Listwise Collaborative Filtering

Shanshan Huang\(^1\)∗
huangshanshansdu
@163.com
Jun Ma\(^1\)
majun@sdu.edu.cn
Zhumin Chen\(^1\)
chenzhumin@sdu.edu.cn
Jari Veijalainen\(^2\)
jari.veijalainen@jyu.fi

\(1\)School of Computer Science and Technology, Shandong University, Jinan, China
\(2\)Department of Computer Science and Information Systems, University of Jyväskylä, Jyväskylä, Finland
\(3\)Microsoft Research Asia, Beijing, China

ABSTRACT

Recently, ranking-oriented collaborative filtering (CF) algorithms have achieved great success in recommender systems. They obtained state-of-the-art performances by estimating a preference ranking of items for each user rather than estimating the absolute ratings on unrated items (as conventional rating-oriented CF algorithms do). In this paper, we propose a new ranking-oriented CF algorithm, called ListCF. Following the memory-based CF framework, ListCF directly predicts a total order of items for each user based on similar users’ probability distributions over permutations of the items, and thus differs from previous ranking-oriented memory-based CF algorithms that focus on predicting the pairwise preferences between items. One important advantage of ListCF lies in its ability of reducing the computational complexity of the training and prediction procedures while achieving the same or better ranking performances as compared to previous ranking-oriented memory-based CF algorithms. Extensive experiments on three benchmark datasets against several state-of-the-art baselines demonstrate the effectiveness of our proposal.

Categories and Subject Descriptors
H.3.3 [Information Storage and Retrieval]: Information Search and Retrieval—Information filtering

Keywords
Recommender systems; Collaborative filtering; Ranking-oriented collaborative filtering

1. INTRODUCTION

In recent years, recommender system has become a key technology behind e-commerce to help customers find their desirable products [14, 22]. As one of the most successful recommendation techniques, collaborative filtering (CF) avoids the necessity of collecting extensive content information about items and users by fully utilizing the user-item rating matrix to make recommendations. As a result, CF can be easily applied to different recommender systems without requiring additional domain knowledge [15]. In the literature, many CF algorithms have been proposed, which roughly fall into two categories: rating-oriented CF and ranking-oriented CF.

Rating-oriented CF, such as user-based CF [1, 5], predicts a user’s potential ratings on unrated items based on the rating information from other similar users. The similarity between two users is calculated based on their rating scores on the set of commonly rated items. A popular similarity measure is the Pearson correlation coefficient [5]. Since this technique focuses on predicting each user’s rating on an unrated item, we refer to it as pointwise CF. Please note that pointwise CF might not be able to achieve the ultimate goal of recommender systems, which is to present a ranking or recommendation list to a user rather than predict the absolute value of the ratings [25]. In fact, it has been observed that improvements in pointwise error metrics such as RMSE (Root Mean Squared Error) do not necessarily lead to improved ranking effectiveness [4].

Different from rating-oriented CF, ranking-oriented CF algorithms [18] directly generate a preference ordering of items for each user without the intermediate step of rating prediction. Related works include memory-based algorithms such as EigenRank [15] and VSRank [27, 28], and model-based algorithms such as CoFiRank [29], ListRank-MF [25] and CCF [30]. In this paper, we focus on memory-based CF algorithms since they have demonstrated many advantages such as strong robustness, interpretability, and competitive performance [6].

Existing memory-based ranking-oriented CF algorithms [15, 27, 28] can also be referred to as pairwise ranking-oriented CF, since they predict pairwise ordering of items based on the correlation between users’ pairwise preferences of the items. A commonly used correlation measure is the Kendall’s \(\tau\) correlation coefficient [17]. Although pairwise ranking-oriented CF has demonstrated significant improvement over previous algorithms in terms of ranking accuracy, it suffers from high computational complexity. Suppose there are \(N\) items, and each user has rated \(n\) items on
average. The average number of commonly rated items for each pair of users is $\pi$, and the number of the neighboring users is $l$. Then the time complexity of computing the correlation between each pair of users is $O(n + \pi^2)$, and the time complexity of the ranking prediction procedure is $O(l(N - n)^2)$ for each user. In most cases, the majority of the items are unrated for each user, resulting in a very large value of $N - n$. As a consequence, the computations of the pairwise ranking-oriented CF algorithms are usually much more expensive than the pointwise ranking-oriented CF algorithms.

Beside the computational complexity, there could be problems with the accuracy of pairwise ranking-oriented CF algorithms too. First, Kendall’s $\tau$ correlation coefficient ignores the ranking positions of the items, which may lead to inaccurate similarities when discovering similar users. Second, Kendall’s $\tau$ correlation coefficient cannot handle the equal preferences tie, which may cause information loss in training and prediction. Third, pairwise ranking-oriented CF algorithms attempt to predict relative preferences between pairs of items rather than the total rankings, which may also result in accuracy loss due to rank aggregation.

To tackle the aforementioned problems with the pairwise CF algorithms, we propose ListCF, a listwise memory-based ranking-oriented CF algorithm, which can reduce the time complexity of both the training and prediction phases while maintaining or even improving the prediction accuracy as compared to conventional pairwise ranking-oriented CF algorithms. ListCF directly predicts a ranking list of items for the target user based on the probability distributions over the permutations of items and thus avoids the accuracy loss caused by the prediction and aggregation of pairwise preferences.

In particular, ListCF utilizes the Plackett-Luce model [17], a widely used permutation probability model in the literature of learning to rank, to represent each user as a probability distribution over the permutations of rated items. In this way, both the ranking positions and equal preferences can be well considered. In ListCF, the similarity between two users is measured based on the Kullback-Leibler (KL) divergence [11] between their probability distributions over the set of commonly rated items. Then for each user, those users who have higher similarity scores are selected as the set of neighborhood users. Given a target user and the neighborhood users, ListCF infers predictions by minimizing the cross entropy loss between their probability distributions over permutations of items with gradient descent. Extensive experiments on three benchmark datasets in comparison with several state-of-the-art algorithms demonstrate the advantages of our approach.

The rest of the paper is organized as follows. Section 2 reviews the related work. Section 3 formulates the listwise collaborative filtering paradigm and presents the details of our algorithm. Section 4 analyzes the relationships between our listwise CF approach and conventional memory-based CF approaches. Section 5 reports the experimental results and analysis, and Section 6 concludes the paper.

2. RELATED WORK

2.1 Recommender Systems

In recent years, many recommendation algorithms have been proposed, which fall into content-based, collaborative filtering (CF) and hybrid algorithms. Content-based algorithms make recommendations based on similarities between the vector representing the user profile and the vectors representing the items, where the vectors for user profile and items are weighted explicit features [16]. CF avoids the necessity of collecting extensive content information about items and users by fully utilizing the user-item rating matrix to make recommendations. Thus, CF can be easily adopted in different recommender systems without requiring additional domain knowledge [15]. Given the effectiveness and convenience, many CF algorithms have been proposed, which are either rating-oriented CF or ranking-oriented CF. In addition, hybrid recommendation algorithms have also been proposed to combine two or more recommendation techniques to gain better performance [2].

