Recommendations Based on Listwise Learning-to-Rank by Incorporating Social Information

Chen Fang, Hengwei Zhang, Ming Zhang and Jindong Wang
Zhengzhou Information Science and Technology Institute, P.R. China
[e-mail: 17756230629@163.com]
*Corresponding author: Hengwei Zhang

Received April 13, 2017; revised July 10, 2017; revised August 2, 2017; accepted September 17, 2017; published January 31, 2018

Abstract

Collaborative Filtering (CF) is widely used in recommendation field, which can be divided into rating-based CF and learning-to-rank based CF. Although many methods have been proposed based on these two kinds of CF, there still be room for improvement. Firstly, the data sparsity problem still remains a big challenge for CF algorithms. Secondly, the malicious rating given by some illegal users may affect the recommendation accuracy. Existing CF algorithms seldom took both of the two observations into consideration. In this paper, we propose a recommendation method based on listwise learning-to-rank by incorporating users' social information. By taking both ratings and order of items into consideration, the Plackett-Luce model is presented to find more accurate similar users. In order to alleviate the data sparsity problem, the improved matrix factorization model by integrating the influence of similar users is proposed to predict the rating. On the basis of exploring the trust relationship between users according to their social information, a listwise learning-to-rank algorithm is proposed to learn an optimal ranking model, which can output the recommendation list more consistent with the user preference. Comprehensive experiments conducted on two

http://doi.org/10.3837/tiis.2018.01.006 **ISSN : 1976-7277**

public real-world datasets show that our approach not only achieves high recommendation accuracy in relatively short runtime, but also is able to reduce the impact of malicious ratings.

Keywords: Collaborative filtering, learning-to-rank, data sparsity, social information, trust relationship

1. Introduction

With the popularity of the Internet and the rapid development of cloud computing technology, the amount of data on the Internet increases dramatically. How to find valuable information from the vast data is becoming more and more challenging for users. In such a context, recommendation system came into being, which could provide users with a recommendation list to meet their preference. Since the recommendation system can effectively solve information overload problem, it has drawn a lot of attention from academia and industry.

Collaborative filtering (CF) is currently one of the most widely used algorithms in the recommendation field. Based on the assumption that similar users have similar interests, CF predicts the ratings of items and then generates the recommendation list according to the descending order of predicted ratings. About rating prediction, many works have been proposed, such as social-aware prediction [1], location-aware prediction [2], time-aware prediction [3], but they all ignore the ordering information of rating data, which is important to improve rating prediction accuracy.

Furthermore, some researchers recently have found that rating-based recommendation is not always accurate. For example, suppose a user rates 2 points for item A and 3 points for item B. By using different algorithms to predict the ratings, we will get different recommendation results. One prediction is item A scoring 2.5 points and item B scoring 3.6 points; the other one is item A scoring 2.5 points and item B scoring 2.4 points. MAE (mean absolute error) is one of the most widely used evaluation criteria of CF algorithms. The two algorithms have the same value of MAE (i.e. $(0.5+0.6)/2=0.55$), meaning that their rating prediction accuracy is the same. However, the order of item A and B in their recommendation lists is quite opposite. This example verifies the conclusion that rating prediction accuracy is difficult to interpret as a measure of recommendation accuracy [4].

In order to solve the problem, researchers began to integrate learning-to-rank (LTR) into CF algorithms, believing that directly optimizing the recommendation list is more in line with the goal of recommendation system. Similar to the principle of machine learning, LTR-based CF algorithms learn an optimal ranking model based on historical rating records. Since the ranking model focuses on the order of items in recommendation list

rather than the rating, it can output a recommendation list more consistent with the user preference. In recent years, LTR-based CF algorithms have received more and more attention [5], but they still suffer from traditional challenges in recommendation field, such as data sparsity problem, malicious ratings, and so on. Some methods have been proposed in related works, such as incorporating users' social networks [6], employing clustering models [7], but they all aim to predict the missing ratings rather than directly optimize the recommendation list. Currently little effort has been devoted to exploiting LTR-based CF in solving those traditional challenges.

In this paper, we propose a recommendation method based on listwise LTR by incorporating social information (ListSo). It considers both the trust relationship between users and ordering information of rating data, and learns an optimal recommendation model by listwise LTR algorithm. The contribution of this paper is three-fold:

(1) Unlike the previous studies, we propose an enhanced measurement to compute the user similarity. By using the Plackett-Luce probability distribution model, the information included in the order of items is fully exploited to find more accurate similar neighbors.

(2) While predicting the rating, the influence of neighbors is integrated into the matrix factorization model, effectively alleviating the data sparsity problem. Meanwhile, the model is also used to explore the trust relationship between users, so as to reduce of impact of malicious rating.

(3) By using the listwise LTR algorithm, an optimal recommendation model is learned from the training dataset. Experiments show that ListSo can be applied to large-scale datasets.

The rest of this paper is organized as follows: Section 2 surveys the related works on recommendation methods. Section 3 shows the framework and details of our approach, including the similarity evaluation, rating prediction, trust evaluation and listwise LTR algorithm. Section 4 describes the experimental results. Finally, we draw conclusions and outline our future work in Section 5.

2. Related Work

Many CF algorithms have been proposed in the recommendation field, which can be divided into two categories: rating-based CF algorithms and LTR-based CF algorithms [9]. Next, we will introduce related works of the two categories in detail.

2.1 Rating-based CF Algorithms

Rating-based CF algorithms can be divided into two categories: memory-based and model-based. Memory-based CF is one of the earliest and most widely used recommendation algorithms, which has been successfully applied to many e-commerce systems. Memory-based CF finds similar users or items as neighbors according to the rating data, and predicts missing ratings based on those neighbors, so it can be further divided into user-based CF and item-based CF. Wang et al. [10] took users' interests and

the timeliness of items into consideration, and found the nearest neighbor set by combining with item-based CF algorithms. Jia et al. [11] proposed a novel CF recommendation algorithm based on the double neighbor choosing strategy. They chose the trustworthy neighbor set of target user by considering the similarity and trust relationship between users at the same time. Li et al. [12] differentiated the non-target users into two types that without recommending ability and with recommending ability. They only cared about the latter users and proposed a "domain nearest neighbor" method to predict the missing ratings of items.

Different from memory-based CF, model-based CF learns a model by training the rating data, and then uses it to predict missing ratings. The popular models include clustering model, Bayesian model, linear regression model, Markov model, matrix factorization model, etc. The matrix factorization model is the most widely used among them. It decomposes the high-dimensional user-item rating matrix into low-dimensional user latent matrix and item latent matrix, thereby reducing the dimensionality of rating data and alleviating the data sparsity problem. Guo et al. [13] combined the community structure and interest clusters information into matrix factorization model by adding a clustering-based regularization term to improve the objective function. Jamali et al. [14] proposed a recommendation model named SocialMF. They assumed that the latent feature vector of the target user was determined by that of his friends, and introduced the concept of trust propagation into the process of matrix factorization. Yang et al. [15] classified the user's friends according to the categories of items, and then gave them different level of trust, but it may further aggravate the data sparsity problem.

In conclusion, either memory-based CF or model-based CF, they both aim to predict missing ratings. However, the recommendation list got by sorting predicted ratings can not accurately reflect user's preference, which has been illustrated by the example in the introduction. So, recently researchers have begun to study how to incorporate learning-to-rank into CF to optimize the recommendation list directly.

2.2 LTR-based CF Algorithms

Learning-to-rank (LTR) is a supervised machine learning algorithm, which aims to get the optimal ranking model by training the dataset [5]. By learning the optimal parameters automatically, LTR can integrate complex features and reduce the defect of considering single factors while ranking, so it has become popular in the recommendation field. According to the input data, LTR can be divided into three categories: pointwise, pairwise and listwise. Pointwise LTR inputs individual items and predicts their missing ratings, which is very similar to rating-based CF. Pairwise LTR defines the partial order between items according to their ratings, and generates the recommendation list by integrating the partial order of all item pairs. Listwise LTR inputs all the items and directly optimizes the entire list, which is the most popular among them.

