of the output layer, and $\hat{r}_{ui}$ denotes the predicted score of interaction r_{ui}. GMF is equivalent to MF[6], especially when a_{out} is an identity function and $\boldsymbol{h}$ is a vector with a value of $\mathbf{1}$. Within the DHA-RS, a_{out} can be a non-linear activation function and $\boldsymbol{h}$ can be learned from training data, therefore, it has greater learning capability than MF.

To differentiate from GMF proposed in Ref. [6], we refer to this model as GMF++. The original GMF relies only on implicit feedback, and the sparsity of users preferences regarding implicit feedback limits the GMF performance. GMF++ adds auxiliary information to the model and takes advantage of SDAEs to obtain user and item latent vectors, therefore achieving better performance.

2.3.2 MLP++

MLP++ and GMF++ use the same method to extract user and item features from auxiliary information. However, MLP++ takes a different learning strategy in the neural collaborative filtering module. Instead of treating user and item latent vectors by MF, MLP++ concatenates a learned user latent vector $\boldsymbol{p}_u$ and item latent vector $\boldsymbol{q}_i$, then adopts a multi-layer perceptron in a collaborative filtering module to learn high-level user-item relations. The collaborative filtering aspect of MLP++ can be defined as follows:

$$\begin{aligned} z_1 &= \phi_1(\boldsymbol{p}_u, \boldsymbol{q}_i) = \begin{bmatrix} \boldsymbol{p}_u \\ \boldsymbol{q}_i \end{bmatrix}, \\ \phi_2(z_1) &= a_2(\boldsymbol{W}_2^{\mathrm{T}} z_1 + \boldsymbol{b}_2), \\ &\cdots \\ \phi_L(z_{L-1}) &= a_L(\boldsymbol{W}_L^{\mathrm{T}} z_{L-1} + \boldsymbol{b}_L), \\ \hat{r}_{ui} &= \sigma(\boldsymbol{h}^{\mathrm{T}} \phi_L(z_{L-1})) \end{aligned} \tag{5}$$

where $\boldsymbol{W}_x$, $\boldsymbol{b}_x$, and a_x denote the weights, bias vector, and activation function for the layer-x, respectively, and "[ ]" denotes the concatenating operation. To differentiate from MLP proposed in Ref. [6], we refer to this model as MLP++. The original MLP uses a multi-layer perceptron to learn user-item relations, but it depends only on implicit feedback. MLP++ employs SDAEs to extract user and item features from auxiliary information, so it is able to learn the critical relation between a user and item.

2.4 Learning DHA-RS

In this section, we define the loss function for DHA-RS and explain how to optimize this function. Generally, a loss function consists of the reconstruction error of feature extraction and the prediction error. The loss of feature extraction contains user and item features extraction. The loss function of SDAE for user feature extraction can be defined as follows:

$$L_u = \|X_c - X_{L^x}\|_{\mathrm{F}}^2 + \lambda_\Omega \|\boldsymbol{\Omega}\|_{\mathrm{F}}^2 \tag{6}$$

where $\boldsymbol{\Omega}$ denotes the model parameter and λ_Ω denotes the regularization-term parameter. Similarly, the loss function of SDAE for the item features extraction can be defined as follows:

$$L_i = \|Y_c - Y_{L^y}\|_{\mathrm{F}}^2 + \lambda_\Psi \|\boldsymbol{\Psi}\|_{\mathrm{F}}^2 \tag{7}$$

where $\boldsymbol{\Psi}$ and λ_Ψ denote the model and regularization-term parameters, respectively.

The neural collaborative filtering process outputs the predicted rating $\hat{r}_{ui}$ for each (u, i) pair. Considering the characteristic of implicit feedback, user ratings can be regarded as labels for the user-item relations, that is, 1 denotes a user relating to the item and 0 otherwise, therefore, the predicted $\hat{r}_{ui}$ can be regarded as the possibility that a user relates to an item. This requires that $\hat{r}_{ui}$ be constrained in the range of [0, 1], which can be realized using a sigmoid activation function. The loss function can be defined as follows:

$$\begin{aligned} L_c = \sum_{(u,i)\in R\cup R^-} & (1 - r_{ui})\log_2(1 - \hat{r}_{ui}) + \\ & r_{ui}\log_2 \hat{r}_{ui} + \lambda_\theta \|\boldsymbol{\theta}\|_{\mathrm{F}}^2 \end{aligned} \tag{8}$$

where R denotes a set of positive instances and R^- denotes a set of negative instances, which can all be (or sampled from) unobserved user-item interactions. $\boldsymbol{\theta}$ and λ_θ denote the regularization-term and model parameters, respectively. This objective function is the same as binary cross-entropy loss, which works well for binary classification problems.