In this paper, we focus on CF recommendation algorithms and propose a listwise CF algorithm which demonstrates advantages in recommendation efficiency and accuracy.

2.2 Rating-oriented CF

Traditional rating-oriented CF predicts the absolute value of ratings of individual users to unrated items, thus we refer to it as pointwise CF. Conventionally, pointwise rating-oriented CF can be either memory-based or model-based.

In memory-based rating-oriented algorithms, two representatives are user-based CF [5] and item-based CF [14, 23], which make predictions by utilizing historical ratings associated with similar users or similar items. The commonly used similarity metrics are Pearson correlation coefficient [5] and cosine similarity [1].

Model-based rating-oriented CF learns a model based on the observed ratings to make rating predictions. Latent factor models such as matrix factorization (MF) [9] that uses dimensionality reduction techniques have become important research directions in this area. A variety of adaptations of the MF model have demonstrated promising prediction performance, such as Probabilistic Matrix Factorization (PMF) [19], Non-negative Matrix Factorization (NMF) [12], and Max-Margin Matrix Factorization (MMMF) [21].

2.3 Ranking-oriented CF

Different from rating-oriented CF, ranking-oriented CF [18] directly predicts a preference ranking of items for each user. Similarly, ranking-oriented CF algorithms can be also categorized into memory-based ranking-oriented CF [15, 27] and model-based ranking-oriented CF [10, 20, 25].

EigenRank [15] is a classic memory-based ranking-oriented CF algorithm, which measures the similarity between users based on the correlation between their pairwise rankings of the co-rated items and ranks items by the preferences of similar users. VSRank [27] adopts the vector space model in representing user pairwise preferences and makes further improvement by considering the relative importance of each pairwise preference. Resulting from the pairwise preference based representations and predictions, we refer to EigenRank and VSRank as pairwise memory-based ranking-oriented CF algorithms. Although they achieve better performance in recommendation accuracy than rating-oriented CF, they suffer from serious efficiency problem. In this paper, we propose a listwise memory-based ranking-oriented CF algorithm to reduce the computational complexity while maintaining or even improving the ranking performance.
A variety of model-based ranking-oriented CF algorithms have been presented by optimizing ranking-oriented objective functions, e.g., bayesian personalized ranking (BPR) [20], CLiMF [24], CoFiRank [29], and ListRank-MF [25]. BPR and CLiMF model the binary relevance data and optimize binary relevance metrics, i.e., Area Under the Curve (AUC) in BPR and Mean Reciprocal Rank (Mean) in CLiMF, which are not suited for graded relevance data. CoFiRank [29] minimizes a convex upper bound of the Normalized Discounted Cumulative Gain (NDCG) [7, 8] loss through matrix factorization while ListRank-MF [25] integrates the learning to rank technique into the matrix factorization model for top-N recommendation.

3. LISTWISE COLLABORATIVE FILTERING

In this section, we propose ListCF, a listwise ranking-oriented collaborative filtering (CF) algorithm based on the memory-based CF framework. Generally, memory-based CF works in two phases [28]: The similarity calculation phase (Phase I) calculates the similarities between pair of users, based on which a set of most similar neighborhood users can be discovered for each user; Then the ranking prediction phase (Phase II) predicts the preference ordering of unrated items for each target user by aggregating preferences of neighborhood users for recommendation purpose.

3.1 Phase I: Similarity Calculation

In ListCF, we utilize Plackett-Luce model [17], a widely used permutation probability model, to represent each user as a probability distribution over the permutations of rated items. It is assumed that given a ranking function on a set of items, any permutation on the item set is possible. However, different permutations may have different probabilities, and more "correct" permutations where items with larger probabilities are ranked higher are assigned with larger probabilities.

Let I be a set of items. A permutation of I is a bijection from I to itself and can be represented as an ordered list \( \pi = (\pi_1, \pi_2, \ldots, \pi_n) \), where \( \pi_i \in I \) denotes the item at position i and \( \pi_i \neq \pi_j \) if \( i \neq j \). The set of all the possible permutations of I is denoted as \( \Omega^I \).

**Definition 1. (Probability of permutation).** Let \( \phi(\cdot) \) be an increasing and strictly positive function. Given a permutation \( \pi = (\pi_1, \pi_2, \ldots, \pi_n) \) and the rating scores of the items \( \{r_{\pi_1}, r_{\pi_2}, \ldots, r_{\pi_n}\} \), the probability of \( \pi \) is defined as follows:

\[
P(\pi) = \prod_{i=1}^{n} \frac{\phi(r_{\pi_i})}{\phi(r_{\pi_k})}.
\]

In this paper, we use the exponential function \( \phi(r) = e^r \).

For a set of n items, the number of different permutations is \( n! \). Thus, it is too time-consuming to calculate the probabilities of all the permutations. To cope with this problem, an alternative solution is the top-k probability model [3] \((1 \leq k \leq n)\), which only focuses on the permutations of items within the top-k positions.

**Definition 2. (Top-k Permutation Set).** Given a set of items I, the top-k permutation set \( G_k(i_1, i_2, \ldots, i_k) \) of I is a set containing all the permutations of I which share the same top-k items, i.e., \( i_1, i_2, \ldots, i_k \) where \( i_1, i_2, \ldots, i_k \in I \). Formally,

\[
G_k(i_1, i_2, \ldots, i_k) = \{ \pi | \pi \in \Omega^I, \pi_j = i_j, j = 1, \ldots, k \}.
\]

Two top-k permutation sets of I are different if the items in the top-k positions are different. Let \( G_k^I \) denote the set of different top-k permutation sets for I. Please note that the cardinality of \( G_k^I \) is \( n!/(n-k)! \), and the cardinality of \( G_k^I \) is \( n!/(n-k)! \).

**Example 1.** Let \( I = \{i_1, i_2, i_3\} \). There are totally \( 3! = 6 \) possible permutations in \( \Omega^I \): \( \langle i_1, i_2, i_3 \rangle, \langle i_1, i_3, i_2 \rangle, \langle i_2, i_1, i_3 \rangle, \langle i_2, i_3, i_1 \rangle, \langle i_3, i_1, i_2 \rangle, \text{and} \langle i_3, i_2, i_1 \rangle \). There are merely \( 3!/2! = 3 \) top-1 permutation sets: \( G_1^I = \{G_1(i_1), G_1(i_2), G_1(i_3)\} \), where \( G_1(i_1) = \{(i_1, i_2, i_3), (i_1, i_3, i_2), (i_2, i_1, i_3), (i_2, i_3, i_1), (i_3, i_1, i_2), (i_3, i_2, i_1)\} \).

