Ranking-based Implicit Regularization for One-Class Collaborative Filtering

Defu Lian, Jin Chen, Kai Zheng, Enhong Chen, *Senior Member, IEEE,* and Xiaofang Zhou, *Fellow, IEEE*

Abstract—One-class collaborative filtering (OCCF) problems are ubiquitous in real-world recommendation systems, such as news recommendation, but suffer from data sparsity and lack of negative items. To address the challenge, the state-of-the-art algorithm assigns uninteracted items with smaller weights of being negative and performs low-rank approximation over the user-item interaction matrix. However, the prior ratings are usually suggested to be zero but may not be well-defined. To avert the direct utilization of prior ratings for uninteracted items, we propose a novel ranking-based implicit regularizer by hypothesizing that users' preference scores for uninteracted items should not deviate a lot from each other. The regularizer is then used in a ranking-based OCCF framework to penalize large differences of preference scores between uninteracted items. To efficiently optimize model parameters in this framework, we develop the scalable alternating least square algorithm and coordinate descent algorithm, whose time complexity is linearly proportional to the data size. Finally, we extensively evaluate the proposed algorithms on six public real-world datasets. The results show that the proposed regularizer significantly improves the recommendation quality of ranking-based OCCF algorithms, such as BPRMF and RankALS. Moreover, the ranking-based framework with the proposed regularizer outperforms the state-of-the-art recommendation algorithms for implicit feedback.

Index Terms—One-class collaborative filtering, implicit regularization, item recommendation, coordinate descent, alternating least square.

1 Introduction

Recommender systems have become one of the most important ways to satisfy user interests and to increase the revenue of service providers. In recent years, plenty of research has concentrated on the recommendation for implicit feedback, since this data is more prevalent but full of noise. Implicit feedback is recorded whenever users interact with information systems. Recommendation for implicit feedback, also referred to as One-Class Collaborative Filtering (OCCF), aims to infer user interest from abundant yet noisy data and to provide each user a ranking list of items by matching user interest with items' latent property.

However, OCCF is very challenging, since implicit feedback is usually very sparse and only includes users' interacted items. Truly negative and potentially positive items are mixed together in uninteracted items [1], [2], [3]. Modeling uninteracted items plays a very important role in developing effective OCCF algorithms. Existing works treat interacted items as positive and uninteracted items as negative. These works can be categorized into two groups. The first regression-based methods optimize a pointwise weighted square loss, which sets the prior zero ratings to uninteracted items and assigns lower confidence to uninteracted items than interacted items [1], [2], [4], [5]. These works differ from each other in how many uninteracted items are considered for optimization. For example, all uninteracted items are considered as negative in [1], [4] and partial samples from them are considered as negative in [2]. Due to the same confidence of uninteracted items being negative, the learning of parameters can be very efficient even when all uninteracted items are considered. The empirical comparison in [5] reveals that leveraging more uninteracted items leads to a higher quality of recommendation. According to the analysis in [6], [7], the explicit prior ratings suggest that prediction scores for uninteracted items should not deviate far from zero values. This constraint is formalized as an implicit regularizer, penalizing non-zero prediction scores for uninteracted items. However, the same and zero-valued prior ratings for all users may be questionable since the prior ratings may vary from user to user, but no other choices of prior ratings can be shown to work better.

Alternative ranking-based methods optimize a squared or non-linear pairwise loss, which also treats uninteracted items as negative. Instead of setting prior ratings to them, ranking-based methods assume that interacted items should be ranked higher than uninteracted items [3], [8], [9], [10]. In other words, the ranking-based methods avert the use of prior ratings. However, only when the square loss function is used, all uninteracted items can be efficiently incorporated [8]. When interacted items are ranked higher than uninteracted ones, we expect the ideal loss should be small, but the square loss may be still large. Therefore, they can not completely guarantee that interacted items are ranked higher than uninteracted items with respect to each user. When the non-linear loss functions are applied, we have to sample some items as negative from a large pool of uninteracted items, to achieve efficient optimization [3], [10]. Due to an extremely large number of candidate items,

- *D. Lian and E. Chen are with the University of Science and Technology of China, Hefei, Anhui 230000, China. E-mail: {liandefu,cheneh}@ustc.edu.cn*
- *J. Chen and K. Zheng are with the School of Computer Science and Engineering, University of Electronic Science and Technology of China. E-mail: chenjin.uestc@gmail.com, zhengkai@uestc.edu.cn.*
- *X. Zhou is with the Department of Computer Science and Engineering, Hong Kong University of Science and Technology. E-mail: zxf@cse.ust.hk.*
- *Corresponding authors: Kai Zheng and Enhong Chen.*

Manuscript received April 19, 2005; revised August 26, 2015.

only a small set of uninteracted items are used as negative for parameter learning. It is obvious that they also cannot guarantee the ranking between interacted items and other uninteracted items for each user.

To this end, in this paper, we propose a novel ranking-based implicit regularizer for ranking-based OCCF, by hypothesizing that prediction scores of uninteracted items for each user should not deviate a lot from each other [11]. The ranking-based implicit regularizer is formulated with squared differences of prediction scores between any two uninteracted items for each user. In this way, the proposed regularizer can implicitly determine personalized prior ratings for each user in an automatic way. Constrained by the regularizer, the ranking-based OCCF loss functions can guarantee that more uninteracted items are ranked lower than interacted items with respect to each user. This can satisfy the ranking-based requirement better and improve recommendation quality.

However, optimizing the ranking-based loss functions with the proposed regularizer is very challenging. For each user, every pair between uninteracted items is required to be considered. Since there are only a few interacted items, the total number of pairs between uninteracted items is extremely large. Therefore, it is time-consuming to directly calculate the regularization and its gradient with respect to latent factors. Without similar consideration to [7], gradient-based optimization approaches, which update parameters for users and items simultaneously, would be of low efficiency. To separate their update for simplification and improving efficiency, we develop alternating least square and coordinate descent algorithms to optimize pairwise loss functions with the proposed regularizer. For the squared pairwise loss function, parameters are updated in an exact closed-form. For the logit pairwise loss function, we alternate between deriving a local variational bound of the logit loss [12] and updating parameters in an approximately closed-form. Based on a novel caching strategy, the time complexity in each iteration of optimizing both loss functions is only linearly proportional to the data size.

To summarize, our main contributions are three-fold:

- We propose a novel ranking-based implicit regularizer and formulate it with squared differences of prediction scores between any two uninteracted items for each user. This regularizer plays an important role in better restricting ranking-based OCCF algorithms, such that more uninteracted items are ranked lower than interacted items with respect to each user. We apply the proposed regularizer for regularizing several ranking-based loss functions, and show that the regularizer significantly improves the quality of recommendation.
- We develop alternating least square and coordinate descent algorithms based on a strategy of first caching and continually updating for efficient optimization. The loss function can be either squared pairwise loss function or logit pairwise loss function. The time complexity of both algorithms is only in linear proportion to data size.
- We extensively evaluate the proposed algorithms on six real-world datasets, and show that the proposed algorithms outperform the competing baselines, including RankALS, BPR, WRMF and SQL_Rank, revealing the effectiveness of the proposed regularizer.

2 RELATED WORK

There are two main tasks in recommender systems, rating prediction and item recommendation. Rating prediction is investigated with explicit feedback datasets, where users rate some items with a specified set of ratings, while item recommendation is usually studied with implicit feedback datasets, where each user interacts with some items.

Rating prediction aims to predict missing ratings in a user-item rating matrix, based on intuitions that like-minded users share many rating patterns and that a user is likely to rate similar items to her rated ones. The kNN methods explicitly predict missing ratings with user-user similarity or item-item similarity [13], and have been improved by removing global effects from the data and globally learning interpolation weights for all nearest neighbors [14], [15]. The matrix factorization methods assume the user-item rating matrix is of low-rank, and directly apply SVD, stochastic gradient descent and alternating least square for learning user and item latent factors [16]. These methods may differ in whether incorporating missing values into loss functions. The matrix factorization methods were extended to model in a Bayesian way [17], such that hyperparameters are automatically learned from rating data. Following that, implicit feedback and side information were incorporated for promoting expressiveness of user latent factors as well as item latent factors [18], [19], [20]. However, these learning-based methods follow a Missing At Random (MAR) assumption, indicating the probability that a rating is missing does not depend on the value of that rating, or the value of any other missing ratings [21]. However, this assumption is usually violated in recommender systems, and the missing ratings are Missing At Not Random (MANR). Therefore, these learning-based methods were improved with error-imputation-based methods [22], [23], an inverse-propensity-scoring based method [24] or a doubly robust method of considering both of them [25].

Item recommendation aims to provide a ranking list of items for each user by learning from implicit feedback datasets. Since no ratings are in implicit feedback, we do not encounter the rating prediction problem, and the MANR assumption is not applicable. However, item recommendation faces the lack of negative items, since only each user's interacted items are provided. The whole community almost agreed with treating interacted items as positive, but disputed much about how to use uninteracted items. As discussed in the introduction, uninteracted items are considered as negative with a lower confidence than interacted items as positive [1], [2], [4], [6], [26], [27], [28], or considered to be ranked in lower positions than interacted items [3], [8], [10], [29]. The ranking-based methods averted the usage of prior ratings for uninteracted items, but how to effectively exploit uninteracted items is very challenging. Though square loss can yield a closed-form updating equation in the alternating least square method when all uninteracted items are considered [8], accurately ranking pairs between interacted items and uninteracted ones may have large loss values. Though logit loss or hinge loss is almost zero for accurately ranking pairs, a small number of items should be sampled from uninteracted items for the sake of efficiency [3], [10]. The sampler should be efficient

for sampling and effective to pick informative samples. The common samplers include the uniform based sampler [3], the popularity based sampler [30], rejection sampling with the uniform proposal [10], dynamic negative sampler [31] and decomposable sampler [32], [33]. In this paper, we do not focus on negative sampling, but on a ranking-based implicit regularizer, which penalizes large differences of prediction scores between any two uninteracted items, to better distinguish interacted items from uninteracted items.

