SQL-Rank++: A Novel Listwise Approach for Collaborative Ranking with Implicit Feedback

Zheng Yuan, Dugang Liu, Weike Pan* and Zhong Ming*
College of Computer Science and Software Engineering, Shenzhen University
Guangdong Laboratory of Artificial Intelligence and Digital Economy (SZ)
Shenzhen, China
yuanzheng2017@email.szu.edu.cn, dugang.ldg@gmail.com, {panweike,mingz}@szu.edu.cn

2022 International Joint Conference on Neural Networks (IJCNN) | 978-1-7281-8671-9/22/$31.00 ©2022 IEEE | DOI: 10.1109/IJCNN55064.2022.9891935

***Abstract*—Collaborative ranking with implicit feedback such as users' clicks is an important recommendation problem in various real-world applications. Most existing approaches are developed based on some pointwise or pairwise preference assumptions, although the listwise assumption is widely accepted as a better alternative due to its consistency with the final delivery result. In this paper, we first identify two fundamental limitations of the most current collaborative listwise approaches, in which their modeling is based on the Plackett-Luce probability. They are too strict and too weak relative preference comparison between the items with the same feedback and between the items with different feedback, respectively. As a response, we propose a novel and improved listwise approach called SQL-Rank++, which is able to learn the user preferences more accurately by leveraging some specifically constructed auxiliary lists, including some positive lists and some negative lists. Specifically, the positive lists have as much semantic consistency as the original list as possible, while the negative lists are the opposite. To construct these auxiliary lists, we design a self-based sampling strategy and a user similarity-based one. Finally, we have four variants of our SQL-Rank++ with different combinations of the auxiliary lists. We then conduct extensive experiments on four public datasets, and find that our SQL-Rank++ achieves very promising performance in comparison with several pointwise, pairwise and listwise approaches. We also study the influence of the two sampling strategies and the key components in our SQL-Rank++.**

***Index Terms*—collaborative ranking, implicit feedback, listwise, auxiliary list**

I. Introduction

Recommender systems usually use a user's historical feedback data to analyze the user's preference, where the data types include explicit feedback and implicit feedback [1], [2]. Explicit feedback can intuitively show how much a user likes an item, but it is difficult to be collected in many application scenarios, such as user ratings for movies or music. On the contrary, implicit feedback usually refers to a user's interaction with an item, and can be easily collected in actual applications, such as a user's click and purchase on an item. Therefore, how to effectively improve the recommendation performance based on implicit feedback is an important research topic that has received more and more attention.

Most recommendation approaches based on implicit feedback are designed by adopting a pointwise or pairwise preference assumption. A pointwise approach only considers a user's feedback label on a specific item to model the user's preference [3]–[7], and a pairwise approach introduces negative sampling to further model the comparison relationship between samples [2], [8]. However, a preference assumption that is more consistent with the goal of a real-world recommender system is listwise, that is, to accurately model a user's preference distribution in a displayed item recommendation list. Compared with the other two forms, there are very few works focusing on modeling the users' implicit feedback from a listwise perspective.

Previous works in this line can be categorized into two classes, including the approaches based on the Plackett-Luce probability and the approaches optimizing a certain ranking metric. The former mainly constructs an item list (usually putting the observed feedback in the first half of the list and the unobserved feedback in the second half) for a user and uses the Plackett-Luce probability to calculate the sorting relationship between the items in the list, and then obtains the matching degree of this list with the user [9]–[12]. The latter usually optimizes a desired ranking metric directly by deriving a smoothed form of the ranking metric [13]–[16]. Since the approaches based on the Plackett-Luce probability have better interpretability and scalability, we focus on improving the performance of these approaches in this paper.

We first identify the approaches based on the Plackett-Luce probability generally have the following two limitations: 1) The constraints on the items with the same feedback in the list are often too strict. These approaches often construct an item list and assume that it obeys a descending preference sorting. Because of this, the items with the same feedback are also modeled by the strict sorting relationship, so any unreasonable position pairs between them will seriously damage the performance of the approach. 2) The constraints between the items with different feedback are usually too weak. The goal of these listwise approaches is to ensure that the overall ranking of the item list is correct, which means that the gap between the observed feedback and the unobserved feedback may be too small, especially the latter part of the observed feedback and the front part of the unobserved feedback.

To solve the above two limitations, by introducing some special auxiliary lists and combining them with the original list for joint learning, we propose a novel listwise approach called SQL-Rank++. Specifically, our approach uses a state-

*: Corresponding author.

of-the-art and representative listwise approach as the skeleton, i.e., SQL-Rank [12], which is based on the Plackett-Luce probability. The introduced auxiliary lists in our SQL-Rank++ are then divided into two types, including some positive lists and some negative lists. The positive lists and the original list maintain the semantic consistency as much as possible, while the negative lists are the opposite.

We propose a self-based sampling strategy and a user similarity-based one to construct some auxiliary lists. The self-based sampling strategy shuffles the observed and unobserved parts of the original list to construct some positive lists, and only samples from the set of unobserved items to obtain some negative lists. Based on collaborative ranking, the user similarity-based sampling strategy defines the neighbors and dissimilar users for each user, and then use the original lists of the neighboring and dissimilar users as the positive lists and the negative lists, respectively.

There are some works that use the pairwise loss as an auxiliary loss function to capture the hidden implicit information [17], [18]. And some works focus on the sampling strategy and the partitioned preference problem of listwise learning to rank [19], [20]. However, our work is fundamentally different from the above works. For the former, we emphasize the distinction between the observed and unobserved parts of the lists, which is different from the traditional pairwise methods and is also more complicated. We show its impact of our SQL-Rank++ in Section V-B3. For the latter, these works mainly focus on reducing the time complexity of the listwise approach, while we focus on improving the effectiveness of the model through the comparison of various feedback and better sampling strategies. At the same time, our SQL-Rank++ has near-linear time complexity and we show the related analysis in Section V-B2.

According to the different combinations of the auxiliary lists, we have four variants, i.e., SQL-Rank++(P), SQL-Rank++(N), SQL-Rank++(P vs. N) and SQL-Rank++(P + N). Finally, extensive empirical studies on four public real-world datasets show that our SQL-Rank++ performs better than several state-of-the-art approaches. We also compare the performance between different variants of our SQL-Rank++, and discuss the influence of different sampling strategies. To fully study the effectiveness of our approach, we conduct quantitative study to discuss the impact of the number of auxiliary lists, and ablation study to show the effect of the key components.

II. Related Work

In this paper we focus on proposing a novel approach to solve the possible limitations of the listwise approaches in the context of implicit feedback. Hence, in this section, we briefly introduce the previous recommendation approaches based on implicit feedback, including pointwise, pairwise and listwise approaches.

Pointwise approaches usually treat implicit feedback as a binary preference assumption, in which an observed feedback is taken as a positive sample, and a user's preference to an unobserved feedback is unknown. Pan et al. introduce a weight to measure the confidence of taking an unobserved feedback as a negative sample to solve this problem [3]. Johnson et al. propose logical matrix factorization to accurately model the likelihood that a user will prefer a specific item [4]. Some researchers focus on the weighting strategy of matrix factorization, e.g., He et al. weight the unobserved items non-uniformly and propose a fast learning approach to reduce the time complexity [21]. Chen et al. use a parameterized neural network to generate more adaptive confidence weights for weighted matrix factorization and develop a new batch-based learning algorithm to support fast and stable learning [22]. In addition, there are some works to further model the user preferences by considering different influencing factors, such as the exposure process of items [5], the similarity between items [6], [23], and the diversity of the user preferences [7].

Instead of modeling a user's preference to one single certain item, pairwise approaches aim to model the difference between the preference of a user to an observed item and that to an unobserved one. One of the most representative approaches is Bayesian personalized ranking (BPR) [2]. It assumes that a user's preference to an observed item is larger than that to an unobserved one, and directly optimizes the user's preference difference. Furthermore, a series of works improve the performance of BPR by considering the different combinations between a single item and an item set, such as an observed item vs. a set of sampled unobserved items [24], a set of sampled observed items vs. a set of sampled unobserved items [25], [26]. In addition, with the popularity of deep learning, most recent recommendation approaches adopt the idea of pairwise preference assumption in the loss function to learn more accurate user preferences [27].