Therefore, the general loss function for the training model is as follows:

$$L = L_c + \alpha L_u + \beta L_i \tag{9}$$

where α and β denote the hyper parameters of the loss function.

Algorithm 1 shows the algorithm used in the DHA-RS framework. $\boldsymbol{P}$ and $\boldsymbol{Q}$ denote the weight matrix for user and item ID embedding, respectively, and λ_W and λ_b are hyper parameters for parameter initialization.

Algorithm 1 Learning algorithm for DHA-RS

```
Input: user feature $X$, item feature $Y$, user ID $v_u$, item ID $v_i$
Output: predicted $\hat{r}_{ui}$
1. Add noise into user feature $X$ and item feature $Y$ by stochastically setting elements in matrix to zero
2. For each layer $l^X$ in SDAE for users:
   (a) Construct weight matrix $W_l^X$ from $N(0, \lambda_W^{-1} I)$,
   (b) Construct bias $b_l^X$ from $N(0, \lambda_b^{-1} I)$,
   (c) Corresponding output layer $X_l = \sigma(X_{l-1} W_l^X + b_l^X)$.
3. For each layer $l^y$ in SDAE for items:
   (a) Construct weight matrix $W_l^Y$ from $N(0, \lambda_W^{-1} I)$,
   (b) Construct bias $b_l^Y$ from $N(0, \lambda_b^{-1} I)$,
   (c) Corresponding output layer $Y_l = \sigma(Y_{l-1} W_l^Y + b_l^Y)$.
4. For each user $u$:
   (a) One-hot encoding of user ID is $v_u$, the embedded user ID is $P^{\mathrm{T}} v_u$,
   (b) Extracted user feature $X_{L^X/2,u*}$,
   (c) Then, user latent vector: $p_u = \begin{bmatrix} X_{L^X/2,u*} \\ P^{\mathrm{T}} v_u \end{bmatrix}$.
5. For each item $i$:
   (a) One-hot encoding of item ID is $v_i$, the embedded item ID is $Q^{\mathrm{T}} v_i$,
   (b) Extract item feature $Y_{L^Y/2,i*}$,
   (c) Then, item latent vector $q_i = \begin{bmatrix} Y_{L^Y/2,i*} \\ Q^{\mathrm{T}} v_i \end{bmatrix}$.
6. For each input user and item latent vectors pair$(p_u, q_i)$, predicted $\hat{r}_{ui} = f(p_u, q_i)$,
   (a) Within GMF++: $\hat{r}_{ui} = a_{out}(h^{\mathrm{T}}(p_u \cdot q_i))$,
   (b) Within MLP++:
       For First Layer: $z_1 = \phi_1(p_u, q_i) = \begin{bmatrix} p_u \\ q_i \end{bmatrix}$,
       For remaining layers:
       $\phi_2(z_1) = a_2(W_2^{\mathrm{T}} z_1 + b_2)$,
       $\phi_L(z_{L-1}) = a_L(W_L^{\mathrm{T}} z_{L-1} + b_L)$,
       $\hat{r}_{ui} = \sigma(h^{\mathrm{T}} \phi_L(z_{L-1}))$.
```

2.5 Making predictions using the trained models

After training the model and learning parameters, we can predict the probability that user will rate an item for a user u and item i pair. As described in Algorithm 1, GMF++ and MLP++ use different strategies to deal with user and item latent vectors. The prediction details may differ for GMF++ and MLP++, but the general prediction function can be written as follows. Given a trained model, for a user u and item i pair with no observed relation, the model output predicted rating is

$$\hat{r}_{ui} = f\left(\begin{bmatrix} \sigma(X_{0,u*}(W_1^X)^* + (b_1^X)^*) \\ (P^*)^{\mathrm{T}} v_u \end{bmatrix}, \begin{bmatrix} \sigma(Y_{0,u*}(W_1^Y)^* + (b_1^Y)^*) \\ (Q^*)^{\mathrm{T}} v_i \end{bmatrix}\right) \tag{10}$$

where "*" denotes trained parameters that differ from the parameters in training.

