Knowledge-Based Systems 152 (2018) 172–185

Contents lists available at ScienceDirect

Knowledge-Based Systems

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

Item-based collaborative ranking

Bita Shams, Saman Haratizadeh*

Faculty of New Sciences and Technologies, University of Tehran, Tehran, Iran

ARTICLE INFO

Article history:
Received 20 August 2017
Revised 10 March 2018
Accepted 9 April 2018
Available online 13 April 2018

Keywords:
Collaborative ranking
Pairwise preference
Item-based recommendation
Random walk

ABSTRACT

Neighbor-based collaborative ranking algorithms exploit users' pairwise preferences to predict how they will rank items. Current neighbor-based algorithms lie in the category of user-based recommendation methods: they calculate users' similarities over pairwise preferences, estimate the unknown pairwise preferences, and finally, infer the ranking of items for the target user. However, it still is an open question how to adapt item-based recommendation for neighbor-based collaborative ranking. The more specific question is how to calculate items' similarities in a pairwise preference dataset, and how these similarities can be employed to infer the total ranking of items for the target user.

This paper presents a novel recommendation approach, PreNIt, that exploits preference networks for item-based collaborative ranking. PreNIt models the users' pairwise preferences as two novel bipartite networks with labeled edges. These labeled edges enable us to model the choice context in which items are preferred/not preferred by the user. Once the networks are constructed, PreNIt finds the transitive similarities of items using a new personalized ranking algorithm in graphs with labeled edges. Experimental results shows the significant outperformance of PreNIt over the state-of-the-art algorithms.

1. Introduction

Recommender systems are intelligent tools which exploit users' feedbacks to learn their interests and predict what they will require more in the future. Traditionally, recommender systems gather users' feedbacks in forms of 1–5 scores and then, they learn a model to predict the numerical values of unknown scores.

Recently, it has been shown that the traditional recommendation approach, called rating-oriented collaborative filtering, suffers from several shortcomings: first, recommendation is inherently a ranking task and so, one can expect that recommendation algorithms only need learn the preference of items over each other no matter what their received ratings are [1–3]; Second, the traditional scoring systems suffer from two flaws: *calibration* and *limited resolution* problems. *Calibration* problem refers to fact that a certain score may have different meanings for different users or even for the same user in different situations [4–6]. *Limited resolution* problem arises when users have to assess and classify all items in few, usually 5, levels of rates. The user must pick one of the two adjacent levels of scores for an item, no matter he thinks that the item is better than those in the lower level, but, not as good as those in the higher level [7]. That can result in some information loss which reduces the systems' accuracy.

These shortcomings result in emergence of a new group of recommendation algorithms, called collaborative ranking that focuses on predicting the rank of items rather than their rates. A large number of researches have been conducted to adapt matrix factorization techniques for prediction using different kinds of user feedbacks such as implicit feedbacks [8,9], heterogeneous implicit feedbacks [10], rates [2,3,11], and pairwise preferences [5,7].

On the other hand, neighbor-based collaborative ranking methods have focused on pairwise preference datasets, as pairwise preferences are the basic elements of ranking and all kinds of feedbacks can be converted to a set of pairwise preferences [12]. Pairwise preference also have been shown to be a more stable kind of data over time and in different situations [7,13]. As neighbor-based collaborative ranking algorithms are applied to pairwise preference datasets, they are also known as neighborhood pairwise preference-based recommendation [7].

Neighbor-based collaborative rating algorithms estimate the preferences of the target user over items by exploring users that are similar to a target user, or items that are similar to a target item. These two attitudes have been used in a large number of researches that are categorized as user-based [14], and item-based neighbor-based collaborative rating [15].

On the other hand, all of existing neighborhood collaborative ranking algorithms lie in the category of user-based recommenda-

* Corresponding author.
E-mail addresses: shams.bita@ut.ac.ir (B. Shams), haratizadeh@ut.ac.ir (S. Haratizadeh).

https://doi.org/10.1016/j.knosys.2018.04.012
0950-7051/

a) a sample of Rating dataset

Users	Titanic	Inception	Batman	Star wars
Jack	5		4	3
Martin		3	2	
Lee	4	2		5

b) a sample of Pairwise preference dataset

Users	Pairwise preferences
Jack	Titanic>Star wars, Inception>Battman
Martin	Inception>Titanic, Titanic>Star wars, Inception>Batman
Lee	Titanic>Batman, Titanic>Star wars
Emma	Inception>Battman, Titanic>star wars

Fig. 1. A Schematic examples to illustrate the difference between users and item profiles in rating and ranking datasets.

tion algorithms. These algorithms calculate users' similarities based on their disagreement/agreement over pairwise preferences. Thereafter, they estimate the values of unknown pairwise preferences, and aggregate these pairwise preferences to obtain a total ranking of items for the target users [1,7,16]. Trivially, this approach adopts the concept of user-based collaborative filtering, which uses the opinions of similar users for inferring the preferences of the target user.

Although several methods have been suggested for user-based recommendation, we are not aware of any framework for item-based neighborhood collaborative ranking. This is despite the fact that item-based class of collaborative rating methods has gained higher accuracy, efficiency, and scalability in many applications, especially when the number of users exceeds the number of items [17].

The main challenge of using an item-based recommendation approach for collaborative ranking is the way in which users are related to the items in a pairwise preference dataset. As it can be seen in Fig. 1a, a rating dataset is comprised of a one-to-one relationship among users and items: a user states his opinion about an item by a single value that reflects his overall evaluation about that item. Such a relationship can be simply represented by a rating matrix $A_{m\times n}$ in which A_{ui} indicates interest of user u to item i. In such a dataset, each item can be simply modeled by a $m \times 1$ vector whose elements represent opinions of users about that item. Once the items are represented by numerical vectors, item-based recommendation algorithms can apply a numerical distance metric (e.g. cosine similarity) to calculate similarities among items and use them for an item-based recommendation to the target user.

But, the situation is different in pairwise preference datasets. As in Fig. 1b, a pairwise preference dataset is comprised of a set of pairwise comparisons among items for each user. In such a dataset, interest of users in items are captured in certain choice contexts. For instance, in Fig. 1b, Martin has preferred Titanic over Star Wars, while, it has not preferred it over Inception. Trivially, there is not a one-to-one relationship among users and items in such a dataset. Therefore, an item can't be easily represented by a numerical vector, and that makes it a complicated task to calculate similarities among items in the pairwise preference datasets. That is why the traditional approach to item-based recommendation is not directly applicable when we are using pairwise preference datasets.

This paper contributes to adapt the item-based recommendation for collaborative ranking when a pairwise preference data set is used. For this purpose, we first analyze how to calculate items similarities in a pairwise preference dataset and then, we show how to rank items for a target user based on calculated similarities. Accordingly, we present a novel framework, called PreNIt, that exploits novel preference network structures for item-based collaborative ranking. The main contributions of this paper can be summarized as below:

- We present an item-based framework for neighborhood collaborative ranking. To our knowledge, this is the first collaborative ranking algorithm that takes into account items' similarities in a pairwise preference dataset.
- We introduce a novel network structure that enables us to model the choice contexts in which a user prefers/not prefers an item.
- We model the behavior of a random surfer with a memory to design a novel personalized ranking algorithm in networks with labeled-edges. Note that most of recommendation algorithms exploit personalized PageRank and its extensions that model the behavior of a memory-less random surfer.

2. Related work

Collaborative ranking algorithms are generally divided into two groups: Matrix factorization and neighborhood collaborative ranking. As this paper contributes to improve neighbor-based collaborative ranking, we first briefly review the matrix factorization approach and then discuss how neighborhood collaborative ranking algorithms work in more details.

2.1. Matrix factorization collaborative ranking

Matrix factorization collaborative ranking algorithms (MFCR) typically learn the latent factors of users and items for optimizing a ranking objective function in different types of feedbacks. These algorithms can be categorized into two groups: list-wise and pairwise MFCR algorithms. The first group of MFCR, called list-wise MFCR, follows a list-wise approach to learn users priorities in case of implicit/explicit feedbacks. CofiRank [11] and List-Rank [2] are the state-of-the-art list-wise approach when rating data is available. CofiRank [11] is the pioneer algorithm in this category that seeks to optimize the normalized discounted cumulative gain (NDCG). List-Rank [2] is another algorithms which optimizes NDCG using a matrix factorization model that predicts Top-1 probabilities of items for each user. CLiMF [9] and xCLimf [18] maximize another objective function, called mean reciprocal rank (MRR), that is applicable in implicit feedbacks. Furthermore, some hybrid list-wise approaches have been proposed that combine rating-oriented matrix factorization and list-wise MFCR algorithms. URM [3] is another algorithm that learns a trade-o between rating and ranking loss functions. UOCCF [19] is another collaborative ranking algorithm that combines probabilistic matrix factorization (PMF) and Climf to improve one-class collaborative filtering. Moreover, it has been proposed to combine SVD++ and XCLimf to improve recommendation quality when both explicit and implicit feedbacks are available [20].

Another group of MFCR algorithms, called pairwise MFCR, has focused on learning the relative preference among items in different types of feedbacks. BPR [8] and PushAtTop [21] learn the pairwise preferences among relevant and irrelevant items when implicit feedbacks are available. The main difference between these algorithms is that PushAtTop weighs pairwise comparisons according to the ranks of their corresponding items. PPMF is another approach in this category that combines PMF and RankRLS to minimize the average number of pairwise preference inversions in the predicted ranking [22]. Moreover, some other algorithms, such as BPR++ [23], ABPR [24], GcBPR [25], have been presented to improve recommendation accuracy when graded/heterogeneous implicit feedbacks are available. MFPref is a recent pairwise MFCR algorithm which seeks to minimize root mean square error (RMSE) in a user-preference utility matrix [7]. To enhance the robustness of this group of algorithms, it has been proposed to suppress

the effect of noisy pairwise preferences by limiting the impact of outlying samples on training process [26]. Also some recent approaches have used social relationships to improve the quality of collaborative ranking algorithms [27,28].