The listwise approaches have received relatively little attention. We present most of the research works in this line in Table I. The approaches optimizing a certain ranking metric such as the normalized discounted cumulative gain [13], the mean reciprocal rank [14]–[16] and the mean average precision [15] usually derive a smooth form of the desired metric, and then directly optimize it. The approaches based on the Plackett-Luce probability first describe the sorting relationship between the observed feedback and the unobserved feedback in a list, and then optimize the Plackett-Luce probability of the list for a user [9]–[12]. A recent representative approach is SQL-Rank [12], which weakens the influence of sorting noise between the observed items by shuffling the observed items and re-sampling the unobserved items in each iteration (called stochastic queuing process). In this paper, we aim to further address the possible limitations of the approaches based on the Plackett-Luce probability.

Although there are many new recommendation frameworks such as graph convolution networks [28], [29] and variational autoencoders [30], [31], we still follow most previous listwise approaches and use the factorization framework for direct comparison. In particular, we use SQL-Rank as the skeleton of the proposed approach.

TABLE I
SUMMARY OF LISTWISE COLLABORATIVE RANKING APPROACHES

Categorization	Typical work	Year	Feedback	Top-N
Approaches based on the Plackett-Luce probability	listRank-MF [9]	2010	Explicit	1
	listPMF [10]	2014	Explicit	1
	listCF [11]	2015	Explicit	1
	SQL-Rank [12]	2018	Explicit, Implicit	K
Approaches optimizing a certain ranking metric	CoFiRank [13]	2010	Explicit	K
	CLiMF [14]	2012	Implicit	K
	CLAPF [15]	2020	Implicit	K
	LRVAE [16]	2020	Implicit	K

III. Preliminaries

In this section, we present the notations and the definition of the studied problem. Taking the state-of-the-art listwise approach SQL-Rank as an example, we introduce the limitations of the listwise approach based on the Plackett-Luce probability in modeling implicit feedback.

A. Notations and Problem Definition

We use $\mathcal{U}$ and $\mathcal{I}$ to denote the sets of users and items, where $n = |\mathcal{U}|$ and $m = |\mathcal{I}|$ are the numbers of users and items, respectively. The set of items preferred by user u is $\mathcal{I}_u$. We have a set $\mathcal{R} = \{(u, i)\}$, which contains the observed implicit feedback from users and items. Our goal is to generate a personalized ranking list of items to each user by learning the user's preference from $\mathcal{R}$. For most listwise approaches, they usually construct a ranking list Π_u of all the observed items and some unobserved items sampled from $\mathcal{I}\backslash\mathcal{I}_u$ for user u and then optimize the sorting relationship in this list. These listwise approaches often assume that the items ranked at higher positions in the list are more preferred by the user. Some notations and their explanations used in the paper are shown in Table II.

B. The Plackett-Luce Probability-based Approach: SQL-Rank

A Plackett-Luce probability-based approach uses the permutation probability of Π_u as the optimization goal [32], which is defined as follows:

$$P(\Pi_u) \stackrel{\text{def}}{=} \prod_{j=1}^{L_u} \frac{\phi(S_{u\Pi_{uj}})}{\sum_{h=j}^{L_u} \phi(S_{u\Pi_{uh}})} \tag{1}$$

where Π_{uj} denotes the j-th item in Π_u ($1 \leqslant j \leqslant L_u$). A user's score on item Π_{uj} is represented by $S_{u\Pi_{uj}}$. Notice that L_u is the number of items in Π_u. And $\phi(\cdot)$ is an increasing and strictly positive function. The intuition of this probability is that the higher the scores on the top-ranked items, the greater the probability values.

SQL-Rank [12] introduces this permutation probability into collaborative ranking by letting $S_{u\Pi_{uj}} = \sigma(\hat{r}_{u\Pi_{uj}})$ and $\phi(\cdot) = \exp(\cdot)$, where $\hat{r}_{u\Pi_{uj}} = U_{u\cdot}V^T_{\Pi_{uj}\cdot}$ is the predicted preference of user u to item Π_{uj}. Notice that $U_{u\cdot} \in \mathbb{R}^{1\times d}$ and $V_{\Pi_{uj}\cdot} \in \mathbb{R}^{1\times d}$ are the latent feature vectors of user u and item Π_{uj}, respectively [12]. $\sigma(x) = 1/(1+e^{-x})$ is the sigmoid function.

TABLE II
SOME NOTATIONS AND EXPLANATIONS

$\mathcal{U}$	set of users
$\mathcal{I}$	set of items
$n = \lvert\mathcal{U}\rvert$	user number
$m = \lvert\mathcal{I}\rvert$	item number
$\mathcal{I}_u$	set of items preferred by user u in training data
$\mathcal{R} = \{(u, i)\}$	set of all observed (user, item) pairs in training data
$\mathcal{R}^{te} = \{(u, i)\}$	set of all observed (user, item) pairs in test data
L_u	number of items in the item ranking list of user u
Π_u	item ranking list of user u
Π_{uj}	item ID of index j in Π_u
$\mathcal{P}_u$	a set of positive lists of user u
$\mathcal{N}_u$	a set of negative lists of user u
$\mathcal{P}_u^g$	g-th list in $\mathcal{P}_u$
$\mathcal{N}_u^g$	g-th list in $\mathcal{N}_u$
$s_p = \lvert\mathcal{P}_u\rvert$	size of $\mathcal{P}_u$
$s_n = \lvert\mathcal{N}_u\rvert$	size of $\mathcal{N}_u$
$d \in \mathbb{R}$	number of latent dimensions
$U_{u\cdot} \in \mathbb{R}^{1\times d}$	user-specific latent feature vector
$V_{i\cdot} \in \mathbb{R}^{1\times d}$	item-specific latent feature vector
$\hat{r}_{ui}$	predicted preference of user u to item i
T	iteration number

The loss function of SQL-Rank is then as follows after the log transformation:

$$\mathcal{L}_{sql} = \sum_{j=1}^{L_u} -\ln \frac{\exp(\sigma(\hat{r}_{u\Pi_{uj}}))}{\sum_{h=j}^{L_u} \exp(\sigma(\hat{r}_{u\Pi_{uh}}))} \tag{2}$$

SQL-Rank also proposes a stochastic queuing process, which shuffles the observed items and re-samples the unobserved items in Π_u in each training iteration. This process helps deal with the sorting noise between the observed items, and the neglect of the unobserved items [12].

However, for collaborative ranking with implicit feedback, the listwise approaches based on the Plackett-Luce probability still have two limitations:

Limitation 1. The preference comparison between the items with the same feedback in the list is often too strict. From implicit feedback, we only know which interaction has been observed and which has not, then the preference comparison relationship for the items with the same feedback is unknown. These approaches simply assume that a user's preference to a certain item is greater than that to all the items behind it. This assumption usually leads to the incorrect comparison between the items with the same feedback. Even though SQL-Rank uses a stochastic queuing process to alleviate the comparison between the observed items, it does not take into account that the unobserved items have the same problem.

Limitation 2. The preference comparison between the items with different feedback are usually too weak. These approaches maintain the overall ranking of an item list. In this list, the observed items are placed in the first half and the unobserved items sampled from $\mathcal{I}\backslash\mathcal{I}_u$ are placed in the second half. This may cause the preference gap between the observed items and the unobserved items to be too small. For example, the items at the back of the observed part and the items at the front of the unobserved part are always treated

similarly. According to collaborative ranking with implicit feedback, the preference comparison between the observed items and the unobserved items is important, which means that weakening this relationship is likely to bring incorrect training guidance. SQL-Rank alleviates this limitation by re-sampling the unobserved items when constructing the new item list in each iteration. However, the new item list still suffers from this limitation, and the model may not learn enough useful information when the unobserved part of the item list changes constantly.