3 Experiments and Results

3.1 Experimental settings

3.1.1 Dataset

Next, we evaluated the performance of the GMF++ and MLP++ models on the MovieLens-1M dataset (https://grouplens.org/dataset/movielens/). As GMF++ and MLP++ are designed for implicit feedback recommendation, we processed the original ratings in the MovieLens-1M dataset into implicit feedback data. If a user has an observed rating action for an item, we labeled the corresponding record 1. GMF++ and MLP++ also use the side information of users and items. Auxiliary user information includes age, occupation, and gender attributes. There are 18 different movie genres in the item features. Tables 2 and 3 summarize the characteristics of the MovieLens-1M dataset.

3.1.2 Evaluation indicators

In this experiment, we used the leave-one-out evaluation method to evaluate the performances of GMF++ and MLP++. We used the latest rating record of each user as the test set and left the remaining records for training. For the top-K recommendation, it is a time-consuming task to obtain the top-K relevant items. Instead, we referenced the experimental strategy used by the authors in Ref. [16] and randomly chose 100 items that are not rated by the user, and then ranked the candidate items to determine the top-K items. Herlocker et al.[9] systematically discussed the

Table 2 MovieLens-1M statistics.

Dataset	Number of users	Number of items	Sparsity (%)
MovieLens-1M	6040	3706	95.8

Table 3 User and item attributes.

Dataset	User attribute	Item attribute
MovieLens-1M	Age, occupation, gender	Genre

evaluation indicators. In this paper, we used the Hit Ratio (HR) and Normalized Discounted Cumulative Gain (NDCG)[22] to indicate the performance of the recommended list. The HR directly measures whether the test item is recommended in the top-K list. NDCG considers the position of the item. The resulting NDCG value is greater, if the item's position is nearer the front. The formula for calculating the NDCG is defined as follows:

$$\text{NDCG} = \frac{\text{DCG}@K}{i\text{DCG}@K} \tag{11}$$

$$\text{DCG}K = \sum_{i=1}^{K} \frac{2^{r(i)} - 1}{\log_2(i+1)} \tag{12}$$

where DCG and iDCG denote the discounted cumulative gain and ideal discount cumulative gain, respectively. $r(i)$ is the graded relevance of the predicted rating at position i. $\log_2(i+1)$ is the logarithmic reduction factor. The default length of the recommendation list is 10, excepting a specific declaration otherwise.

3.1.3 Compared methods

We compared GMF++ and MLP++ with the following related methods in this area:

(1) **ItemKNN**[23]. This is the standard item-based collaborative filtering method, which is employed by Amazon Inc. We used the settings reported in Ref. [24] to apply this method using implicit feedback data.

(2) **BPR**[25]. Bayesian Personalized Ranking (BPR) does not optimize item ratings, but directly optimizes personalized rankings, and we used this model as a comparative baseline for item recommendation.

(3) **GMF**[6]. This is a model proposed within the NCF framework, which learns the non-linear relation between users and items in a way similar to MF.

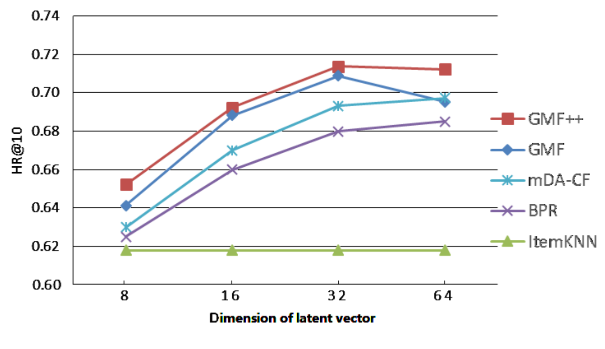
(4) **mDA-CF**[7]. This method employs SDAE to extract features from user and item auxiliary information and uses MF to determine user-item relations