2.2. Neighborhood collaborative ranking

The general framework of neighborhood collaborative ranking is originally proposed in [1]. This framework is comprised of three steps: calculating users' similarities, estimating unknown pairwise preferences based on opinions of neighbors, and finally, aggregating the pairwise preferences to infer a total ranking of items. Kruskal's gamma (GK) and its variants are the most popular similarity measure in neighbor-based collaborative ranking [7]. This metric, that is also known as Kendall correlation [1,16,29], measures the users' similarities based on their agreement/disagreement over pairwise preferences. Wang et al. [16] improved this metric by assigning higher weights to the pairwise preferences with higher *specialty* and *degree* values. *Specialty* defines how rare a pairwise preference is in the data. For instance, a pairwise preference $A > B$ have a lower specialty if all users agree with it. On the other hand, *degree* reflects how strong a pairwise preference is for a user. For example, the *degree* of $A > B$ is higher than that of $A > C$ for u if he has rated $A = 5$, $B = 1$, and $C = 4$. Kalloori et al. [7] used EDRC to calculate users' similarities in a pairwise preference dataset. EDRC weighs the pairwise preference p according to the rank of the p's involved items [30]. Shams and Haratizadeh [31] presented an extended graph-based similarity measure to calculate users' similarity in sparse pairwise preferences datasets.

Once the similarities are calculated, the recommendation algorithms estimate the concordance of each pairwise preference based on a weighted averaging over opinions of neighbor users. Neighbors are typically defined as k most similar users [1,16,32] or all users who has stated their opinion over a certain pairwise comparison [7].

Finally, calculated concordance values for the pairwise preferences are aggregated to infer a total ranking of items. Some algorithms estimate the score of an item i through minimizing values of those pairwise preferences in which i has lost [7,32] from estimated values of pairwise preferences in which i has won [7,32]. Liu et al. modified this approach such that their algorithm eliminates items with highest score and then recalculate the score of the remaining items until all items are ranked [1]. Some other algorithms use the exponential ranking algorithm on signed network $G =< V, E >$ whose edges are associated with a signed weight; The weight of edge e_{ij} indicates the estimated preference of j over i [1,31].

As an alternative to the traditional three step framework for user-based neighborhood recommendation, it has been suggested to represent a preference dataset as a tripartite preference graph, and then use a personalized PageRank method on this graph to rank items for each target user [12]. IteRank [33] is another recommendation approach which exploits a two-step approach to traditional user-based recommendation approach. IteRank first uses an iterative approach to estimate users' similarities and preferences concordances. Then, it exploits the initial ranking of items to simultaneously adjust the concordance of preferences and the ranking of items. As mentioned before, none of these algorithms have tried to estimate items' similarities for neighborhood pairwise preference-based recommendation.

Our algorithm lies in the class of neighborhood collaborative ranking algorithms which exploit graph structure to capture extended similarities among entities. But unlike its predecessors [12,31,33], PreNIt takes into account items' similarities to make recommendation while previous methods use users' similarities. PreNIt introduces novel network structures, called LNet and WNet, which connect users and items using edges that reflect the choice contexts in which items have been preferred by users.

3. Preliminaries

From the NCR perspective, a recommender system is represented by a set of users $U = \{u_1, \ldots, u_m\}$, a set of items $I = \{i_1, \ldots, i_n\}$, and a pairwise preference dataset $O = \{o =< u, i, j > , u \in U, i \in I, j \in I\}$ in which $o =< u, i, j >$ reflects that the user u has preferred item i over item j. For simplicity, we call the first item, i, as o's *winner item*, and the second item, j, as o's *losing item*. Table 1 illustrates an schematic example of pairwise preference datasets. Note that a feedback data set usually can be simply converted to a pairwise preference dataset O [12].

An Item-based neighborhood recommendation algorithm should define a scoring function f: $U \times I \mapsto R$ to capture the interest of the target user u to an item, i, based on similarities between i and other items that u has compared them. We are going to adapt the concept of item-based recommendation for neighborhood collaborative ranking. For this purpose, we review the basic rules og the item-based approach in rating-oriented recommender systems:

- Two items are similar if they are given similar ratings by each particular user.
- The target user u is interested in an item i if it is similar to items that u has rated as good ones.

Similarly, we can define some rules to calculate items' similarities in a pairwise preference dataset. The main idea behind our item-based similarity measure is that when two items like A and B are preferred/not preferred in similar choice contexts, that is as an evidence for possible similarity of A and B. When the number of such evidences in the data set grows, one's expectation for a higher degree of similarity between A and B grows too. Accordingly, we can adapt recommendation rules for collaborative ranking as follow:

- Two items are similar if users have similar opinions about them in the same choice contexts. For instance, we can infer that item i is similar to item j if a number of users have preferred to the same set of other items.
- A user u is interested to an item i if i has a high similarity to items which u has preferred over other items and it has a low similarity to items over which u has preferred other items.

Accordingly, in order to use an item-based collaborative ranking method we need to answer two main questions: first, how to calculate items' similarities in a pairwise preference dataset, and second, how to use these similarities to score and rank items in such a dataset. The notation used in this paper is summarized in Table 2.

4. PreNIt: preference networks for item-based collaborative ranking

PreNIt, is a novel framework for making recommendations that exploits novel preference networks for item-based collaborative ranking. As explained earlier, from the viewpoint of item-based collaborative ranking, the target user u is interested in an item i if i has a high similarity to the winning items of his preferences and a low similarity to the losing items of them. To capture these similarities, PreNIt models users' pairwise preferences as two distinct networks, one for capturing the similarity to the losing items and one for capturing the similarity to the winning items. Each network is a bipartite network with labeled edges between users and items. Thereafter, PreNIt uses a novel graph-based personalized ranking algorithm, MemoRank, to calculate items' similarities.

Finally, it score items based on their similarity to u's winning and losing items.

4.1. Graph construction

Let $P_u = \{< i, j > \mid < u, i, j > \in O\}$ be the set of pairwise preferences of user u in dataset O. We also define the set $W_u = \{i \mid \exists j, < i, j > \in P_u\}$ containing items that u has preferred them at least in one pairwise comparison. Similarly, we define $L_u = \{i \mid \exists j, < j, i > \in P_u\}$ as the set of items that u has preferred at least one item over them. As mentioned before, score of an item i can be estimated based on its similarity to items that belong to L_u and W_u. For simplicity, we call items in these sets as the *losing* and *wining* items of the user u, respectively.

From the viewpoint of collaborative filtering, we should calculate items' similarities based on the history of the interactions of users with items. Bipartite networks are effective tools for calculating similarities among a set of target objects (i.e. items), based on their interactions with objects of another type. (i.e. users). Typical bipartite networks have been widely used to calculate items' similarities in rating-oriented recommender systems [34]. However, they are not applicable in pairwise preference-based recommender systems as they do not model the choice context in which a user has preferred/not preferred an item. More clearly, ordinary bipartite networks contain a set of links between each user u and all items i he has interacted with. This will result in information loss in pairwise preference datasets as it does not capture the information about the items over which u has preferred i, that is the context in which u has decided to interact with i. Although some network structures have been recently presented to model pairwise preferences [12,31,33], none of them include nodes for representing items, and so they are not suitable for calculating items' similarities.

To resolve this problem, we suggest to model the preference dataset O as two distinct bipartite networks, called LNet and WNet, that respectively model the users' interactions with their losing items and winning items. Additionally, each edge is associated with a label that reflects the choice context in which the edge's corresponding user has preferred/not preferred the edge's corresponding item. The formal definitions of WNet and LNet are given in Definition 1. Each edge in WNet connects a user node u to an item node w with a label i given that user u has preferred item w over item i. Also each edge in LNet connects a user node u to an item node l with a label i given that user u has preferred item l over item i.

Definition 1. $WNet =< (U, I), E_W >$ is a bipartite network *LNet* where V is the set of vertices containing nodes for users and items, and $E_W = \{(< u, i >, c) \mid < u, i, c > \in O\}$ is the set of edges, such that each edge connects a user node u to an item node i with a label c. Similarly, $LNet =< (U, I), E_L >$ is a bipartite network where V is the set of vertices containing nodes for users and items, and $E_L = \{(< u, i >, c) \mid < u, c, i > \in O\}$ is the set of edges between users and their losing items.

It is worth noting that each user might be connected to an item with multiple edges as he probably has preferred/not preferred that item over multiple items (i.e. in multiple choice contexts). Fig. 2 illustrate the WNet and LNet representations for the data in Table 1. As it can be seen, WNet contains two edges between Jacob and E as Jacob has preferred E over both of B and C in the dataset (See Table 1). Similarly, Emma is connected with two edges to D in LNet as she has preferred both B and C over D.

Similarities to the target user's winning and losing items can be effectively captured through paths from these items in WNet and LNet, respectively. For instance, from the winning perspective, it can be implied that B and C are similar as Louis has preferred both of them over A, and, Emma has preferred both of them over D (see Fig. 2a). On the other hand, B and C are similar from the losing perspective as Jacob has preferred E over both of them (see Fig. 2b).

Table 1
A schematic example of pairwise preference datasets.

User	Winner item	Loser item
Emma	A	B
Emma	B	D
Emma	C	D
Louis	A	D
Louis	B	A
Louis	C	A
Louis	E	C
Jacob	D	A
Jacob	E	C
Jacob	E	B

Formally, we can infer that an item i is similar to a winning/losing item j if WNet/LNet contains a path in form of $< i - e_1 - u - e_2 - j >$ where $e_1.label = e_2.label$. This form of path indicates a direct similarity between two items and so, we call it a *direct item-based recommendation path*. However, similarity is a transitive relation; that is if i is similar to j and j is similar to k, it can be inferred that i is also similar to k. The transitive nature of the similarity can be captured through transitive item-based recommendation paths as explained in Definition 2.

Definition 2. *Transitive item-based recommendation paths* are obtained through concatenation of several direct item-based recommendation paths in LNet/WNet and formally can be represented as $\phi =< i_1 - e_1 - u_1 - e_2 - i_2 - e_3 - u_2 \cdots - i_{l-1} - e_{l-1} - u_l - e_l - i_l >$ such that $e_k.label = e_{k+1}.label$ for all ϕ's sub-paths in form of $\pi =< e_k - u - e_{k+1} >$.

Note that the main characteristic of transitive item-based recommendation paths is that they enter and exit a user node with edges having the same labels. As expected, these paths connect items that are similarly evaluated by users in similar choice contexts (represented by labels of edges) and so, they are good indicators to infer transitive similarities among items.

4.2. MemoRank: personalized ranking using a random surfer with a memory.

Once the LNet and WNet are constructed, we can score items according to the frequency and length of item-based recommendation paths from losing/winning items of the target user u to other items. Typically, graph-based recommendation algorithms use Personalized PageRank as a proximity measure to determine similarities of nodes to a set of query nodes in such a situation. However, Personalized PageRank is not applicable in our problem as it ignores the labels of edges and measures similarities among nodes based on the length and frequency of all available paths in the graph, not the item-based recommendation paths as desired here.