3 Ranking-based Implicit Regularizer

3.1 Implicit Regularizer

In this paper, we investigate item recommendation for implicit feedback dataset. Assume user-item interaction matrix $\boldsymbol{R} \in \{0,1\}^{M \times N}$ available, where M denotes the number of users and N denotes the number of items. Each element r_{ui} indicates whether the user u interacts with the item i. $U = \{u_1, u_2, ..., u_M\}$ denotes all users and $E = \{e_1, e_2, ..., e_N\}$ denote all items. $E_u = \{i \in E | r_{ui} > 0\}$ denotes the items which the user u interacts with and $U_i = \{u \in U | r_{ui} > 0\}$ denotes the users who have interacted with the item i.

The SOTA regression-based methods treat all uninteracted items as negative, but assign them a lower confidence being negative [1], [4], [6]. In particular, they optimize the objective function as follows:

$$\sum_{u,i} w_{ui}(r_{ui} - \boldsymbol{p}_u^T \boldsymbol{q}_i)^2 + \lambda(\|\boldsymbol{P}\|_F^2 + \|\boldsymbol{Q}\|_F^2), \tag{1}$$

where $\boldsymbol{p}_u$ is the latent factor of user u, and $\boldsymbol{q}_i$ is the latent factor of item i. $\boldsymbol{P}$ and $\boldsymbol{Q}$ respectively stack $\boldsymbol{p}_u$ and $\boldsymbol{q}_i$ by row. w_{ui} equals α when user u interacts with item i and 1 otherwise. α is generally greater than 1.

According to [6], [34], Eq (1) is decomposed into a prediction error term and an implicit regularizer term, which is defined as:

$$\sum_u \sum_{i \notin E_u} (\boldsymbol{p}_u^T \boldsymbol{q}_i - 0)^2. \tag{2}$$

The recommendation for implicit feedback is interpreted as explicit problems with the additional implicit regularizer [6], which penalizes non-zero prediction scores of uninteracted items for each user, and plays an important role in improving deep matching network for recommendation [7].

3.2 Ranking-based Implicit Regularizer

It may be questionable to set the prior ratings of all uninteracted items for all users to zero, since prior ratings for uninteracted items should be as least different from user to user. However, it is difficult to set appropriate prior ratings for uninteracted items. Observing the implicit regularizer actually implies some "similarity" among uninteracted items (preference scores approaching the prior rating), we propose a *ranking-based implicit regularizer*, by hypothesizing each user's preference scores for uninteracted items are close to each other. Formally, the ranking-based implicit regularizer is defined as follows:

$$\begin{aligned} \Omega &= \frac{1}{2} \sum_u \sum_{i<j \notin E_u} (\boldsymbol{p}_u^T \boldsymbol{q}_i - \boldsymbol{p}_u^T \boldsymbol{q}_j)^2 \\ &= \frac{1}{4} \sum_u \sum_{i,j \notin E_u} (\boldsymbol{p}_u^T \boldsymbol{q}_i - \boldsymbol{p}_u^T \boldsymbol{q}_j)^2. \end{aligned} \tag{3}$$

The expression can be written in a symmetric way due to no contribution of the case $i = j$ and symmetry between $i < j$ and $i > j$. Compared with Eq (2), the ranking-based regularizer eliminates the usage of prior ratings for uninteracted items through difference of preference scores between any two uninteracted items. When Ω is decreased, preference scores of uninteracted items are closer to each other. Therefore, the ranking-based implicit regularizer penalizes large differences of preference scores between any two of uninteracted items, playing a similar role to the implicit regularizer.

However, directly computing the regularizer and deriving its gradient is very time-consuming, since the regularizer involves $\mathcal{O}(MN^2)$ differences of preference scores. Below we investigate how to reduce a great number of double computations. Before delving into details, we first rewrite the ranking-based implicit regularizer as follows:

$$\Omega = \frac{1}{4}(\Omega_0 - 2\Omega_1 + \Omega_2),$$

where

$$\begin{aligned} \Omega_0 &= \sum_u \sum_{i,j} (\boldsymbol{p}_u^T \boldsymbol{q}_i - \boldsymbol{p}_u^T \boldsymbol{q}_j)^2 \\ \Omega_1 &= \sum_u \sum_{i \in E_u} \sum_j (\boldsymbol{p}_u^T \boldsymbol{q}_i - \boldsymbol{p}_u^T \boldsymbol{q}_j)^2 \\ \Omega_2 &= \sum_u \sum_{i,j \in E_u} (\boldsymbol{p}_u^T \boldsymbol{q}_i - \boldsymbol{p}_u^T \boldsymbol{q}_j)^2 \end{aligned}$$

The regularizer is then decomposed into summation over a full set of items and summation over each user's interacted items. We will follow such a decomposition to derive gradient and derivative of the regularizer with respect to parameters. The gradient is used for alternating least square while the derivative is used for coordinate descent.

3.3 Regularizer Gradient

We take each user's latent factor or each item's latent factor as a whole and derive the gradient of the regularizer with respect to user latent factors and item latent factors. Firstly, we consider the gradient of Ω_1 with respect to $\boldsymbol{p}_u$, latent factor of user u, which is derived as follows:

$$\begin{aligned} \frac{\partial \Omega_1}{\partial \boldsymbol{p}_u} =& 2 \sum_{i \in E_u} \sum_j (\boldsymbol{q}_i - \boldsymbol{q}_j)(\boldsymbol{p}_u^T \boldsymbol{q}_i - \boldsymbol{p}_u^T \boldsymbol{q}_j) \\ =& 2 \left(N \boldsymbol{Q}_u^T \boldsymbol{Q}_u + N_u \boldsymbol{Q}^T \boldsymbol{Q} - \tilde{\boldsymbol{q}} \tilde{\boldsymbol{q}}_u^T - \tilde{\boldsymbol{q}}_u \tilde{\boldsymbol{q}}^T \right) \boldsymbol{p}_u \end{aligned}$$

where $\tilde{\boldsymbol{q}} = \sum_j \boldsymbol{q}_j$ and $\tilde{\boldsymbol{q}}_u = \sum_{i \in E_u} \boldsymbol{q}_i$. $\boldsymbol{Q}_u \in \mathbb{R}^{N_u \times K}$, denoting $\boldsymbol{Q}_{[E_u]}$, is a submatrix of $\boldsymbol{Q}$ extracted by E_u. $N_u = |E_u|$ denotes the number of interacted items of user u and $C_u = |N - N_u|$ denotes the number of uninteracted items. Deriving gradient of other two parts with respect to $\boldsymbol{p}_u$ is similar. Aggregating them together, we obtain gradient of the ranking-based implicit regularizer with respect to $\boldsymbol{p}_u$:

$$\begin{aligned} \frac{\partial \Omega}{\partial \boldsymbol{p}_u} &= C_u \left(\boldsymbol{Q}^T \boldsymbol{Q} - \boldsymbol{Q}_u^T \boldsymbol{Q}_u \right) \boldsymbol{p}_u - (\tilde{\boldsymbol{q}} - \tilde{\boldsymbol{q}}_u)(\tilde{\boldsymbol{q}} - \tilde{\boldsymbol{q}}_u)^T \boldsymbol{p}_u \\ &= C_u (\boldsymbol{S} - \boldsymbol{S}^u) \boldsymbol{p}_u - (\tilde{\boldsymbol{q}} - \tilde{\boldsymbol{q}}_u)(\tilde{\boldsymbol{q}} - \tilde{\boldsymbol{q}}_u)^T \boldsymbol{p}_u \end{aligned} \tag{4}$$

where $\boldsymbol{S} = \boldsymbol{Q}^T \boldsymbol{Q}$ and $\boldsymbol{S}^u = \boldsymbol{Q}_u^T \boldsymbol{Q}_u$. Due to independence of user latent vectors, the matrix $\boldsymbol{S}$ and the vector $\tilde{\boldsymbol{q}}$ can be computed prior to updating user latent factors.

The gradient of the ranking-based implicit regularizer with respect to item latent factor is as follows:

$$\begin{aligned}\frac{\partial\Omega}{\partial \boldsymbol{q}_l} =& \sum_u C_u \boldsymbol{p}_u \boldsymbol{p}_u^T \boldsymbol{q}_l - \sum_u \boldsymbol{p}_u \boldsymbol{p}_u^T (\tilde{\boldsymbol{q}} - \tilde{\boldsymbol{q}}_u) \\ &- \sum_{u\in U_l} C_u \boldsymbol{p}_u \boldsymbol{p}_u^T \boldsymbol{q}_l + \sum_{u\in U_l} \boldsymbol{p}_u \boldsymbol{p}_u^T (\tilde{\boldsymbol{q}} - \tilde{\boldsymbol{q}}_u) \\ =& (\boldsymbol{H} - \boldsymbol{H}^l)\boldsymbol{q}_l - (\boldsymbol{T} - \boldsymbol{T}^l)\tilde{\boldsymbol{q}} + \boldsymbol{P}^T \tilde{\boldsymbol{r}} - \boldsymbol{P}_l^T \tilde{\boldsymbol{r}}_l.\end{aligned} \tag{5}$$

where $\boldsymbol{H} = \boldsymbol{P}^T \text{diag}(\boldsymbol{c}) \boldsymbol{P}$, $\boldsymbol{H}^l = \boldsymbol{P}_l^T \text{diag}(\boldsymbol{c}_l) \boldsymbol{P}_l$, $\boldsymbol{T} = \boldsymbol{P}^T \boldsymbol{P}$, and $\boldsymbol{T}^l = \boldsymbol{P}_l^T \boldsymbol{P}_l$. The u-th element of $\boldsymbol{c}$ is C_u. $\tilde{\boldsymbol{r}} = \text{diag}(\boldsymbol{P}\tilde{\boldsymbol{Q}}^T)$, where $\tilde{\boldsymbol{Q}}$, of the same size as $\boldsymbol{P}$, stacks $\tilde{\boldsymbol{q}}_u$ by row. U_l is used to extract a submatrix $\boldsymbol{P}_l$ from $\boldsymbol{P}$, a subvector $\tilde{\boldsymbol{r}}_l$ from $\tilde{\boldsymbol{r}}$ and a subvector $\boldsymbol{c_l}$ of $\boldsymbol{c}$. Compared to the implicit regularizer in Eq (2), gradient with respect to latent factors is not symmetric any more and becomes much more complicated. The gradient with respect to an item latent factor is correlated with other item latent factors, so that parallel update among items is not feasible any more.