3.1.4 Parameter setting

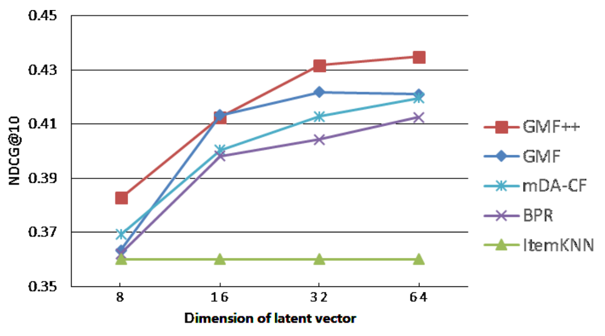
For the training set, we sampled four negative instances for each positive instance. We randomly initialized the model parameters using a Gaussian distribution with mean of 0 and standard deviation of 0.01. With reference to Ref. [26], we used a mini-batch Adam method to optimize the model and set the learning rate to 0.001 and the batch size to 256. In the feature-extraction step, we tested [4, 8, 16, 32] neurons in the middle layer. In neural collaborative step, we defined the latent vector dimension as the number of neurons in the last neural collaborative filtering layer of neural collaborative filtering. We tested the dimensions of [8, 16, 32, 64] latent vectors. For an MLP model with three hidden layers, if the dimension of latent vector is set to 8, then the architecture of the neural collaborative filtering layers is 32→16→8 and the dimension of the user and item latent vector is 16. In addition, we chose ReLU as the activation function, as had the authors in Ref. [6].

3.2 Experimental results and analysis

Because the experiments in Ref. [6] reveal that GMF performed better than MLP, we compared GMF++ with the other related methods. Here, we focus on the performance of GMF++ for a different dimension of the latent vector and length of recommendation list. Figure 3 shows the recommendation precisions of different methods with different latent vector dimensions. Since ItemKNN[23] is an item-based method and does not refer to latent vectors, we tested its performance using different neighborhood sizes and chose the best ItemKNN results. As shown in Fig. 3,



(a) MovieLens-HR@10



(b) MovieLens-NDCG@10

Fig. 3 Performances of HR@10 and NDCG@10 w.r.t. the dimension of latent vector.

GMF++ performs better than the other four models at HR@10 and NDCG@10. Compared with mDA-CF, GMF++ improved its average HR@10 and NDCG@10 performances by 2.9% and 3.7%, respectively, which proves that GMF++ yields better recommendation quality. GMF++ also performed better than GMF, which means that GMF++ with auxiliary information effectively enhances prediction precision. With respect to the latent vector dimension, the performances of HR@10 and NDCG@10 are maximized when the dimensions of the latent vector are 32 and 64, respectively.

The length of the recommendation list also affects prediction precision. Figure 4 shows that all five models perform better when K, the length of the recommendation list, increases. Compared to the other models, GMF++ achieves the best performance with respect to HR and NDCG when K is greater than five. However, GMF++ does not perform as well as mDA-CF, MGF, and BPR when K is less than 5. The performances of mDA-CF, GMF, and BPR are similar. From the tendency shown in Fig. 4, we can conclude that GMF++ can provide good recommendations when the recommendation list is long.

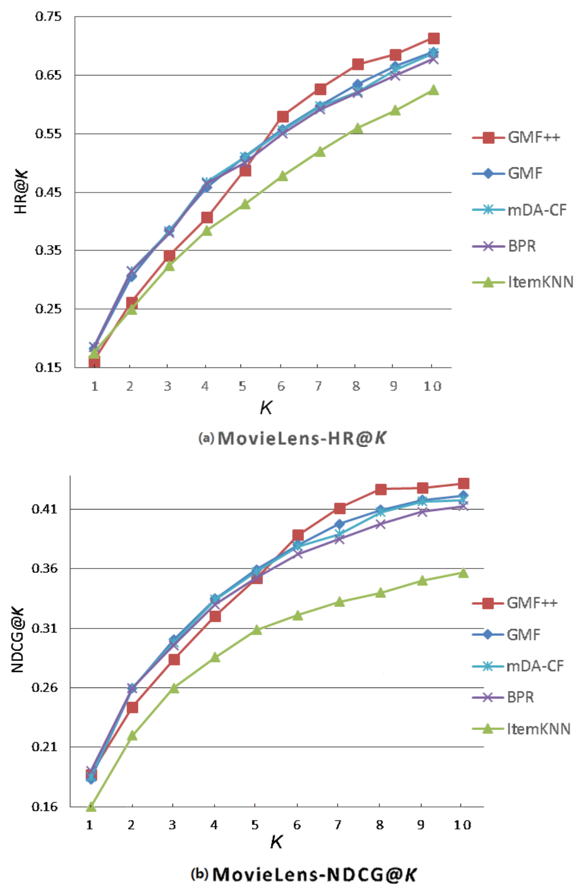


Fig. 4 Performances of HR@K and NDCG@K w.r.t. K.