We propose a novel metric, called MemoRank, which modifies personalized PageRank in a way that it only considers item-based recommendation paths to calculate items similarities. For this purpose, we first review the idea behind personalized PageRank and then explain how to modify it to score items for an item-based recommendation in LNet/WNet.

Personalized PageRank of a node i is equal to the probability that a random surfer, starting at one of the query nodes, will reach node i. More formally, personalized PageRank models the behavior of a random surfer who starts at one of the query nodes and then either follows a random edge of the current node to go to an ad-

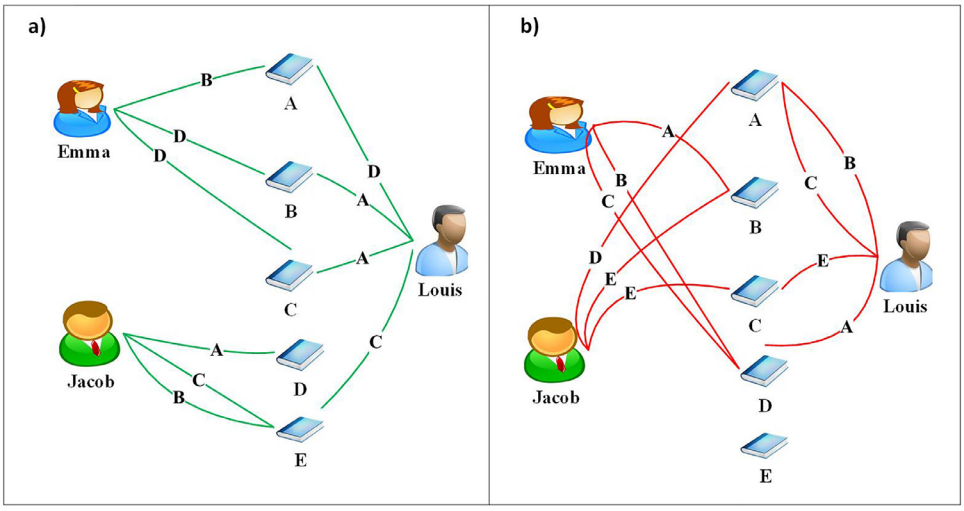

Fig. 2. Network representations for the pairwise preference dataset in Table 1. (a) WNet representation (b)LNet representation.

Table 2
Mathematical ntations.

Symbol	Descriptions
U	The set of users
I	The set of items
P	The set of pairwise preferences
O	The pairwise preference dataset
n	Number of users
m	Number of items
s	Size of pairwise preference dataset (i.e. O)
P_u	The set of all pairwise preference stated by user u
W_u	The set of all items that u has preferred in at least one choice context
L_u	The set of all items that u has not preferred in at least one choice context
C	Choice context

jacent node with a given probability α or restarts his walk at one of the query nodes with probability of $1-\alpha$. If this process is iterated for an infinite number of times, then the probability of the random surfer reaching each node converges to a stationary distribution. Note that this random surfer may pass any possible path over the network without considering the types and labels of the edges.

Our proposed approach, MemoRank, models the behavior of a random surfer whose walk is consistent with item-based recommendation paths. Please note that item-based recommendation paths are obtained through concatenation of sub-paths in form of$\pi = < e_k - u - e_{k+1} >$ where $e_k.label = e_{k+1}.label$. Therefore, when the random surfer reaches at a user node by passing an edge with label c, he should leave that user node using an edge labeled c. To enable the random surfer to do this, we suggest to equip him with a one step memory to remember the label of the last edge that he has passed.

Formally, MemoRank's random surfer starts at a query node, jumps to a user node u with an edge labeled c. Then, he randomly moves to an item node which is also connected to u by an edged labeled c. This process is iterated until we can obtain the stationary distribution of the probabilities that the random surfer will reach each node in the network. Note that similar to personalized PageRank, at each step, the random surfer might jump to one of the adjacent nodes with probability α or restart his walk at a query node with probability $1-\alpha$.

The key step to model the behavior of a random surfer on a network is to define his transition probabilities. Therefore, we should define two transition matrices: T_c^U indicating the probability to move from user nodes to item nodes using edges with label c, and T_c^I to define the probability to move from item nodes to user nodes through edges labeled c.

More formally, T_c^U is a $n \times m$ matrix in which $T_c^U(i,u)$ indicates the transition probability from u to i if the random surfer has reached u using an edge labeled c.

$$T_c^U(i,u) = \begin{cases} \frac{1}{d_c(u)} & e:(u,i) \in E \quad and \quad e.label = c \\ 0 & Otherwise \end{cases} \tag{1}$$

Where $d_c(u)$ be the number of edges with label c that are connected to u.

Similarly, T_c^I is a $m \times n$ matrix whose element $T_c^I(u,i)$ indicates the probability that a random surfer will jump from an item node i to a user node u through an edge labeled c. More formally, We can define $T_c^I(u,i)$ using below equation

$$T_c^I(u,i) = \begin{cases} \frac{1}{d(i)} & e(u,i) \in E \quad and \quad e.label = c \\ 0 & Otherwise \end{cases} \tag{2}$$

where $d(v) = \sum_{c=1}^{n} d_c(v)$ be the total degree of node v. As the random surfer should check the label equality at user nodes, we decompose the rank of user node to n types of rank in a way that each type of rank refers to one possible label $c \in I$. That is not applied to item nodes, and each item has a single rank.

Let $M_t^{U,c}$ be a $m \times 1$ vector indicating the rank of user nodes with regards to label c at time t. From the perspective of random surfer, $M_t^{U,c}$ indicates the probability of reaching a node u through an edge labeled c in time t. Similarly, we define M_t^I as $n \times 1$ vector whose elements determines the total probability to reach item nodes at time t. Now, we can obtain the stationary probabilities of reaching each node through iteratively recalculation of Eqs. (3) and

(4)

$$M_{t+1}^{U,c} = \alpha T_c^I M_t^I \tag{3}$$

$$M_{t+1}^{I} = \alpha \sum_{c=1}^{n} T_c^U M_t^{U,c} + (1-\alpha)R \tag{4}$$

Where α is the damping factor and it is usually set to 0.85 in graph-based ranking algorithms [12,31,35–37]. Moreover, R is the vector indicating the probability that the random surfer will restart his walk at each query node. Note that all query nodes are items as PreNIt seeks to find similarity to item nodes. Therefore, the random surfer never restarts his walk at a user node. That is why $M_{t+1}^{U,c}$ is not affected by restart vector.

Algorithm 1 gives the pseudo-code for MemoRank calculation.

ALGORITHM 1: MemoRank.

Input: Graph $G :< (U, I), E >$, Restarting probabilty vector: R
graphs, the target user u
Output: MemoRank Vector of items: M_t^I
`/* Ensure that R sum up to 1 (i.e. ||R|| = 1) */`
foreach $c \in I$ **do**
$R_c = \frac{R_c}{\sum_{k \in I} R_k}$
end
Randomly initial vectors of $M_0^{U,c}$ and M_0^I such that
$M_0^I + \sum_{c \in I} M_0^{U,c} = 1$
Define transition matrix T_c^U and T_c^I through Eqs. (1) and (2)
repeat
foreach $c \in I$ **do**
$M_{t+1}^{U,c} = \alpha T_c^I M_{t+1}^I$
end
foreach $c \in I$ **do**
$M_{t+1}^I = M_{t+1}^I + T_t^U M_t^{U,c}$
end
$M_{t+1}^I = \alpha M_{t+1}^I + (1-\alpha)R$
t=t+1
until *converge*;

As an input, MemoRank takes a bipartite graph and a restart vector R as input. Note that R indicates the restart probabilities, and its entries should sum up to 1. After initializing the transition matrices and ranking vectors, it first updates the rank of user nodes with regards to the label c. Then, it calculates $\sum_{(c=1)}^{n} T_c^U M_t^{U,c}$ and updates the overall rank of items using Eq. (4). This process iterates until rank values converge to the stationary distribution. Finally, the MemoRank of items is returned. It is worth reminding that MemoRank of items indicate their similarity to the query nodes with regards to item-based recommendation paths over LNet/WNet.

Note that MemoRank scores nodes based on the stationary probability that the random surfer will reach each node. Therefore, it is a non-symmetric graph-based measure which considers the local properties of query nodes to score other items. Clearly, higher ranks are given to items that are more reachable from query nodes through item-based recommendation paths.

Proposition 1. *It is ensured that MemoRank will converge.*

Proof. *If $q = (m+1) \times n$, we can define M_{t+1} and H as a $q \times 1$ vector as*

$$M_{t+1} = \begin{bmatrix} M_{t+1}^{U,1} \\ \dots \\ M_{t+1}^{U,c} \\ \dots \\ M_{t+1}^{U,n} \\ M_{t+1}^{I} \end{bmatrix}, H = \begin{bmatrix} 0 \\ \dots \\ 0 \\ \dots \\ 0 \\ R \end{bmatrix}$$

We also define the transition matrix T as

$$T = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & T_1^I \\ 0 & 0 & 0 & 0 & 0 & \dots \\ 0 & 0 & 0 & 0 & 0 & T_c^I \\ 0 & 0 & 0 & 0 & 0 & \dots \\ 0 & 0 & 0 & 0 & 0 & T_n^I \\ T_1^U & \dots & T_c^U & \dots & T_n^U & 0 \end{bmatrix}$$

whose all elements are zero except the last row and column blocks. Now, we can compute the vectors of M_{t+1}^I and $M_{t+1}^{U,c}$ through iteratively calculation of $M = \alpha TM + (1-\alpha)H$. More clearly, the vector Mcan be obtained through personalized PageRank of the graph with transition matrix of T and personalization vector of H. Let $B = \alpha T + (1-\alpha)H\mathbf{1}$, where $\mathbf{1}$ is a row vector of all ones. We can write $M_{t+1} = BM_t$. Note that B is a stochastic matrix as it is a linear combination of two column stochastic matrix with two coefficient that sum to 1. Therefore, the algorithm will converge to a unique solution M that is the right eigenvector corresponding to B's largest eigenvalue [35,38]. □

4.3. Recommendation

As stated before, we score items based on their similarity to winning/losing items of the target user. Algorithm 2 gives the

ALGORITHM 2: PreNIt recommendation.