3.4 Regularizer Derivative

Next we separately consider each dimension of latent factors, and calculate its partial derivative of the regularizer for subsequent usage in coordinate descent algorithms. We consider the derivative of the regularizer with respect to p_{uf}, the f-th dimension of user latent vector $\boldsymbol{p}_u$.

$$\begin{aligned}\frac{\partial\Omega}{\partial p_{uf}} =& C_u(S_{ff} - S_{ff}^u)p_{uf} + C_u \sum_{k\neq f} p_{uk}(S_{kf} - S_{kf}^u) \\ &- (\tilde{q}_f - \tilde{q}_{uf})^2 p_{uf} - (\tilde{q}_f - \tilde{q}_{uf}) \sum_{k\neq f} p_{uk}(\tilde{q}_k - \tilde{q}_{uk})\end{aligned} \tag{6}$$

where S_{kf} is the (k,f) element in the matrix $\boldsymbol{S} = \sum_j \boldsymbol{q}_j \boldsymbol{q}_j^T = \boldsymbol{Q}^T\boldsymbol{Q}$. S_{kf}^u is the (k,f) element in the matrix $\boldsymbol{S}^u = \sum_{i\in E_u} \boldsymbol{q}_i \boldsymbol{q}_i^T = \boldsymbol{Q}_u^T \boldsymbol{Q}_u$. $\tilde{q}_k$ is the k-th element of the vector $\tilde{\boldsymbol{q}} = \sum_j \boldsymbol{q}_j$ and $\tilde{q}_{uf}$ is the f-th element of the vector $\tilde{\boldsymbol{q}}_u = \sum_{i\in E_u} \boldsymbol{q}_i$.

Similarly, the derivative of the regularizer Ω with respect to q_{lf}, the f-th dimension of the l-th item latent vector is :

$$\begin{aligned}\frac{\partial\Omega}{\partial q_{lf}} =& (H_{ff} - H_{ff}^l)q_{lf} + \sum_{k\neq f}(H_{kf} - H_{kf}^l)q_{lk} \\ &- (\boldsymbol{t}_f - \boldsymbol{t}_f^l)^T \tilde{\boldsymbol{q}} + \tilde{\boldsymbol{r}}^T \boldsymbol{P}[:,f] - \tilde{\boldsymbol{r}}_l^T \boldsymbol{P}_l[:,f]\end{aligned} \tag{7}$$

where H_{kf} denote the (k,f) element of the matrix $\boldsymbol{H} = \sum_u C_u \boldsymbol{p}_u \boldsymbol{p}_u^T$, H_{kf}^l denotes the (k,f) element of the matrix $\boldsymbol{H}^l = \sum_{u\in U_l} C_u \boldsymbol{p}_u \boldsymbol{p}_u^T$, $\boldsymbol{t}_f$ denotes the f-th row of the matrix $\boldsymbol{T} = \sum_u \boldsymbol{p}_u \boldsymbol{p}_u^T$ and $\boldsymbol{t}_f^l$ denotes the f-th row of the matrix $\boldsymbol{T}^l = \sum_{u\in U_l} \boldsymbol{p}_u \boldsymbol{p}_u^T$. $\boldsymbol{P}[:,f]$ denotes the f-th column of the matrix $\boldsymbol{P}$. Given these derivatives, we can optimize ranking-based losses with the proposed regularizer by the coordinate descent algorithms.

4 One-Class Collaborative Filtering With Ranking-based Implicit Regularizer

We first introduce the loss functions for the ranking-based collaborative filtering with the normalization strategy to deal with the imbalance problem between the interacted and uninteracted items. Then we develop the alternating least square and coordinate descent algorithms for the square and logit loss respectively for efficient optimization.

4.1 Loss function

Ranking-based methods do not use explicit prior ratings for uninteracted items, and exactly align with the ranking-based implicit regularizer. Basically, ranking-based methods assume interacted items are ranked higher than uninteracted items. Therefore, ranking-based OCCF usually optimizes the following objective function:

$$\mathcal{L}_1 = \sum_u \sum_{i\in E_u} \sum_{j\notin E_u} \ell(r_{ui} - r_{uj}, \hat{r}_{ui} - \hat{r}_{uj}) + \lambda R(\boldsymbol{P}, \boldsymbol{Q})$$

where r_{ui} denotes the true rating, $\hat{r}_{ui} = \boldsymbol{p}_u^T \boldsymbol{q}_i$ denotes the predicted rating and $R(\boldsymbol{P}, \boldsymbol{Q})$ is the regularization term of ℓ_1-norm or ℓ_2-norm. Note that we do not use the absolute ratings but the relative ratings and avert explicit prior ratings for uninteracted items. In the RankALS [8], $r_{ui} - r_{uj} = 1$ and $\ell(y,\hat{y}) = (y-\hat{y})^2$, where item popularity is incorporated for weighting each loss. In this case, all uninteracted items can be incorporated with efficiency guarantee, but correctly ranking pairs may also lead to large loss values. In the BPR [3], $\ell(y,\hat{y}) = -(y \log\sigma(\hat{y}) + (1-y)\log(1-\sigma(\hat{y})))$ and $r_{ui} - r_{uj} = 1$. In this case, sampling from uninteracted items is a necessary condition of efficient training. In other words, whichever of these loss functions is used, they can not guarantee that interacted items are ranked higher than uninteracted ones. Therefore, we will optimize the ranking-based loss function with the proposed ranking-based implicit regularizer as follows

$$\mathcal{L} = \sum_u \sum_{i\in E_u} \sum_{j\notin E_u} \ell(1, \hat{r}_{ui} - \hat{r}_{uj}) + \lambda R(\boldsymbol{P}, \boldsymbol{Q}) + \alpha\Omega \tag{8}$$

where we assume $r_{ui} - r_{uj} = 1$. α controls effect of the ranking-based implicit regularizer.

However, we observe that the magnitude of implicit regularizer is much larger than the loss, particularly the BPR loss, in Eq (8). Therefore, the coefficient α may be too sensitive to tune. According to our experience, the coefficient is usually very small to achieve best recommendation performance in the validation set. To reduce sensitivity of the coefficient, and intrinsically address the imbalance between interacted items and uninteracted items, we introduce a normalization strategy, which assign user-specific weight to each loss and each implicit regularizer. Particularly, we finally optimize the following objective function,

$$\begin{aligned}\mathcal{L} =& \overbrace{\sum_u \sum_{i\in E_u} \sum_{j\notin E_u} z_u \ell(1, \hat{r}_{ui} - \hat{r}_{uj})}^{\mathcal{L}'} + \lambda R(\boldsymbol{P}, \boldsymbol{Q}) \\ &+ \frac{1}{2}\alpha \underbrace{\sum_u \sum_{i<j\notin E_u} y_u (\hat{r}_{ui} - \hat{r}_{uj})^2}_{\Omega'}\end{aligned} \tag{9}$$

where $z_u = f(C_u \times N_u)$ and $y_u = f(C_u \times C_u)$ is a normalization coefficient for the loss and the regularizer, respectively. Note that N_u is the number of interacted items of user u and C_u is the number of uninteracted items. Here, we focus on

three strategies of the normalization coefficient, by setting $f(x) = \frac{1}{x}$, $f(x) = \frac{1}{\sqrt{x}}$ and $f(x) = \frac{1}{\log x}$. When $f(x) = \frac{1}{x}$, it simply computes the average loss and regularizer for each user, but weakens contribution of users who have interacted with many items to the loss. The other two strategies can alleviate the problem to some extent.

Though we introduce a normalization coefficient into the regularizer, the gradient and derivative of regularizer with respect to parameters can be easily obtained by following the chain rule. Particularly, letting $\breve{\boldsymbol{p}}_u = \sqrt{y_u}\boldsymbol{p}_u$ and explicitly showing parameters of the regularizer, $\breve{\Omega}(\boldsymbol{P}, \boldsymbol{Q}) = \Omega(\breve{\boldsymbol{P}}, \boldsymbol{Q})$, and the gradient can be derived as follows

$$\begin{aligned} \frac{\partial \breve{\Omega}(\boldsymbol{P}, \boldsymbol{Q})}{\partial \boldsymbol{p}_u} &= \sqrt{y_u}\frac{\partial \Omega(\breve{\boldsymbol{P}}, \boldsymbol{Q})}{\partial \breve{\boldsymbol{p}}_u} = y_u \frac{\partial \Omega(\boldsymbol{P}, \boldsymbol{Q})}{\partial \boldsymbol{p}_u} \\ \frac{\partial \breve{\Omega}(\boldsymbol{P}, \boldsymbol{Q})}{\partial \boldsymbol{q}_i} &= \frac{\partial \Omega(\breve{\boldsymbol{P}}, \boldsymbol{Q})}{\partial \boldsymbol{q}_i}. \end{aligned} \tag{10}$$

The derivation can be obtained similarly. However, such a trick is difficult to use in the loss functions, so we consider the loss functions one by one.