3.3 Parameters used in training the model

In this section, we discuss the parameters we used in training the GMF++ model. Specifically, we focus on how the number of iterations and training weights, α and β, affect the performance of our methods. First, we compare the performances of GMF++ and GMF for different numbers of iterations.

As shown in Fig. 5, the performances of HR@10 and NDCG@10 for both GMF and GMF++ improve when the number of iterations increases and HR@10 and NDCG@10 tend toward stability. However, GMF++ performs better than GMF due to the introduction of user and item side information. We note that the performances of HR@10 and NDCG@10 for both GMF and GMF++ fluctuate after 10 iterations. This may be due to overfitting of the training model.

Figure 6 shows the effect of the number of iterations on the training loss. We can see that the training loss drops rapidly with less than 10 iterations and then tends to stabilize afterwards. Although the training loss continues to decrease, as we can see from the figure,

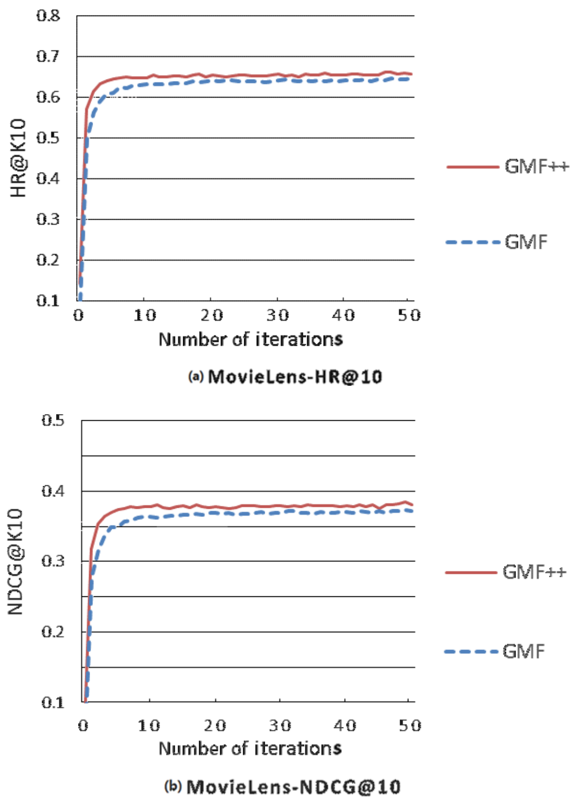


Fig. 5 HR@10 and NDCG@10 w.r.t the number of iterations.

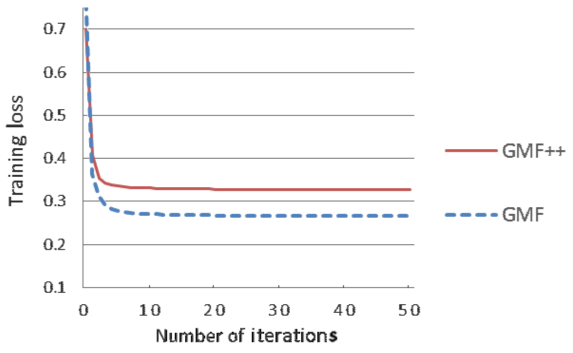


Fig. 6 Training loss w.r.t. number of iterations.

this may cause overfitting. Therefore, the number of iterations should not be too large. We note that the training loss of GMF++ contains the reconstruction error of the user and item features extraction which leads to a greater loss than GMF.

This is because the last two terms in Eq. (9), representing the errors of the user and item features extraction, are symmetric. In addition, the user and item inputs of the MovieLens-1M dataset are similar in size, so we set the parameters α and β as trade-off. In the experiment, we set the trade-off to [0.1, 1, 10, 100, 1000] and the dimension of the latent vector to eight. Figure 7 shows how this trade-off affects the performances of HR@10 and NDCG@10.