Based on the Definition 2, the probability of the top-k permutation \( P(G_k(i_1, i_2, \ldots, i_k)) \) equals to the sum of the probabilities of permutations in which the items \( i_1, i_2, \ldots, i_k \) are ranked in the top-k positions. To calculate all the probabilities of top-k permutation sets in \( G_k^I \), we need consider \( n!/(n-k)! \) top-k permutation sets, each containing \( (n-k)! \) permutations of I. Therefore we still have to calculate \( n!/(n-k)! \cdot (n-k)! = n! \) probabilities of permutations in total. In [3], an efficient way to calculate the probability of the top-k permutation set \( P(G_k(i_1, i_2, \ldots, i_k)) \) is shown in Lemma 1.

**Lemma 1.** The probability of the top-k permutation set \( P(G_k(i_1, i_2, \ldots, i_k)) \) equals to the probability of that the items \( i_1, i_2, \ldots, i_k \) are ranked in the top-k positions respectively. Formally,

\[
P(G_k(i_1, i_2, \ldots, i_k)) = \prod_{j=1}^{k} \frac{\phi(r_{\pi_j})}{\sum_{l=1}^{n} \phi(r_{\pi_l})}, \forall j = 1, \ldots, k : \pi_j = i_j
\]

where \( r_{\pi_j} \) is the rating score of the item which is ranked in position \( l, l = 1, 2, \ldots, n \).

Obviously, the probabilities of the top-k permutation sets in \( G_k^I \) form a probability distribution, which can be calculated much more efficiently than the probability distribution over \( G^I \). In this study, we refer to the probability distribution over the top-k permutation sets as the top-k probability model, which uses the top-k permutation sets instead of the full permutations of all the items.

In recommender systems, for each pair of users \( u \) and \( v \), let \( I_{u,v} \) be the set of their commonly rated items. Then \( G_{k,u,v}^I \) represents the set of top-k permutation sets of \( I_{u,v} \). We use \( P_u \) and \( P_v \) to denote the probability distributions of \( u \) and \( v \) over \( G_{k,u,v}^I \) according to the top-k probability model, which can be calculated with Equation (1) based on their rating scores. Thus the similarity between user \( u \) and \( v \) can be estimated as the similarity between the probability distributions of \( P_u \) and \( P_v \).

In this study, we use Kullback-Leibler (KL) divergence [11] based metric for similarity calculation, since it is a common measure of the difference between two probability distributions in probability theory and information theory. Given two users \( u \) and \( v \), the KL divergence of \( P_u \) from \( P_v \)
is defined as follows:

$$D_{KL}(P_u||P_v) = \sum_{g\in G_{u,v}^k} P_u(g) \log_2 \left( \frac{P_u(g)}{P_v(g)} \right).$$

where $P_u(g)$ and $P_v(g)$ is the probability of the top-$k$ permutation set $g \in G_{u,v}^k$ for $u$ and $v$ respectively. $D_{KL}(P_u||P_v)$ can be calculated in the similar way. Obviously, the KL divergence is asymmetric, i.e., $D_{KL}(P_u||P_v) \neq D_{KL}(P_v||P_u)$.

In this study, we define a symmetric similarity metric based on the KL divergence, which is shown as follows:

$$s(u,v) = 1 - \frac{1}{2} \left[ D_{KL}(P_u||P_v) + D_{KL}(P_v||P_u) \right].$$  \hspace{1cm} (2)

The similarity metric in Equation (2) suffers from computing high similarity between users with few ratings in common. Like in [5], this can be alleviated by multiplying the similarity function with $\min(|I_{u,v}|/c,1)$, where $c$ is a threshold. When the number of commonly rated items falls below this threshold, the similarity is defined as follows:

$$s(u,v) = 1 \quad \text{if} \quad |I_{u,v}| < c.$$  \hspace{1cm} (3)

By definition, the similarity between each pair of users is no greater than 1. Formally, $\forall u,v \in U : s(u,v) \leq 1$. The similarity between $u$ and $v$ reaches the maximum bound ($s(u,v) = 1$) only if their probability distributions $P_u$ and $P_v$ over the top-$k$ permutation sets are the same. We also note that $s(u,v)$ may be negative and has no lower bound. However, this is not an important issue in our model because we are just interested in discovery of limited number of users with high similarities to the target user as the neighborhood users.

### 3.2 Phase II: Ranking Prediction

For a target user $u$, let $N_u$ be the set of neighborhood users and $T_u = \{t_1,t_2,\ldots,t_k\}$ be the set of items to be predicted. The ultimate goal of ranking-oriented CF is to predict a preference ranking of items in $T_u$ for $u$.

Let $P_u$ be the probability distribution over the set of top-$k$ permutation sets $G_{u,v}^k$ to be predicted for $u$. In order to guarantee that the sum of the probabilities in $P_u$ is equal to 1, for each top-$k$ permutation set $\forall g \in G_{u,v}^k$, the probability $P_u(g)$ is denoted as follows:

$$\hat{P}_u(g) = \frac{\varphi_{u,g}}{\sum_{g'\in G_{u,v}^k} \varphi_{u,g'}},$$

where $\{\varphi_{u,g} | g \in G_{u,v}^k\}$ are unknown variables to be predicted for user $u$ and $\sum_{g\in G_{u,v}^k} \hat{P}_u(g) = 1$ holds.

In this study, following the same assumption of memory-based CF, we make predictions based on a set of neighborhood users with similar ranking preferences to the target user. If users have similar ranking preferences in the past, they are more likely to have similar ranking preferences in the future, which means that $\hat{P}_u$ should be as close as possible to the probability distributions of the neighborhood users over their sets of top-$k$ permutation sets.

We use the cross entropy loss function to make predictions, as it is widely-used for optimizing similarity/distance between probability distributions in machine learning, and it has been successfully used in the text mining [3], multimedia retrieval [13], and pattern recognition [26] fields.

Given a target user $u$ with a set of unrated items $T_u$, and a neighborhood user $v \in N_u$, let $I_v$ be the set of items that the neighborhood user $v$ has rated, and $T_{u,v} = T_u \cap I_v$. Thus $G_{u,v}^k$ is the set of top-$k$ permutation sets of $T_{u,v}$. Let $P_u'$ and $P_v'$ be the respective probability distributions of $u$ and $v$ over $G_{u,v}^k$. The cross entropy between $\hat{P}_u'$ and $P_v'$ can be calculated as follows:

$$E(\hat{P}_u', P_v') = - \sum_{g\in G_{u,v}^k} P_v'(g) \log_2 (\hat{P}_u'(g)).$$  \hspace{1cm} (4)

Please note that in the above equation, $P_v'$ is different from $P_u$ that is used in similarity calculation. As they are based on different sets of items, where $P_v'$ is based on $T_{u,v}$ while $P_u$ is based on the commonly rated items $I_{u,v} = I_u \cap I_v$ of user $u$ and $v$. Obviously $\sum_{g\in G_{u,v}^k} P_v'(g) = 1$ holds.