In order to solve these two limitations in the listwise approaches based on the Plackett-Luce probability, we propose an improved approach called SQL-Rank++, which contains four variants, i.e., SQL-Rank++(P), SQL-Rank++(N), SQL-Rank++(P vs. N) and SQL-Rank++(P + N). We also propose two sampling strategies for efficient training. In the following section, we will introduce our SQL-Rank++ and the sampling strategies, and analyze how they solve these two limitations.

IV. The Proposed Approach

To learn more accurate user preferences and solve the two limitations mentioned above, we introduce some special auxiliary preference lists and then propose a novel listwise approach called SQL-Rank++. The overall optimization problem is as follows:

$$min \sum_{u=1}^{n} (\mathcal{L}_{sql} + \mathcal{L}_{aux}) + R_\theta \tag{3}$$

where $\mathcal{L}_{sql}$ represents the loss function of SQL-Rank [12], and $\mathcal{L}_{aux}$ denotes the loss function contributed by the auxiliary lists. For SQL-Rank++(P), SQL-Rank++(N), SQL-Rank++(P vs. N) and SQL-Rank++(P + N), $\mathcal{L}_{aux}$ can be formalized as $\mathcal{L}_P$, $\mathcal{L}_N$, $\mathcal{L}_{Pvs.N}$ and $\mathcal{L}_{P+N}$, respectively. Notice that $R_\theta = \frac{\alpha_u}{2} ||U_{u\cdot}||_F^2 + \frac{\alpha_v}{2} ||V_{j\cdot}||_F^2$ is an L_2 regularization term used to avoid the overfitting, where α_u and α_v are the hyper-parameters.

For each user u, we sample some lists with similar preference as Π_u to form a positive preference set $\mathcal{P}_u$, and some lists with different preference from Π_u to form a negative preference set $\mathcal{N}_u$. Each auxiliary list consists of two parts: the first part corresponds to the observed part of the original list, and the second part corresponds to the unobserved part. These two parts use different sampling processes, which will be introduced in Section IV-A. In Section IV-B, we will describe the four variants of our SQL-Rank++. We illustrate our SQL-Rank++ in Figure 1 and describe the complete algorithm in Section IV-C.

A. Sampling Strategies for Auxiliary Lists

In this section, we present two sampling strategies used in our SQL-Rank++ to construct the set of auxiliary lists, including a self-based strategy and a user similarity-based strategy. The former samples from the original list and the unobserved item set, and the latter samples from the neighbors' and the dissimilar users' lists. Each strategy has different sampling processes for the positive list set $\mathcal{P}_u$ and the negative list set $\mathcal{N}_u$. We illustrate these two sampling strategies in Figure 2.

1) Self-based sampling strategy: In this strategy, we sample the positive list set $\mathcal{P}_u$ from the user u's original list Π_u, and the negative list set $\mathcal{N}_u$ from the unobserved item set $\mathcal{I} \backslash \mathcal{I}_u$ of user u.

Specifically, for each positive list, we shuffle the observed and unobserved parts of the original list Π_u, and combine them to form a positive list. Performing this operation for s_p times, we can get the positive list set $\mathcal{P}_u$ of size s_p. For each negative list, we randomly sample L_u items from the unobserved item set $\mathcal{I} \backslash \mathcal{I}_u$ of user u to construct a negative list. Performing this operation for s_n times, we can get the negative list set $\mathcal{N}_u$ of size s_n.

The intuition of this sampling strategy is that a user's preference to the items with the same feedback may be similar, and the user's preference to the observed items may be larger than that to the unobserved ones.

2) User similarity-based sampling strategy: In this strategy, we first define an indicator for distinguishing the neighbors and the dissimilar users, and then randomly sample a set of neighbors or dissimilar users for user u. We construct the positive list set $\mathcal{P}_u$ and the negative list set $\mathcal{N}_u$ according to the original lists of these neighbors or dissimilar users. We use $\{c_{uw} = |\mathcal{I}_u \cap \mathcal{I}_w| / |\mathcal{I}_u \cup \mathcal{I}_w| \mid w = 1, 2, \ldots, n, w \neq u\}$ to denote the similarity between user u and the other users. We sort this set in descending order and then select the highest H users as the neighbors of user u, while other users as the dissimilar users. Following [33], we set $H = 50$ in the experiments.

For each positive list, we first randomly sample a neighbor w from the neighbor set of user u. Then we randomly sample $|\mathcal{I}_u|$ items from the observed part of user w's original list Π_w to form the observed part of the positive list, and $L_u - |\mathcal{I}_u|$ items from the unobserved part of Π_w to form the unobserved part. For each negative list, we first randomly sample a dissimilar user from the dissimilar user set of user u and perform the same operation as above. Notice that the size of the two parts in the auxiliary list should be the same as that in the original list. Therefore, when the length of the sampled list is not enough, repeated sampling is needed.

The intuition of this sampling strategy is that a user's preference may be similar to the preference of their neighbors, rather than the users who are dissimilar.

B. Four Variants of SQL-Rank++

Since a positive auxiliary list maintains the semantic consistency with the original list, we expect the model to make the original list and the positive auxiliary list be close as much as possible. For the negative auxiliary list and the original list, we expect the model to reduce the similarity between them due to the semantic inconsistency.

Because the preference comparison relationship for the unobserved items is unknown, it is unable to define a clear relative relationship between the unobserved part of the original list and this part of any auxiliary list. Therefore, we treat

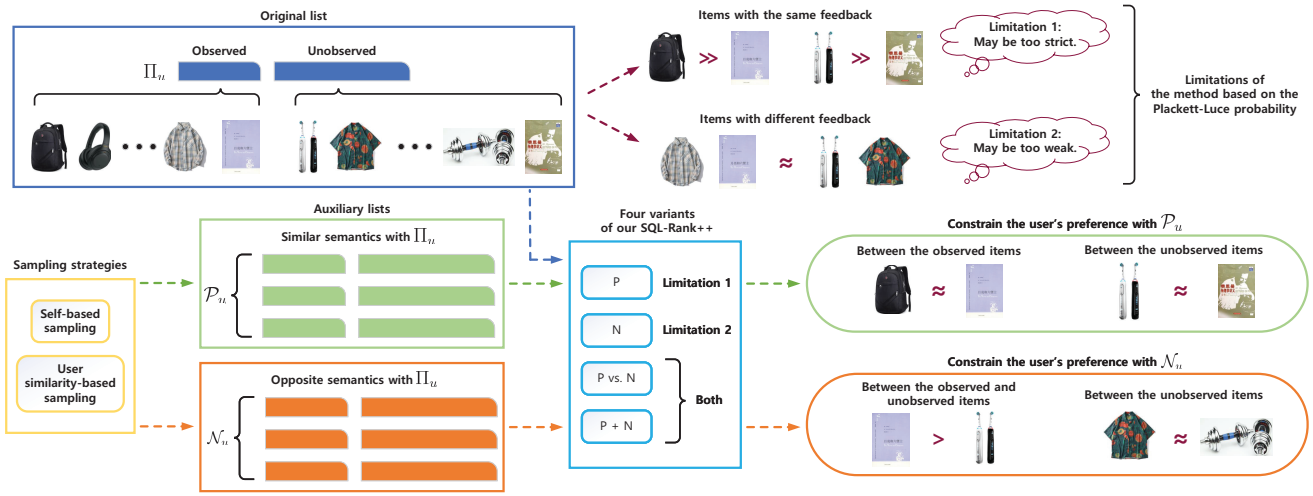


Fig. 1. Illustration of our SQL-Rank++. The approaches based on the Plackett-Luce probability, e.g., SQL-Rank (shown in the upper), have too strict constraints between the items with the same feedback (e.g., backpack vs. light blue book, and toothbrush vs. yellow book), and too weak constraints between the items with different feedback (e.g., grey shirt and light blue book vs. toothbrush and color shirt). We propose two sampling strategies to construct some positive auxiliary lists which have similar semantics with the original list and some negative auxiliary lists which are opposite (shown in the bottom left corner). We then have four variants of our SQL-Rank++ to jointly learn from the auxiliary lists and the original list to address these limitations (shown in the bottom half). Specifically, our SQL-Rank++ constrains the user's preference to the items with the same feedback to be close to each other, as well as expands the preference difference between the items with different feedback (shown in the bottom right corner).