Input: The pairwise preference dataset O, HyBNet
representation of dataset, the target user u
Output: score of items S
Define P_u ,W_u, and L_u
Initialize zero vector of R_W
foreach $i \in W_u$ **do**
$R_W[i] = CalculateWinningfrequency(O, u, i)$.
end
$S_w = MemoRank(WNet, R_W)$
Initialize zero vector of R_L
foreach $i \in L_u$ **do**
$R_L[i] = CalculateLosingFrequency(O, u, i)$.
end
$S_L = MemoRank(LNet, R_L)$
$S = S_W - S_L$

pseudo-code for PreNIt's recommendation. As can be seen, PReNit calculates the similarity to u's winning items. For this purpose, it extracts W_u in order to define the restart vector for the winning-similarity calculation phase. More clearly, PreNIt biases the restart probability (i.e. R) at query nodes such that items that are similar to more important query nodes will get higher MemoRank values. In PReNits recommendation, items that are similar to more preferred items, get higher scores. Accordingly, PreNIt weighs each member of W_u based on the number of comparisons in which it has been preferred over other items. PreNIt uses the function *CalculateWinningFrequency(O,u,i)* to weigh each winning item. This function calculates the size of the pairwise comparison set $P_u^w(i) = \{< i, j > | < i, j > \in O_u\}$ that contains all comparisons in which u has preferred i over other items. Thereafter, it uses Algorithm 1 to

calculate the overall similarity of items to W_U based on their paths in WNet. Similarly, PReNit can calculate items' similarities to L_U in LNet. It uses the function *CalculateLosingFrequency(O,u,i)* to find the number of comparisons in which i has not been preferred by the target user. Then, it estimates the overall similarity of items to losing items using the mentioned technique for MemoRank calculation in LNet. Finally, it scores items based on their similarity to winning/losing items of u; PreNIt gives higher score to those items that are similar to u's wining items and not similar to his losing items. The score of item i is calculated through below equation

$$S(i) = S_W(i) - S_L(i) \tag{5}$$

where $S_W(i)$ and $S_L(i)$ are similarities of an item i to u's winning and losing items, respectively.

4.4. Computational complexity

We first analyze the computational complexity of PreNIt's two main phases: graph construction and MemoRank calculation. In case of using adjacency lists, constructing a graph $G =< V, E >$ would have a time complexity of $O(v + e)$ where v and e denote the number of vertices and edges, respectively. Let m be the number of users, n be the number of items, and s be the size of pairwise preference dataset. We also note that these networks are simply updated at computational complexity of $O(1)$.

Similar to personalized PageRank, MemoRank can be calculated at computational complexity of $O(te)$ where t is the number of iterations till MemoRank converges. In our experiments, MemoRank converges in less than 20 iterations. Roughly speaking, at each iteration, MemoRank of each node should be updated once for each incoming edge. Therefore, we have $2e$ updates at each iteration of MemoRank calculation. Therefore, MemoRank will be computed at computational complexity of $o(ts)$ where s is the size pairwise preference dataset.

In summary, initial graphs are constructed at computational complexity of $O(m + n + s)$ but they can be updated in online systems at $O(1)$. Therefore, graph construction does not affect the speed of PreNIt recommendation. PreNIt makes recommendation to the target user through fetching his winning and losing items, and then, computing his personalized MemoRank in LNet and WNet. As the losing/winning items of each user can be stored in a dynamic updated list, we can fetch them at computational complexity of $O(1)$, and so, the computational complexity of PreNIt mainly depends on MemoRank's computational complexity that is $o(ts)$. This indicates that PreNIt is more scalable than user-based approaches which are implemented at computational complexity of $O(mn^2 + kn^2 + n^2)$ where k is the size of users' neighborhood [31]. Note that the pairwise preference dataset O will contain at most $s = mn^2$ preferences. However, the real-world datasets are too sparse and $s = cn^2$ where c is a small constant, and so, PreNIt seems to be more applicable in online recommender systems.

5. Experimental setting and result

We provide a comprehensive set of experiments to assess the performance of PreNIt compared to the state of the art collaborative ranking algorithms. In the following, we first explain our experimental setting and evaluation methodology. Then, we analyze how PreNIt performs in terms of accuracy and scalability.

5.1. Experimental setting

We compare PreNIt to the state-of-the-art collaborative ranking algorithms using *weak generalization* setting that is widely used in collaborative ranking literature [3,11,21]. In *weak generalization* approach, we split the dataset into training and test sets such that the training set contains a fixed number of ratings $UPL = \{10, 20, 30, 40\}$ that are randomly sampled for each user as training set, and the remaining ratings of each user are considered as test samples. We should ensure that each user has at least 10 ratings in his test set and so, we remove users whose profile does not meet this condition for each UPL value [18,39].*Weak Generalization* protocol is widely used in collaborative ranking literature for two reasons: first, it enables us to analyze the performance of algorithms in datasets with different number of users, different number of items, and different degrees of sparsity [33]. Second, it evaluates algorithms under the situation in which all users have higher number of ratings in their test set than in their training set. So, it simulates the real world applications in which users have already seen a small number of items and recommender systems should rank a large number of unseen items and recommend a few ones [21,33].

5.1.1. Datasets

We conducted our experiments over publicly available datasets that are widely used in collaborative ranking literature: Movielens100k, Movielens1M, and FilmTrust. Movielens100K is comprised of 100,000 ratings from 943 users to 1682 items [40], while, Movielens1M contains 1,000,000 ratings from 6040 users to 3952 items. Both of these datasets contain ratings in 1 to 5 scale. On the other hand, FilmTrust [41] is comprised of 1508 users and 2071 items and ratings are gathered in 8 levels in form of $0.5, 1, \ldots, 4$.

5.1.2. Baseline algorithms

We compare PreNIt to the state of the art neighbor-based and graph-based collaborative ranking algorithms.

- **EigenRank:** EigenRank [1] is the first neighborhood collaborative ranking algorithm which calculate users' similarity through Kendall correlation, and then, exploits a random walk approach to aggregate the estimated pairwise preferences of target user.
- **SibRank:** SibRank [31] calculates users' extended similarity through personalized ranking over a signed bipartite network which models users' pairwise preferences. Thereafter, it follows the EigenRank's framework to score items.
- **NN-GK**: NN-GK [7] is another user-based collaborative ranking algorithm, which uses Kruskal's gamma (GK) to calculate users' similarities over a pairwise preference dataset, and then, it ranks items according to their estimated preferences over other items.
- **IteRank**: IteRank [33] is the state of the art user-based neighborhood recommendation algorithms which follows a two-step approach to score item. It first uses an iterative method to refine similarities among users and concordance of pairwise preferences. Thereafter, it scores items through adjusting the concordance of pairwise preferences and the significance of items' representatives.
- **GRank**: GRank [12] adapts a graph-based recommendation framework for collaborative ranking. It models users' pairwise preferences as a tripartite preference graph and then ranks items through calculating personalized PageRank of the target user.

For further analysis, we compare our algorithm to the state of the art matrix factorization collaborative ranking (MFCR) algorithms:

- **CofiRank** is the most popular MFCR algorithm that exploits matrix factorization approach to optimize a structured loss function[11]. CofiRank has been adapted to learn the pairwise and list-wise ranking loss functions: CofiRank-ordinal minimizes a loss function which refers to the number of disconcordant pairwise preferences while CofiRank-NDCG optimizes the normalized discounted cumulative gain which scores the ranked list of

items inferred for a user compared to the optimum recommendation list for that user.

- **PushCR** is one of the state-of-the art MFCR algorithms which learns pairwise preferences among items with a focus on items that are at the top of the recommendation list [21]. Inf-push, p-push and rev-push are the three variations of PushCR framework. Inf-push and P-push focus on minimizing the largest or the p-norm of the number of relevant items that are ranked below a non-relevant item for each user, while, rev-push minimizes the number of non-relevant items that are ranked above a relevant item for each user.

5.1.3. Evaluation criteria

We assess the top-N recommendation list of algorithms through measuring the average NDCG@topN of the recommendation lists to users. NDCG@topN evaluates the performance of algorithms based on comparing the quality of their top-N recommendation list to the ideal one. More clearly, we can evaluate the top-N recommendation to the target user u using Eq. (5)

$$NDCG@TopN_u = \frac{1}{\delta_u}\sum_{i=1}^{topN}\frac{2^{r_i^u}-1}{log(i+1)} \tag{6}$$

where r_i^u is the u's real rating to the ith item in the generated recommendation list and δ_u is the normalization factor that ensures the NDCG would be equal to 1 for the ideal recommendation list. To remove the impact of random sampling, we measured the performance of algorithms over 5 random training sets and reported their average and standard deviation in Figs. 3–5. We note that NDCG is the most well-known evaluation metric when rating or graded implicit feedbacks are available [2,3,11,21,24,33].

Additionally, we conduct a set of paired t-tests on the results to verify if the difference between the performance of PreNIt and other algorithms is significant. We mention that using t-test is a valid approach to check the significance of differences among recommendation algorithms as the performance results are obtained based on a set of experiments on sufficiently large samples of data, and different algorithms are tested on common sets of samples [12,42,43]. Let $a_1,\dots a_N$ and $b_1,\dots,b_N$ be the performance result of two algorithms A and B over N set of samples derived from the dataset. Also, assume that μ_d and σ_d be the average and variance of differences $d_i = a_i - b_i$. To determine whether there is a significant difference between the performance of A and B, we consider the null hypothesis as $\mu_d = 0$ in contrast to the alternative hypothesis $\mu_d \neq 0$. The null hypothesis is rejected in favor of the alternative hypothesis if the p-value corresponding to the t-statistic $t = \sigma_d/\sqrt{N}$, is lower than the significance threshold (e.g. 0.05). Table 3 shows the p-values that are obtained through applying t-test over the performance results of PreNIt and other algorithms.

5.2. Performance analysis

We applied the well-known weak generalization protocol to compare the top-N recommendation quality of PreNIt and other collaborative ranking algorithms. As shown in Figs. 3–5, the performance of PreNIt has been higher than its competitors in most of evaluation conditions. We also applied a paired t-test to verify whether its improvements over other algorithms are significant or not [12,42,43]. The results imply that PreNIt significantly outperforms traditional neighborhood collaborative ranking methods (i.e. SibRank, EigenRank, and NN-GK) as well as the matrix factorization approaches in most of evaluation conditions, while the significance of its improvement over graph-based neighborhood algorithms (i.e. GRank and IteRank) is arguable depending on UPL values (See Table 3). In the following, we will discuss the relative performance of PreNIt over other algorithms in more details:

5.2.1. Comparison of PreNIt and traditional neighborhood collaborative ranking

PreNIt significantly outperforms neighborhood collaborative ranking (tNCR) algorithms that follow the traditional three-step user-based framework. For example, PreNIt shows improvement of 3% over SibRank, 4% over NN-GK, and 5% over EigenRank in terms of NDCG@3 in Movielens100k where UPL=20.