ListCF attempts to make predictions by minimizing the weighted sum of the cross entropy loss between the probability distributions of the target user and his neighborhood users over the set of the top-$k$ permutation sets of $T_{u,v}$. For a target user $u$ and the set of unrated items $T_u$, the cross entropy-based loss function is:

$$\arg \min_{\varphi_u} \sum_{v \in N_u} s(u,v) \cdot E(\hat{P}_u', P_v')$$

s.t. $\forall g \in G_{u,v}^k : \varphi_{u,g} \geq 0.$  \hspace{1cm} (5)

According to Equation (2) and Equation (4), the objective function can be specified as follows:

$$F(\varphi_u) = \sum_{v \in N_u} s(u,v) \cdot E(\hat{P}_u', P_v')$$

$$= - \sum_{v \in N_u} s(u,v) \sum_{g \in G_{u,v}^k} P_v'(g) \log_2 (\hat{P}_u'(g))$$

$$= - \sum_{v \in N_u} s(u,v) \sum_{g \in G_{u,v}^k} P_v'(g) \log_2 \left( \sum_{g' \in G_{u,v}^k} \frac{\varphi_{u,g'}}{\varphi_{u,g}} \right)$$

$$= \sum_{v \in N_u} s(u,v) \sum_{g \in G_{u,v}^k} P_v'(g) \left[ \log_2 \left( \sum_{g' \in G_{u,v}^k} \varphi_{u,g'} \right) - \log_2 \left( \varphi_{u,g} \right) \right].$$  \hspace{1cm} (6)

We optimize Equation (6) with the gradient descent method. The derivative of $F$ with respect to $\varphi_{u,g}$ is:

$$\frac{\partial F}{\partial \varphi_{u,g}}$$

$$= \sum_{v \in N_u} \left( \frac{s(u,v)}{\ln 2 \cdot \sum_{g' \in G_{u,v}^k} \varphi_{u,g'}} \right) - \frac{\sum_{v \in N_u} s(u,v) P_v'(g)}{\ln 2 \cdot \varphi_{u,g}}$$

$$= \sum_{v \in N_u} \frac{s(u,v)}{\ln 2 \cdot \sum_{g' \in G_{u,v}^k} \varphi_{u,g'}} - \frac{\sum_{v \in N_u} s(u,v) P_v'(g)}{\ln 2 \cdot \varphi_{u,g}}.$$  \hspace{1cm} (7)

Given a learning rate $\eta$, the update formula of the gradient descent method is:

$$\varphi_{u,g} \leftarrow \varphi_{u,g} - \eta \frac{\partial F}{\partial \varphi_{u,g}}.$$  \hspace{1cm} (8)
Algorithm 1: The ListCF Algorithm

Input: An item set $I$, a user set $U$, and a rating matrix $R \in \mathbb{R}^{M \times N}$. A set of rated items $I_u \subseteq I$ by each user $u \in U$. The maximal number of iterations $maxIteration$ and error threshold $\epsilon$.

Output: A ranking $\hat{r}_u$ of items for each user $u \in U$.

1. for $u \in U$ do
2.   for $v \in U$ and $u \neq v$ do
3.     $P_u, P_v \leftarrow \text{TopKProDist}(I_u, I_v, R)$ /* Eq. 1 */
4.     $\text{sim}(u, v) \leftarrow \text{Similarity}(P_u, P_v)$ /* Eq. 2 */
5.   end
6. $N_u \leftarrow \text{SelectNeighbors}(\{\text{sim}(u, v)\}_{v \in U/u})$
7. end
8. for $u \in U$ do
9.   $t = 1$
10. repeat
11.    $\epsilon = 0$
12.    Initialize($\varphi^0_u$)
13.    for $g \in G^\epsilon_u$ do
14.       $\varphi^t_{u,g} \leftarrow \text{Update}(N_u, \text{sim}, R)$ /* Eq. 8 */
15.    end
16.    $\epsilon^+ = \sqrt{\sum_k (\varphi^t_{u,k} - \varphi^{t-1}_{u,k})^2}$
17.    $t \leftarrow t + 1$
18. until $t > maxIteration$ or $\epsilon < \epsilon$;
19. for $t \in T_u$ do
20.   $P(t) \leftarrow \text{Aggregation}(\{\varphi_{u,g}\}_{g \in G^t_u})$
21. end
22. $\hat{r}_u \leftarrow \text{Ordering}(\{P(t)\}_{t \in T_u})$
23. end

After updating $\varphi_{u,g}$, we can easily get the probabilities of $\hat{P}_u(g)$ according to Equation (3), which meets the constraint of $\sum_{g \in \varphi^t_u} \hat{P}_u(g) = 1$.

In order to obtain an ultimate ranking of $T_u$ for each user $u$, the probabilities of the top-$k$ permutation sets need to be aggregated. One intuitive way is to get the probabilities of the top-$l$ permutation sets and rank the items in $T_u$ according to the descending order of the corresponding probabilities. The probability of the top-$l$ permutation set $P(G^t_u(i_l))$ actually equals to the sum of the probabilities of the top-$k$ ($k > 1$) permutation sets with the item $i_l$ in $T_u$ ranked in the top-$l$ position. Here gives an example.

**Example 2.** Suppose user $u$ has items $I = \{i_1, i_2, i_3\}$ to be recommended. We have probability distribution $\{0.2, 0.3, 0.1, 0.1, 0.1, 0.2\}$ over the set of top-2 permutation sets $G^2_u = \{G_2(i_1, i_2), G_2(i_1, i_3), G_2(i_2, i_1), G_2(i_2, i_3), G_2(i_3, i_1), G_2(i_3, i_2)\}$. We can obtain the probabilities of that each item is ranked in the top-1 position: $P(G_1(i_1)) = 0.2 + 0.3 = 0.5$, $P(G_1(i_2)) = 0.1 + 0.1 = 0.2$, and $P(G_1(i_3)) = 0.1 + 0.2 = 0.3$. Since $P(G_1(i_1)) > P(G_1(i_3)) > P(G_1(i_2))$, the recommendation list is ordered as $(i_1, i_3, i_2)$.

### 3.3 Pseudocode and Discussion

The pseudocode of the ListCF algorithm is shown in Algorithm 1, where lines 1-7 calculate the similarities between users and discover the neighbor users for each target user, and lines 8-23 predict the ranking of items to make recommendations. When $k > 1$ in the top-$k$ probability model, lines 19-21 aggregate the probabilities of the top-$k$ permutation sets to obtain the probabilities of the top-1 permutation sets, that are the probabilities of each item $t \in T_u$ ranked in the first position.