4.2 Optimizing Square Loss

The square loss is widely used in many machine learning models, due to yielding closed-form solutions and being convenience for theoretical analysis. When it is applied for ranking-based methods in one-class collaborative filtering, it is formulated as follows:

$$\mathcal{L}' = \frac{1}{2}\sum_u \sum_{i \in E_u} \sum_{j \notin E_u} z_u \left(1 - (\hat{r}_{ui} - \hat{r}_{uj})\right)^2 \tag{11}$$

Because of the large number of uninteracted items, straightforwardly computing gradient is very time-consuming. The use of square loss can make gradient computation efficient. Since we observe that both alternating least square and coordinate descent work better than gradient descent for optimizing WRMF [1], the well-known method for recommendation from implicit feedback, below we develop an alternating least square method and a coordinate descent method for efficient parameter learning.

4.2.1 Alternating Least Square

The optimization procedure of the alternating least square alternates between updating user latent vector and updating item latent vector until convergence. The ALS method requires us to derive the gradient of the objective function with respect to latent vectors. The gradient of the regularizer is given in Section 3.3, so below we derive the gradient of the loss $\mathcal{L}'$ in Eq (11). The gradient with respect to $\boldsymbol{p}_u$ is elaborated as follows:

$$\begin{aligned} \frac{\partial \mathcal{L}'}{\partial \boldsymbol{p}_u} =& \sum_{i \in E_u} \sum_{j \notin E_u} -z_u(\boldsymbol{q}_i - \boldsymbol{q}_j)\left(1 - (\boldsymbol{p}_u^T\boldsymbol{q}_i - \boldsymbol{p}_u^T\boldsymbol{q}_j)\right) \\ =& z_u\left(N_u\tilde{\boldsymbol{q}} - N\tilde{\boldsymbol{q}}_u\right) + z_u\left(N_u\boldsymbol{S} + (N - 2N_u)\,\boldsymbol{S}^u\right)\boldsymbol{p}_u \\ & - z_u\left(\tilde{\boldsymbol{q}}_u\left(\tilde{\boldsymbol{q}} - \tilde{\boldsymbol{q}}_u\right)^T + \left(\tilde{\boldsymbol{q}} - \tilde{\boldsymbol{q}}_u\right)\tilde{\boldsymbol{q}}_u^T\right)\boldsymbol{p}_u. \end{aligned} \tag{12}$$

Setting $\frac{\partial \mathcal{L}}{\partial \boldsymbol{p}_u} = \frac{\partial \mathcal{L}'}{\partial \boldsymbol{p}_u} + \alpha\frac{\partial \Omega}{\partial \boldsymbol{p}_u} + \lambda\boldsymbol{p}_u$ to zero, we can obtain the closed-form solution for updating latent vector of user u in a form of $\boldsymbol{A}_u\boldsymbol{p}_u = \boldsymbol{b}_u$. Note that $\tilde{\boldsymbol{q}}$ and $\boldsymbol{Q}^T\boldsymbol{Q}$ should be precomputed for efficient computation. Regarding the gradient with respect to the item latent vector $\boldsymbol{q}_l$, we elaborate the similar process as follows :

$$\begin{aligned} \frac{\partial \mathcal{L}'}{\partial \boldsymbol{q}_l} =& \sum_{u \notin U_l} \sum_{i \in E_u} z_u\boldsymbol{p}_u\left(1 - \left(\boldsymbol{p}_u^T\boldsymbol{q}_i - \boldsymbol{p}_u^T\boldsymbol{q}_l\right)\right) \\ & - \sum_{u \in U_l} \sum_{j \notin E_u} z_u\boldsymbol{p}_u\left(1 - \left(\boldsymbol{p}_u^T\boldsymbol{q}_l - \boldsymbol{p}_u^T\boldsymbol{q}_j\right)\right) \\ =& N(\mathring{\boldsymbol{T}} - \mathring{\boldsymbol{T}}^l)\boldsymbol{q}_l - (\mathring{\boldsymbol{H}} - 2\mathring{\boldsymbol{H}}^l)\boldsymbol{q}_l - \mathring{\boldsymbol{T}}^l\tilde{\boldsymbol{q}} \\ & - \boldsymbol{P}^T\mathrm{diag}(\boldsymbol{z})\tilde{\boldsymbol{r}} + \boldsymbol{P}_l^T\mathrm{diag}(\boldsymbol{z}_l)\tilde{\boldsymbol{r}}_l \\ & + \boldsymbol{P}^T(\boldsymbol{z} \odot \boldsymbol{n}) - N\boldsymbol{P}_l^T\boldsymbol{z}_l \end{aligned} \tag{13}$$

where the u-th element of $\boldsymbol{n}$ is N_u. $\mathring{\boldsymbol{T}}$ and $\mathring{\boldsymbol{H}}$ is obtained by substituting $\boldsymbol{p}_u$ with $\mathring{\boldsymbol{p}}_u = \sqrt{z_u}\boldsymbol{p}_u$ in $\boldsymbol{T}$ and $\boldsymbol{H}$, respectively. $\mathring{\boldsymbol{T}}^l$ and $\mathring{\boldsymbol{H}}^l$ respectively denote submatrix of $\mathring{\boldsymbol{T}}$ and $\mathring{\boldsymbol{H}}$ extracted by the user set U_l. Setting $\frac{\partial \mathcal{L}}{\partial \boldsymbol{q}_l} = \frac{\partial \mathcal{L}'}{\partial \boldsymbol{q}_l} + \alpha\frac{\partial \Omega}{\partial \boldsymbol{q}_l} + \lambda\boldsymbol{q}_l$ to zero, we can obtain the closed-form solution for updating latent vector of item l in a form of $\boldsymbol{A}_l\boldsymbol{q}_l = \boldsymbol{b}_l$. Due to independence of item l, $\mathring{\boldsymbol{T}}$, $\mathring{\boldsymbol{H}}$ and $\boldsymbol{b} = \boldsymbol{P}^T(\boldsymbol{z} \odot \boldsymbol{n})$ can be precomputed. However, it is infeasible to precompute $\tilde{\boldsymbol{q}}$, $\tilde{\boldsymbol{r}}$ and $\mathring{\boldsymbol{b}} = \boldsymbol{P}^T\mathrm{diag}(\boldsymbol{z})\tilde{\boldsymbol{r}}$, since they are continuously changing when each item latent vector is updated. We develop a strategy of first caching and continual updating to address this challenge. The details of procedure are referred in Algorithm 1, where each line is commented with time cost.

Algorithm 1: ALS for Square Loss

Input: The rating matrix $\boldsymbol{R} \in \mathbb{R}^{M \times N}$
Output: latent factors $\boldsymbol{P} \in \mathbb{R}^{M \times K}$, and $\boldsymbol{Q} \in \mathbb{R}^{N \times K}$

Randomly initialize $\boldsymbol{P}$, $\boldsymbol{Q}$;
for $u = \{1, \cdots, M\}$ **do**
  $\tilde{\boldsymbol{Q}}[u,:] \leftarrow \boldsymbol{Q}_u^T\boldsymbol{1}_{N_u}$;
  $\tilde{\boldsymbol{r}}[u] \leftarrow \langle\tilde{\boldsymbol{Q}}[u,:], \boldsymbol{p}_u\rangle$;
$\tilde{\boldsymbol{q}} \leftarrow \boldsymbol{Q}^T\boldsymbol{1}_N$;
repeat
  $\boldsymbol{S} \leftarrow \boldsymbol{Q}^T\boldsymbol{Q}$; // $\mathcal{O}(NK^2)$
  for $u = \{1, \cdots, M\}$ **do**
    $\boldsymbol{S}^u \leftarrow \boldsymbol{Q}_u^T\boldsymbol{Q}_u$; // $\mathcal{O}(N_uK^2)$
    Compute $\boldsymbol{A}_u$ and $\boldsymbol{b}_u$; // $\mathcal{O}(K^2)$
    Solve $\boldsymbol{A}_u\boldsymbol{p}_u = \boldsymbol{b}_u$; // $\mathcal{O}(K^3)$
  $\breve{\boldsymbol{H}} \leftarrow \boldsymbol{P}^T\mathrm{diag}(\boldsymbol{y} \odot \boldsymbol{c})\boldsymbol{P}$; // $\mathcal{O}(MK^2)$
  $\breve{\boldsymbol{T}} \leftarrow \boldsymbol{P}^T\mathrm{diag}(\boldsymbol{y})\boldsymbol{P}$; // $\mathcal{O}(MK^2)$
  $\breve{\boldsymbol{b}} \leftarrow \boldsymbol{P}^T\mathrm{diag}(\boldsymbol{y})\tilde{\boldsymbol{r}}$; // $\mathcal{O}(MK)$
  $\mathring{\boldsymbol{H}} \leftarrow \boldsymbol{P}^T\mathrm{diag}(\boldsymbol{z} \odot \boldsymbol{c})\boldsymbol{P}$; // $\mathcal{O}(MK^2)$
  $\mathring{\boldsymbol{T}} \leftarrow \boldsymbol{P}^T\mathrm{diag}(\boldsymbol{z})\boldsymbol{P}$; // $\mathcal{O}(MK^2)$
  $\mathring{\boldsymbol{b}} \leftarrow \boldsymbol{P}^T\mathrm{diag}(\boldsymbol{z})\tilde{\boldsymbol{r}}$; // $\mathcal{O}(MK)$
  $\boldsymbol{b} \leftarrow \boldsymbol{P}^T(\boldsymbol{z} \odot \boldsymbol{n})$; // $\mathcal{O}(MK)$
  for $l = 1 : N$ **do**
    Compute $\breve{\boldsymbol{H}}^l$, $\breve{\boldsymbol{T}}^l$, $\mathring{\boldsymbol{H}}^l$ and $\mathring{\boldsymbol{T}}^l$; // $\mathcal{O}(N_lK^2)$
    Compute $\boldsymbol{A}_l$ and $\boldsymbol{b}_l$; // $\mathcal{O}(K^2)$
    Solve $\boldsymbol{A}_l\boldsymbol{q}_l = \boldsymbol{b}_l$; // $\mathcal{O}(K^3)$
    $\tilde{\boldsymbol{q}} \leftarrow \tilde{\boldsymbol{q}} - \boldsymbol{q}_l^{old} + \boldsymbol{q}_l$; // $\mathcal{O}(K)$
    $\tilde{\boldsymbol{Q}}_{U_l} \leftarrow \tilde{\boldsymbol{Q}}_{U_l} - \boldsymbol{q}_l^{old} + \boldsymbol{q}_l$; // $\mathcal{O}(N_lK)$
    $\tilde{\boldsymbol{r}}_{U_l} \leftarrow \tilde{\boldsymbol{r}}_{U_l} + \boldsymbol{P}_l^T(-\boldsymbol{q}_l^{old} + \boldsymbol{q}_l)$; // $\mathcal{O}(N_lK)$
    $\breve{\boldsymbol{b}} \leftarrow \breve{\boldsymbol{b}} + \breve{\boldsymbol{T}}^l(-\boldsymbol{q}_l^{old} + \boldsymbol{q}_l)$; // $\mathcal{O}(K^2)$
    $\mathring{\boldsymbol{b}} \leftarrow \mathring{\boldsymbol{b}} + \mathring{\boldsymbol{T}}^l(-\boldsymbol{q}_l^{old} + \boldsymbol{q}_l)$; // $\mathcal{O}(K^2)$
until *Convergent*;