As seen in Figs. 3–5, the relative improvement of PreNIt over tNCR algorithm is higher in sparse datasets. For instance, in Movielens100k dataset, PreNIt improves NDCG@5 of EigenRank about 9% in case of UPL=10, while, its improvement is less than 2% when UPL is 40. The first reason is that users rarely have common pairwise comparisons in sparse datasets. Therefore, in such a dataset, EigenRank and NN-GK can not accurately estimate the users' similarities and so, they might perform as random recommendation algorithms. Although SibRank tries to resolve this issue using its graph-based similarity measure, it still suffers from another problem that is called *limited coverage/low discrimination flaw* [33]. As mentioned before, tNCR algorithms predict the preference of the target user based on opinions of the top-k most similar users, called neighbors. However, these neighbors have stated their opinions about a small number of pairwise preferences. Therefore, tNCR algorithms can not make any prediction about pairwise preferences that are not covered by the neighbors of the target users. PreNIt resolves these issues as it uses MemoRank to calculate transitive similarities among items. Moreover, PreNIt do not limit the size of neighborhood and uses all available data to calculate items' similarities and rank them.

5.2.2. Comparison of PreNIt and graph-based neighborhood collaborative ranking

Comparing the performances of graph-based algorithms implies that PreNIt significantly outperforms other GCR algorithms in sparse dataset. For instance, in case of UPL=10 in Movielens1M dataset (see Fig. 4), PreNIt achieved NDCG@1 of 72% while IteRank and GRank gained NDCG@1 of 65%, and 64%, respectively. But, in denser datasets GRank and IteRank slightly perform better than PreNIt in case of UPL=40 in Movielens1m dataset. Please note that both GRank and IteRank exploit the relations among pairwise preferences that share a common item to adjust the estimation about pairwise preferences and the overall desirability of items. Experimental results implied that rank propagation through these graphs are more informative and reliable when the data is dense (e.g. UPL=40 in Movielens1M) while in sparse data, the rank propagation using unreliable paths may decrease the performance of GRank and IteRank (e.g. in UPL=10 in Movielens1M). On the other hand, PreNIt propagates the rank through paths that are guaranteed to be consistent with item-based recommendation semantics in a pairwise preference dataset even when the data is sparse.

5.2.3. Comparison of PreNIt and MFCR algorithms

We also compare PreNIt with CofiRank and PushCR as two Matrix factorization approaches to collaborative ranking. Note that there is a critical difference between CofiRank and PushCR algorithms; CofiRank algorithms seek to learn the relative preference of rated/relevant items while PushCR algorithms focus on the relative preferences among relevant and irrelevant items for each user. Therefore, we individually analyze the relative performance of PreNIt over CofiRank and PushCR algorithms.

As seen in Figs. 3–5, PreNIt outperforms CofiRank in all evaluation conditions but its improvement increases in denser datasets. For example, in Movielens100k dataset, PreNIt shows improvement of 10% over both CofiRank-NDCG and CofiRank-ordinal in terms of NDCG@10 when UPL is 10 while its performance is only about 5% better than that of CofiRank-NDCG and 8% better CofiRank-Ordinals in case of UPL=10(See Fig. 3d). Please notice that that CofiRank

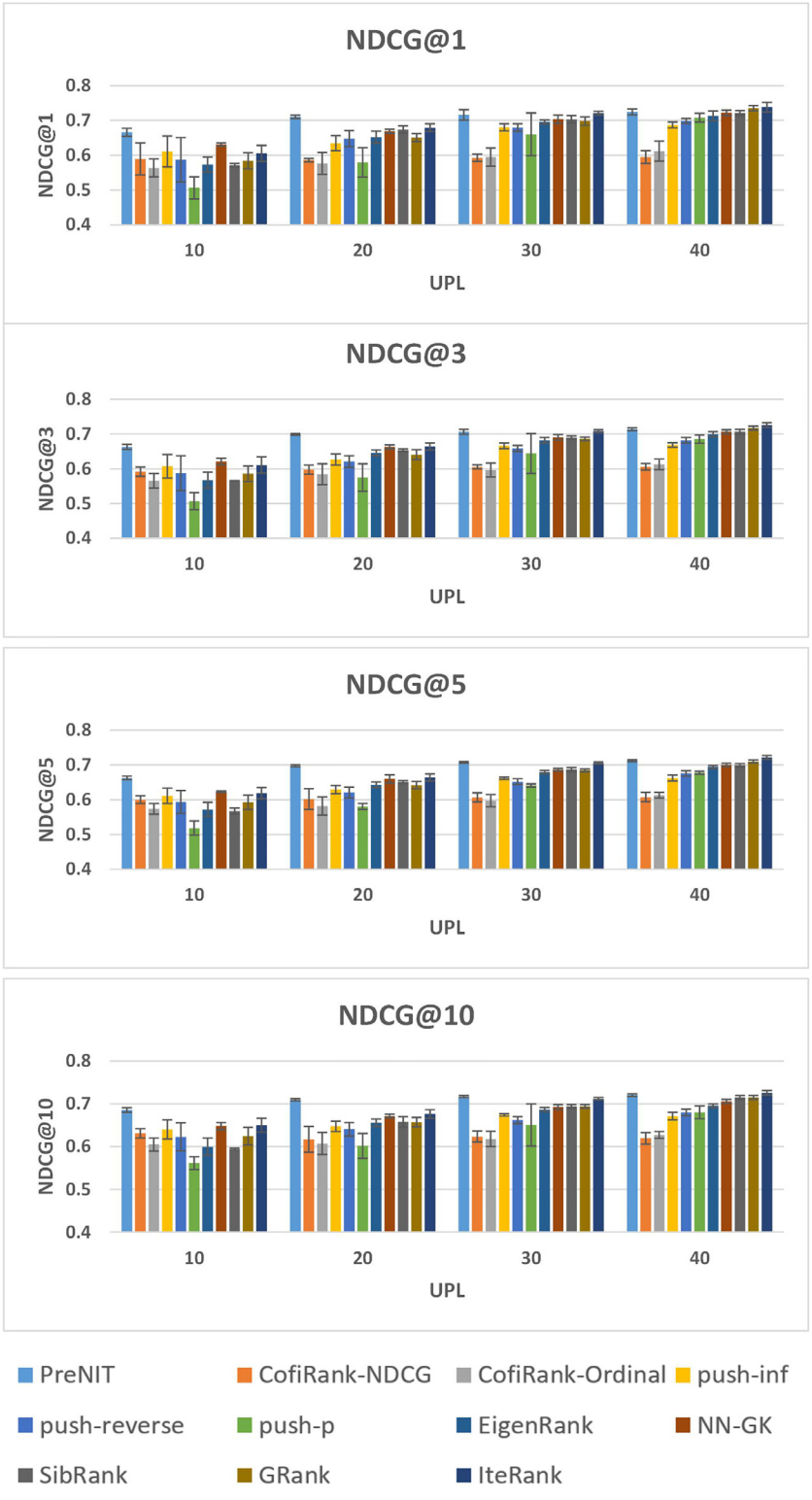

Fig. 3. Comparison of algorithms in Movielens100k dataset.

typically learns the global tastes of users and recommend items that are of interest of many users. On the other hand, GCR algorithms, including PreNIt, simultaneously incorporates personal and global users' tastes with a more importance on local tastes of users. In sparse datasets, information is not rich enough to accurately recognize the personal tastes of users. Therefore, the recommendation list of PreNIt is mainly affected by the common tastes of users which also forms the key element of CofiRank's recommendation. Therefore, PreNIt shows more improvement over CofiRank in dense datasets where the data is rich enough for PreNIt to correctly recognize the local and personal tastes of the users.

Additionally, PreNIt Significantly outperforms all modifications of PushCR framework. For instance, its performance is about 4% better than the performance of inf-push, 5% better than that of p-