Liu et al. [4] predicted the partial order of unrated item pairs based on similar users, and generated a recommendation list by incorporating all the partial orders. Pan et al. [16]

converted the user-item matrix into the partial order matrix according to the user's browsing records, and proposed a personalized ranking algorithm based on Bayesian theory. Both [4] and [16] have high recommendation accuracy, but their time complexity will be high when being applied to large datasets. Shi et al. [17] combined listwise LTR with matrix factorization and got the recommendation list by minimizing the loss function representing the uncertainty between training lists and predicted lists. Weimer et al. [18] proposed an algorithm called CoFiRank which directly optimized the evaluation criteria NDCG. Huang et al. [19] directly predicted a total order of items for each user based on similar users' probability distributions over permutations of the items.

As a kind of machine learning algorithms, LTR can effectively process large-scale datasets, but it still suffers from data sparsity problem, as users only rate a few items. In this paper, we will address this issue by mining the information included in the order of items and incorporating social information into recommendation.

3. The Recommendation Approach

Traditional CF-based recommendation algorithms find similar users according to the historical ratings, and then predict missing ratings for the target user based on those similar users. However, if some neighbors give malicious ratings on some items, the recommendation accuracy will be severely affected. Aiming at this problem, both similarity and trust relationship are taken into consideration, thereby improving the recommendation accuracy and reducing the impact of malicious ratings.

3.1 Framework of Our Approach

Now we describe the framework of our approach as shown in **Fig. 1**. For simplicity, we suppose that the rating dataset is already provided or acquired.

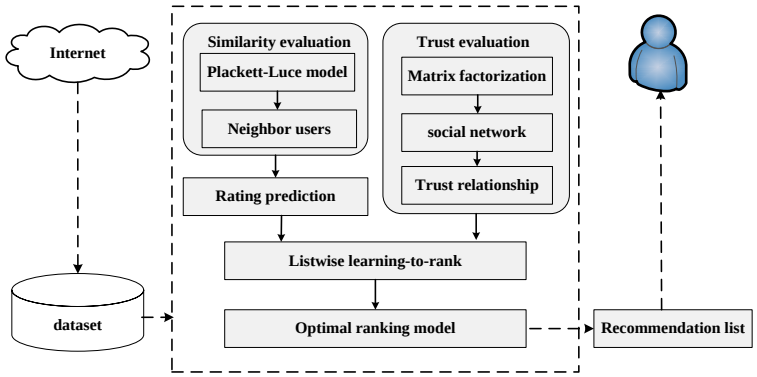


Fig. 1. Framework of our approach

Our approach has four subcomponents: similarity evaluation, rating prediction, trust evaluation and listwise learning-to-rank algorithm. The similarity evaluation employs the Plackett-Luce probability distribution model to calculate the user similarity, thereby making full use of the information included in the order of items and finding more accurate neighbors. The rating prediction is performed by integrating the influence of neighbors into matrix factorization model, effectively alleviating the data sparsity problem. The trust evaluation explores the trust relationship between users from their social information, so as to reduce the impact of malicious ratings. Based on the rating and trust relationship, the listwise LTR algorithm learns an optimal ranking model from the training dataset. Finally, the ranking model will output a recommendation list more consistent with the user preference. The major symbols used in this paper are listed in **Table 1**.

Table 1. Major symbols and definitions

Symbols	Definitions
$\boldsymbol{R}=(r_{ij})_{m\times n}$	user-item matrix, where m is the number of users and n is the number of items.
r_{ij}	the rating of user u_i for item j
$\boldsymbol{T}=\{t_{ik}\}_{m\times m}$	user-trust matrix, where $t_{ik}\in\{0,1\}$ indicates whether user u_i trusts user u_k or not
$\boldsymbol{U}$	$d\times m$ dimensional user latent matrix
$\boldsymbol{U}_i=[u_{i1},u_{i2},\cdots,u_{id}]^T$	the *i-th* column vector of $\boldsymbol{U}$, representing the latent vector of user u_i
$\boldsymbol{V}$	$d\times n$ dimensional item latent matrix
$\boldsymbol{V}_j=[v_{j1},v_{j2},\cdots,v_{jd}]^T$	the *j-th* column vector of $\boldsymbol{V}$, representing the latent vector of item j
$\boldsymbol{Z}$	$d\times m$ dimensional trust latent matrix
$\boldsymbol{Z}_k=[z_{k1},z_{k2},\cdots,z_{kd}]^T$	the *k-th* column vector of $\boldsymbol{Z}$, representing the trust latent vector of user u_k
d	dimensionality of latent vectors

3.2 Similarity Evaluation

The key step in CF-based recommendation algorithms is finding similar users. Methods of calculating the user similarity include Pearson correlation, cosine similarity, modified cosine similarity, etc [12]. The Pearson correlation is most widely used in related papers, which is defined as follows:

Definition 1 (Pearson correlation) If r_{ui} and r_{vi} respectively denotes the rating of user u and v for item i, $\overline{r}_u$ and $\overline{r}_v$ respectively denotes the average rating of user u and v, $I_{u,v}$ is the

commonly rated item set of user *u* and *v*, then the similarity between user *u* and *v* based on Pearson correlation is:

$$PCC(u,v)=\frac{\sum_{i\in I_{u,v}}(r_{ui}-\overline{r}_u)(r_{vi}-\overline{r}_v)}{\sqrt{\sum_{i\in I_{u,v}}(r_{ui}-\overline{r}_u)^2}\sqrt{\sum_{i\in I_{u,v}}(r_{vi}-\overline{r}_v)^2}} \tag{1}$$

However, in this paper we argue that using the Pearson correlation to compute the user similarity is not accurate. For example, assume user *A*, *B* and *C* commonly rate item *a*, *b*, *c*, *d*, and the ratings range from 0 to 100. Their rating vectors for the four items are respectively denoted as $A=\{0,20,80,100\}, B=\{10,0,80,100\}, C=\{0,22,100,89\}$. As shown in the Fig. 2, by sorting the items according to their ratings, the recommendation list for each user is obtained. It can be seen that the recommendation order of user *B* for item *a* and *b* is contrary to that of user *A*, while the recommendation order of user *C* for item *c* and *d* is contrary to that of user *A*. As a result, user *B* would recommend item *d* to user A, while user *C* would recommend item *c* to user *A*. It can be seen that the recommendation of user *B* is more consistent with user *A* preference, so the user similarity between *A* and *B* should be larger. However, according to Definition 1, we can get *PCC(A,B)=PCC(A,C)=0.97*, meaning the similarity between *A* and *B* is the same as the similarity between *A* and *C*, which is unreasonable. In the actual recommendation system, users are more concerned about the items ranked ahead in the recommendation list, such as *c* and *d*, while paying less attention to those items ranked behind in the list, such as *a* and *b*, because low rating means the items do not meet the user preference. So the order of items in the recommendation list should also be considered while computing the user similarity.

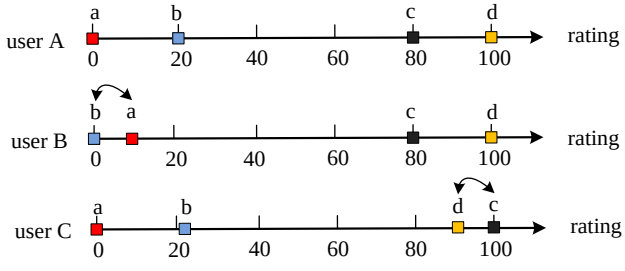


Fig. 2. Example of ratings

In this paper, both ratings and order of items are taken into account, so as to find more accurate similar users. By using the Plackett-Luce model in [20] to represent the user with the probability distribution over the permutations of rated items, the user similarity is computed more accurately.

The main idea of Plackett-Luce model is as follows: given a set *I* containing *n* items and the corresponding ranking function, the item set *I* has $n!$ kinds of permutations. Each permutation can be denoted as $\pi=<\pi_1,\pi_2,\ldots,\pi_n>$, where $\pi_i\in I$ denotes the item ranked

at position *i*. The set of all possible permutations is denoted as Ω^I. Each item in the permutation has its own rating, so different permutation has different probability. Those permutations where items with larger ratings are ranked higher should have larger probability.