When we set the trade-off parameter to 100, we can see that the performance of HR@10 is the highest. Similarly, the performance of NDCG@10 is the highest when the trade-off value is 1. However, variations in the trade-off parameter had no significant influence on the HR@10 and NDCG@10 values. Therefore, we conclude that the model is not sensitive to trade-off and we set the trade-off to 100 in our final model.

3.4 Number of layers in MLP++

In this section, we evaluate the effect of layer number on the performance of MLP++, because the experiment results in Ref. [6] indicated that GMF performs better than MLP. Here, we only discuss the relation between the performance of MLP++ and the number of neural layers.

Figure 8 shows the performances of HR@10 and NDCG@10 of MLP++ when the number of layers is increased from 1 to 4. Specifically, we conducted the experiments using the output layer dimensions of [8, 16, 32, 64]. The performances of HR and NDCG improve when the dimension of the output layer is increased

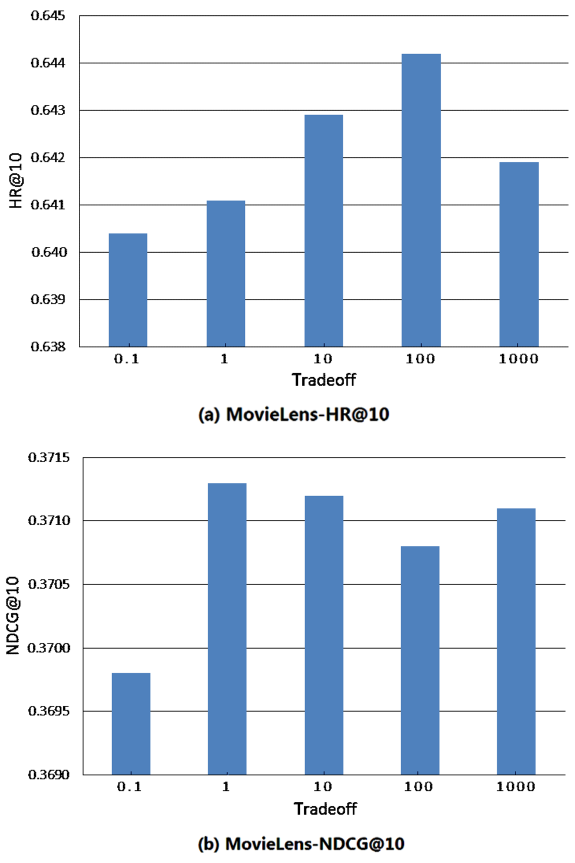


Fig. 7 Trade-off effect on HR@10 and NDCG@10.

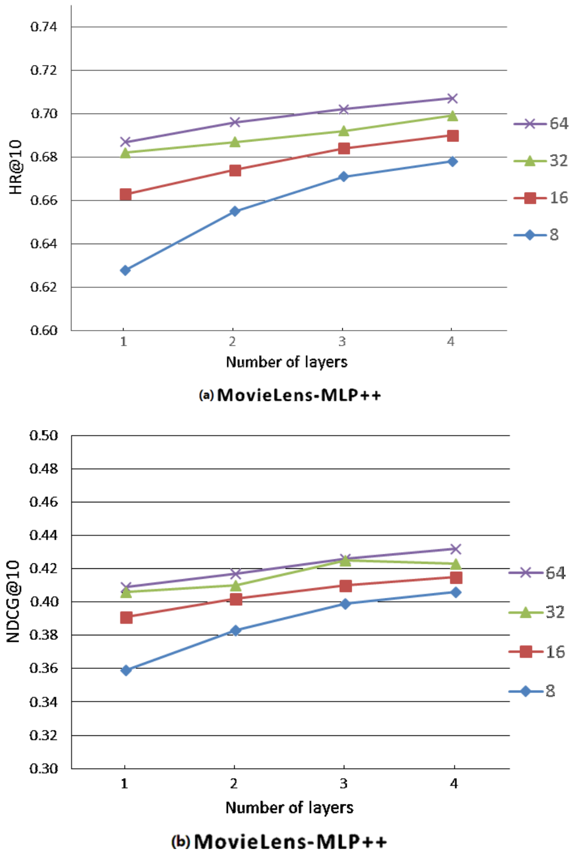


Fig. 8 Performances of HR@10 and NDCG@10 w.r.t. the number of hidden layers.

and MLP++ exhibits better recommendation precision with more hidden layers. Although the performance of MLP++ does not surpass that of GMF++, the performance of MLP++ continues to improve with increases in the number of hidden layers. Therefore, we can say that MLP++ has room for more improvement.