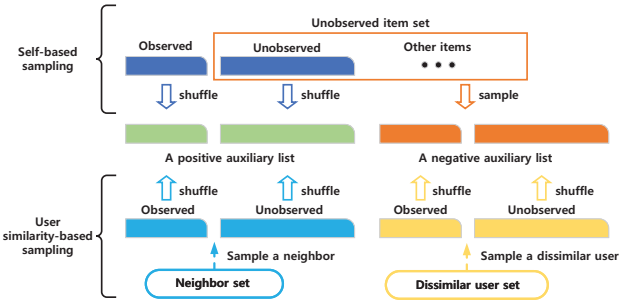


Fig. 2. Illustration of the self-based sampling strategy (shown in the upper) and the user similarity-based sampling strategy (shown in the bottom).

the unobserved part of these auxiliary lists equally, regardless of whether it comes from a positive one or a negative one.

The positive auxiliary list prompts the items with the same feedback closer, thereby weakening the too strict constraints described in **Limitation 1**. The negative auxiliary list introduces more comparisons between the items with different feedback, which in turn strengthens the too weak constraint described in **Limitation 2**. Through these, we can train a model that better captures the user's true preference and then address the two limitations faced by the listwise approaches based on the Plackett-Luce probability.

1) SQL-Rank++(P): In this variant, we use the auxiliary lists with similar preference as Π_u in the positive preference set $\mathcal{P}_u$ to solve the limitation of the too strict comparison between the items with the same feedback. As previously described, we treat the observed and unobserved parts of each list separately. The loss function of SQL-Rank++(P) is composed of these two parts: $\mathcal{L}_P = \mathcal{L}_P(ob) + \mathcal{L}_P(un)$.

Observed part. Let $\mathcal{P}_u^g$ denote the g-th list in the positive preference set $\mathcal{P}_u$. We expect that a user's preference on the observed part of the original list Π_u and the positive list $\mathcal{P}_u^g$ are close. The objective function of this strategy is:

$$\mathcal{L}_P(ob) = \frac{1}{|\mathcal{I}_u|}\sum_{j=1}^{|\mathcal{I}_u|}\frac{1}{s_p}\sum_{g=1}^{s_p} -\ln(1-\sigma(|\hat{r}_{u\Pi_{uj}} - \hat{r}_{u\mathcal{P}_{uj}^g}|)) \quad (4)$$

where $\hat{r}_{u\Pi_{uj}}$ and $\hat{r}_{u\mathcal{P}_{uj}^g}$ denote the predicted preference of user u to item Π_{uj} and item $\mathcal{P}_{uj}^g$, respectively. Notice that s_p is the size of $\mathcal{P}_u$.

Unobserved part. Since the relative relationship between the original list and a positive auxiliary list on the unobserved part is not as intuitive as that on the observed part, in order to avoid introducing unnecessary noise in the modeling process, we treat the unobserved part of these two equally, i.e., constrain a user's preference on the unobserved part to be also close. The objective function is as follows:

$$\mathcal{L}_P(un) = \frac{1}{L_u - |\mathcal{I}_u|}\sum_{j=|\mathcal{I}_u|+1}^{L_u}\frac{1}{s_p}\sum_{g=1}^{s_p} -\ln(1-\sigma(|\hat{r}_{u\Pi_{uj}} - \hat{r}_{u\mathcal{P}_{uj}^g}|)) \quad (5)$$

2) SQL-Rank++(N): In this variant, we use the auxiliary lists with different preference from Π_u in the negative preference set $\mathcal{N}_u$ to solve the limitation of the too weak preference comparison between the observed items and the unobserved items. Same as SQL-Rank++(P), we have $\mathcal{L}_N = \mathcal{L}_N(ob) + \mathcal{L}_N(un)$.

As mentioned before, the target of the unobserved part on all variants of our SQL-Rank++ is similar, the only difference of $\mathcal{L}_N(un)$ and $\mathcal{L}_P(un)$ is the set of auxiliary lists they use. Hence, we only describe the observed part of SQL-Rank++(N).

Observed part. Let $\mathcal{N}_u^g$ denotes the g-th list in the negative preference set $\mathcal{N}_u$. We make a user's preference to the j-th observed item in the original list Π_u larger than that to its corresponding item in the negative list $\mathcal{N}_u^g$ to improve the preference comparison between the observed items and the unobserved items. Furthermore, such a comparison relationship can make better use of the important unobserved items in implicit feedback. The objective function of this strategy is:

$$\mathcal{L}_N(ob) = \frac{1}{|\mathcal{I}_u|}\sum_{j=1}^{|\mathcal{I}_u|}\frac{1}{s_n}\sum_{g=1}^{s_n} -\ln(\sigma(\hat{r}_{u\Pi_{uj}} - \hat{r}_{u\mathcal{N}_{uj}^g})) \tag{6}$$

where s_n is the size of $\mathcal{N}_u$. The difference between $\mathcal{L}_P(ob)$ and $\mathcal{L}_N(ob)$ is that $\mathcal{L}_P(ob)$ needs to make the user's preference to the two corresponding items close to each other, while $\mathcal{L}_N(ob)$ needs to make the user's preference to the observed item in the original list strictly greater than that to its corresponding item in the negative list.

3) SQL-Rank++(P vs. N): When considering the combined use of the positive auxiliary lists and negative auxiliary lists, inspired by contrastive learning [27], [30], we propose a joint training form called SQL-Rank++(P vs. N). The loss function of SQL-Rank++(P vs. N) is $\mathcal{L}_{Pvs.N} = \mathcal{L}_{Pvs.N}(ob) + \mathcal{L}_{Pvs.N}(un)$. For the same reason as that in Section IV-B2, we only describe $\mathcal{L}_{Pvs.N}(ob)$.

Observed part. A contrastive learning approach usually increases the similarity between the original sample and the positive sample, and reduces the similarity between the original sample and the negative sample. Similar to the idea of contrastive learning, we propose a contrastive preference function $obj(r_1, r_2)$. Suppose we have two items p and q sampled from the observed part of the original list Π_u and an auxiliary list $\mathcal{P}_u^g$ (or $\mathcal{N}_u^g$), respectively, the contrastive preference function of $(u,\ q,\ p)$ can be written as:

$$obj(\hat{r}_{up},\ \hat{r}_{uq}) = \begin{cases} 1-\sigma(|\hat{r}_{up} - \hat{r}_{uq}|), & \text{if } q \in \mathcal{P}_u^g \\ 1-\sigma(\hat{r}_{up} - \hat{r}_{uq}), & \text{if } q \in \mathcal{N}_u^g \end{cases} \tag{7}$$

If q belongs to a positive auxiliary list $\mathcal{P}_u^g$, then the closer the preference of user u to items p and q, the greater the value of $obj(\hat{r}_{up},\ \hat{r}_{uq})$. In the other case, if q belongs to a negative auxiliary list $\mathcal{N}_u^g$, then the larger the preference of user u to item p than that to item q, the smaller the value of $obj(\hat{r}_{up},\ \hat{r}_{uq})$. According to this contrastive preference function, we have the following loss function:

$$\mathcal{L}_{Pvs.N}(ob) = \frac{1}{|\mathcal{I}_u|}\sum_{j=1}^{|\mathcal{I}_u|} -\ln\frac{\frac{1}{s_p}\sum_{g=1}^{s_p}\phi(\mathcal{P}_{uj}^g)}{\frac{1}{s_p}\sum_{g=1}^{s_p}\phi(\mathcal{P}_{uj}^g) + \sum_{g=1}^{s_n}\phi(\mathcal{N}_{uj}^g)} \tag{8}$$

where $\phi(\mathcal{P}_{uj}^g) = \exp(obj(\hat{r}_{u\Pi_{uj}},\ \hat{r}_{u\mathcal{P}_{uj}^g}))$ and $\phi(\mathcal{N}_{uj}^g) = \exp\ (obj(\hat{r}_{u\Pi_{uj}},\ \hat{r}_{u\mathcal{N}_{uj}^g}))$. Based on the descriptions in SQL-Rank++(P) and SQL-Rank++(N), optimizing Equation (8) can not only weaken the preference comparison between the observed items but also strengthen the preference comparison between the observed items and the unobserved items. Both **Limitation 1** and **Limitation 2** can thus be solved at the same time.