Definition 2 (Permutation Probability) Given a permutation $\pi =< \pi_1, \pi_2, \ldots, \pi_n >$ and the rating of the corresponding items $\{r_{\pi_1}, r_{\pi_2}, \ldots, r_{\pi_n}\}$, the probability of π is defined as:

$$P(\pi) = \prod_{i=1}^{n} \frac{\varphi(r_{\pi_i})}{\sum_{k=i}^{n} \varphi(r_{\pi_k})} \tag{2}$$

Where $\varphi(\bullet)$ is a monotonically increasing and strictly positive function. In this paper, we assume $\varphi(x) = e^x$.

In the real network environment, the number of items in the set *I* is often very large, so computing the probability of $n!$ different permutations will consume lots of time. Cao et al. [21] proposed a permutation model, which only focused on the items in the top-*k* positions of the permutation.

Definition 3 (Top-*k* permutation set) Given an item set $I=\{i_1,i_2,i_3,\ldots,i_n\}$, a top-*k* permutation set $T_k(i_1, i_2, \ldots, i_k)$ of *I* includes all the permutations where the top-*k* items are $i_1, i_2, \ldots, i_k \in I$, which can be denoted as

$$T_k(i_1, i_2, \ldots, i_k) = \{\pi \,|\, \pi \in \Omega^I, \pi_j = i_j, j = 1, 2, \ldots, k\} \tag{3}$$

According to Definition 3, two top-*k* permutation sets are different if the top-*k* items are different. So for set *I* of *n* items, there are $n!/(n-k)!$ different top-*k* permutation sets in all, which are denoted as set T_k^I. For example, let $I=\{i_1,i_2,i_3\}$. There are *3!=6* different permutations of *I*: $< i_1,i_2,i_3>$, $< i_1,i_3,i_2>$, $< i_2,i_1,i_3>$, $< i_2,i_3,i_1>$, $< i_3,i_2,i_1>$, $< i_3,i_1,i_2>$. If *k=1*, there are *3!/(3-1)!=3* different top-*1* permutation sets: $T_1^I = \{T_1(i_1), T_1(i_2), T_1(i_3)\}$, where $T_1(i_1) = \{< \underline{i_1}, i_2, i_3 >, < \underline{i_1}, i_3, i_2 >\}, T_1(i_2) = \{< \underline{i_2}, i_1, i_3 >, < \underline{i_2}, i_3, i_1 >\}, T_1(i_3) = \{< \underline{i_3}, i_1, i_2 >, < \underline{i_3}, i_2, i_1 >\}$.

Cao et al. [21] gave an efficient way to compute the probability of the top-*k* permutation set.

LEMMA The probability of the top-k permutation set $P(T_k(i_1, i_2, \ldots, i_k))$ is the probability of items $i_1, i_2, \ldots, i_k$ being ranked in the top-k positions, which can be computed as:

$$P(T_k(i_1, i_2, ..., i_k)) = \prod_{j=1}^{k} \frac{\varphi(r_{\pi_j})}{\sum_{l=j}^{n} \varphi(r_{\pi_l})}, \forall j = 1, \ldots, k : \pi_j = i_j \tag{4}$$

Where r_{π_j} denotes the rating of the item ranked in position *j*. Thus, computing the probability of the top-*k* permutation sets in T_k^I is much more efficient than computing the probabilities of all the permutations of *I*.

The above is an introduction of Plackett-Luce model. Next we will illustrate how to apply Plackett-Luce model to compute the user similarity.

In a recommendation system, for any two users *u* and *v*, let $I_{u,v}$ be the set of items they commonly rated, and $T_k^{I_{u,v}}$ be the set of top-*k* permutation sets of $I_{u,v}$. Given the rating of *u* and *v*, their probability distributions over $T_k^{I_{u,v}}$ can be computed by Eq.(4), denoted as P_u and P_v respectively. Then the similarity between *u* and *v* can be measured by the distance between the probability distribution P_u and P_v. The closer P_u and P_v is, the more similar *u* and *v* is.

The Kullback-Leibler (KL) distance [22] is commonly used to measure the distance of probability distributions in probability theory, so we use it to compute the user similarity in this paper. For user *u* and *v*, the KL distance between P_u and P_v is computed as follows:

$$D_{KL}(P_u \| P_v) = \sum_{g \in T_k^{S_{u,v}}} P_u(g) \log_2 (\frac{P_u(g)}{P_v(g)}) \tag{5}$$

Where $P_u(g)$ and $P_v(g)$ represents the probability of the top-k permutation set $g \in T_k^{I_{u,v}}$ for *u* and *v* respectively, which can be computed by Eq.(4). According to Eq.(5), it can be seen that the KL distance is asymmetric, i.e., $D_{KL}(P_u \| P_v) \neq D_{KL}(P_v \| P_u)$. Therefore a symmetric similarity measurement based on KL distance is proposed.

Definition 4 (User similarity) The user similarity between *u* and *v* is defined as follows:

$$s(u,v) = 1 - \frac{1}{2}(D_{KL}(P_u \| P_v) + D_{KL}(P_v \| P_u)) \tag{6}$$

However, according to Eq.(6), when the number of commonly rated items of *u* and *v* is small, the user similarity will be too large. In order to solve the problem, we multiply $s(u,v)$ by $\min\{\frac{|I_{u,v}|}{c}, 1\}$, where *c* is threshold of $|I_{u,v}|$, i.e.

$$s'(u,v) = s(u,v) \times \min\{\frac{|I_{u,v}|}{c}, 1\} \tag{7}$$

When the number of commonly rated items is less than *c*, the similarity will be reduced to the normal level.

For each user u_i, compute the similarity $s'(u,v)$ between u_i and each other user. Select the top-*k* users with larger similarity as the neighbors of u_i, denoted as N_i, which will be used in subsequent rating prediction.

Using the algorithm to compute the user similarity in **Fig. 2**, we can get $s'(A,B) = 0.95, s'(A,C) = 0.82$, which verifies the conclusion that the user similarity between A and B should be larger.

Algorithm 1. User Similarity Computation

Input:

user *u*; user *v*; common rated item set $I_{u,v}$; the rating data of user *u* and user *v*; the threshold *c*; parameter *k*

Output:

the similarity between u and v: $s'(u,v)$

1: $D_{KL}(P_u \| P_v) \leftarrow 0; D_{KL}(P_v \| P_u) \leftarrow 0$; $s(u,v) \leftarrow 0$; $n_1 \leftarrow |I_{u,v}|$;

2: $T_k^{I_{u,v}} \leftarrow \{T_k(i_1,i_2,\ldots,i_j,\ldots,i_k) \mid i_j \in I_{u,v}\}$; // get all the possible top-k permutation sets of $I_{u,v}$ according to Definition 3

3: for each top-k permutation set $g \in T_k^{I_{u,v}}$ do

4: compute the probability of $g \in T_k^{I_{u,v}}$ for user u by Eq.(4), denoted as $P_u(g)$;

5: compute the probability of $g \in T_k^{I_{u,v}}$ for user v by Eq.(4), denoted as $P_v(g)$;

6: $D_{KL}(P_u \| P_v) \leftarrow D_{KL}(P_u \| P_v) + P_u(g)\log_2(\frac{P_u(g)}{P_v(g)})$; $D_{KL}(P_v \| P_u) \leftarrow D_{KL}(P_v \| P_u) + P_v(g)\log_2(\frac{P_v(g)}{P_u(g)})$;

7: End for

8: $s(u,v) \leftarrow 1 - \frac{1}{2}(D_{KL}(P_u \| P_v) + D_{KL}(P_v \| P_u))$;

9: $s'(u,v) \leftarrow s(u,v) \times \min\{\frac{|I_{u,v}|}{c}, 1\}$;

10: return $s'(u,v)$;

3.3 Rating Prediction

Based on the idea "users with similar rating behavior tend to give similar ratings on items [23]", CF-based recommendation algorithms predict missing ratings according to similar users. However, they still suffer from data sparsity problem. There exists massive items in recommendation system, but most users only rate a few of them, so their rating data is very sparse, making CF-based algorithms hard to find accurate similar users.