In the proposed ListCF model, we use the top-$k$ ($k \geq 1$) probability model [3] in the similarity calculation and ranking prediction phases. ListCF with different values of $k$ may achieve different recommendation performances. On one hand, generally the accuracy will be higher when a larger $k$ value is used. On the other hand, the computational complexity of similarity calculation and ranking prediction procedures will be exponentially increased. In particular, for a user $u$ associated with $n$ items, the number of permutation sets in the top-$k$ probability model is $n!/(n-k)!$, and specially this number is merely $n$ when $k = 1$. Thus the time complexity of ListCF based on the top-$k$ ($k > 1$) probability model is $(n-1)!/(n-k)!$ times higher than that of ListCF based on the top-1 probability model.

| Table 1: Accuracy and efficiency comparison of ListCF with different $k$. |
|---|---|---|
| $k=1$ | $k=2$ | $k=3$ |
| Accuracy (NDCG@5) | 0.7866 | 0.7953 | 0.8052 |
| Runtime of Phase I | 8s | 71s | 346s |
| Runtime of Phase II | 1.5s | 20.2s | 731.7s |

For illustration purpose, we randomly selected 1000 users from the MovieLens-1M dataset and conducted experiments to compare the ranking accuracy as well as the runtime of the similarity calculation and ranking prediction phases (Phase I and II respectively) of ListCF when $k = 1, 2$ and $3$. The size of neighborhood users was set to be 50, and 50 unrated items were predicted for each user in Phase II. The results are reported in Table 1.

From the table we can see that, when $k > 1$ in ListCF, the improvement in recommendation accuracy is very subtle, but the degeneration of efficiency is extremely significant. For example, ListCF with $k = 3$ achieves about 0.02 improvement in NDCG@5 over ListCF with $k = 1$, but it takes 433 and 488 times longer in Phase (I) and Phase (II) respectively. Obviously, the loss outweighs the gain. Thus we set $k = 1$ when compare ListCF with other recommendation algorithms in the experimental section.

### 4. COLLABORATIVE FILTERING: LISTWISE VS. POINTWISE AND PAIRWISE

#### 4.1 Relationship

Although ListCF, as a listwise memory-based CF algorithm, is completely different from pointwise and pairwise memory-based CF algorithms in similarity calculation and ranking prediction, these three categories of algorithms inherently share similar prediction strategy.

**Theorem 1.** ListCF predicts the probability of a top-$k$ permutation set as the weighted average of the neighbors’ probabilities of the corresponding top-$k$ permutation sets, if
\( \forall v \in N_u : T_{u,v} \equiv T_u \) holds. Formally, for \( \forall g \in G_k^{T_u} \),
\[
\hat{P}_u(g) = \frac{\sum_{v \in N_u} s(u, v) P'_v(g)}{\sum_{v \in N_u} s(u, v)}.
\]

**Proof.** If \( \forall v \in N_u : T_{u,v} \equiv T_u \), then \( \forall v \in N_u : G_k^{T_{u,v}} \equiv G_k^{T_u} \). For the optimization problem in Equation (5), set the derivative of \( F \) (Equation 7) to be 0:
\[
\sum_{v \in N_u} \ln 2 \cdot \sum_{g' \in G_k^{T_{u,v}}} \varphi_{u,g'}^v = \sum_{v \in N_u} \frac{s(u, v) P'_v(g)}{\ln 2 \cdot \varphi_{u,g'}} = 0.
\]

Thus
\[
\sum_{v \in N_u} \frac{s(u, v)}{\ln 2 \cdot \varphi_{u,g'}} = \frac{\sum_{v \in N_u} s(u, v) P'_v(g)}{\sum_{g' \in G_k^{T_{u,v}}} \varphi_{u,g'}} = \hat{P}_u(g).
\]

Theorem 1 presents the prediction formula of ListCF in a special case when all the neighbor users have rated the items in test set \( T_u \) of user \( u \), i.e., \( \forall v \in N_u : T_{u,v} \equiv T_u \) holds. In this case, for \( \forall g \in G_k^{T_u} \), the probability \( \hat{P}_u(g) \) can be predicted as the weighted average of the neighbors’ corresponding probabilities, which is the same as the pairwise preference prediction formula in pairwise memory-based CF [15, 27, 28] as well as the rating prediction formula in the pointwise memory-based CF after performing the normalization of the ratings, i.e., the average rating of the user \( u \) has been deducted from each rating \( r_{u,i} \) of \( u \), formally \( r^{(N)}_{u,i} = r_{u,i} - \tau_u \).

Now we explain why we cannot obtain such a result without the constraint of \( \forall v \in N_u : T_{u,v} \equiv T_u \) in ListCF while we can with the constraint. In pointwise CF and pairwise CF, the ratings and the pairwise preferences of users are constant. However, in listwise CF, the probability distributions of each pair of users \( u \) and \( v \) in the ranking prediction phase vary with respect to the set of items \( T_{u,v} \): (1) Given a target user \( u \), for any two different neighborhood users \( v_1 \) and \( v_2 \), since \( T_{u,v_1} \neq T_{u,v_2} \) and \( G_k^{T_{u,v_1}} \neq G_k^{T_{u,v_2}} \), the length of the probability distribution \( P_u \) of user \( u \) could change and the values in \( P_u \) could be different; (2) Given two different target users \( u_1 \) and \( u_2 \) who have a same neighborhood user \( v \), the probability distribution of \( P_v \) for user \( v \) could also be different. Thus we cannot simply make predictions with a weighted average formula as pointwise and pairwise CF do. However, with the constraint of \( \forall v \in N_u : T_{u,v} \equiv T_u \), \( G_k^{T_{u,v}} \equiv G_k^{T_u} \), the probabilities \( \hat{P}_u(g) \) and \( P'_v(g) (\forall g \in G_k^{T_u}) \) in the ranking prediction formula are invariable, which is the same as those in pointwise CF and pairwise CF. In this case, we can obtain the same result shown in Theorem 1.

**4.2 Advantages of ListCF**

In comparison with conventional pairwise ranking-oriented CF, listwise CF demonstrates many advantages in efficiency (when \( k = 1 \)) and accuracy on account of the listwise representation. On the efficiency side, let \( M \) and \( N \) be the number of users and items. Suppose each user has rated \( n \) items on average, and the number of commonly rated items for each pair of users is \( \Pi \). In the similarity calculation phase, the time complexity of ListCF is \( O(M^2(n + \Pi)) \). Recall that the complexity of pairwise CF is \( O(M^2(n + \Pi^2)) \), which is \( (n + \Pi^2)/(n + \Pi) \) times higher than that of ListCF. Suppose each user has \( t \) neighbor users on average, and each prediction process maximally performs \( d \) iterations for gradient descent optimization. In the ranking prediction phase, the time complexity of ListCF is \( O(M(N - n)d) \), where \( N - n \) is the number of items to recommend. Recall that the complexity of pairwise CF is \( O(M(N - n)^2d) \), which is \( (N - n)/d \) times higher than that of ListCF, and \( d < N - n \) in our experiments. Since \( d \) is a constant, the ratio of the complexity of pairwise CF to that of ListCF increases linearly with the growth of the number of items to be predicted in the ranking prediction phase.