Complexity Since $\tilde{\boldsymbol{q}}$ and $\boldsymbol{Q}^T\boldsymbol{Q}$ are precomputed, updating user latent factor $\boldsymbol{p}_u$ only costs $\mathcal{O}(N_uK^2 + K^3)$, where we assume solving the system of linear equation costs $\mathcal{O}(K^3)$. By caching the global quantities and dynamically updating them, updating $\boldsymbol{q}_l$ only costs $\mathcal{O}(N_lK^2+K^3)$. To summarize, the time complexity of the proposed learning algorithm is $\mathcal{O}\left(\|\boldsymbol{R}\|_0K^2 + (M+N)K^3\right)$, where $\|\boldsymbol{R}\|_0$ denotes the number of non-zero entries in the interaction matrix $\boldsymbol{R}$. In other words, the optimization algorithm is only in linear proportion to the total number of users' interacted items.

Regarding of the space complexity, the algorithm needs $\mathcal{O}(MK+NK+K^2)$ memory spaces for storing recommendation models. More specifically, The user and item latent matrices occupies $\mathcal{O}(MK)$ and $\mathcal{O}(NK)$, respectively. We also need $\mathcal{O}(K^2)$ extra space for storing $\boldsymbol{Q}^T\boldsymbol{Q}$. Since we exploit the caching and updating strategy, we need $\mathcal{O}(MK)$, $\mathcal{O}(K)$, $\mathcal{O}(M)$ extra memory spaces for these matrices $\tilde{\boldsymbol{Q}}$, $\tilde{\boldsymbol{q}}$, $\tilde{r}_l$, respectively.

4.2.2 Coordinate Descent

In spite of being linearly proportional to the number of interacted items, alternating least square is still cubic with the dimension K, due to the computation of matrix inversion. To improve the efficiency, we develop an coordinate descent algorithm, which updates an element of latent factor at a time given others fixed. Below we first obtain the derivative with respect to p_{uf}

$$\begin{aligned}\frac{\partial \mathcal{L}'}{\partial p_{uf}} =& z_uN_uS_{ff}p_{uf} + z_u(N-2N_u)S^u_{ff}p_{uf} \\ &- 2z_u\tilde{q}_{uf}(\tilde{q}_f - \tilde{q}_{uf})p_{uf} + z_uN_u\tilde{q}_f \\ &- z_uN\tilde{q}_{uf} + z_u(N-2N_u)\sum_{k\neq f}p_{uk}S^u_{kf} \\ &+ z_uN_u\sum_{k\neq f}p_{uk}S_{kf} - z_u\tilde{q}_{uf}\sum_{k\neq f}p_{uk}\tilde{q}_k \\ &- z_u\tilde{q}_f\sum_{k\neq f}p_{uk}\tilde{q}_{uk} + 2z_u\tilde{q}_{uf}\sum_{k\neq f}p_{uk}\tilde{q}_{uk}\end{aligned} \tag{14}$$

We note that the time complexity of computing $\boldsymbol{S}^u$ is $\mathcal{O}(N_uK^2)$. Since each time we only update one factor, the computation can be accelerated. In particular, we do not compute $\boldsymbol{S}^u$ in advance, instead we directly compute S^u_{ff} and $\sum_{k\neq f}p_{uk}S^u_{kf}$ as follows:

$$\begin{aligned}S^u_{ff} &= \sum_{i\in E_u} q^2_{if}, \\ \sum_{k\neq f}p_{uk}S^u_{kf} &= \sum_{i\in E_u}\sum_{k\neq f}p_{uk}q_{ik}q_{if} = \sum_{i\in E_u}\hat{r}^f_{ui}q_{if}.\end{aligned}$$

where $\hat{r}^f_{ui} = \hat{r}_{ui} - p_{uf}q_{if}$. As long as we first cache prediction scores for each user's interacted items, these two quantities can be computed in $\mathcal{O}(N_u)$ for each factor.

Next we obtain the derivatives with respect to q_{lf},

$$\begin{aligned}\frac{\partial \mathcal{L}'}{\partial q_{lf}} &= N(\mathring{T}_{ff} - \mathring{T}^l_{ff})q_{lf} - (\mathring{H}_{ff} - 2\mathring{H}^l_{ff})q_{lf} - \mathring{\boldsymbol{t}}^l_f\tilde{\boldsymbol{q}} \\ &+ N\sum_{k\neq f}(\mathring{T}_{kf} - \mathring{T}^l_{kf})q_{lk} - \sum_{k\neq f}(\mathring{H}_{kf} - 2\mathring{H}^l_{kf})q_{lk} \\ &- \mathring{b}_f + \boldsymbol{P}_l[:,f]^T\text{diag}(\boldsymbol{z}_l)\tilde{\boldsymbol{r}}_l + b_f - N\boldsymbol{P}_l[:,f]^T\boldsymbol{z}_l\end{aligned} \tag{15}$$

Algorithm 2: Coordinate Descent for Square Loss

Input: The rating matrix $\boldsymbol{R} \in \mathbb{R}^{M\times N}$
Output: Latent factors $\boldsymbol{P} \in \mathbb{R}^{M\times K}$, and $\boldsymbol{Q} \in \mathbb{R}^{N\times K}$

Randomly initialize $\boldsymbol{P}$, $\boldsymbol{Q}$;
for $u = \{1, \cdots, M\}$ **do**
  $\tilde{\boldsymbol{Q}}[u,:] \leftarrow \boldsymbol{Q}_u^T\boldsymbol{1}_{N_u}$;
  $\tilde{\boldsymbol{r}}[u] \leftarrow \langle\tilde{\boldsymbol{Q}}[u,:], \boldsymbol{p}_u\rangle$;
$\tilde{\boldsymbol{q}} \leftarrow \boldsymbol{Q}^T\boldsymbol{1}_N$;
repeat
  $\boldsymbol{S} \leftarrow \boldsymbol{Q}^T\boldsymbol{Q}$; // $\mathcal{O}(NK^2)$
  for $u = \{1, \cdots, M\}$ **do**
    $\hat{\boldsymbol{r}}_{E_u} \leftarrow \boldsymbol{Q}_u\boldsymbol{p}_u$; // $\mathcal{O}(N_uK)$
    for $f = \{1, \cdots, K\}$ **do**
      Compute $S^u_{ff}, \sum_{k\neq f}p_{uk}S^u_{kf}$; // $\mathcal{O}(N_u)$
      Update p_{uf} ; // $\mathcal{O}(K)$
      $\hat{\boldsymbol{r}}_{E_u} \leftarrow \hat{\boldsymbol{r}}_{E_u} + (p_{uf} - p^{old}_{uf})\boldsymbol{Q}_u[:,f]$;
  $\breve{\boldsymbol{H}} \leftarrow \boldsymbol{P}^T\text{diag}(\boldsymbol{y}\odot\boldsymbol{c})\boldsymbol{P}$; // $\mathcal{O}(MK^2)$
  $\breve{\boldsymbol{T}} \leftarrow \boldsymbol{P}^T\text{diag}(\boldsymbol{y})\boldsymbol{P}$; // $\mathcal{O}(MK^2)$
  $\breve{\boldsymbol{b}} \leftarrow \boldsymbol{P}^T\text{diag}(\boldsymbol{y})\tilde{\boldsymbol{r}}$; // $\mathcal{O}(MK)$
  $\mathring{\boldsymbol{H}} \leftarrow \boldsymbol{P}^T\text{diag}(\boldsymbol{z}\odot\boldsymbol{c})\boldsymbol{P}$; // $\mathcal{O}(MK^2)$
  $\mathring{\boldsymbol{T}} \leftarrow \boldsymbol{P}^T\text{diag}(\boldsymbol{z})\boldsymbol{P}$; // $\mathcal{O}(MK^2)$
  $\mathring{\boldsymbol{b}} \leftarrow \boldsymbol{P}^T\text{diag}(\boldsymbol{z})\tilde{\boldsymbol{r}}$; // $\mathcal{O}(MK)$
  $\boldsymbol{b} \leftarrow \boldsymbol{P}^T(\boldsymbol{z}\odot\boldsymbol{n})$; // $\mathcal{O}(MK)$
  for $l = \{1, \cdots, N\}$ **do**
    $\tilde{\boldsymbol{r}}^0_{U_l} \leftarrow \tilde{\boldsymbol{r}}_{U_l}$; // $\mathcal{O}(N_l)$
    $\hat{\boldsymbol{r}}_{U_l} \leftarrow \boldsymbol{P}_l\boldsymbol{q}_l$; // $\mathcal{O}(N_lK)$
    $\bar{\boldsymbol{r}}_{U_l} \leftarrow \boldsymbol{P}_l\tilde{\boldsymbol{q}}$; // $\mathcal{O}(N_lK)$
    for $f = \{1, \cdots, K\}$ **do**
      Compute $\mathring{H}^u_{ff}, \sum_{k\neq f}p_{uk}\mathring{H}^u_{kf}$; // $\mathcal{O}(N_l)$
      Compute $\breve{H}^u_{ff}, \sum_{k\neq f}p_{uk}\breve{H}^u_{kf}$;
      Compute $\mathring{T}^u_{ff}, \sum_{k\neq f}p_{uk}\mathring{T}^u_{kf}$;
      Compute $\breve{T}^u_{ff}, \sum_{k\neq f}p_{uk}\breve{T}^u_{kf}$;
      Compute $\mathring{\boldsymbol{t}}^l_f\tilde{\boldsymbol{q}}$ and $\breve{\boldsymbol{t}}^l_f\tilde{\boldsymbol{q}}$; // $\mathcal{O}(N_l)$
      $\breve{b}_f \leftarrow \breve{b}_f + \boldsymbol{P}_l[:,f]^T\text{diag}(\boldsymbol{y})(\tilde{\boldsymbol{r}}_{U_l} - \tilde{\boldsymbol{r}}^0_{U_l})$;
      $\mathring{b}_f \leftarrow \mathring{b}_f + \boldsymbol{P}_l[:,f]^T\text{diag}(\boldsymbol{z})(\tilde{\boldsymbol{r}}_{U_l} - \tilde{\boldsymbol{r}}^0_{U_l})$;
      Update q_{lf} ; // $\mathcal{O}(K)$
      $\tilde{q}_f \leftarrow \tilde{q}_f + q_{lf} - q^{old}_{lf}$;
      $\tilde{\boldsymbol{Q}}[U_l,f] \leftarrow \tilde{\boldsymbol{Q}}[U_l,f] + q_{lf} - q^{old}_{lf}$;
      $\tilde{\boldsymbol{r}}_{U_l} \leftarrow \tilde{\boldsymbol{r}}_{U_l} + \boldsymbol{P}_l[:,f](q_{lf} - q^{old}_{lf})$; // $\mathcal{O}(N_l)$
      $\hat{\boldsymbol{r}}_{U_l} \leftarrow \hat{\boldsymbol{r}}_{U_l} + \boldsymbol{P}_l[:,f](q_{lf} - q^{old}_{lf})$; // $\mathcal{O}(N_l)$
      $\bar{\boldsymbol{r}}_{U_l} \leftarrow \bar{\boldsymbol{r}}_{U_l} + \boldsymbol{P}_l[:,f](q_{lf} - q^{old}_{lf})$; // $\mathcal{O}(N_l)$
until *Convergent*;