In this paper, the matrix factorization model is used to alleviate the data sparsity problem. The main idea is to decompose the high-dimensional user-item matrix $\boldsymbol{R}$ into low-dimensional user latent matrix $\boldsymbol{U}$ and item latent matrix $\boldsymbol{V}$, which is as follows:

$$\boldsymbol{R} \approx \boldsymbol{U}^T\boldsymbol{V} \tag{8}$$

Then the rating of user u_i for item j can be predicted as:

$$\hat{r}_{ij} = g(\boldsymbol{U}_i^T\boldsymbol{V}_j) \tag{9}$$

During the training process, the model constantly adjusts the user latent matrix $\boldsymbol{U}$ and item latent matrix $\boldsymbol{V}$, so as to minimize the bias between predicted ratings and actual ratings. However, in the actual recommendation system, the rating behavior of the target user is often influenced by his similar neighbors. In order to further improve the rating prediction accuracy, the influence of similar users is integrated into the process of matrix factorization, modifying Eq.(9) as follows:

$$\hat{r}_{ij} = g(\alpha\boldsymbol{U}_i^T\boldsymbol{V}_j + (1-\alpha)\sum_{k\in N_i} s'(i,k)\boldsymbol{U}_k^T\boldsymbol{V}_j) \tag{10}$$

Where N_i is the neighbor set of user u_i ; $s'(i,k)$ is the user similarity between u_i and u_k ,computed by Eq.(7); α controls the influence of similar neighbors; $g(x)$ is the

logistic function, assumed as $g(x) = 1/1 + e^{-x}$ in this paper. It can be seen that when the neighbor user u_k gives a high rating on item j ($\boldsymbol{U}_k^T \boldsymbol{V}_j$ is large), the predicted rating $\hat{r}_{ij}$ is supposed to be increased as well.

3.4 Trust Evaluation

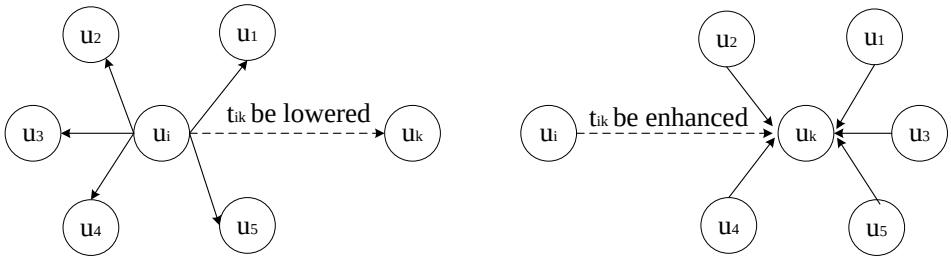
In traditional CF-based recommendation algorithms, if there are some neighbors giving malicious ratings on items, the recommendation accuracy will be severely affected. There are also works about the trust-based recommendation [11][15], but they still suffer from data sparsity problem, due to the fact that trust relationship between users is also sparse under the big data environment.

To solve the problem, in this section the matrix factorization model is also used to predict the unknown trust relationship between users. By decomposing the high-dimensional user-trust matrix $\boldsymbol{T}$ into low-dimensional user latent matrix $\boldsymbol{U}$ and trust latent matrix $\boldsymbol{Z}$, the trust relationship between u_i and u_k can be predicted as follows:

$$\hat{t}_{ik} = g(\boldsymbol{U}_i^T \boldsymbol{Z}_k) \tag{11}$$

Where $\boldsymbol{U}$ is the same user latent matrix in rating prediction. $g(x)$ is the logistic function that bounds the range of $\boldsymbol{U}_i^T \boldsymbol{Z}_k$ within [0,1], assumed as $g(x) = 1/1 + e^{-x}$ in this paper.

In real social network, the trust relationship between users is asymmetric, because A trusts B does not mean B trusts A. So $t_{ik} \in \{0,1\}$ can not accurately reflect the trust between users. The structure of social network needs to be taken into account while computing the trust value between users. For example, as shown in **Fig. 3,** when user u_i trusts many users, the trust value t_{ik} between user u_i and user u_k should be lowered. When user u_k is trusted by many users, t_{ik} should be enhanced.



(1)u_i trusts many users (2)u_k is trusted by many users

Fig. 3. Two cases of trust relationship

Based on the above analysis, the original trust value $t_{ik} \in \{0,1\}$ is modified as follows [24]:

$$t_{ik}^{*} = \sqrt{\frac{d^{-}(u_k)}{d^{+}(u_i)+d^{-}(u_k)}} \times t_{ik} \tag{12}$$

Where $d^{+}(u_i)$ denotes the number of users that u_i trusts, and $d^{-}(u_k)$ denotes the number of users who trust user u_k.

3.5 Listwise Learning-to-rank Algorithm

Listwise LTR algorithm can directly optimize the final recommendation list, so as to avoid the inaccuracy caused by generating recommendation list based on the descending order of ratings. To employ listwise LTR algorithm, we need to take the cross entropy between the predicted list and correct list as the loss function [5]. By minimizing the loss function, we finally get an optimal ranking model, which outputs a recommendation list more consistent with the user preference. However, listwise LTR algorithm still faces the challenge of data sparsity problem and malicious rating. In this section, we incorporate the rating data and trust relationship into listwise LTR algorithm. Before introducing the specific algorithm, some definitions are given as follows.

Definition 5 (Top-one probability) It is the probability of an item with rating r_{ij} being ranked at the top-one position in the user's recommendation list, which is computed as follows:

$$P_{l_i}(r_{ij}) = \frac{\exp(r_{ij})}{\sum_{k=1}^{n}\exp(r_{ik})} \tag{13}$$

Where l_i denotes the recommendation list of user u_i, and n is the number of items in the recommendation list.

On the one hand, we get the correct list according to the rating and trust relationship provided by the training dataset. On the other hand, we get the predicted list according to the rating and trust relationship predicted by our approach. Based on top-one probability, the cross entropy between the predicted list and the correct list is taken as the loss function, which is as follows:

$$L = -\sum_{i=1}^{m}\sum_{j=1}^{n} I_{ij}^{R} P_{l_i}(r_{ij}) \log_2(\hat{P}_{l_i}(\hat{r}_{ij})) - \lambda_t \sum_{i=1}^{m}\sum_{k=1}^{m} I_{ik}^{T} P_{l_i}(t_{ik}^{*}) \log_2(\hat{P}_{l_i}(\hat{t}_{ik})) + \frac{\lambda}{2}(\|\boldsymbol{U}\|_F^2 + \|\boldsymbol{V}\|_F^2 + \|\boldsymbol{Z}\|_F^2) \tag{14}$$

Where r_{ij} is the rating provided by the training dataset, $\hat{r}_{ij}$ is the predicted rating computed by Eq.(10), and I_{ij}^{R} is an indicator function, which equals 1 if user u_i rated item j, and 0 otherwise. t_{ik}^{*} is the modified trust value computed by Eq.(12), $\hat{t}_{ik}$ is the predicted trust value computed by Eq.(11), and I_{ik}^{T} equals 1 if user u_i trusts user u_k, and 0 otherwise. λ_t is a parameter that controls the importance of trust regularization, and λ is the regularization coefficient used to reduce over-fitting. $\|\bullet\|_F^2$ denotes the Frobenius norm.

Having formulated the non-convex loss function as shown in Eq.(14), the next step is the training process. Compute the gradient of each latent vector, such

as $\boldsymbol{U}_i$、$\boldsymbol{V}_j$、$\boldsymbol{Z}_k$ defined in Section **3.1**, and learn them by stochastic gradient descent as shown in Eq.(15), from which the local minimum of the cross-entropy loss function can be obtained.

$$\begin{aligned}
\boldsymbol{U}_i &\leftarrow \boldsymbol{U}_i - \eta \cdot \frac{\partial L}{\partial \boldsymbol{U}_i}, \quad \frac{\partial L}{\partial \boldsymbol{U}_i} = \left[\frac{\partial L}{\partial u_{i1}}, \frac{\partial L}{\partial u_{i2}}, \cdots, \frac{\partial L}{\partial u_{id}} \right]^T \\
\boldsymbol{V}_j &\leftarrow \boldsymbol{V}_j - \eta \cdot \frac{\partial L}{\partial \boldsymbol{V}_j}, \quad \frac{\partial L}{\partial \boldsymbol{V}_j} = \left[\frac{\partial L}{\partial v_{j1}}, \frac{\partial L}{\partial v_{j2}}, \cdots, \frac{\partial L}{\partial v_{jd}} \right]^T \\
\boldsymbol{Z}_k &\leftarrow \boldsymbol{Z}_k - \eta \cdot \frac{\partial L}{\partial \boldsymbol{Z}_k}, \quad \frac{\partial L}{\partial \boldsymbol{Z}_k} = \left[\frac{\partial L}{\partial z_{k1}}, \frac{\partial L}{\partial z_{k2}}, \cdots, \frac{\partial L}{\partial z_{kd}} \right]^T
\end{aligned} \tag{15}$$

Where η denotes the learning rate. Repeat the iteration and compute the MAE of loss function *L*. When the MAE is less than the predefined threshold ε or the maximum iterations *t-max* is reached, the training process is completed. By sorting items according to the descending order of their top-one probability, we can get a recommendation list more consistent with the user preference.