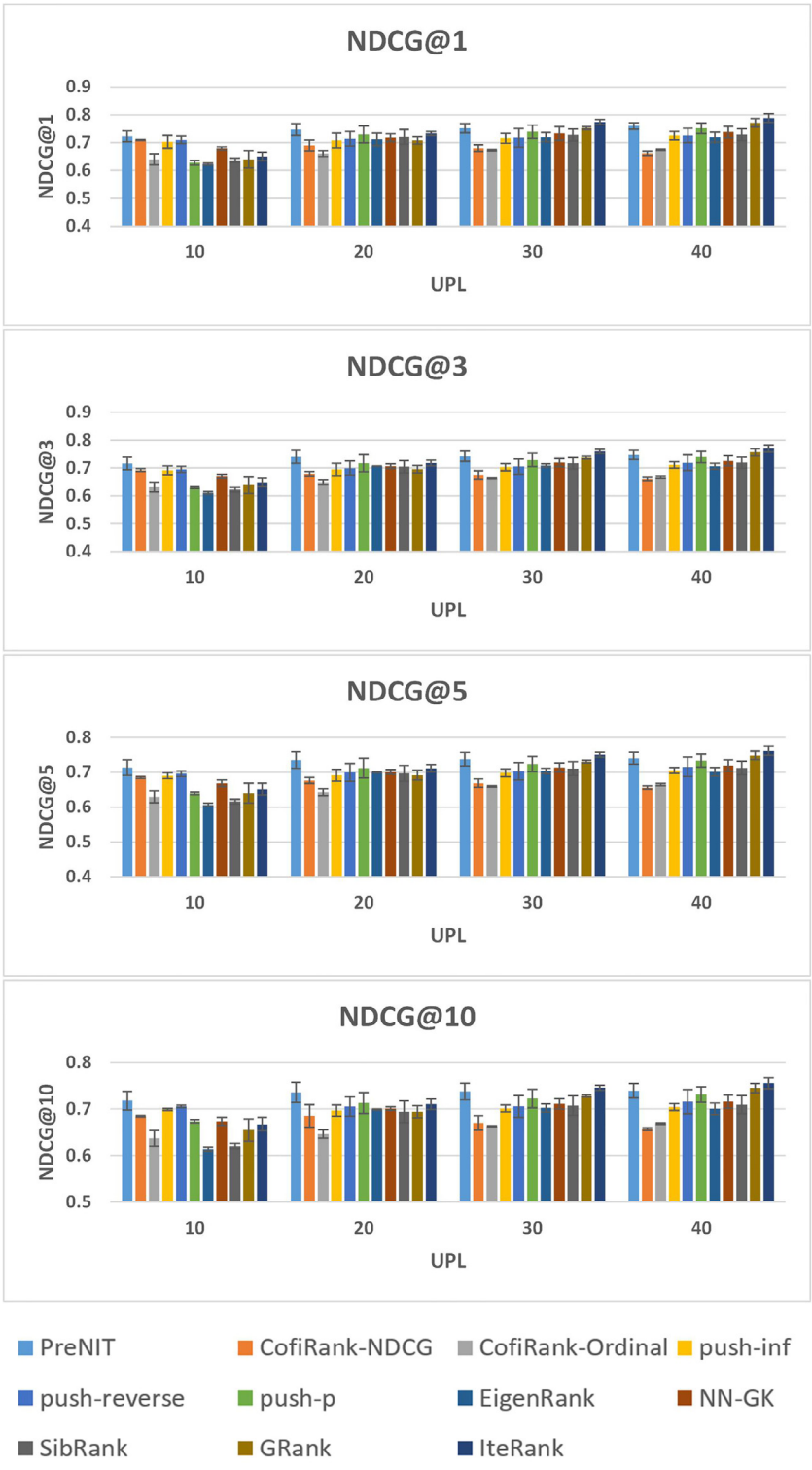

Fig. 4. Comparison of algorithms in Movielens1M dataset.

push and 6% better than rev-pushs performance when UPL is 30 in Movielens100K dataset. The reason can be that PushCR algorithms focus on learning the difference relevant and irrelevant items and ignore the priorities over relevant items. On the other hand, PreNIt seeks to learn the pairwise preferences over relevant items and so, it can achieve a higher performance in comparison to all PushCR algorithms.

5.3. Scalability analysis

We also studied the scalability of NCR algorithms through analyzing their computational complexity and the running time of the model construction and recommendation phases of each algorithm. Note that it is not acceptable to compare the running time of neighborhood collaborative ranking algorithms with model-based approaches [33] as neighborhood algorithms typically can make an

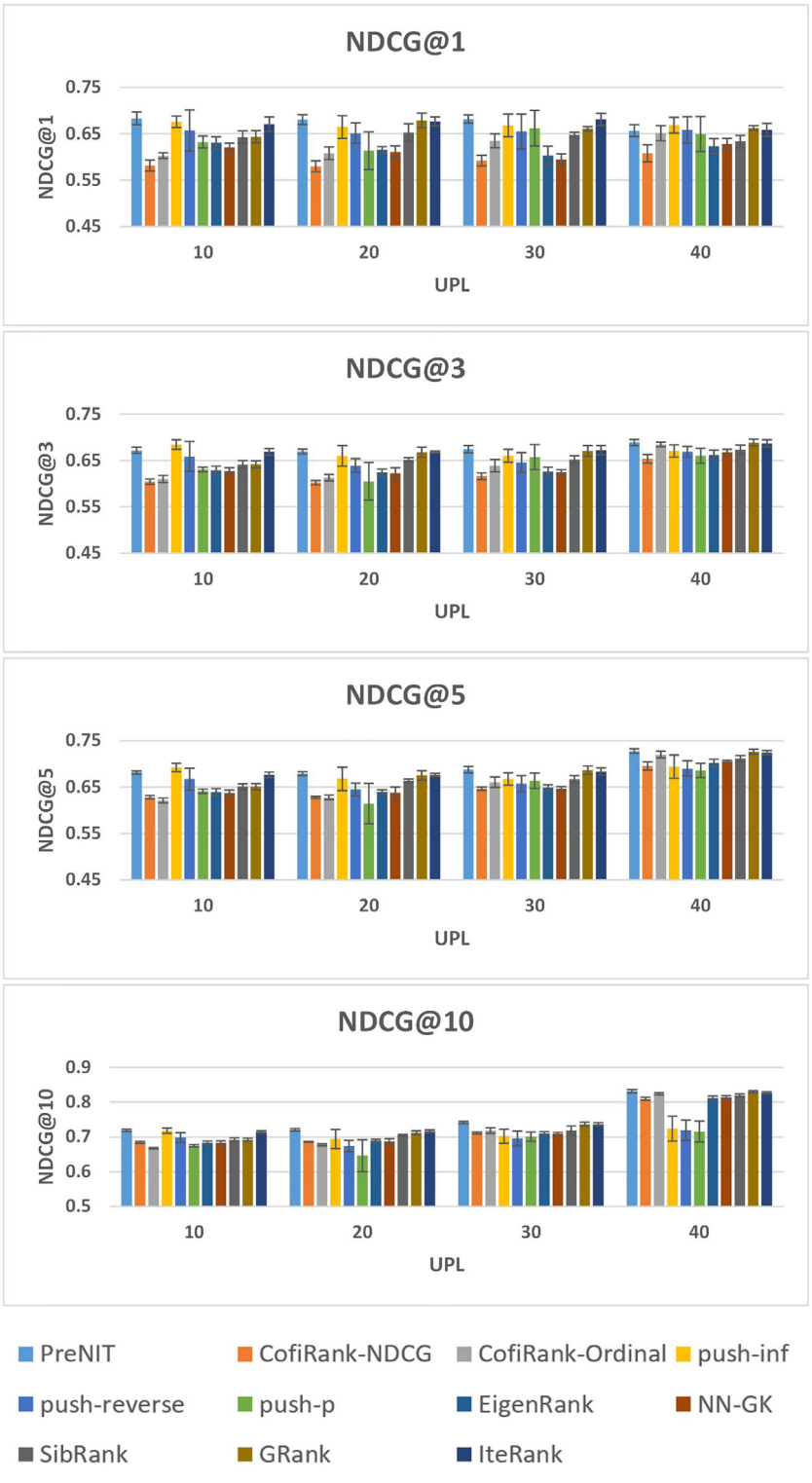

Fig. 5. Comparison of algorithms in FilmTrust dataset.

up-to-date recommendation without model reconstruction, while, model-based algorithms require to reconstruct their models in order to make up-to-date recommendations [17].

5.3.1. Model construction

Typically, neighborhood recommendation algorithms, such as EigenRank and NN-GK, do not construct any model for recommendation but some NCR methods may need to keep the data in a certain data structure, like a graph. However it is usually not a time-consuming task to construct the graph structure required by graph-based NCR methods. For instance, in Movielens1M, it takes 7.49 s for SiBRank, 32.56 s for PreNIt, 113.83 seconds for GRank and 142.66 s for IteRank on a Linux based PC running an Intel core i7-5820K processor at 3.3 GHz with 32GB of RAM.

Also, these algorithms can simply insert each new instance of data into their graph structure to keep them up-to-date. PreNIt

Table 3
P-values obtained for the paired *t*-test under each evaluation condition. $P \leq 0.05$ indicates the significant difference PreNIt and other algorithms. Bolded numbers indicate the significant outperformance of PreNIt, while, underlined numbers indicate significant improvement of other algorithm over PreNIt.