We also do not compute $\mathring{\boldsymbol{H}}^l$ and $\mathring{\boldsymbol{T}}^l$ in advance, we directly compute the related quantities as follows

$$\begin{aligned}\mathring{T}^l_{ff} &= \sum_{u\in U_l} z_up^2_{uf}, \\ \mathring{H}^l_{ff} &= \sum_{u\in U_l} C_uz_up^2_{uf}, \\ \sum_{k\neq f}q_{lk}\mathring{T}^l_{kf} &= \sum_{u\in U_l} z_u(\hat{r}^f_{ul})p_{uf}, \\ \sum_{k\neq f}q_{lk}\mathring{H}^l_{kf} &= \sum_{u\in U_l} C_uz_u(\hat{r}^f_{ul})p_{uf}, \\ \mathring{\boldsymbol{t}}^l_f\tilde{\boldsymbol{q}} &= \sum_{u\in U_l} z_up_{uf}\boldsymbol{p}^T_u\tilde{\boldsymbol{q}}.\end{aligned}$$

As long as we additionally cache $\bar{r}_u = \boldsymbol{p}^T_u\tilde{\boldsymbol{q}}$, these quantities are computed in $\mathcal{O}(N_l)$. By observing only users in U_l are

used for computation, we suggest updating $\bar{r}_u$ only for users in U_l instead of the whole users after updating each factor. This dramatically reduces the computational cost compared with the ALS method. The details of procedure are referred in Algorithm 2.

Complexity Compared with alternating least square method, matrix inversion is not required in the coordinate descent optimization method. By caching the global quantities and dynamically updating them, the time complexity of parameter learning is now reduced to $O(|R|K + (M + N)K^2)$, K times faster than alternating least square. However, it may require more iterations for convergence. The space complexity of optimizing recommendation models is the same as ALS in Section 4.2.1, i.e., $\mathcal{O}(MK + NK + K^2)$.

4.3 Optimizing Logit Loss

The square loss can yield closed-form equations for updating parameters, but may lead to large loss values for accurately ranking pairs. In this section, we investigate how to optimize the ranking-based logit loss with the proposed regularizer. Different from the square loss, it is impossible to incorporate all uninteracted items at a time. To address the issue, we sample the same size of uninteracted items as interacted items, and assume they should be ranked lower than interacted items in the ranking-based logit loss, following previous work [3]. Then, the ranking-based logit loss with the proposed regularizer is formulated as follows:

$$\mathcal{L}' = \sum_u \sum_{i \in E_u} \sum_{j \in F_u} z_u \log\left(1 + \exp\left(-\left(\hat{r}_{ui} - \hat{r}_{uj}\right)\right)\right)$$

where F_u is the set of sampled negative items for user u. Different from the Bayesian Personalize Ranking (BPR) loss, which compares an interacted item with a sampled uninteracted item, the loss function compares an interacted item with all items in F_u.

Another issue of the rankingb-based logit loss lies in difficulty of directly deriving the closed-form updating equations. Here, we seek a upper variational quadratic bound of the logit loss [12], [34] and then derive the closed-form updating equations for alternating least squares and coordinate descent methods. Particularly,

$$\begin{aligned}
&\log\left(1 + e^{-\hat{r}_{uij}}\right) = \log(1 + e^{\hat{r}_{uij}}) - \hat{r}_{uij} \\
\leq &\lambda\left(\hat{\epsilon}_{uij}\right)\left(\hat{r}_{uij}^2 - \hat{\epsilon}_{uij}^2\right) - \frac{1}{2}\left(\hat{r}_{uij} + \hat{\epsilon}_{uij}\right) + \log(1 + e^{\hat{\epsilon}_{uij}})
\end{aligned}$$

where $\lambda(x) = \frac{1}{4x}\tanh\left(\frac{x}{2}\right) = \frac{1}{2x}\left(\sigma(x) - \frac{1}{2}\right)$ and $\sigma(x)$ is the sigmoid function. The equality holds only if $\hat{\epsilon}_{uij} = \hat{r}_{uij}$. By setting $\hat{r}_{uij} = (\hat{r}_{ui} - \hat{r}_{uj})$, the ranking-based logit loss is bounded from the above by

$$\hat{\mathcal{L}}' = \sum_u \sum_{i \in E_u} \sum_{j \in F_u} z_u \left(\lambda\left(\hat{\epsilon}_{uij}\right)\left(\hat{r}_{ui} - \hat{r}_{uj}\right)^2 - \frac{1}{2}\left(\hat{r}_{ui} - \hat{r}_{uj}\right)\right)$$

where $\hat{\epsilon}_{uij}$ is the most recent estimation of $\hat{r}_{uij}$. Next we elaborate the ALS and CD algorithms for optimizing the ranking-based logit loss.

4.3.1 Alternating Least Square

The gradient of $\hat{\mathcal{L}}'$ with respect to user latent vector $\boldsymbol{p}_u$ is derived as follows:

$$\begin{aligned}
\frac{\partial \hat{\mathcal{L}}'}{\partial \boldsymbol{p}_u} =& \sum_{i \in E_u} \sum_{j \in F_u} 2 z_u \lambda(\hat{\epsilon}_{uij}) \left(\boldsymbol{q}_i - \boldsymbol{q}_j\right)\left(\boldsymbol{q}_i - \boldsymbol{q}_j\right)^T \boldsymbol{p}_u \\
&- \sum_{i \in E_u} \sum_{j \in F_u} \frac{1}{2} z_u \left(\boldsymbol{q}_i - \boldsymbol{q}_j\right) \\
=& 2 z_u \sum_{i \in E_u} \left(\sum_{j \in F_u} \lambda(\hat{\epsilon}_{uij})\right) \boldsymbol{q}_i \boldsymbol{q}_i^T \boldsymbol{p}_u \\
&+ 2 z_u \sum_{j \in F_u} \left(\sum_{i \in E_u} \lambda(\hat{\epsilon}_{uij})\right) \boldsymbol{q}_j \boldsymbol{q}_j^T \boldsymbol{p}_u \\
&- 2 z_u \sum_{i \in E_u} \sum_{j \in F_u} \lambda(\hat{\epsilon}_{uij}) \left(\boldsymbol{q}_i \boldsymbol{q}_j^T \boldsymbol{p}_u + \boldsymbol{q}_j \boldsymbol{q}_i^T \boldsymbol{p}_u\right) \\
&- \frac{1}{2} z_u N_u \sum_{i \in E_u} \boldsymbol{q}_i + \frac{1}{2} z_u N_u \sum_{j \in F_u} \boldsymbol{q}_j
\end{aligned} \tag{16}$$

For efficient computation, we cache prediction scores for positive items in a sparse matrix $\hat{\boldsymbol{R}}^+$ and negative items in a sparse matrix $\hat{\boldsymbol{R}}^-$, so that $\lambda(\epsilon_{uij})$ is computed in $\mathcal{O}(1)$. Setting $\frac{\partial \mathcal{L}}{\partial \boldsymbol{p}_u} = \frac{\partial \hat{\mathcal{L}}'}{\partial \boldsymbol{p}_u} + \alpha \frac{\partial \Omega}{\partial \boldsymbol{p}_u} + \lambda \boldsymbol{p}_u$ to zero, we can obtain the closed-form solution for updating latent vector of user u in a form of $\boldsymbol{A}_u \boldsymbol{p}_u = \boldsymbol{b}_u$. Computing a part of $\boldsymbol{A}_u$ in Eq (16) is achieved with a two-layer loop, which costs $\mathcal{O}(N_u^2 K^2)$.