Algorithm 2. listwise learning-to-rank algorithm

Input:

training dataset; calculated user similarity by **Algorithm 1**, learning rate η; maximum iterations *t-max;* $\lambda_t, \lambda, \alpha$ *;*

Output:

the recommendation list for each user;

1. randomly initialize ***U***, ***V***, ***Z***
2. compute the initial loss function *L* by Eq.(14)
3: for *t*←1 to *t-max* do
4. update $\boldsymbol{U}_i$、$\boldsymbol{V}_j$、$\boldsymbol{Z}_k$ by Eq.(15)
5. record the loss function in the last iteration as *L*
6: for each user $u_i \in \boldsymbol{U}$
7: for each item $j \in \boldsymbol{V}$
8: update the predicted rating $\hat{r}_{ij}$ by Eq.(10)
9: end for
10: end for
11: for each user $u_i \in \boldsymbol{U}$
12: for each user $u_k \in \boldsymbol{U}, k \neq i$
13: update the predicted trust value $\hat{t}_{ik}$ by Eq.(11)
14: end for
15: end for
16: update the loss function L' by Eq.(14)
17: *if* $(\|L' - L\| < \varepsilon)$ then

18: break;
19: $t \leftarrow t+1$
20: end for //the iteration completes
21: for each user $u_i \in \boldsymbol{U}$
22: sort items according to the descending order of their top-one probability
23: return the recommendation list.
24: end for

3.6 Complexity Analysis

Evaluating the complexity comprises the computation process of ListSo: (1) The computational complexity of user similarity is $O(\frac{mn_1!}{(n_1-k)!})$, where m is the number of users, n_1 is the average number of items that are commonly rated by two users, and k denotes the parameter in the top-k model. (2) Computing the loss function L is of complexity $O(|R|d+|R|m_1d+|T|d+|T|m_2d)$, where $|R|$ is the number of observed ratings in matrix R, $|T|$ is the number of observed trust relationship in matrix T, d is the dimensionality of latent vector, m_1 is the average number of neighbor users, and m_2 is the average number of users that a user trust. (3) Computing the gradients $\frac{\partial L}{\partial U_i}, \frac{\partial L}{\partial V_j}, \frac{\partial L}{\partial Z_k}$ are of complexity $O(|R|d+|R|m_1d+|T|d+|T|m_2d)$, $O(|R|d+|R|n_2m_1d)$, $O(|T|d+|T|m_2d)$, respectively, where n_2 is the average number of items that a user rated. (4) Computing the top-one probability of the items to be predicted is of complexity $O(n_3d+n_3m_1d)$, where n_3 is the number of items to be predicted.

Actually, the value of k is usually small, which will be illustrated in experiments. So it can be inferred that the total computational complexity has a linear correlation with the number of users, the number of items to be predicted and the number of observed data in matrix R and T. Therefore, the proposed approach can be applied to large-scale dataset.

4. Experiments

4.1 Dataset

During our experiments, we adopt two public real-world datasets, FilmTrust [25] and Epinion [26]. They contain rating of items and trust relationship between users. The rating of Epinion is an integer between 1 and 5, while the rating of FilmTrust is a number from 0.5 to 4.0 with an interval of 0.5. The details are shown in **Table 2**.

Table 2. Data Statistics

Dataset	Users	Items	Ratings	Density	Trust
FilmTrust	1,508	2,071	35,497	1.1400%	1,853
Epinion	40,163	139,738	664,824	0.0118%	487,183

4.2 Evaluation Metric

Our proposed approach directly optimizes the recommendation list by listwise LTR algorithm, so we employ NDCG (Normalized Discounted Cumulative Gain) [27] and ERR (Expected Reciprocal Rank) [28], which are widely used in LTR algorithms, as the criteria to measure the recommendation accuracy. The larger the NDCG and ERR are, the higher the recommendation accuracy will be.

NDCG. In recommendation system, the rating can be regarded as the relevance. Assume there are m users, and r_u^j denotes the rating of the item ranked at position j in the recommendation list of user u. $k \in R$ denotes the cutoff threshold. For user u, the $NDCG_u$ is defined as:

$$NDCG_u @ k = \sum_{j=1}^{k} \frac{2^{r_u^j} - 1}{\log_2(j+2)} \tag{16}$$

$NDCG@k$ denotes the average of $NDCG_u @ k$ for all users, defined as

$$NDCG@k = \frac{1}{m} \sum_{u=1}^{m} NDCG_u @ k \tag{17}$$

ERR. In practice, users usually consider recommended items from front to back, until a satisfactory item is found. So the probability that the user considers the item at i-th position is related to the user's satisfaction with those items before i-th position. ERR is used to evaluate the relevance of recommended items.

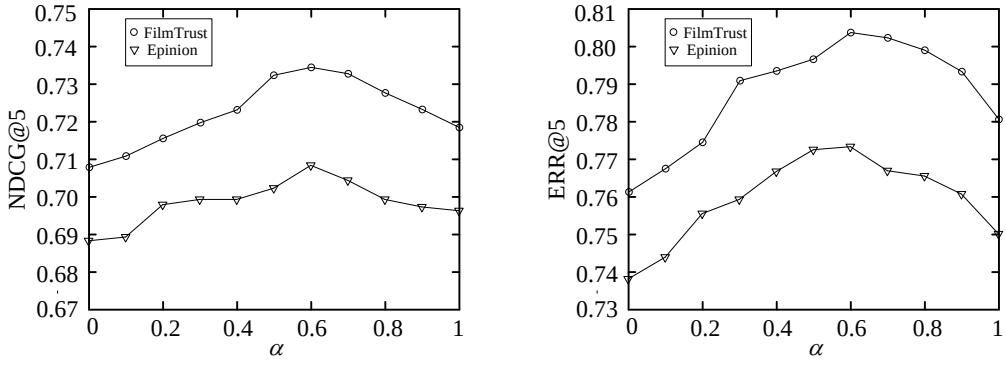
$$\begin{aligned} ERR@k &= \sum_{j=1}^{k} \frac{1}{j} e_j \prod_{l=1}^{j-1} (1 - e_l) \\ e_j &= \frac{2^{r_u^j} - 1}{2^{r_{\max}}} \end{aligned} \tag{18}$$

Where $r_{\max}$ denotes the maximal value of ratings. $r_{\max} = 100$ in most recommendation systems.

4.3 Impact of Parameters

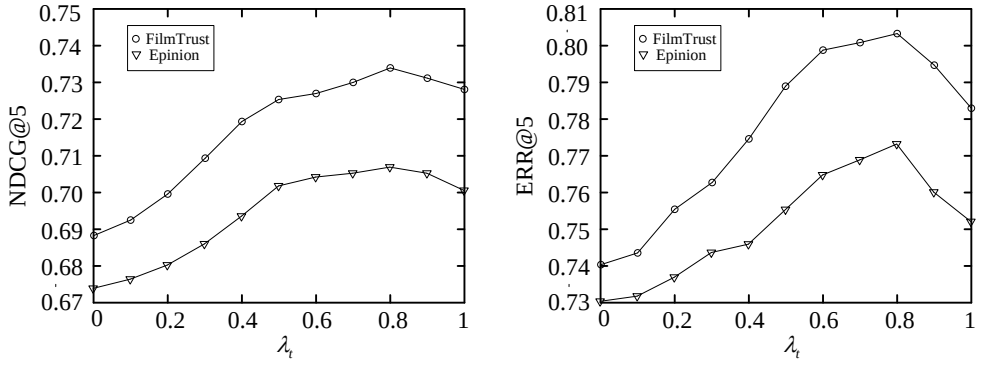
In order to study the impact of parameters, for each user, we randomly select ten items for testing, and reserve the remaining data for training.