Dataset	UPL	topN	Cofi-NDCG	Cofi-Ordinal	inf-push	p-push	rev-push	EigenRank	SibRank	NN-GK	GRank	IteRank
ML100K	10	1	***0.01***	***0.001***	***0.048***	0.079	***0.000***	***0.002***	***0.001***	***0.011***	***0.002***	***0.004***
	10	3	***0.00***	***0.000***	***0.026***	***0.05***	***0.000***	***0.000***	***0.000***	***0.000***	***0.001***	***0.005***
	10	5	***0.000***	***0.000***	***0.024***	***0.041***	***0.000***	***0.000***	***0.000***	***0.000***	***0.002***	***0.006***
	10	10	***0.000***	***0.000***	***0.015***	***0.023***	***0.000***	***0.000***	***0.000***	***0.000***	***0.002***	***0.006***
	20	1	***0.000***	***0.005***	***0.006***	***0.017***	***0.012***	***0.012***	***0.004***	0.517	***0.005***	***0.026***
	20	3	***0.000***	***0.004***	***0.005***	***0.004***	***0.011***	***0.001***	***0.000***	0.785	***0.003***	***0.006***
	20	5	***0.001***	***0.002***	***0.004***	***0.005***	***0.01***	***0.002***	***0.001***	0.780	***0.005***	***0.010***
	20	10	***0.01***	***0.003***	***0.004***	***0.004***	***0.007***	***0.001***	***0.000***	0.212	***0.003***	***0.006***
	30	1	***0.000***	***0.000***	***0.019***	***0.036***	0.198	***0.045***	***0.018***	0.399	0.198	0.511
	30	3	***0.000***	***0.000***	***0.001***	***0.000***	***0.113***	***0.002***	***0.000***	0.287	***0.012***	0.681
	30	5	***0.000***	***0.000***	***0.000***	***0.001***	0.087	***0.000***	***0.000***	0.709	***0.005***	.564
	30	10	***0.000***	***0.000***	***0.000***	***0.000***	***0.055***	***0.000***	***0.000***	0.327	***0.000***	***0.000***
	40	1	***0.000***	***0.000***	***0.000***	***0.002***	0.109	0.288	0.717	0.798	0.206	0.105
	40	3	***0.000***	***0.000***	***0.000***	***0.003***	***0.006***	***0.045***	0.143	0.529	0.406	0.008
	40	5	***0.000***	***0.000***	***0.001***	***0.001***	***0.004***	***0.024***	***0.041***	0.938	0.554	0.008
	40	10	***0.000***	***0.000***	***0.000***	***0.000***	***0.006***	***0.000***	***0.004***	0.267	***0.046***	0.052
ML1M	10	1	0.238	***0.024***	0.077	***0.0372***	***0.000***	***0.001***	***0.021***	***0.008***	***0.034***	***0.024***
	10	3	0.136	***0.025***	**0.021**	0.053	**0.003**	***0.001***	0.053	***0.008***	***0.043***	***0.040***
	10	5	0.095	***0.024***	0.053	0.1259	**0.005**	***0.001***	0.066	***0.007***	***0.045***	***0.052***
	10	10	0.054	0.021	0.145	0.3106	**0.017**	**0.000**	0.054	**0.005**	**0.0468**	0.061
	20	1	***0.042***	***0.012***	**0.005**	**0.016**	**0.118**	***0.001***	***0.011***	***0.014***	0.106	0.402
	20	3	***0.024***	***0.011***	**0.000**	**0.001**	**0.021**	0.064	***0.021***	***0.000***	0.094	0.279
	20	5	***0.034***	***0.012***	**0.002**	**0.0019**	**0.009**	0.067	***0.029***	***0.000***	0.106	0.263
	20	10	0.114	**0.010**	**0.008**	**0.002**	**0.001**	**0.039**	**0.031**	**0.000**	0.0965	0.216
	30	1	***0.015***	***0.002***	***0.002***	0.098	0.068	***0.002***	***0.032***	***0.004***	0.947	***0.009***
	30	3	***0.027***	***0.003***	***0.004***	***0.032***	***0.022***	***0.015***	***0.003***	***0.001***	0.538	0.052
	30	5	***0.022***	***0.004***	***0.005***	***0.017***	***0.006***	***0.010***	***0.006***	***0.000***	0.477	0.115
	30	10	0.028	***0.004***	***0.008***	***0.007***	***0.001***	***0.005***	***0.006***	***0.000***	0.3244	0.295
	40	1	***0.001***	***0.000***	***0.002***	***0.026***	0.203	***0.002***	***0.027***	***0.016***	0.017	0.001
	40	3	***0.001***	***0.002***	***0.003***	***0.023***	0.051	***0.001***	***0.006***	***0.002***	0.018	0.001
	40	5	***0.001***	***0.003***	***0.006***	***0.028***	***0.008***	***0.001***	***0.002***	***0.000***	0.047	0.002
	40	10	***0.002***	***0.003***	***0.005***	***0.0225***	***0.003***	***0.000***	***0.000***	***0.000***	0.116	***0.003***
FT	10	1	***0.000***	***0.000***	0.557	0.329	***0.000***	***0.000***	***0.001***	***0.001***	***0.001***	0.081
	10	3	***0.000***	***0.000***	0.154	0.454	***0.000***	***0.000***	***0.000***	***0.000***	***0.001***	0.216
	10	5	***0.000***	***0.000***	0.082	0.276	***0.000***	***0.000***	***0.000***	***0.000***	***0.000***	0.055
	10	10	***0.000***	***0.000***	0.748	***0.050***	***0.000***	***0.000***	***0.000***	***0.000***	***0.000***	***0.000***
	20	1	***0.000***	***0.000***	0.139	***0.038***	***0.023***	***0.000***	***0.000***	***0.003***	0.842	0.318
	20	3	***0.000***	***0.000***	0.303	***0.008***	0.023	***0.000***	***0.000***	***0.000***	0.747	0.597
	20	5	***0.000***	***0.000***	0.325	***0.005***	0.024	***0.000***	***0.000***	***0.000***	0.270	***0.003***
	20	10	***0.000***	***0.000***	0.075	***0.002***	***0.018***	***0.000***	***0.000***	***0.000***	***0.005***	***0.001***
	30	1	***0.000***	***0.004***	0.254	0.252	0.308	***0.000***	***0.000***	***0.000***	***0.013***	0.753
	30	3	***0.000***	***0.000***	0.062	0.063	0.224	***0.000***	***0.000***	***0.001***	0.247	0.184
	30	5	***0.000***	***0.003***	***0.017***	***0.038***	***0.015***	***0.000***	***0.000***	***0.009***	0.710	***0.015***
	30	10	***0.000***	***0.003***	***0.011***	***0.011***	***0.001***	***0.000***	***0.000***	***0.019***	0.089	***0.001***
	40	1	***0.007***	0.480	0.315	0.920	0.750	***0.017***	***0.014***	***0.028***	0.372	0.325
	40	3	***0.000***	0.357	***0.031***	***0.018***	***0.036***	***0.002***	***0.000***	***0.030***	0.975	0.316
	40	5	***0.000***	0.107	0.027	***0.004***	***0.004***	***0.004***	***0.000***	***0.011***	0.805	***0.019***
	40	10	***0.000***	0.025	***0.002***	***0.000***	***0.000***	***0.000***	***0.000***	***0.003***	0.46	***0.002***

and SibRank update their graphs at computational complexity of $O(1)$ and GRank and IteRank update their graphs at computational complexity of $O(1)$ for inserting a new user or a new instance of pairwise preference, and at computational complexity of $O(n)$ for adding a new item. Therefore, Therefore, NCR algorithms can make an up-to-date recommendation without any model reconstruction, and so, the overall computational complexity of graph-based NCR algorithms mainly depends on the approach they use to score items for a given user.

5.3.2. Recommendation

Fig. 6 illustrates the recommendation time of neighborhood collaborative ranking algorithms measured in seconds on a Linux based PC running an Intel core i7-5820K processor at 3.3 GHz with 32GB of RAM. As shown in Fig. 6, PreNIt makes recommendation faster than any other NCR algorithm in all evaluation conditions. For instance, in case of UPL=10, PreNIt makes a recommendation to a target user in about 25 milliseconds in Movielens1M datasets where the recommendation time is about 2 seconds for traditional user-based approaches, 6 seconds for IteRank, and 10 seconds for GRank, that is an expected observation as PreNIt makes recommendation at complexity of $O(ts)$, GRank and IteRank at computational complexity of $O(t(s+n^2))$ and neighborhood approach at computational complexity of $O(mn^2+kn^2+t'n^2)$ where t is the number of iterations each algorithm requires to converge.

We also analyze the scalability of NCR algorithms by varying the size of the pairwise preference dataset that they use. For this purpose, we filter out users who have rated less than 50 items and 100 pairwise comparisons in Movielens1M dataset and then, measure the running time of recommendation phase when the average number of pairwise preferences (i.e. $k = \frac{s}{m}$) varies from 20 to 100 (see Table 4). As shown in Table 4, PreNIt makes recommendation in less than a second, much faster than other algorithms. Note that running time of GRank and IteRank slightly increase over different k values as $s \ll n^2$ and so their computational complexity is highly affected by the number of items rather than the size of dataset. We also remind that s is maximally equal to mn^2 and so, the running time of PreNIt would be always less than other baseline NCR algorithms.

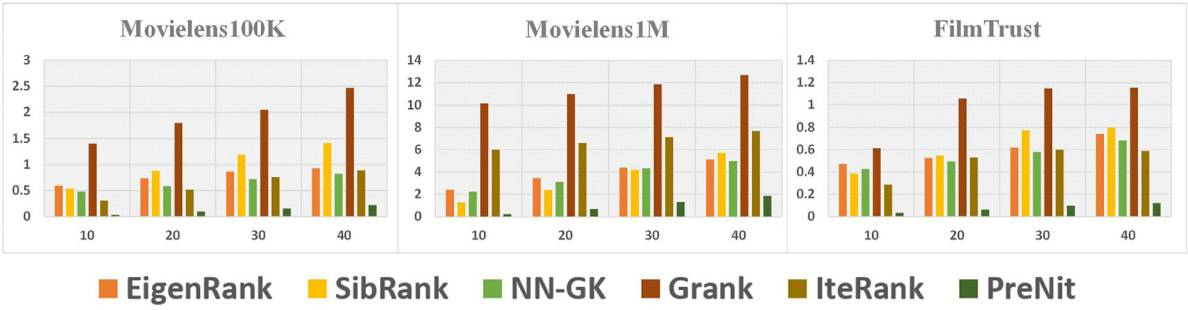

Fig. 6. Running time of neighborhood collaborative ranking algorithms.

Table 4
The running time recommendation phase in collaborative ranking algorithms based on average number of pairwise preferences for each user.

	Computational Complexity	k=20	k=40	k=60	k=80	k=100
PreNIt	$O(t's)$	0.06	0.10	0.15	0.18	0.21
EigenRank	$O(mn^2 + kn^2 + t'n^2)$	6.17	6.60	6.93	7.37	7.97
SibRank	$O(ts + kn^2 + t'n^2)$	2.37	2.53	2.80	2.90	3.08
NN-GK	$O(mn^2 + kn^2 + n^2)$	5.61	6.53	6.78	7.22	7.81
GRank	$O(t(s + n^2))$	8.73	8.81	8.83	8.88	9.08
IteRank	$O(ts + t'n^2)$	13.95	13.97	14.18	14.28	14.38

6. Conclusion

In this paper, we proposed a novel framework, called PreNIt, that adapts item-based recommendation for neighborhood collaborative ranking. PreNIt is the first algorithm which takes into account items' similarities rather than users' similarities in pairwise preference datasets. PreNIt first models the pairwise preference dataset as two bipartite networks with labeled edges, called LNet and WNet. Then, it uses a novel personalized ranking algorithm, called MemoRank, to calculate items' transitive similarities based on the choice context in which items are preferred/not preferred. Thereafter, it gives high scores to items that are more similar to the preferred items of the target user and not similar to the items over which the target user has preferred other items.

Experimental results demonstrated that PreNIt outperforms traditional neighborhood collaborative ranking (tNCR) methods and Matrix factorization approaches in all evaluation condition. However, its improvement over tNCR algorithm is more significant in sparse datasets while its improvement over CofiRank is higher in dense datasets. The reason is that PreNIt, as a graph-based algorithm, pays attention to local tastes of each user as well as the global tastes of all users depending on the degree of sparsity of the data while the recommendation by other methods, tNCR or CofiRank, is mainly based on just one of those kinds of taste analysis. Therefore, PreNIt can effectively exploit the global tastes of users in sparse datasets where local information is not enough for tNCR algorithms to make good recommendation. On the other hand, it can properly recognize the personal and local tastes of users and improves the performance of CofiRank in dense datasets. The results also implied that though PreNIt can significantly outperforms GRank and IteRank in sparse datasets, GRank and IteRank can slightly perform better in dense datasets. That relies on the fact that GRank and IteRank explore other types of preference-item relations that are reliable in dense datasets.