4) SQL-Rank++(P + N): We can also directly add the loss functions of SQL-Rank++(P) and SQL-Rank++(N) together to achieve the goal of simultaneously solving **Limitation 1** and **Limitation 2**. Then the objective function can be represented as follows:

$$\mathcal{L}_{P+N} = \mathcal{L}_P + \mathcal{L}_N \tag{9}$$

C. Algorithm

We use the stochastic gradient descent (SGD) based algorithm to solve the optimization problem, and describe the complete algorithm of SQL-Rank++ in Algorithm 1.

Algorithm 1: The algorithm of SQL-Rank++.

Input: $\mathcal{R} = \{(u, i)\}$.
Output: $U \in \mathbb{R}^{n\times d}$ and $V \in \mathbb{R}^{m\times d}$.

for $t = 1, 2, \ldots, T$ **do**
  for $u = 1, 2, \ldots, n$ **do**
    Generate a new original list Π_u by stochastic queuing process [12].
    Generate a new positive auxiliary list set $\mathcal{P}_u$ and a new negative auxiliary list set $\mathcal{N}_u$ by sampling.
  end for
  for $u = 1, 2, \ldots, n$ **do**
    Update U via the gradients of Equation (3) by stochastic gradient descent.
  end for
  for $u = 1, 2, \ldots, n$ **do**
    Update V via the gradients of Equation (3) by stochastic gradient descent.
  end for
end for

V. Empirical Evaluation

We conduct experiments to study the following three research questions (RQs). **RQ1**) Does our SQL-Rank++ achieve the state-of-the-art results? **RQ2**) How does the number of auxiliary lists affect the performance of our SQL-Rank++? **RQ3**) What is the impact of the observed and unobserved parts in our SQL-Rank++?

A. Experiment Setup

1) Datasets: We use four public datasets, i.e., MovieLens 1M[1], Netflix[2], Alibaba2015[3] and Tmall[3], to study the effectiveness of our SQL-Rank++. MovieLens 1M and Netflix include 5-star digital ratings for movies provided by users from two different movie platforms. We follow [26] and keep ratings larger than 3 as the observed feedback. Alibaba2015 contains the records of users' purchases and examinations to items on the e-commerce platform. We treat both purchase and examination records as the observed feedback. Tmall contains users' visit labels to merchants on the e-commerce platform,

[1]http://www.grouplens.org/
[2]https://www.netflix.com/
[3]https://tianchi.aliyun.com/dataset/

including repeat and non-repeat buying. We treat the label of both repeat and non-repeat buying as the observed feedback.

For MovieLens 1M and Netflix, we randomly sample half of the observed feedback as the training data, and the rest as the test data. We then randomly sample one record for each user on average from the training data as the validation data [26]. For Alibaba2015 and Tmall, we randomly sample 60% observed feedback as the training data, 20% as the test data, and the rest 20% as the validation data [34]. We repeat the above procedure three times to get three copies of each dataset. All the experiment results are averaged on these three copies. We report the statistics of all the processed datasets in Table III.

TABLE III
STATISTICS OF THE DATASETS USED IN THE EXPERIMENTS, WHERE n, m, $|\mathcal{R}|$ AND $|\mathcal{R}^{te}|$ DENOTE THE NUMBERS OF USERS, ITEMS, TRAINING RECORDS AND TEST RECORDS, RESPECTIVELY.

Dataset	n	m	$\lvert\mathcal{R}\rvert$	$\lvert\mathcal{R}^{te}\rvert$	$\lvert\mathcal{R}\rvert/(nm)$
MovieLens 1M	6,040	3,952	287,641	287,640	1.21%
Netflix	5,000	5,000	77,936	77,936	0.31%
Alibaba2015	7,475	5,257	59,417	14,854	0.15%
Tmall	10,000	10,000	100,726	25,182	0.10%

2) Evaluation Metrics: We use two ranking-oriented top-K evaluation metrics: precision and normalized discounted cumulative gain (NDCG), denoted by Prec@K and NDCG@K. Prec@K counts the proportion of items that are interacted by a user in a top-K recommendation list. NDCG@K pays attention to the positions of the items that a user interacts with in a recommendation lists. Considering that a user usually only cares about the first few items in a recommendation list in an actual interactive environment, we set $K = 5$ in the experiments.

3) Baselines: We compare our SQL-Rank++ with seven state-of-the-art baselines modeling implicit feedback, including two pointwise approaches, three pairwise approaches, and two listwise approaches.

LogMF [4] is a pointwise approach that uses a logistic loss function to increase the predicted score of an observed (user, item) pair and reduce that of an unobserved (user, item) pair to learn the latent representation of users and items.

FISMrmse [6] is a pointwise approach, which uses the items observed by a user to represent the profile of the user and RMSE as the loss function.

FISMauc [6] is a pairwise variant of FISMrmse, using AUC as the loss function.

BPR [2] is a pairwise approach, which assumes that a user's preference to an observed item should be larger than that to an unobserved item.

SetRank [24] defines the pairwise comparison relationship between an observed item and a set of sampled unobserved items.

SQL-Rank [12] is a listwise approach based on the Plackett-Luce probability. SQL-Rank learns a user's preference by constructing a list with all the observed items and some sampled unobserved items, and maximizing the Plackett-Luce probability.

CLAPF-MRR [15] integrates the listwise loss based on the mean reciprocal rank and the pairwise ranking to combine the advantages of the two.

4) Implementation Details: We perform grid search according to the performance of NDCG@5 on the first copy of validation data for each dataset, to search the best values of the hyper-parameters in all approaches. For LogMF, BPR, FISMrmse, FISMauc and CLAPF-MRR, we search the regularization weights (α_u, α_v) from $\{0.1, 0.01, 0.001\}$, and the learning rate γ from $\{0.1, 0.01, 0.001\}$ [34]. We do not add a decay rate to the above approaches [15]. For SetRank, SQL-Rank and our SQL-Rank++, we search the regularization weights (α_u, α_v) from $\{0.2, 0.6, 1.0, 1.4, 1.8\}$, and the learning rate γ from $\{0.1, 0.2, 0.3, 0.4, 0.5\}$. We fix the decay rate of the learning rate to 0.99 [24]. For CLAPF-MRR, we search the best value of the iteration number $T \in \{1000, 2000, \ldots, 100000\}$ [15], and $T \in \{10, 20, \ldots, 1000\}$ for the other approaches. For all approaches in the experiments, we fix the dimension $d = 50$.

B. Experimental Results

1) Performance Comparison (RQ1): We use S1 to represent the self-based sampling strategy, and S2 to represent the user similarity-based sampling strategy. The main experimental results on the four datasets are shown in Table IV, from which we can have the following observations.

The performance of our SQL-Rank++ with two sampling strategies and four variants (a total of eight forms) is greatly improved compared with the most closely related work SQL-Rank in all cases. This shows that our analysis of the two limitations (i.e., the preference comparison between the items with the same feedback in the list is often too strict, and the preference comparison between the items with different feedback is usually too weak) of the listwise approaches based on the Plackett-Luce probability is reasonable. Meanwhile, our solution of leveraging some auxiliary preference lists to constrain a user's preference can help address the above two limitations and train the model more appropriately.