According to Eq.(10), parameter α controls the weight of the influence of neighbors in the rating prediction. In order to further explore the influence of α on the recommendation accuracy, we vary the value of α from 0 to 1, with a step value of 0.1, and conduct experiments in the two datasets. **Fig. 4** shows that no matter in which dataset, our approach always achieves the highest recommendation accuracy when $\alpha = 0.6$, indicating that considering the influence of neighbors can effectively improve the recommendation accuracy.



(a) Varying α on NDCG@5 (b) Varying α on ERR@5

Fig. 4. Impact of α on the recommendation accuracy



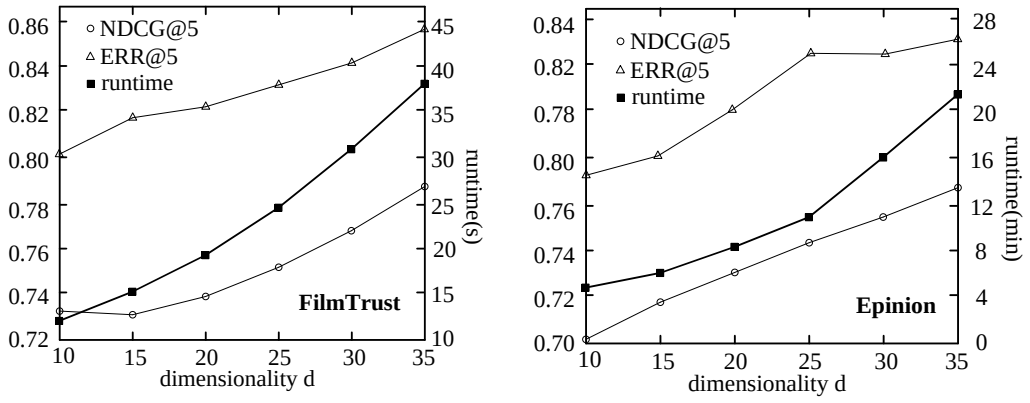
(a) Varying λ_t on NDCG@5 (b) Varying λ_t on ERR@5

Fig. 5. Impact of λ_t on the recommendation accuracy

According to Eq.(14), parameter λ_t controls the importance of trust regularization. We also vary the value of λ_t from 0 to 1, with a step value of 0.1, and conduct experiments in the two datasets. **Fig. 5** shows that our approach always achieves the highest recommendation accuracy when $\lambda_t = 0.8$, indicating that incorporating the trust

relationship can effectively improve the recommendation accuracy.

Parameter *d* is the dimensionality of latent vector. We vary the value of *d* from 10 to 35, with a step value of 5, and record the runtime and recommendation accuracy when testing our approach in the two datasets. Note that the time units in FilmTrust and Epinion datasets are second(s) and minute(min) respectively.



(a) Varying *d* in FilmTrust (b) Varying *d* in Epinion

Fig. 6. Impact of *d*

According to **Fig. 6**, no matter in which dataset, when the value of *d* increases, the runtime of our approach increases much more rapidly than the recommendation accuracy. For example, in FilmTrust dataset, when *d* increases from 10 to 35, the value of NDCG@5 only increases by 7.2%, but the runtime increases by 204%. This shows that just a small value of *d* can ensure relatively high recommendation accuracy in short runtime. Therefore, we set *d=10* in all the experiments.

4.4 Runtime Comparison

The top-*k* probability model is used when computing the user similarity. The larger the *k* is, the more accurate the user similarity will be. But according to Section 3.6, the computational complexity of user similarity is $O(\frac{mn_1!}{(n_1-k)!}) \cong O(mn_1^k)$, which means the computational complexity increases with the value of *k* by exponential growth. In order to get the reasonable value of *k*, we let *k*=1,2,3 respectively. For each user, we also randomly select ten items for testing, and reserve the remaining data for training.

Table 3. The influence of *k* for the recommendation performance

Dataset	FilmTrust			Epinion		
	NDCG@5	ERR@5	runtime	NDCG@5	ERR@5	runtime
k=1	**0.732**	**0.804**	**12.5s**	**0.708**	**0.773**	**5.6min**
k=2	0.758	0.837	51.3s	0.736	0.805	17.2min
k=3	0.772	0.851	123s	0.758	0.832	33.3min

According to **Table 3**, no matter in which dataset, with the increase of *k*, the runtime increases much more rapidly than the recommendation accuracy. For example, in FilmTrust dataset, the value of NDCG@5 when *k=2* is 3.6% larger than that when *k=1*, but the runtime increases by 311%. When *k=1*, our approach can already achieve relatively high recommendation accuracy in short runtime. So we set *k=1* in our experiments.

In order to evaluate the efficiency of our approach, we compare our approach ListSo with the following classical recommendation methods:

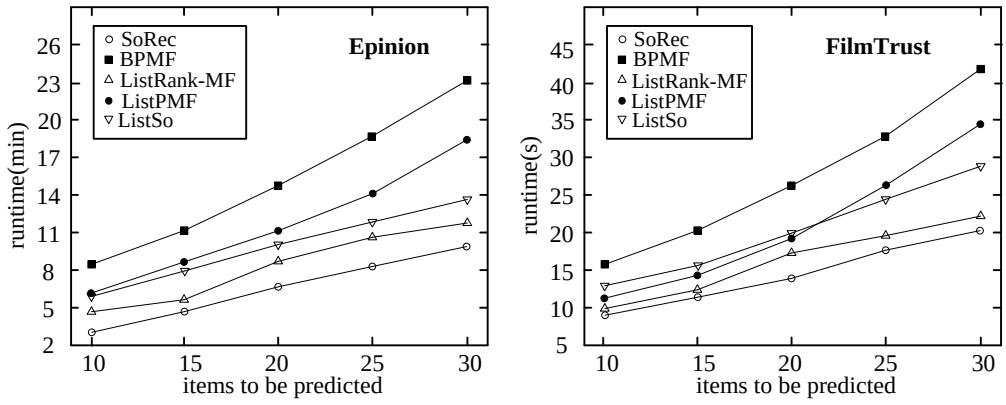
①SoRec [24]: It is a social-aware recommendation method fusing the user-item rating records with the user's social network by probabilistic matrix factorization, which belongs to traditional rating-based CF algorithms.

②BPMF [29]: It is a Bayesian Probabilistic Matrix Factorization algorithm by using Markov chain Monte Carlo (MCMC) methods to estimate the parameters, which belongs to traditional rating-based CF algorithms.

③ListRank-MF [17]: It combines a listwise LTR algorithm with matrix factorization, and gets a recommendation list by minimizing the loss function representing the uncertainty between training lists and output lists. It belongs to LTR-based CF algorithms.

④ListPMF [30]: This method employs an improved probability matrix factorization model to predict the user's preference sequence, and recommends the items by maximizing the posterior probability of the predicted preference sequence for the observed preference sequence. It belongs to LTR-based CF algorithms.

The main parameters of our approach are set as follows: $\alpha = 0.7, \beta = 0.6, d = 10, k = 1$. The other compared methods are set to the optimal parameters described in their original paper. We vary the number of items to be predicted from 10 to 30, with a step value of 5, and conduct experiments in the FilmTrust and Epinion datasets.



(a) Experiment in FilmTrust (b) Experiment in Epinion

Fig. 7. Runtime comparison

According to **Fig. 7**, the runtime of ListSo is shorter than that of BPMF and ListPMF, but longer than that of SoRec and ListRank-MF. The reason is as follows: compared with SoRec, ListSo adds the similarity evaluation based on probability distribution model; compared with ListRank-MF, ListSo adds the trust evaluation. ListSo is a hybrid CF algorithm, involving the similarity evaluation, rating prediction and trust evaluation simultaneously. So ListSo consumes larger amount of computation sources while achieving higher recommendation accuracy. But according to **Fig. 7** and the complexity analysis in Section 3.6, it can be seen that with the increase of the number of items to be predicted, the runtime of ListSo still keeps a linear growth trend, indicating that ListSo can be applied to large-scale datasets.