On the accuracy side, firstly, permutation probability based similarity metric considers the ranking position information of items, which can discover more accurate neighbor users with similar preferences. Secondly, permutation probability based representation can easily handle the the case of equal preferences on items, i.e., assigns equal probability to the permutations in which the top items have equal ranking scores. Thirdly, ListCF attempts to make top-\( n \) recommendation by optimizing listwise loss function, which is more natural and accurate. The following example demonstrates the accuracy loss of the Kendall’s \( \tau \) correlation coefficient because of not considering the ranking position information of items.

**Example 3.** Suppose there are three users \( U = \{u_1, u_2, u_3\} \) and three items \( I = \{i_1, i_2, i_3\} \), where \( u_1 \) assigned ratings of \( \{5, 3, 4\} \) to items, \( u_2 \) assigned \( \{5, 4, 3\} \), and \( u_3 \) assigned \( \{4, 3, 5\} \).

In pointwise CF, the similarity (Pearson correlation coefficient) between \( u_1 \) and \( u_2 \) and the similarity between \( u_1 
and \( u_3 \) are \( \rho(u_1, u_2) = \rho(u_1, u_3) = 0.5 \).

In pairwise CF, the similarity (Kendall’s \( \tau \) correlation coefficient) between \( u_1 \) and \( u_2 \) and the similarity between \( u_1 
and \( u_3 \) are \( \tau(u_1, u_2) = \tau(u_1, u_3) = 0.333 \).

In ListCF, according to Equation (1), the top-1 probability distributions of users \( u_1, u_2 \) and \( u_3 \) are \( P_{u_1} = (0.665, 0.090, 0.245) \), \( P_{u_2} = (0.665, 0.245, 0.090) \), and \( P_{u_3} = (0.245, 0.090, 0.665) \). According to Equation (2), \( s(u_1, u_2) = 0.776 \) and \( s(u_1, u_3) = 0.395 \), and thus \( s(u_1, u_2) > s(u_1, u_3) \).

Since recommender systems aim to recommend the items that users may be most interested in, we should emphasize
the items with highest scores in the similarity calculation and ranking prediction phases, and thus $u_3$ is more similar to $u_1$ than $u_3$ in Example 3. Obviously, ListCF can successfully recognize it while pointwise CF and pairwise CF cannot.

5. EXPERIMENTS

5.1 Experimental Settings

5.1.1 Datasets

<table>
<thead>
<tr>
<th></th>
<th>Movielens-1M</th>
<th>EachMovie</th>
<th>Netflix</th>
</tr>
</thead>
<tbody>
<tr>
<td>#users</td>
<td>6,040</td>
<td>36,656</td>
<td>99,884,940</td>
</tr>
<tr>
<td>#items</td>
<td>3,952</td>
<td>1,000,209</td>
<td>429,584</td>
</tr>
<tr>
<td>#ratings</td>
<td>1,000,209</td>
<td>2,580,222</td>
<td>99,884,940</td>
</tr>
<tr>
<td>#ratings/user</td>
<td>165.6</td>
<td>70.4</td>
<td>232.5</td>
</tr>
<tr>
<td>#ratings/item</td>
<td>253.1</td>
<td>1,589.8</td>
<td>5621.0</td>
</tr>
<tr>
<td>sparsity</td>
<td>93.7%</td>
<td>95.7%</td>
<td>98.7%</td>
</tr>
</tbody>
</table>

We conducted a series of experiments on three real-world rating datasets: Movielens-1M, EachMovie, and Netflix. All of these datasets are commonly used benchmark datasets in evaluating recommendation performance. The ratings in Movielens-1M and Netflix are on a scale from 1 to 5, while the EachMovie rating scale is from 1 to 6. In order to guarantee that there are adequate number of ratings per user for training and testing, we removed those users who have rated less than 20 items from these datasets. Several statistics on the resulting datasets are presented in Table 2, where the sparsity levels in the last row are calculated as follows [23]:

\[
\text{Sparsity} = 1 - \frac{\text{#rating scores}}{\text{#users} \times \text{#items}}.
\]

5.1.2 Comparison Algorithms

In our experiments, we mainly compared the performance of ListCF with three memory-based CF algorithms of PointCF\[1\], EigenRank \[15\] and VSRank \[27, 28\], a classic pointwise rating-oriented algorithm and two state-of-the-art pairwise ranking-oriented algorithms respectively. In particular, PointCF uses the Pearson correlation coefficient to calculate the similarity between users, and ranks the items according to the predicted ratings for each user. In EigenRank and VSRank, the greedy aggregation method is used to aggregate the predicted pairwise preferences of items into total ranks. In the memory-based CF algorithms, the number of neighborhood users was set to be 100 in Movielens-1M and EachMovie, and 300 in Netflix. Since ListCF is also a memory-based CF algorithm, a direct comparison of them will provide valuable and irreplaceable insights.

Besides, we also chose CoFiRank \[29\] and ListRank-MF \[25\], two state-of-the-art model-based ranking-oriented CF algorithms for comparison to further demonstrate the promising performance of ListCF. Like \[29\], the dimensionality of latent features was set to be 10.

5.2 Efficiency Comparison

In order to evaluate the efficiency of our model, we chose PointCF and EigenRank as our comparison partners, as they share the same memory-based CF framework with ListCF. Since CoFiRank\[1\] and ListRank-MF\[2\] are two model-based CF algorithms and the implementation of them is based on the publicly available software packages written in different program languages from us, we did not include them in this section.

We conducted experiments to compare their runtime of the similarity calculation phase and ranking prediction phase on the datasets. Due to the huge size of the Netflix data, we selected users from Netflix who had rated no more than 50 items to form an extremely sparse dataset containing about 100,000 users. As analyzed in Section 4.2, when the number of items each user has rated increases the difference between EigenRank and ListCF would be more distinct, and we omit the result here. Experiments were run on a computer with 4 core 2 GHz CPU and 64 GB RAM.