The best-performing SQL-Rank++ variant outperforms all seven pointwise, pairwise and listwise baselines on the four datasets. This shows that the Plackett-Luce probability-based listwise approaches has great potential. It is worth mentioning that the experimental results are not completely consistent with those reported in SetRank [24] and SQL-Rank [12]. The reason is that they remove a lot of (user, item) pairs and inactive users during their data processing in order to obtain adequate positive feedback. Taking MovieLens 1M as an example, they only keep users with more than 60 ratings, and randomly sample 50 of them in training [12]. And such a dense dataset may not be common in real-world scenarios.

For the two sampling strategies, the performance of the user similarity-based sampling strategy is better than that of the self-based sampling strategy in most cases. One reason is that

TABLE IV
RECOMMENDATION PERFORMANCE OF OUR SQL-RANK++ AND SEVEN BASELINES ON FOUR DATASETS. THE BEST RESULTS ARE MARKED IN BOLD. NOTICE THAT OUR SQL-RANK++ CAN BE CONFIGURED WITH A SELF-BASED SAMPLING STRATEGY (S1) OR A USER SIMILARITY-BASED SAMPLING STRATEGY (S2), AS WELL AS ONE OF THE FOUR COMBINATIONS OF POSITIVE (P) AND NEGATIVE (N) AUXILIARY LISTS.

Models		MovieLens 1M		Netflix		Alibaba2015		Tmall	
		Prec@5	NDCG@5	Prec@5	NDCG@5	Prec@5	NDCG@5	Prec@5	NDCG@5
LogMF		0.4209±0.0022	0.4317±0.0023	0.2279±0.0020	0.2490±0.0025	0.0307±0.0004	0.0536±0.0017	0.0180±0.0008	0.0285±0.0010
FISMrmse		0.3866±0.0052	0.4039±0.0065	0.2110±0.0025	0.2276±0.0033	0.0279±0.0005	0.0478±0.0008	0.0159±0.0002	0.0241±0.0007
FISMauc		0.3847±0.0007	0.3943±0.0018	0.2152±0.0009	0.2286±0.0013	0.0233±0.0004	0.0403±0.0002	0.0150±0.0004	0.0223±0.0007
SetRank		0.3808±0.0034	0.3909 ±0.0036	0.1870±0.0020	0.1970±0.0022	0.0261±0.0006	0.0434±0.0008	0.0152±0.0001	0.0233±0.0007
BPR		0.4324±0.0005	0.4439±0.0005	0.2354±0.0027	0.2540±0.0038	0.0298±0.0010	0.0515±0.0015	0.0172±0.0006	0.0265±0.0005
SQL-Rank		0.4416±0.0022	0.4590±0.0022	0.2216±0.0042	0.2335±0.0049	0.0248±0.0006	0.0420±0.0020	0.0154±0.0002	0.0236±0.0008
CLAPF-MRR		0.4329±0.0018	0.4447±0.0030	0.2379±0.0011	0.2570±0.0019	0.0330±0.0008	0.0571±0.0005	0.0180±0.0010	0.0279±0.0015
SQL-Rank++(S1)	P	0.4473±0.0032	0.4618±0.0030	0.2439±0.0031	0.2637±0.0040	0.0315±0.0002	0.0533±0.0011	0.0176±0.0003	0.0275±0.0005
	N	0.4551±0.0011	0.4737±0.0013	0.2273±0.0037	0.2467±0.0055	0.0313±0.0004	0.0532±0.0008	0.0173±0.0003	0.0266±0.0008
	P vs. N	0.4565±0.0032	0.4736±0.0034	0.2413±0.0027	0.2621±0.0041	0.0335±0.0010	0.0568±0.0008	0.0184±0.0003	0.0285±0.0008
	P + N	0.4605±0.0022	0.4768±0.0023	0.2471±0.0033	0.2671±0.0036	0.0335±0.0008	0.0571±0.0006	0.0188±0.0006	0.0293±0.0011
SQL-Rank++(S2)	P	0.4610±0.0016	0.4769±0.0015	0.2473±0.0030	0.2676±0.0038	0.0315±0.0003	0.0537±0.0006	0.0181±0.0003	0.0280±0.0004
	N	**0.4666**±0.0024	0.4840±0.0026	0.2480±0.0015	0.2685±0.0019	0.0338±0.0010	0.0573±0.0009	0.0180±0.0004	0.0279±0.0003
	P vs. N	0.4658±0.0035	**0.4841**±0.0033	0.2450±0.0023	0.2661±0.0031	0.0339±0.0010	0.0578±0.0008	0.0191±0.0003	0.0293±0.0004
	P + N	0.4611±0.0033	0.4761±0.0028	**0.2550**±0.0026	**0.2760**±0.0039	**0.0346**±0.0010	**0.0593**±0.0010	**0.0192**±0.0005	**0.0300**±0.0006

the auxiliary list constructed based on the neighbors or the dissimilar users can introduce more comparison relationships, so that the model can get more useful information. In particular, the self-based sampling strategy has a fixed number of candidates in the observed part, while the user similarity-based sampling strategy has a larger coverage.

The relative performance comparison between different variants of our SQL-Rank++ is different on different datasets. In most cases, the approaches that co-constraining the positive and negative auxiliary lists (i.e., SQL-Rank++(P vs. N) and SQL-Rank++(P + N)) are better than the approaches that only constraining a certain auxiliary list (i.e., SQL-Rank++(P) and SQL-Rank++(N)). These results can be observed in using both sampling strategies on Alibaba2015 and Tmall, and the self-based sampling strategy on MovieLens 1M and Netflix. For the approaches with the user similarity-based sampling strategy on MovieLens 1M and Netflix, the performance of SQL-Rank++(P vs. N) is similar or even worse than that of SQL-Rank++(P) or SQL-Rank++(N). Moreover, although SQL-Rank++(P + N) is the best performing model in most cases, it is worse than SQL-Rank++(P vs. N) and SQL-Rank++(N) on MovieLens 1M with the user similarity-based sampling strategy. One reason is that MovieLens 1M contains a smaller number of items and Netflix contains a smaller number of users, and there is a larger probability of introducing noise when using the auxiliary lists owned by the neighboring and dissimilar users.

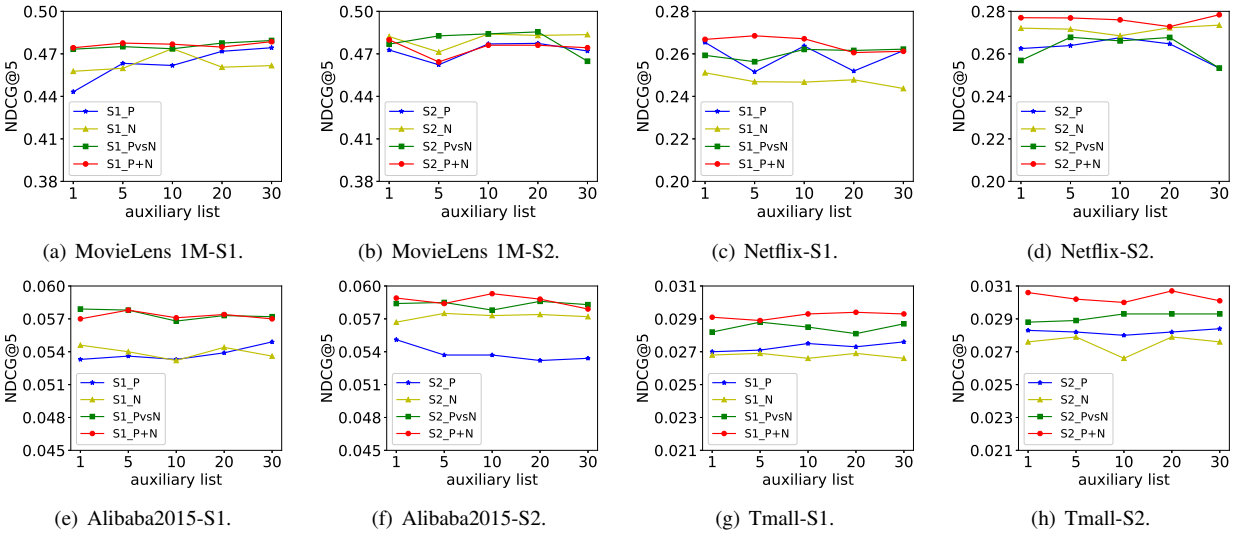
2) Quantitative Study (RQ2): It can be seen from Equation (4) that the number of auxiliary lists is related to the complexity of our SQL-Rank++. We study the influence of the number of auxiliary lists, and report the results in Figure 3.