4.5 Recommendation Accuracy Comparison

In order to evaluate the proposed user similarity computation method, we record the method, which replaces the proposed user similarity computation method with Pearson correlation given in Definition 1, as ListPe. By conducting experiments in FilmTrust and Epinion, we compare the recommendation accuracy of ListSo, ListPe and the other four recommendation methods in Section 4.4, shown as **Table 4**. The experimental procedure is the same as that described in Section 4.4.

Table 4. Experimental results on NDCG@5

items to be predicted	Dataset	SoRec	BPMF	ListRank-MF	ListPe	ListPMF	**ListSo**
n_3=10	FilmTrust	0.694	0.699	0.701	0.703	0.711	**0.732**
	Epinion	0.658	0.665	0.669	0.672	0.679	**0.708**
n_3=15	FilmTrust	0.698	0.704	0.705	0.708	0.713	**0.741**
	Epinion	0.659	0.663	0.671	0.673	0.674	**0.713**
n_3=20	FilmTrust	0.689	0.695	0.693	0.699	0.703	**0.726**
	Epinion	0.647	0.653	0.652	0.656	0.657	**0.694**
n_3=25	FilmTrust	0.714	0.718	0.722	0.724	0.729	**0.763**
	Epinion	0.664	0.668	0.677	0.686	0.681	**0.727**
n_3=30	FilmTrust	0.704	0.715	0.714	0.719	0.720	**0.751**
	Epinion	0.667	0.673	0.678	0.679	0.683	**0.718**

Table 5: Experimental results on ERR@5

items to be predicted	Dataset	SoRec	BPMF	ListRank-MF	ListPe	ListPMF	**ListSo**
n_3=10	FilmTrust	0.752	0.761	0.763	0.771	0.775	**0.804**
	Epinion	0.720	0.729	0.731	0.735	0.743	**0.773**
n_3=15	FilmTrust	0.757	0.762	0.765	0.773	0.782	**0.812**
	Epinion	0.722	0.728	0.732	0.735	0.747	**0.779**
n_3=20	FilmTrust	0.748	0.754	0.759	0.765	0.773	**0.798**
	Epinion	0.709	0.715	0.722	0.723	0.728	**0.761**
n_3=25	FilmTrust	0.767	0.777	0.781	0.784	0.797	**0.826**
	Epinion	0.726	0.734	0.737	0.741	0.753	**0.785**
n_3=30	FilmTrust	0.765	0.772	0.780	0.783	0.794	**0.822**
	Epinion	0.736	0.741	0.744	0.751	0.761	**0.794**

According to **Table 4** and **Table 5**, we can get the following conclusions:

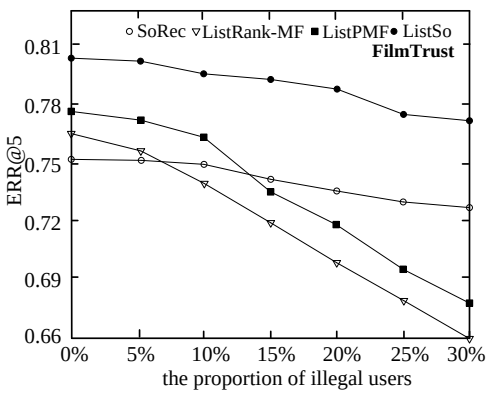
(1) Almost all the methods perform better in FilmTrust than in Epinion. The reason lies in the larger data sparsity of Epinion, which lowers the accuracy of recommendation models.

(2) Compared with ListPe, the recommendation accuracy of ListSo gets improved averagely by more than 4% in FilmTrust and more than 5% in Epinion. This indicates that our proposed user similarity computation method is efficient in improving the recommendation accuracy.

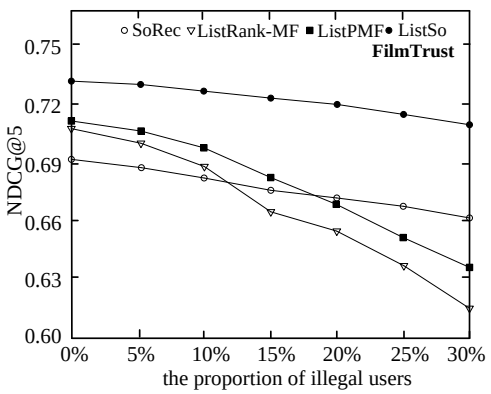
(3) ListSo has the largest value of NDCG@5 and ERR@5, indicating that it outperforms the other five methods in recommendation accuracy. ListSo fully utilizes the ordering information of items to find more accurate neighbors, and employs listwise LTR algorithm to directly optimize the recommendation list, thereby achieving the highest recommendation accuracy.

4.6 The Ability to Resist Malicious Ratings

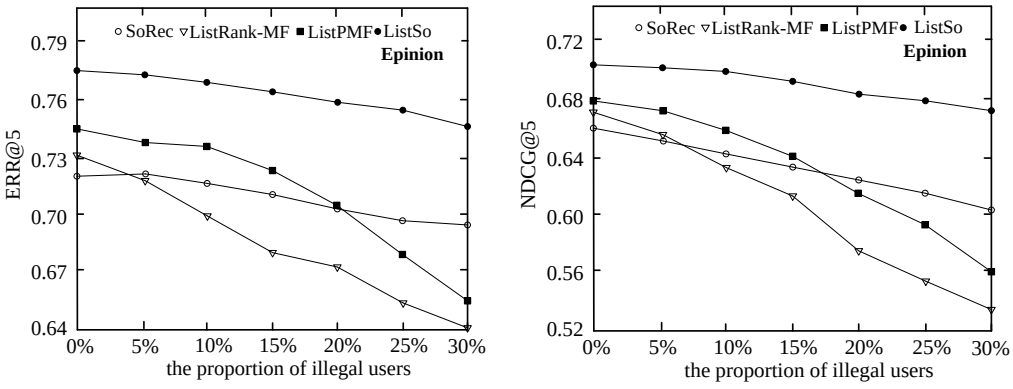
SoRec, ListRank-MF, ListPMF and ListSo all belong to CF algorithms based on neighbor users. In this kind of algorithms, there may be some illegal users giving malicious rating due to the purpose of profit. Once the system regards them as neighbor users, the recommendation accuracy will be severely affected. In order to measure their ability to resist malicious ratings, we compare ListSo with SoRec, ListRank-MF and ListPMF by gradually increasing the proportion of illegal users in the system. The main parameters of our approach are set as follows: $\alpha = 0.7, \beta = 0.6, d = 10, k = 1$. The other compared methods are set to the optimal parameters described in their original paper. For each user, we also randomly select ten items for testing, and reserve the remaining data for training.



(a) Experiment on NDCG@5 in FilmTrust



(b) Experiment on ERR@5 in FilmTrust



(c) Experiment on NDCG@5 in Epinion (d) Experiment on ERR@5 in Epinion

Fig. 8. Anti-attack ability comparison

According to **Fig. 8**, with the increase of illegal users, the recommendation accuracy of ListRank-MF and ListPMF decreases rapidly, because they don't establish any defense mechanism. Once the neighbor user gives malicious ratings, the recommendation accuracy will be severely affected. SoRec fuses the social network while making recommendation, so it has anti-attack ability, but it fails to mine the ordering information of items. ListSo uses the probability distribution model to compute the user similarity, and combines with the social information to recommend the items, so it not only has higher recommendation accuracy, but also can resist malicious ratings.

5. Conclusions

This paper presents a recommendation method based on listwise learning-to-rank by incorporating social information. Firstly, in view of the low accuracy of Pearson correlation, both of the rating and order of items are taken into account by Plackett-Luce model, so as to find more accurate similar users. Secondly, in order to alleviate the data sparsity problem, the improved matrix factorization model by integrating the influence of similar users is proposed to predict the rating. In addition, we also employ the matrix factorization model to explore the trust relationship between users from their social information. Finally, based on the predicted rating and trust relationship, a listwise learning-to-rank algorithm is proposed to learn an optimal ranking model, which can output the recommendation list more consistent with the user preference. Comprehensive experiments conducted on two public real-world datasets show that our approach outperforms previous rating-based CF algorithms and LTR-based CF algorithms.