Figure 1 demonstrates the respective runtime of the similarity calculation phase of PointCF, ListCF and EigenRank on three datasets. Note that the units of the runtime are Second (s), Minute (min) and Hour (h) for the datasets of Movielens-1M, EachMovie and Netflix respectively. From Figure 1, we can observe that in the similarity calculation phase:

- The runtime of ListCF is comparable with PointCF, where ListCF is merely around 1.2 times higher than PointCF. The slight increase of the runtime of ListCF mainly results from an extra step of computing the top-1 probability distributions for each pair of users.
- The runtime of ListCF is much lower than that of EigenRank, saving almost 40% time. This observation is consistent with our theoretical analysis in Section 4.2, which is that in the similarity calculation phase, the computational cost of pairwise CF is \((n + \pi^2)/(n + \pi)\) higher than that of ListCF, where \(n\) is the average number of items rated by users, and \(\pi\) is the average number of co-rated items by each pair of users.

Figure 2 shows the runtime comparison of the ranking prediction phase of the three CF algorithms, where the horizontal axis in these four sub-figures is the number of items to be predicted for each user. The vertical axis in Figure 2(a), 2(b) and 2(c) is the runtime of the algorithms, and the vertical axis in Figure 2(d) is the runtime ratio of EigenRank to ListCF. From the figures we can see that, in the ranking prediction phase ListCF demonstrates significant improvement in prediction efficiency in comparison with EigenRank.

1http://www.CoFiRank.org/
2http://mmc.tudelft.nl/users/yue-shi
From Figure 2(a), 2(b) and 2(c) we can see that the runtime of both ListCF and PointCF is linearly increased with the growth of the number of items to be predicted, while the increase of EigenRank (the pairwise CF) is quadratic. This observation is consistent with our theoretical analysis in Section 4.2 that, in the ranking prediction phase, the computational costs of ListCF and pairwise CF are $O(M(N − n)ld)$ and $O(M(N − n)^2t)$ respectively, where $M$ is the number of users, and $N − n$ is the number of items to be predicted. $l$ is the size of the neighbor set, and $d$ is the maximal number of iterations in the gradient descent based optimization method.

From Figure 2(d) we can see that the runtime ratio of EigenRank to ListCF increases linearly with the growth of the number of items to be predicted in the ranking prediction phase. This observation is consistent with our analyzed result $(N − n)/d$ in Section 4.2. $N − n$ is the number of items to be predicted, and $d$ is a constant, i.e., the maximal number of iterations in the gradient descent optimization method. The gradients of the curves are slightly different because some iterative optimization processes could converge before reaching the maximal number of iterations $d$.

The runtime of ListCF is significantly reduced as compared to EigenRank even when no more than 200 items are predicted. For example, for the EachMovie dataset, ListCF takes 20min to predict the total rankings of 200 items for 36656 users, while the runtime of EigenRank is 423min, which is 21 times longer. The results on Movielens-1M and Netflix are 19 times and 16 times longer respectively.

### 5.3 Prediction Accuracy Comparison

To demonstrate the recommendation accuracy of our model, we compare ListCF with all of the state-of-the-art recommendation algorithms mentioned in Section 5.1.2 on three datasets of Movielens-1M, EachMovie and Netflix. For each dataset, we randomly select each user’s 10 rated items and their corresponding ratings to form test set, and the remaining ratings are used to form train set.

#### 5.3.1 Evaluation Metric

For rating-based CF algorithms, the standard evaluation criteria are the Mean Absolute Error (MAE) and the Root Mean Square Error (RMSE). Both criteria measure the error between true ratings and predicted ratings. Since our study focuses on improving item rankings instead of rating prediction, we employ the Normalized Discounted Cumulative Gain (NDCG) [7, 8] as the evaluation metric, which is the most popular accuracy metric in information retrieval for evaluating ranking of retrieved documents with multi-level relevance scores.

In the context of collaborative filtering, item ratings assigned by users can naturally serve as relevance judgments. The NDCG metric is evaluated over some number of the top items on the ranked list. Let $U$ be the set of users and $r_u^p$ be the rating score assigned by user $u$ to the item at the $p$th position of the ranked list. The NDCG value at the $n$th position with respect to the given user $u$ is defined as follows:

$$NDCG_u@n = \frac{\sum_{p=1}^{n} 2^r_u^p - 1}{\log(1 + p)}$$

where $Z_u$ is a normalization factor calculated so that the NDCG value of the optimal ranking is 1. $NDCG_u@n$ takes the mean of the $NDCG_u@n$ over the set of users $U$, which is defined as follows:

$$NDCG@n = \frac{1}{|U|} \sum_{u \in U} NDCG_u@n,$$

where $|U|$ is the cardinality of the set of users.

#### 5.3.2 Performance on Movielens-1M and EachMovie

Figure 3 shows the performance comparison results in terms of NDCG@1, 3 and 5 on the Movielens-1M and EachMovie datasets. From the figure we can observe that:

- Ranking-oriented CF algorithms generally achieve more accurate top recommendations than the rating-oriented CF algorithm (PointCF) under most conditions except NDCG@3 and 5 on Movielens-1M.
- ListCF outperforms all of the comparison partners, including the pairwise ranking-oriented CF algorithms.
and also listwise model-based CF algorithms. The improvement is more distinct on EachMovie dataset. For example, ListCF achieves improvements of 9.25% over EigenRank and 5.9% over ListRank-MF on EachMovie in terms of NDCG@5.

- In comparison with all the other algorithms, the improvements of ListCF are more significant in terms of NDCG at higher positions. For example, for Movielens-1M, ListCF improves 19.88%, 14.97%, and 13.41% in terms of NDCG@1, 3 and 5 in comparison with PointCF, and these numbers are 16.29%, 15.51%, and 13.93% respectively in comparison with EigenRank. For EachMovie, we can get similar trends. This is because ListCF considers the ranking position issue in the similarity calculation phase, which is demonstrated in Example 3 in Section 4.2.

5.3.3 Performance on Netflix

In previous experiments, we show that ListCF can generate superior performance in top-n recommendation on Movielens-1M and EachMovie datasets compared with other recommendation algorithms. In order to compare our model with the other algorithms thoroughly, distinct from previous validations focusing on comparing global quality of recommendations over all users, here we are particularly interested in testing the performance of respective algorithms in regard to different groups of users. We first select user groups based on the number of observed ratings in the training set. We use group “20-50”, “50-100” and “100-200” as experimental subjects, and each user group contains about 90 thousand users. User group “20-50” denotes users who have rated more than 20 and less than 50 items in the training set. User groups represent different datasets with different sparsities. The sparsity of user group “20-50” is 99.87%, and it is the sparsest dataset among three datasets.

The detailed experimental results are reported in Table 3. From the table we can see that:

- All of the recommendation algorithms perform worse on sparser datasets. For example, our memory-based algorithm ListCF achieves 0.7666 in NDCG@1 on group “100-200” while 0.7004 on group “20-50”, and the numbers of the model-based algorithm ListRank-MF are 0.7357 on group “100-200” while 0.6929 on group “20-50”.