The gradient of $\hat{\mathcal{L}}'$ with respect to item latent vector $\boldsymbol{q}_l$ is derived as follows:

$$\begin{aligned}
\frac{\partial \hat{\mathcal{L}}'}{\partial \boldsymbol{q}_l} =& \sum_{u \in U_l} \sum_{j \in F_u} 2 z_u \lambda\left(\hat{\epsilon}_{ulj}\right) \left(\boldsymbol{p}_u^T \boldsymbol{q}_l - \boldsymbol{p}_u^T \boldsymbol{q}_j\right) \boldsymbol{p}_u - \frac{1}{2} z_u \boldsymbol{p}_u \\
&- \sum_{u \in V_l} \sum_{i \in E_u} 2 z_u \lambda\left(\hat{\epsilon}_{uil}\right) \left(\boldsymbol{p}_u^T \boldsymbol{q}_i - \boldsymbol{p}_u^T \boldsymbol{q}_l\right) \boldsymbol{p}_u + \frac{1}{2} z_u \boldsymbol{p}_u \\
=& 2 \sum_{u \in U_l} z_u \sum_{j \in F_u} \lambda(\hat{\epsilon}_{ulj}) \boldsymbol{p}_u \boldsymbol{p}_u^T \boldsymbol{q}_l \\
&+ 2 \sum_{u \in V_l} z_u \sum_{i \in E_u} \lambda(\hat{\epsilon}_{uil}) \boldsymbol{p}_u \boldsymbol{p}_u^T \boldsymbol{q}_l \\
&- 2 \sum_{u \in U_l} z_u \boldsymbol{p}_u \sum_{j \in F_u} \lambda(\hat{\epsilon}_{ulj}) \hat{r}_{uj}^- \\
&- 2 \sum_{u \in V_l} z_u \boldsymbol{p}_u \sum_{i \in E_u} \lambda(\hat{\epsilon}_{uil}) \hat{r}_{ui}^+ \\
&- \frac{1}{2} \sum_{u \in U_l} z_u N_u \boldsymbol{p}_u + \frac{1}{2} \sum_{u \in V_l} z_u N_u \boldsymbol{p}_u
\end{aligned} \tag{17}$$

where V_l is a set of users without interacting item l. We follow the similar first-caching-and-continual-updating strategy to efficiently compute high-cost quantities. Setting $\frac{\partial \mathcal{L}}{\partial \boldsymbol{q}_l} = \frac{\partial \hat{\mathcal{L}}'}{\partial \boldsymbol{q}_l} + \alpha \frac{\partial \Omega}{\partial \boldsymbol{q}_l} + \lambda \boldsymbol{q}_l$ to zero, we can obtain the closed-form solution for updating $\boldsymbol{q}_l$ in a form of $\boldsymbol{A}_l \boldsymbol{q}_l = \boldsymbol{b}_l$. The details of procedure are referred in Algorithm 3.

Complexity According to Algorithm 3, it is easy to understand that time complexity of updating user latent factor is $\mathcal{O}(\sum_u N_u^2 K^2 + MK^3)$. Regarding to updating latent factor for all items, the two-layer loop costs $\mathcal{O}\left(K^2(\sum_{l=1}^N \sum_u^M \delta_{ul}^+ \sum_{j=1}^N \delta_{uj}^- + \sum_{l=1}^N \sum_u^M \delta_{ul}^- \sum_{j=1}^N \delta_{uj}^+)\right)$

Algorithm 3: ALS for Logit-Loss

Input: The rating matrix $\boldsymbol{R} \in \mathbb{R}^{M \times N}$
Output: Latent factors $\boldsymbol{P} \in \mathbb{R}^{M \times K}$, and $\boldsymbol{Q} \in \mathbb{R}^{N \times K}$

1 Randomly initialize $\boldsymbol{P}$, $\boldsymbol{Q}$;
2 **for** $u \in \{1, \cdots, M\}$ **do**
3 $\quad \tilde{\boldsymbol{Q}}[u,:] \leftarrow \boldsymbol{Q}_u^T \mathbf{1}_{N_u}$;
4 $\quad \tilde{\boldsymbol{r}}[u] \leftarrow \langle \tilde{\boldsymbol{Q}}[u,:], \boldsymbol{p}_u \rangle$;
5 $\tilde{\boldsymbol{q}} \leftarrow \boldsymbol{Q}^T \mathbf{1}_N$;
6 **repeat**
7 $\quad$ **for** $u \in \{1, \cdots, M\}$ **do**
8 $\quad\quad$ Sample item set F_u from $E \backslash E_u$;
9 $\quad \boldsymbol{S} \leftarrow \boldsymbol{Q}^T \boldsymbol{Q}$; // $\mathcal{O}(NK^2)$
10 $\quad$ **for** $u \in \{1, \cdots, M\}$ **do**
11 $\quad\quad \hat{\boldsymbol{r}}^+_{E_u}, \hat{\boldsymbol{r}}^-_{F_u} \leftarrow \boldsymbol{Q}[E_u,:]\boldsymbol{p}_u, \boldsymbol{Q}[F_u,:]\boldsymbol{p}_u$;
12 $\quad\quad \boldsymbol{A}_u \leftarrow$ Hessian matrix of $\left(\alpha\Omega + \frac{1}{2}\lambda\|\boldsymbol{p}_u\|^2\right)$;
13 $\quad\quad$ **for** $i \in E_u$ **do**
14 $\quad\quad\quad$ **for** $j \in F_u$ **do**
15 $\quad\quad\quad\quad \lambda(\hat{e}_{uij}) \leftarrow \lambda(\hat{r}^+_{ui} - \hat{r}^-_{uj})$; // $\mathcal{O}(1)$
16 $\quad\quad\quad\quad$ Update $\boldsymbol{A}_u$; // $\mathcal{O}(K^2)$
17 $\quad\quad$ Compute $\boldsymbol{b}_u$; // $\mathcal{O}(N_u K)$
18 $\quad\quad$ Solve $\boldsymbol{A}_u \boldsymbol{p}_u = \boldsymbol{b}_u$; // $\mathcal{O}(K^3)$
19 $\quad\quad \hat{\boldsymbol{r}}^+_{E_u}, \hat{\boldsymbol{r}}^-_{F_u} \leftarrow \boldsymbol{Q}[E_u,:]\boldsymbol{p}_u, \boldsymbol{Q}[F_u,:]\boldsymbol{p}_u$
20 $\quad \breve{\boldsymbol{H}} \leftarrow \boldsymbol{P}^T \text{diag}(\boldsymbol{y} \odot \boldsymbol{c}) \boldsymbol{P}$; // $\mathcal{O}(MK^2)$
21 $\quad \breve{\boldsymbol{T}} \leftarrow \boldsymbol{P}^T \text{diag}(\boldsymbol{y}) \boldsymbol{P}$; // $\mathcal{O}(MK^2)$
22 $\quad \breve{\boldsymbol{b}} \leftarrow \boldsymbol{P}^T \text{diag}(\boldsymbol{y}) \tilde{\boldsymbol{r}}$; // $\mathcal{O}(MK)$
23 $\quad$ **for** $l = 1 : N$ **do**
24 $\quad\quad \hat{\boldsymbol{r}}^+_{U_l}, \hat{\boldsymbol{r}}^-_{V_l} \leftarrow \boldsymbol{P}[U_l,:]\boldsymbol{q}_l, \boldsymbol{P}[V_l,:]\boldsymbol{q}_l$;
25 $\quad\quad \boldsymbol{A}_l \leftarrow$ Hessian matrix of $\left(\alpha\Omega + \frac{1}{2}\lambda\|\boldsymbol{q}_l\|^2\right)$;
26 $\quad\quad \boldsymbol{b}_l \leftarrow -\frac{1}{2}\sum_{u \in U_l} z_u N_u \boldsymbol{p}_u + \frac{1}{2}\sum_{u \in V_l} z_u N_u \boldsymbol{p}_u$;
27 $\quad\quad$ **for** $u \in U_l$ **do**
28 $\quad\quad\quad$ **for** $j \in F_u$ **do**
29 $\quad\quad\quad\quad \lambda(\hat{e}_{ulj}) \leftarrow \lambda(\hat{r}^+_{ul} - \hat{r}^-_{uj})$; // $\mathcal{O}(1)$
30 $\quad\quad\quad\quad$ Update $\boldsymbol{A}_l, \boldsymbol{b}_l$; // $\mathcal{O}(K^2)$
31 $\quad\quad$ **for** $u \in V_l$ **do**
32 $\quad\quad\quad$ **for** $i \in E_u$ **do**
33 $\quad\quad\quad\quad \lambda(\hat{e}_{uil}) \leftarrow \lambda(\hat{r}^+_{ui} - \hat{r}^-_{ul})$; // $\mathcal{O}(1)$
34 $\quad\quad\quad\quad$ Update $\boldsymbol{A}_l, \boldsymbol{b}_l$; // $\mathcal{O}(K^2)$
35 $\quad\quad$ Solve $\boldsymbol{A}_l \boldsymbol{q}_l = \boldsymbol{b}_l$; // $\mathcal{O}(K^3)$
36 $\quad\quad$ **for** $u \in U_l$ **do**
37 $\quad\quad\quad$ Update score of item l in $\hat{\boldsymbol{r}}^+_{E_u}, \hat{\boldsymbol{r}}^-_{F_u}$;
38 $\quad\quad \tilde{\boldsymbol{q}} \leftarrow \tilde{\boldsymbol{q}} - \boldsymbol{q}_l^{old} + \boldsymbol{q}_l$; // $\mathcal{O}(K)$
39 $\quad\quad \tilde{\boldsymbol{Q}}_{U_l} \leftarrow \tilde{\boldsymbol{Q}}_{U_l} - \boldsymbol{q}_l^{old} + \boldsymbol{q}_l$; // $\mathcal{O}(N_l K)$
40 $\quad\quad \tilde{\boldsymbol{r}}_{U_l} \leftarrow \tilde{\boldsymbol{r}}_{U_l} + \boldsymbol{P}_l^T(-\boldsymbol{q}_l^{old} + \boldsymbol{q}_l)$; // $\mathcal{O}(N_l K)$
41 $\quad\quad \breve{\boldsymbol{b}} \leftarrow \breve{\boldsymbol{b}} + \breve{\boldsymbol{T}}^l(-\boldsymbol{q}_l^{old} + \boldsymbol{q}_l)$; // $\mathcal{O}(K^2)$
42 **until** *Convergent*;