In the future, more detailed information about users will be taken into account, such as the user's label. We will also consider the influence of time factor, so as to enhance the applicability of our approach in dynamic environment.

References

[1] Zhao Z, Yang Q, Cai D, et al, "Expert finding for community-based question answering via ranking metric network learning," in *Proc. of International Joint Conference on Artificial Intelligence*, pp. 3000-3006, July 9-15, 2016. Article (CrossRef Link).

[2] Liu J, Tang M, Zheng Z, et al, "Location-aware and personalized collaborative filtering for web service recommendation," *International Journal of Computer Engineering In Research Trends*, vol. 3, no. 5, pp. 356-360, 2016. Article (CrossRef Link).

[3] Hu Y, Peng Q, Hu X, et al, "Time aware and data sparsity tolerant web service recommendation based on improved collaborative filtering," *IEEE Transactions on Services Computing*, vol. 8, no. 5, pp. 782-794, 2015. Article (CrossRef Link).

[4] Liu N N, Yang Q, "EigenRank: a ranking-oriented approach to collaborative filtering," in *Proc. of International ACM SIGIR Conference on Research and Development in Information Retrieval ACM*, pp. 83-90, July 20-24, 2008. Article (CrossRef Link).

[5] Grotov A, Rijke M D, "Online learning to rank for information retrieval: SIGIR 2016 Tutorial," in *Proc. of International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 1215-1218, July 17-21, 2016. Article (CrossRef Link).

[6] Zhao Z, Zhang L, He X, et al, "Expert finding for question answering via graph regularized matrix completion," *IEEE Transactions on Knowledge & Data Engineering*, vol.27, no.4, pp. 993-1004, 2015. Article (CrossRef Link).

[7] Lei Y, Wang Z, Meng L, et al, "Clustering and recommendation for semantic web service in time series," *KSII Transactions on Internet And Information Systems*, vol.8, no.8, pp. 2743-2762, 2014. Article (CrossRef Link).

[8] Kang G, Tang M, Liu J, et al, "Diversifying web service recommendation results via exploring service usage history," *IEEE Transactions on Services Computing*, vol. 9, no. 4, pp. 566-579, 2016. Article (CrossRef Link).

[9] Ricci F, Rokach L, Shapira B, "Introduction to recommender systems handbook," *Recommender Systems Handbooks*, vol.22, no.1, pp. 1-4, 2011. Article (CrossRef Link).

[10] Wang Y B, Meng X W, Hu X, "Information aging-based collaborative filtering recommendation algorithm," *Journal of Electronics & Information Technology*, vol. 35, no. 10, pp. 2391-2396, 2013. Article (CrossRef Link).

[11] Jia D, Zhang F, "A collaborative filtering recommendation algorithm based on double neighbor choosing strategy," *Journal of Computer Research & Development*, vol.50, no.5, pp. 1076-1084, 2013. Article (CrossRef Link).

[12] Cong L, Liang C, Li M, "A collaborative filtering recommendation algorithm based on domain nearest neighbor," *Journal of Computer Research & Development*, vol.45, no.9, pp. 1532-1538, 2008. Article (CrossRef Link).

[13] Guo H Y, Liu G S, Su B, et al, "Collaborative filtering recommendation algorithm combining community structure and interest clusters," *Journal of Computer Research and Development*, vol.53, no.8, pp. 1664-1672, 2016. Article (CrossRef Link).

[14] Jamali M, Ester M, "A matrix factorization technique with trust propagation for recommendation in social networks," in *Proc. of ACM Conference on Recommender Systems*, pp. 135-142, September 26-30, 2010. Article (CrossRef Link).

[15] Yang X, Steck H, Liu Y, "Circle-based recommendation in online social networks," in *Proc. of ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. pp. 1267-1275, August 12-16, 2012. Article (CrossRef Link).

[16] Pan W, Zhong H, Xu C, et al, "Adaptive bayesian personalized ranking for heterogeneous implicit feedbacks," *Knowledge-Based Systems*, vol.73, no.1, pp. 173-180, 2015. Article (CrossRef Link).

[17] Shi Y, Larson M, Hanjalic A, "List-wise learning to rank with matrix factorization for collaborative filtering," in *Proc. of ACM Conference on Recommender Systems*, pp. 269-272, September 26-30, 2010. Article (CrossRef Link).

[18] Weimer M, Karatzoglou A, Le Q V, et al, "COFI RANK, maximum margin matrix factorization for collaborative ranking," in *Proc. of International Conference on Neural Information Processing Systems*, pp. 1593-1600, December 03-06, 2007. Article (CrossRef Link).

[19] Huang S, Wang S, Liu T Y, et al, "Listwise collaborative filtering," in *Proc. of International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 343-352, August 09-13, 2015. Article (CrossRef Link).

[20] Mollica C, Tardella L, "Bayesian mixture of Plackett-Luce models for partially ranked data," *Statistics*, 2015. Article (CrossRef Link).

[21] Cao Z, Qin T, Liu T Y, et al, "Learning to rank: from pairwise approach to listwise approach," in *Proc. of International Conference on Machine Learning*, pp. 129-136, June 20-24, 2007. Article (CrossRef Link).

[22] Morita C, Tsukimoto H, "Knowledge discovery from numerical data," *Knowledge-Based Systems*, vol.10, no.7, pp. 413-419, 1998. Article (CrossRef Link).

[23] Zheng X, Luo Y, Xu Z, et al, "Tourism destination recommender system for the cold start problem," *KSII Transactions on Internet And Information Systems*, vol.10, no.7, pp. 3192-3212, 2016. Article (CrossRef Link).

[24] Ma H, Yang H, Lyu M R, et al, "SoRec: social recommendation using probabilistic matrix factorization," in *Proc. of ACM Conference on Information & Knowledge Management*, pp. 931-940, October 26-30, 2008. Article (CrossRef Link).

[25] Barinova O, Lempitsky V, Kholi P, "On detection of multiple object instances using hough transforms," *IEEE Transactions on Pattern Analysis & Machine Intelligence*, vol. 34, no. 9, pp. 1773-1784, 2012. Article (CrossRef Link).

[26] Oren M, Papageorgiou C, Sinha P, et al, "Pedestrian detection using wavelet templates," in *Proc. of 1997 Conference on Computer Vision and Pattern Recognition*, pp. 193, June 17-19, 1997. Article (CrossRef Link).

[27] Kalervo Järvelin and et al, "IR evaluation methods for retrieving highly relevant documents," in *Proc. of International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 41-48, July 24-28, 2000. Article (CrossRef Link).

[28] Chapelle O, Metlzer D, Zhang Y, et al, "Expected reciprocal rank for graded relevance," in *Proc. of ACM Conference on Information and Knowledge Management*, pp. 621-630, November 02-06, 2009. Article (CrossRef Link).

[29] Salakhutdinov R, Mnih A, "Bayesian probabilistic matrix factorization using markov chain monte carlo," in *Proc. of International Conference on Machine Learning*, pp. 880-887, July 05-09, 2008. Article (CrossRef Link).

[30] Liu J, Wu C, Xiong Y, et al, "List-wise probabilistic matrix factorization for recommendation," *Information Sciences*, vol. 278, pp. 434-447, 2014.
Article (CrossRef Link).

Chen Fang received the B.S. degree in electronic science and technology from Zhengzhou information science and technology institute, Henan, China, in 2015. He is pursuing the M.S. degree in information security at Zhengzhou information science and technology institute. His research interests include cloud service management and data mining.

Hengwei Zhang received the B.S., M.S. and Ph.D degrees in information security from Zhengzhou information science and technology institute, Henan, China, in 2000, 2006, and 2015, respectively. He is now an associate professor at the Zhengzhou information science and technology institute. His research interests include information security and cloud computing.

Ming Zhang received the B.S. degree in electronic science and technology from Zhengzhou information science and technology institute, Henan, China, in 2016. He is pursuing the M.S. degree in information security at Zhengzhou information science and technology institute. His research interests include cloud resource scheduling and trust evaluation.

Jindong Wang received the B.S., M.S. degrees in information security from Zhengzhou information science and technology institute, Henan, China, in 1983, 1990 respectively. He is now a professor at the Zhengzhou information science and technology institute. His research interests include cloud computing, cyberspace security, game theory. He has published 40 papers and 1 book in these areas.