From Figure 3, we can see that the performance of our SQL-Rank++ is similar when using different numbers of auxiliary lists. As the number of auxiliary lists increases, the performance of the different variants of our SQL-Rank++ fluctuates slightly within a certain range. The reason for this situation is that we re-construct the auxiliary list in each iteration, which can generate enough comparison relationships to help the training of the model when using a few auxiliary lists. Therefore, only a small number of auxiliary lists (or even just one) are needed to achieve the desired performance, too many auxiliary lists may introduce noise and increase the difficulty of training.

The time complexity of SQL-Rank is $O(nL_u d)$ [12], and that of our SQL-Rank++ is $O(nL_u d(1+g))$ because it needs to calculate g additional auxiliary lists on the basis of SQL-Rank. However, the number g is often a small constant, which makes our SQL-Rank++ have the same near-linear time complexity $O(nL_u d)$ as SQL-Rank.

3) Ablation Study (RQ3): To figure out the contribution of L_{ob} and L_{un} to the performance of our SQL-Rank++, we conduct an ablation study as shown in Figure 4. Taking SQL-Rank++(P) as an example, we compare the performance of $\min \sum_{u=1}^{n}(\mathcal{L}_{sql} + \mathcal{L}_P(ob)) + R_\theta$ (denoted as "observed" in Figure 4) , $\min \sum_{u=1}^{n}(\mathcal{L}_{sql} + \mathcal{L}_P(un)) + R_\theta$ (denoted as "unobserved" in Figure 4) and the complete SQL-Rank++(P) (denoted as "both" in Figure 4) on four datasets. Notice that "observed" aims to solve the too strict constraints between the observed items or the too weak constraints between the observed items and the unobserved items. The purpose of "unobserved" is only to solve the too strict constraints between the unobserved items (i.e., we treat all unobserved items equally). And "both" combines "observed" and "unobserved" together.

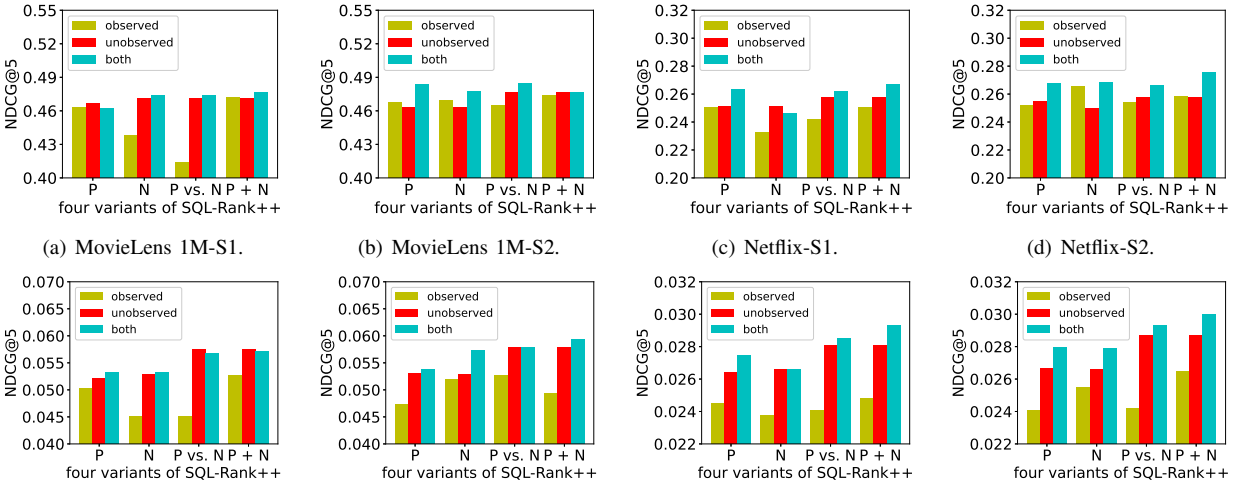
For "observed" vs. "unobserved", the experimental results show that a user's preference to the unobserved items needs to be more constrained than that to the observed ones. In Figure 4, the NDCG@5 values of "unobserved" are comparable



(a) MovieLens 1M-S1. (b) MovieLens 1M-S2. (c) Netflix-S1. (d) Netflix-S2.

(e) Alibaba2015-S1. (f) Alibaba2015-S2. (g) Tmall-S1. (h) Tmall-S2.

Fig. 3. Recommendation performance (NDCG@5) of our SQL-Rank++ with different numbers of auxiliary lists on four datasets.



(a) MovieLens 1M-S1. (b) MovieLens 1M-S2. (c) Netflix-S1. (d) Netflix-S2.

(e) Alibaba2015-S1. (f) Alibaba2015-S2. (g) Tmall-S1. (h) Tmall-S2.

Fig. 4. Recommendation performance (NDCG@5) of our SQL-Rank++ with different architectures (i.e., ablation studies) on four datasets.

to or larger than that of "observed" in most cases. This means that making a user's preference to the unobserved items close to each other is important for learning the true preference of the user [17]. Mentions that the unobserved part of the original list and the auxiliary lists can create more comparison relationships than the observed part because of the large number of the unobserved data, which can bring more useful information to the training. In addition, the stochastic queuing process of SQL-Rank has helped smooth the constraints between the observed items. Therefore, the improvement of "unobserved" is more significant than that of "observed". We also find that the user similarity-based sampling strategy has the potential to capture the users' true preferences from the neighbors' and the dissimilar users' observed items. This feature makes it possible for "observed" to surpasses "unobserved" (e.g., SQL-Rank++(P) and SQL-Rank++(N) in Figure 4(b), and SQL-Rank++(N) in Figure 4(d)).

For "both" vs. "observed" and "unobserved", we can observe two different results depending on the sampling strategy in Figure 4. For the self-based sampling strategy, in most cases, constraining the observed and unobserved items at the same time is better than constraining one of them separately. However, the performance of "unobserved" is better than that of "both" on some variants (e.g., SQL-Rank++(P) in Figure 4(a), SQL-Rank++(N) in Figure 4(c), and SQL-Rank++(P vs. N) and SQL-Rank++(P + N) in Figure 4(e)). Since the self-based sampling strategy can only generate a fixed number of candidates when constructing the auxiliary lists, constraining the observed and unobserved items at the same time may add more penalty on some unobserved items. Compared with constraining observed items or unobserved items separately, this may result in the loss of some potential preferred items. For

the user similarity-based sampling strategy, the performance of "both" is comparable to or better than that of "observed" and "unobserved" in all cases. This means that constraining the observed and unobserved items at the same time ("both") can achieve better performance when the comparison relationships are adequate.

VI. Conclusions and Future Work

In this paper, we propose a novel listwise recommendation approach, i.e., SQL-Rank++, for modeling users' implicit feedback. Our SQL-Rank++ addresses two fundamental limitations in the approaches based on the Plackett-Luce probability, i.e., too strict and too weak constraints between the items with the same feedback and different feedback, respectively. Specifically, we design two sampling strategies to construct some special auxiliary lists, including some positive lists and some negative lists, to assist learning of the user preferences from the original list, and finally obtain four variants of our SQL-Rank++ with different combinations of the positive and negative lists. Extensive empirical studies on four public datasets show the effectiveness of our SQL-Rank++ over several competitive pointwise, pairwise and listwise approaches.