4 Conclusion and Future Work

In this paper, we proposed a novel recommender framework DHA-RS, which incorporates implicit feedback and user and item auxiliary information to effectively learn user and item features. Neural collaborative filtering enhances the learning capacity of collaborative filtering and incorporates user and item auxiliary information to improve the performance of user preference predictions. Using different settings for the neural collaborative filtering process, we proposed two models: GMF++ and MLP++. GMF++ merges user and item latent vectors to generate an element-wise product, which is used as the input of a fully connected perceptron. MLP++ concatenates the user and item latent vectors and employs a multi-layer network to learn the user-item relationship. Our experimental results show that GMF++ performs better than other state-of-the-art methods. Although MLP++ performs less well than GMF++, the experimental results for different numbers of hidden layers used indicate that MLP++ has considerable room for improvement. Our work shows that neural collaborative filtering with auxiliary information can enhance recommendation precision.

DHA-RS is not limited to textual auxiliary information, and can be extended to other types of auxiliary information. In future work, we will try to incorporate more auxiliary information, such as knowledge ontology and images. By combining more representative side information into the recommender system, we expect that recommendation precision can be further improved.

Acknowledgment

This work was supported by the National Natural Science Foundation of China (No. 61370077) and Collaborative Innovation Center of Novel Software Technology and Industrialization.

References

[1] J. W. Lu, Y. Guo, Z. Q. Mi, and Y. Yang, Trust-enhanced matrix factorization using PageRank for recommender system, in *Proc. 2017 Int. Conf. Computer, Information and Telecommunication Systems*, Dalian, China, 2017, pp. 123–127.

[2] G. Linden, B. Smith, and J. York, Amazon.com recommendations: Item-to-item collaborative filtering, *IEEE Internet Comput.*, vol. 7, no. 1, pp. 76–80, 2003.

[3] 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.

[4] R. Salakhutdinov, A. Mnih, and G. Hinton, Restricted Boltzmann machines for collaborative filtering, in *Proc. 24th Int. Conf. Machine Learning*, Corvalis, OR, USA, 2007, pp. 791–798.

[5] Y. X. Ouyang, W. Q. Liu, W. G. Rong, and Z. Xiong, Autoencoder-based collaborative filtering, in *Proc. Int. Conf. Neural Information Processing*, Kuching, Malaysia, 2014, pp. 284–291.

[6] X. N. He, L. Z. Liao, H. W. Zhang, L. Q. Nie, X. Hu, and T. S. Chua, Neural collaborative filtering, in *Proc. 26th Int. Conf. World Wide Web*, Perth, Australia, 2017, pp. 173–182.

[7] S. Li, J. Kawale, and Y. Fu, Deep collaborative filtering via marginalized denoising auto-encoder, in *Proc. 24th ACM Int. Conf. Information and Knowledge Management*, Melbourne, Australia, 2015, pp. 811–820.

[8] F. Z. Zhang, N. J. Yuan, D. F. Lian, X. Xie, and W. Y. Ma, Collaborative knowledge base embedding for recommender systems, in *Proc. 22nd ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining*, 2016, pp. 353–362.

[9] J. L. Herlocker, J. A. Konstan, L. G. Terveen, and J. T. Riedl, Evaluating collaborative filtering recommender systems, *ACM Trans. Inf. Syst.*, vol. 22, no. 1, pp. 5–53, 2004.

[10] H. Wang, N. Y. Wang, and D. Y. Yeung, Collaborative deep learning for recommender systems, in *Proc. 21th ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining*, Sydney, Australia, 2015, pp. 1235–1244.

[11] D. D. Lee and H. S. Seung, Algorithms for non-negative matrix factorization, in *Proc. 13th Int. Conf. Neural Information Processing Systems*, Denver, CO, USA, 2001, pp. 556–562.