Unfortunately, PreNIt can only incorporates the pairwise preference datasets that does not contain any conflict. However, it is expected that users' pairwise preferences change over the time. For instance, a user might prefer a romantic movie a over a comic one b when he is with the family, while, he prefers b over a when he is with his friends. Therefore, it seems to be useful to design novel graph structures and algorithms that can incorporate this type of situation in which some inconsistencies exist in the datasets possibly by considering the content and context information. Additionally, PreNIt only considers the choice context in which items are preferred/not preferred to calculate items' similarities and neglects the degree of certainty of user for each preference. For example, a user might view three items a, b and c and then he buys a and like b in some session. Though he prefers both of a and b over c, his preference of a over c is stronger than his preference of b over c. We may be able to design more accurate similarity measures through considering the strength of preferences. Furthermore, PreNIt does not integrate items' attributes (e.g. movie's genre, director, etc.) with pairwise preferences to calculate items' similarities. So, designing algorithms and graph structures to simultaneously handle items' attributes and users' pairwise preferences is another potential extension to the current research.

References

[1] N. Liu, Q. Yang, EigenRank : a ranking-oriented approach to collaborative filtering, in: Proceedings of the 31st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, 2008, pp. 83–90. http://dl.acm.org/citation.cfm?id=1390351.

[2] Y. Shi, M. Larson, A. Hanjalic, List-wise learning to rank with matrix factorization for collaborative filtering, in: Proceedings of the Fourth ACM Conference on Recommender System, 2010, pp. 269–272. http://dl.acm.org/citation.cfm?id=1864764.

[3] Y. Shi, M. Larson, A. Hanjalic, Unifying rating-oriented and ranking-oriented collaborative filtering for improved recommendation, Inf. Sci. (Ny) 229 (2013) 29–39.

[4] A. Ammar, D. Shah, Efficient rank aggregation using partial data, ACM SIGMETRICS Perform. Eval. Rev. 40 (1) (2012) 355–366.

[5] A.K. Jain, S. Jain, J. Yi, R. Jin, Inferring users preferences from crowdsourced pairwise comparisons : a matrix completion approach, in: Proceedings of the First AAAI Conference on Human Computation and Crowdsourcing., 2013.

[6] S. Hacker, L.V. Ahn, Matchin : eliciting user preferences with an online game, in: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems. ACM, 2009, pp. 1207–1216.

[7] S. Kalloori, F. Ricci, M. Tkalcic, Pairwise preferences based matrix factorization and nearest neighbor recommendation techniques, in: Proceedings of the 10th ACM Conference on Recommender Systems - RecSys '16, 2016, pp. 143–146, doi:10.1145/2959100.2959142.

[8] S. Rendle, C. Freudenthaler, Z. Gantner, L. Schmidt-thieme, BPR : Bayesian personalized ranking from implicit feedback, in: Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence, 2009, pp. 452–461, doi:10.1145/1772690.1772773.

[9] Y. Shi, A. Karatzoglou, L. Baltrunas, Climf: learning to maximize reciprocal rank with collaborative less-is-more filtering, in: Proceedings of the Sixth ACM Con-

ference on Recommender Systems. ACM, 2012, pp. 139–146. http://dl.acm.org/citation.cfm?id=2365981.
[10] W. Pan, H. Zhong, C. Xu, Z. Ming, Adaptive bayesian personalized ranking for heterogeneous implicit feedbacks, Knowl. Based Syst. 73 (2015) 173–180, doi:10.1016/j.knosys.2014.09.013.
[11] M. Weimer, A. Karatzoglou, Maximum margin matrix factorization for collaborative ranking, Adv. Neural Inf. Process. Syst. (2007) 1–8.
[12] B. Shams, S. Haratizadeh, Graph-based collaborative ranking, Expert. Syst. Appl. 67 (2017) 59–70, doi:10.1016/j.eswa.2016.09.013.
[13] N. Jones, A. Brun, A. Boyer, Comparisons instead of ratings: towards more stable preferences, in: Proceedings of the 2011 IEEE/WIC/ACM International Conference on Web Intelligence, WI 2011, 1, 2011, pp. 451–456, doi:10.1109/WI-IAT.2011.13.
[14] A. Satsiou, L. Tassiulas, Propagating Users' similarity towards improving recommender systems, in: Proceedings of the 2014 IEEE/WIC/ACM International Joint Conferences on Web Intelligence (WI) and Intelligent Agent Technologies (IAT), 2014, pp. 221–228, doi:10.1109/WI-IAT.2014.37.
[15] M. Gori, A. Pucci, V. Roma, I. Siena, ItemRank: a random-walk based scoring algorithm for recommender engines, IJCAI 7 (2007) 2766–2771.
[16] S. Wang, J. Sun, B.J. Gao, VSRank : a novel framework for ranking-based collaborative filtering, ACM Trans. Intel. Syst. Technol. 5 (3) (2014) 51.
[17] C. Desrosiers, G. Karypis, A comprehensive survey of neighborhood-based recommendation methods, in: Recommender Systems Handbook, 2011, pp. 107–144..
[18] Y. Shi, A. Karatzoglou, L. Baltrunas, M. Larson, xCLiMF : optimizing expected reciprocal rank for data with multiple levels of relevance, in: Proceedings of the 7th ACM Conference on Recommender Systems, 2013, pp. 431–434.
[19] G. Li, Z. Zhang, L. Wang, Q. Chen, J. Pan, One-class collaborative filtering based on rating prediction and ranking prediction, Knowl. Based Syst. 124 (2017) 46–54, doi:10.1016/j.knosys.2017.02.034.
[20] G. Li, Q. Chen, Exploiting explicit and implicit feedback for personalized ranking, Math. Probl. Eng. 2016 (2016), doi:10.1155/2016/2535329.
[21] K. Christakopoulou, A. Banerjee, Collaborative ranking with a push at the top, in: Proceedings of the 24th International Conference on World Wide Web, 2015, pp. 205–215, doi:10.1145/2736277.2741678.
[22] G. Li, W. Ou, Pairwise probabilistic matrix factorization for implicit feedback collaborative filtering, Neurocomputing 204 (2016) 17–25, doi:10.1016/j.neucom.2015.08.129.
[23] L. Lerche, D. Jannach, Using graded implicit feedback for Bayesian personalized ranking, in: Proceedings of the 8th ACM Conference on Recommender systems. ACM, 2014, pp. 353–356.
[24] W. Pan, H. Zhong, C. Xu, Z. Ming, Adaptive Bayesian personalized ranking for heterogeneous implicit feedbacks, Knowl. Based Syst. 73 (2015) 173–180.
[25] G. Guo, H. Qiu, Z. Tan, Y. Liu, J. Ma, X. Wang, Resolving data sparsity by multi-type auxiliary implicit feedback for recommender systems, Knowl. Based Syst. 138 (2017) 202–207, doi:10.1016/j.knosys.2017.10.005.
[26] G. Li, L. Wang, W. Ou, Robust personalized ranking from implicit feedback, Int. J. Pattern Recognit Artif Intell. 30 (01) (2016) 1659001, doi:10.1142/S0218001416590011.
[27] D. Rafailidis, F. Crestani, Joint collaborative ranking with social relationships in top-n recommendation, in: Proceedings of the 25th ACM International on Conference on Information and Knowledge Management, ACM, 2016, pp. 1393–1402.
[28] D. Rafailidis, F. Crestani, Collaborative ranking with social relationships for Top-N recommendations, in: Proceedings of the 39th International ACM SIGIR conference on Research and Development in Information Retrieval - SIGIR '16, 2016, pp. 785–788, doi:10.1145/2911451.2914711.
[29] A. Ukkonen, Clustering algorithms for chains, J. Mach. Learn. Res. 12 (2011) 1389–1423. http://dl.acm.org/citation.cfm?id=2021045.
[30] B. Ackerman, Y. Chen, Evaluating rank accuracy based on incomplete pairwise preferences, in: Proceedings of the CEUR Workshop Proceedings, 811, 2011, pp. 74–77. http://ceur-ws.org/Vol-811/paper11.pdf.
[31] B. Shams, S. Haratizadeh, Sibrank: signed bipartite network analysis for neighbor-based collaborative ranking, Physica A 458 (2016) 364–377, doi:10.1016/j.physa.2016.04.025.
[32] I. AvSegal, Z. Katzir, K. Gal, EduRank : a collaborative filtering approach to personalization in e-learning, in: Proceedings of the Educational Data Mining 2014, in: Edm, 2014, pp. 68–75.
[33] B. Shams, S. Haratizadeh, Iterank: an iterative network-oriented approach to neighbor-based collaborative ranking, Knowl. Based Syst. 0 (2017) 1–13, doi:10.1016/j.knosys.2017.05.002.
[34] X. He, M. Gao, M.-y. Kan, D. Wang, Birank : towards ranking on bipartite graphs, IEEE Trans. Knowl. Data Eng. 29 (1) (2017) 57–71, doi:10.1109/TKDE.2016.2611584.
[35] F.L. Cruz, C.G. Vallejo, F. Enríquez, J.A. Troyano, Polarityrank: finding an equilibrium between followers and contraries in a network, Inform. Process. Manage. 48 (2) (2012) 271–282, doi:10.1016/j.ipm.2011.08.003.
[36] A. Silva, M. Zaki, ProfileRank : finding relevant content and influential users based on information diffusion, in: Proceedings of the 7th Workshop on Social Network Mining and Analysis. ACM, 2013, pp. 1–9, doi:10.1145/2501025.2501033.
[37] S. Lee, S. Park, M. Kahng, S.-g. Lee, Pathrank : ranking nodes on a heterogeneous graph for flexible hybrid recommender systems, Expert Syst. Appl. 40 (2013) 684–697.
[38] I.C.F. Ipsen, R.S. Wills, Mathematical properties and analysis of google s pagerank, Bol. Soc. Esp. Mat. Apl 34 (2006) 191–196.
[39] M.N. Volkovs, R.S. Zemel, Collaborative Ranking With 17 Parameters, in: Advances in Neural Information Processing Systems, 2012, pp. 2294–2302.
[40] F.M. Harper, J.A. Konstan, The movielens datasets: history and context, ACM Trans. Interact. Intel. Syst. (TiiS) 5 (4) (2016) 19.
[41] G. Guo, J. Zhang, N. Yorke-Smith, A novel bayesian similarity measure for recommender systems, in: Proceedings of the 23rd International Joint Conference on Artificial Intelligence (IJCAI), 2013, pp. 2619–2625.
[42] G. Guo, J. Zhang, N. Yorke-Smith, Leveraging multiviews of trust and similarity to enhance clustering-based recommender systems, Knowl. Based Syst. 74 (2015) 14–27, doi:10.1016/j.knosys.2014.10.016.
[43] G. Shani, A. Gunawardana, Evaluating recommendation systems, in: Recommender Systems Handbook, 2011, pp. 257–298, doi:10.1007/978-0-387-85820-3_8.