- Our ListCF can significantly outperform other memory-based algorithms, both pointwise (PointCF) and pairwise CF (EigenRank and VSRank), in all evaluation metrics of NDCG@1, 3 and 5. In addition, similar to the results on Movielens-1M and EachMovie, ranking-oriented algorithms, both pairwise and listwise CF, achieve more accurate predictions than rating-oriented CF (PointCF).

- ListCF can outperform the model-based ranking-oriented algorithms under most conditions except NDCG@3 and NDCG@5 on group “20-50” of ListRank-MF. The reason might be that ListCF, as a memory-based algorithm, cannot find useful neighborhood users and make accurate predictions if the dataset is too sparse because of that the commonly rated items between users are too rare. However, even on this dataset with the sparsity of 99.87%, ListCF can also achieve the best performance in NDCG@1, with improvements of 2.56% and 1.08% over CoFiRank and ListRank-MF respectively.

- As a whole, the improvements of ListCF are more distinct on denser datasets in comparison with other algorithms. For example, on the densest dataset (group “100-200”), the improvements of ListCF are 13.32%, 11.10% and 9.30% in terms of NDCG@1, 3 and 5 respectively compared with PointCF, while on the sparsest dataset (group “20-50”), these numbers become 6.42%, 4.49% and 3.40%. The reason is similar to the previous observation, which is that the number of co-rated items between users are larger on a denser dataset, and thus ListCF can find more accurate neighborhood users to make predictions.

<table>
<thead>
<tr>
<th>User group</th>
<th>Metric</th>
<th>PointCF</th>
<th>EigenRank</th>
<th>VSRank</th>
<th>CoFiRank</th>
<th>ListRank-MF</th>
<th>ListCF</th>
</tr>
</thead>
<tbody>
<tr>
<td>20-50</td>
<td>NDCG@1</td>
<td>0.6581</td>
<td>0.6600</td>
<td>0.6590</td>
<td>0.6824</td>
<td>0.6929</td>
<td>0.7004</td>
</tr>
<tr>
<td></td>
<td></td>
<td>6.42%</td>
<td>6.12%</td>
<td>6.28%</td>
<td>2.56%</td>
<td>1.08%</td>
<td></td>
</tr>
<tr>
<td></td>
<td>NDCG@3</td>
<td>0.6866</td>
<td>0.6791</td>
<td>0.6723</td>
<td>0.7106</td>
<td>0.7218</td>
<td>0.7174</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4.49%</td>
<td>5.64%</td>
<td>6.71%</td>
<td>0.96%</td>
<td>-0.61%</td>
<td></td>
</tr>
<tr>
<td></td>
<td>NDCG@5</td>
<td>0.7233</td>
<td>0.7144</td>
<td>0.7126</td>
<td>0.7322</td>
<td>0.7570</td>
<td>0.7502</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3.40%</td>
<td>4.69%</td>
<td>4.95%</td>
<td>1.72%</td>
<td>-0.90%</td>
<td></td>
</tr>
<tr>
<td>50-100</td>
<td>NDCG@1</td>
<td>0.6693</td>
<td>0.6898</td>
<td>0.6833</td>
<td>0.6990</td>
<td>0.7233</td>
<td>0.7372</td>
</tr>
<tr>
<td></td>
<td></td>
<td>10.14%</td>
<td>8.87%</td>
<td>7.10%</td>
<td>5.46%</td>
<td>1.92%</td>
<td></td>
</tr>
<tr>
<td></td>
<td>NDCG@3</td>
<td>0.7057</td>
<td>0.7085</td>
<td>0.7117</td>
<td>0.7153</td>
<td>0.7359</td>
<td>0.7563</td>
</tr>
<tr>
<td></td>
<td></td>
<td>7.17%</td>
<td>6.75%</td>
<td>6.27%</td>
<td>5.73%</td>
<td>2.77%</td>
<td></td>
</tr>
<tr>
<td></td>
<td>NDCG@5</td>
<td>0.7445</td>
<td>0.7364</td>
<td>0.7406</td>
<td>0.7591</td>
<td>0.7679</td>
<td>0.7817</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5.00%</td>
<td>6.15%</td>
<td>5.55%</td>
<td>2.98%</td>
<td>1.80%</td>
<td></td>
</tr>
<tr>
<td>100-200</td>
<td>NDCG@1</td>
<td>0.6765</td>
<td>0.6900</td>
<td>0.7014</td>
<td>0.6954</td>
<td>0.7357</td>
<td>0.7666</td>
</tr>
<tr>
<td></td>
<td></td>
<td>13.32%</td>
<td>11.10%</td>
<td>9.30%</td>
<td>10.24%</td>
<td>4.26%</td>
<td></td>
</tr>
<tr>
<td></td>
<td>NDCG@3</td>
<td>0.7093</td>
<td>0.7174</td>
<td>0.7228</td>
<td>0.7235</td>
<td>0.7494</td>
<td>0.7739</td>
</tr>
<tr>
<td></td>
<td></td>
<td>9.11%</td>
<td>7.88%</td>
<td>7.07%</td>
<td>6.97%</td>
<td>3.27%</td>
<td></td>
</tr>
<tr>
<td></td>
<td>NDCG@5</td>
<td>0.7491</td>
<td>0.7565</td>
<td>0.7641</td>
<td>0.7755</td>
<td>0.7794</td>
<td>0.7987</td>
</tr>
<tr>
<td></td>
<td></td>
<td>6.62%</td>
<td>5.58%</td>
<td>4.53%</td>
<td>2.99%</td>
<td>2.48%</td>
<td></td>
</tr>
</tbody>
</table>
6. CONCLUSION

In this paper, we propose ListCF, a listwise ranking-oriented model for collaborative filtering. ListCF measures the user-user similarity based on the Kullback-Leibler divergence between users’ probability distributions over permutations of commonly rated items. It predicts item rankings for each user by minimizing the cross entropy loss between the target user and his neighbors with weighted similarities. Experimental results on three benchmark datasets show that ListCF demonstrates many advantages in recommendation efficiency and ranking accuracy. In addition, we prove that ListCF shares similar prediction strategy with pointwise and pairwise memory-based CF under certain constraints.

In the future, we plan to explore other possible listwise loss functions in ListCF. In addition, it is also interesting to investigate model-based listwise CF algorithms based on the top-$k$ probability representations.

7. ACKNOWLEDGMENTS

This work was supported by the Natural Science Foundation of China (61272240, 61103151, 71171122), Humanity and Social Science Foundation of Ministry of Education of China (61272240, 61103151, 71402083, 71171122), the Natural Science Foundation of Shandong province (ZR2012FM037, BS2012DX012), the Doctoral Fund of Ministry of Education of China (20110131110028).

8. REFERENCES