For future works, we are interested in generalizing our SQL-Rank++ from a shallow model to a deep one in order to capture more complex relations among the users and items. We are also interested in exploiting more types of implicit feedback such as adds-to-cart and purchases [29] and pre-training user representations [35] in the proposed listwise learning paradigm so as to learn the user preferences more accurately.

Acknowledgment

We thank the support of National Natural Science Foundation of China Nos. 62172283 and 61836005.

References

[1] Y. Koren, R. Bell, and C. Volinsky, "Matrix factorization techniques for recommender systems," *Computer*, 2009, 42, 8, pp. 30-37.

[2] S. Rendle, C. Freudenthaler, Z. Gantner, and L. Schmidt-Thieme, "BPR: Bayesian personalized ranking from implicit feedback," *UAI'09*, 2009, pp. 452-461.

[3] R. Pan, Y. Zhou, B. Cao, N. Liu, R. Lukose, M. Scholz, and Q. Yang, "One-class collaborative filtering," *ICDM'08*, 2008, pp. 502-511.

[4] C. C. Johnson, "Logistic matrix factorization for implicit feedback data," *NeurIPS'14*, 2014, 78, pp. 1-9.

[5] D. Liang, L. Charlin, J. McInerney, and D. M. Blei, "Modeling user exposure in recommendation," *WWW'16*, 2016, pp. 951-961.

[6] S. Kabbur, X. Ning, and G. Karypis, "FISM: Factored item similarity models for top-N recommender systems," *KDD'13*, 2013, pp. 659-667.

[7] Z. Mai, G. Wu, K. Luo, and S. Sanner, "Attentive autoencoders for multifaceted preference learning in one-class collaborative filtering," *ICDMW'20*, 2020, pp. 165-172.

[8] R. He, W. Kang, and J. McAuley, "Translation-based recommendation," *RecSys'17*, 2017, pp. 161-169.

[9] Y. Shi, M. Larson, and A. Hanjalic, "List-wise learning to rank with matrix factorization for collaborative filtering," *RecSys'10*, 2010, pp. 269-272.

[10] J. Liu, C. Wu, Y. Xiong, and W. Liu, "List-wise probabilistic matrix factorization for recommendation," *Information Sciences*, 2014, 278, pp. 434-447.

[11] S. Huang, S. Wang, T. Liu, J. Ma, Z. Chen, and J. Veijalainen, "Listwise collaborative filtering," *SIGIR'15*, 2015, pp. 343-352.

[12] L. Wu, C. Hsieh, and J. Sharpnack, "SQL-Rank: A listwise approach to collaborative ranking," *ICML'18*, 2018, pp. 5315-5324.

[13] A. Mnih, and R. R. Salakhutdinov, "CoFiRank - Maximum margin matrix factorization for collaborative ranking," *NeurIPS'07*, 2007, pp. 222-230.

[14] Y. Shi, A. Karatzoglou, L. Baltrunas, M. Larson, N. Oliver, and A. Hanjalic, "CLiMF: Learning to maximize reciprocal rank with collaborative less-is-more filtering," *RecSys'12*, 2012, pp. 139-146.

[15] R. Yu, Q. Liu, Y. Ye, M. Cheng, E. Chen, and J. Ma, "Collaborative list-and-pairwise filtering from implicit feedback," *IEEE Transactions on Knowledge and Data Engineering*, 2022, 34, 6, pp. 2667-2680.

[16] X. Sun, H. Liu, L. Jing, and J. Yu, "Deep generative recommendation with maximizing reciprocal rank," *KSEM'20*, 2020, pp. 123-130.

[17] J. Chen, D. Lian, and K. Zheng, "Improving one-class collaborative filtering via ranking-based implicit regularizer," *AAAI'19*, 2019, 01, pp. 37-44.

[18] C. Chen, C. Wang, M. Tsai, and Y. Yang, "Collaborative similarity embedding for recommender systems," *WWW'19*, 2019, pp. 2637-2643.

[19] A. Rawat, A. Menon, A. Veit, F. Yu, S. J. Reddi, and S. Kumar, "Doubly-stochastic mining for heterogeneous retrieval," *arXiv preprint arXiv:2004.10915*, 2020.

[20] J. Ma, X. Yi, W. Tang, Z. Zhao, L. Hong, E. Chi, and Q. Mei, "Learning-to-rank with partitioned preference: Fast estimation for the Plackett-Luce model," *AISTATS'21*, 2021, pp. 928-936.

[21] X. He, J. Tang, X. Du, R. Hong, T. Ren, and T. Chua, "Fast matrix factorization with nonuniform weights on missing data," *IEEE Transactions on Neural Networks and Learning Systems*, 2019, 31, 8, pp. 2791-2804.

[22] J. Chen, C. Wang, S. Zhou, Q. Shi, J. Chen, Y. Feng, and C. Chen, "Fast adaptively weighted matrix factorization for recommendation with implicit feedback," *AAAI'20*, 2020, 04, pp. 3470-3477.

[23] X. He, Z. He, J. Song, Z. Liu, Y. Jiang, and T. Chua, "NAIS: Neural attentive item similarity model for recommendation," *IEEE Transactions on Knowledge and Data Engineering*, 2018, 30, 12, pp. 2354-2366.

[24] C. Wang, H. Zhu, C. Zhu, C. Qin, and H. Xiong, "SetRank: A setwise Bayesian approach for collaborative ranking from implicit feedback," *AAAI'20*, 2020, 04, pp. 6127-6136.

[25] L. Chen, L. Wu, K. Zhang, R. Hong, and M. Wang, "Set2setRank: Collaborative set to set ranking for implicit feedback based recommendation," *SIGIR'21*, 2021, pp. 585¨C594.

[26] W. Pan, L. Chen, and Z. Ming, "Personalized recommendation with implicit feedback via learning pairwise preferences over item-sets," *Knowledge and Information Systems*, 2019, 2, pp. 295-318.

[27] Y. Qin, P. Wang, and C. Li, "The world is binary: Contrastive learning for denoising next basket recommendation," *SIGIR'21*, 2021, pp. 859-868.

[28] X. He, K. Deng, X. Wang, Y. Li, Y. Zhang, and M. Wang, "LightGCN: Simplifying and powering graph convolution network for recommendation," *SIGIR'20*, 2020, pp. 639-648.

[29] B. Jin, C. Gao, X. He, D. Jin, and Y. Li, "Multi-behavior recommendation with graph convolutional networks," *SIGIR'20*, 2020, pp. 659-668.

[30] Z. Xie, C. Liu, Y. Zhang, H. Lu, D. Wang, and Y. Ding, "Adversarial and contrastive variational autoencoder for sequential recommendation," *WWW'21*, 2021, pp. 449-459.

[31] D. Liang, R. G. Krishnan, M. D. Hoffman, and T. Jebara, "Variational autoencoders for collaborative filtering," *WWW'18*, 2018, pp. 689-698.

[32] Z. Cao, T. Qin, T. Liu, M. Tsai, and H. Li, "A super-peer model for resource discovery services in large-scale grids," *ICML'07*, 2007, pp. 129-136.

[33] T. Bai, J. Wen, J. Zhang, and W. Zhao, "A neural collaborative filtering model with interaction-based neighborhood," *CIKM'17*, 2017, pp. 1979-1982.

[34] W. Pan, M. Liu, and Z. Ming, "Transfer learning for heterogeneous one-class collaborative filtering," *IEEE Intelligent Systems*, 2016, 4, pp. 43-49.

[35] Z. Qiu, X. Wu, J. Gao, and W. Fan, "U-BERT: Pre-training User Representations for Improved Recommendation," *AAAI'21*, 2021, pp. 4320-4327.