[12] Y. Koren, R. Bell, and C. Volinsky, Matrix factorization techniques for recommender systems, *Computer*, vol. 42, no. 8, pp. 30–37, 2009.

[13] R. Salakhutdinov and A. Mnih, Probabilistic matrix factorization, in *Proc. 20th Int. Conf. Neural Information Processing Systems*, Vancouver, Canada, 2008, pp. 1257–1264.

[14] H. Larochelle, D. Erhan, A. Courville, J. Bergstra, and Y. Bengio, An empirical evaluation of deep architectures on problems with many factors of variation, in *Proc. 24th Int. Conf. Machine Learning*, Corvalis, OR, USA, 2007, pp. 473–480.

[15] K. Georgiev and P. Nakov, A non-IID framework for collaborative filtering with restricted Boltzmann machines, in *Proc. 30th Int. Conf. Machine Learning*, Atlanta, GA, USA, 2013, pp. 1148–1156.

[16] Y. Wu, C. DuBois, A. X. Zheng, and M. Ester, Collaborative denoising auto-encoders for top-N recommender systems, in *Proc. 9th ACM Int. Conf. Web Search and Data Mining*, San Francisco, CA, USA, 2016, pp. 153–162.

[17] P. Vincent, H. Larochelle, Y. Bengio, and P. A. Manzagol, Extracting and composing robust features with denoising autoencoders, in *Proc. 25th Int. Conf. Machine Learning*, Helsinki, Finland, 2008, pp. 1096–1103.

[18] R. N. He and J. McAuley, VBPR: Visual Bayesian personalized ranking from implicit feedback, in *Proc. 30th AAAI Conf. Artificial Intelligence*, Phoenix, AZ, USA, 2016, pp. 144–150.

[19] M. D. Zeiler and R. Fergus, Visualizing and understanding convolutional networks, in *Proc. European Conf. Computer Vision*, Zurich, Switzerland, 2014, pp. 818–833.

[20] A. Krizhevsky, I. Sutskever, and G. E. Hinton, ImageNet classification with deep convolutional neural networks, in *Proc. 25th Int. Conf. Neural Information Processing Systems*, Lake Tahoe, NV, USA, 2012, pp. 1097–1105.

[21] Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle, Greedy layer-wise training of deep networks, in *Proc. 19th Int. Conf. Neural Information Processing Systems*, 2007, pp. 153–160.

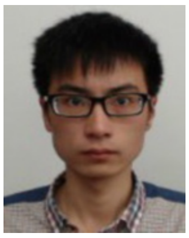
[22] X. N. He, T. Chen, M. Y. Kan, and X. Chen, TriRank: Review-aware explainable recommendation by modeling aspects, in *Proc. 24th ACM Int. Conf. Information and Knowledge Management*, Melbourne, Australia, 2015, pp. 1661–1670.

[23] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl, Item-based collaborative filtering recommendation algorithms, in *Proc. 10th Int. Conf. World Wide Web*, Hong Kong, China, 2001, pp. 285–295.

[24] S. Rendle, Factorization machines, in *Proc. 10th Int. Conf. Data Mining*, Sydney, Australia, 2010, pp. 995–1000.

[25] S. Rendle, C. Freudenthaler, Z. Gantner, and L. Schmidt-Thieme, BPR: Bayesian personalized ranking from implicit feedback, in *Proc. 25th Conf. Uncertainty in Artificial Intelligence*, Montreal, Canada, 2009, pp. 452–461.

[26] D. P. Kingma and J. Ba, Adam: A method for stochastic optimization, arXiv preprint arXiv: 1412.6980, 2014.



Yu Liu is currently a master student in School of Computer Science and Engineering of Southeast University. His research interests include machine learning and data mining.

Shuai Wang received the MSc degree from Southeast University, China, in 2017. His research interests include business intelligence and data mining.

Jieyue He received the BSc and MSc degrees from Nanjing University, China, and PhD degree from Southeast University, China, in 1985, 1988, and 2006. She is currently a professor of the School of Computer Science and Engineering, Southeast University, China. Her current research interests include bioinformatics, data mining, machine learning, and big data.

M. Shahrukh Khan is currently a master student in School of Computer Science and Engineering of Southeast University. His research interests include data mining and recommendation system.