in total, where we use δ^+_{uj} indicate item j is positive and δ^-_{uj} indicate item j is negative. The time complexity equals $\mathcal{O}(\sum_u N_u^2 K^2)$ by exchanging the first two summations. One thing to pay special attention is Line 37 in Algorithm 3, where we additionally store element index of E_u and F_u to ensure $\mathcal{O}(1)$ update. Therefore, the total time complexity of the optimization algorithm is $\mathcal{O}(\sum_u N_u^2 K^2 + (M+N)K^3)$. The space complexity of optimizing recommendation models is the same as ALS in Section 4.2.1, i.e., $\mathcal{O}(MK + NK + K^2)$.

4.3.2 Coordinate Descent

Next we investigate how to optimize the objective function with coordinate descent. We first derive the derivative of the objective function with respect to p_{uf} as follows:

$$\begin{aligned}\frac{\partial \hat{\mathcal{L}}'}{\partial p_{uf}} = &\sum_{i \in E_u}\sum_{j \in F_u} 2z_u \lambda(\hat{\epsilon}_{uij})\left(q_{if} - q_{jf}\right)^2 p_{uf} \\ &+ \sum_{i \in E_u}\sum_{j \in F_u} 2z_u \lambda(\hat{\epsilon}_{uij})\left(q_{if} - q_{jf}\right)\left(\hat{r}^f_{ui} - \hat{r}^f_{uj}\right) \\ &- \sum_{i \in E_u}\sum_{j \in K_u} \frac{1}{2} z_u (q_{if} - q_{jf})\end{aligned} \tag{18}$$

where $\hat{r}^f_{ui} = \hat{r}_{ui} - p_{uf} q_{if}$. The derivative with respect to q_{lf} is given as:

$$\begin{aligned}\frac{\partial \hat{\mathcal{L}}'}{\partial q_{lf}} = &\sum_{u \in U_l}\sum_{j \in F_u} 2z_u \lambda(\hat{\epsilon}_{ulj})\left(\hat{r}^f_{ul} p_{uf} + p^2_{uf} q_{lf} - \hat{r}_{uj} p_{uf}\right) \\ &+ \sum_{u \in V_l}\sum_{i \in E_u} 2z_u \lambda(\hat{\epsilon}_{uil})\left(\hat{r}^f_{ul} p_{uf} + p^2_{uf} q_{lf} - \hat{r}_{ui} p_{uf}\right) \\ &- \sum_{u \in U_l} \frac{1}{2} z_u N_u p_{uf} + \sum_{u \in V_l} \frac{1}{2} z_u N_u p_{uf}\end{aligned} \tag{19}$$

Setting $\frac{\partial \mathcal{L}}{\partial p_{uf}} = \frac{\partial \hat{\mathcal{L}}'}{\partial p_{uf}} + \alpha \frac{\partial \Omega}{\partial p_{uf}} + \lambda p_{uf} = 0$ and $\frac{\partial \mathcal{L}}{\partial q_{lf}} = \frac{\partial \hat{\mathcal{L}}'}{\partial q_{lf}} + \alpha \frac{\partial \Omega}{\partial q_{lf}} + \lambda q_{lf} = 0$, we can take turn to update each factor of latent factors until convergence. Different from alternating learning square, the variational parameter $\lambda(\hat{\epsilon}_{uij})$ should be updated whenever a factor is updated. This means $\lambda(\hat{\epsilon}_{uij})$ is computed K times as many as the alternating least square method. However, when only one factor is updated each time, the loss could be closer to the upper bound. According to Algorithm 2 and Algorithm 3, it is easy to figure out the detailed procedure of the coordinate descent algorithm, so that we omit it for space limitation. Based on the analysis of Algorithm 2, it is easy to deduce that the time complexity of the coordinate descent algorithm is $O(\sum_u N_u^2 K + (M+N)K^2)$, which should be K times faster than the alternating least square method. The space complexity of optimizing recommendation models is the same as ALS in Section 4.2.1, i.e., $\mathcal{O}(MK + NK + K^2)$.

5 Experiments

5.1 Experimental Settings

5.1.1 Datasets

We perform experiments on two types of datasets to verify the effectiveness of our methods. The first type of dataset is explicit feedback dataset, including MovieLens, Amazon and Yelp, where the ratings range from 1 to 5.

- **MovieLens**: The MovieLens dataset is from the famous MovieLens10M dataset, including 10,000,054 ratings from 71,567 users for 10,681 items.
- **Yelp**: The Yelp dataset includes 2,685,066 ratings from 409,117 users for 85,539 points of interest, such as restaurants, hotels, and shopping malls.
- **Netflix**: The Netflix dataset is another famous movie rating dataset. It contains 463,770 users and 17,764 items, which is larger than the MovieLens and Yelp dataset.

TABLE 1
The Statistics of the Datasets

Type	Datasets	Ratings	Users	Items	Sparsity
Explicit datasets	MovieLens	9,983,739	69,838	8,939	83.67%
	Yelp	2,103,895	77,277	45,638	99.94%
	Netflix	100,396,329	463,770	17,764	98.78%
Implicit datasets	CiteUlike	204,986	5,551	16,980	99.78%
	Gowalla	1,027,464	29,858	40,988	99.92%
	LastFM	16,574,711	351,422	61,257	99.92%

These explicit feedback datasets are converted into implicit feedback datasets following [3], by considering user's rated item as positive. The other type of dataset for experiments is implicit feedback dataset, including CiteUlike, Gowalla and LastFM, where user's interacted items are considered as positive.

- **CiteUlike**: The CiteULike dataset is from user's individual library of articles [35].
- **Gowalla**: The Gowalla dataset includes users' check-ins at locations in a location-based social network.
- **LastFM**: The dataset is a famous music recommendation dataset [36] with 360,000 users.

All the datasets are publicly available and we filter the users and items with fewer than 10 interactions. Table 1 summarizes the statistics of the datasets.

When optimizing the ranking-based logit loss, we randomly pick N_u items from uninteracted ones for each user during each iteration and treat them as negative samples.

For each user, we randomly sample his 80% ratings as training set and the rest 20% as testing test. We fit a model to the training set and evaluate it in the test set. We repeat 5 random splits and report the averaged performances.

5.1.2 Baselines

- **RankALS** [8] optimizes the ranking-based square loss with an alternating least square algorithm. The parameter s_j for each item is set to 1 so that they can be fairly compared with the proposed methods.
- **WRMF** [1], [2] optimizes regression-based square loss, by considering all uninteracted items as negative and assigning them lower confidence being negative. The optimization is based on alternating least square, which scales linearly with the number of interaction.
- **BPR** [3] optimizes the ranking-based logit loss with stochastic gradient descent. The negative samples are randomly picked from uninteracted items.
- **SQL_RANK** [37] exploits a listwise approach for recommendation which cast listwise collaborative ranking as maximum likelihood under a permutation model.

By presenting the top k candidate items for each user, two standard metrics are used to assess the quality of ranking: NDCG and Recall. The parameters are fine-tuned according to the metric of NDCG@200.

5.2 Results and Analysis

5.2.1 Comparison with baselines

We tune hyperparameters of the proposed methods and baselines in a validation set, which consists of 5% training data. Note that the proposed method with the square loss is trained for 20 iterations and the proposed method with logit loss is trained for 50 iterations. Results are shown in Table 2, where we set the dimension of latent space in all methods to 32 and report NDCG and Recall at cutoffs 10, 50, and 200.

On all the datasets, our proposed algorithm outperforms all the baselines. In particular, the square loss with the ranking-based implicit regularizer (Square-RIR) is much better than RanKALS. The relative improvements are 120.1% on average with respect to NDCG@10. This shows the effectiveness of the proposed regularizer. The logit loss with the ranking-based implicit regularizer (Logit-RIR) is also much better than BPR. The relative improvements are 51.7% on average with respect to NDCG@10. The superior performance of BPR to RankALS may lie in logit loss, which can yield small loss values for accurately ranking pairs. WRMF is the best among the baselines, since it imposes an implicit regularizer $\sum_u \sum_{i \notin E_u} (\boldsymbol{p}_u^T \boldsymbol{q}_i)^2$ to penalize non-zero preference prediction. However, the proposed algorithms still outperform WRMF, and the average relative improvements are 5.5% in the explicit feedback datasets, and up to 8.3% in the implicit feedback datasets in terms of NDCG@10. This indicates that the ranking-based implicit regularizer is more suitable for sparser implicit feedback datasets. The logit loss can not consistently beat the square loss, but BPR is better than RankALS. This indicates that the ranking-based implicit regularizer benefits the square loss more than the logit loss, to reduce large loss values for accurately ranking pairs.

Furthermore, we investigate how the performance of the proposed algorithm varies with the increasing dimension of latent space. The results of NDCG@200 on MovieLens and Gowalla datasets are reported in Figure 1, where K is varied in the set $\{16, 32, 48, 64, 128\}$. As the dimension of latent space increases, the performance of the proposed method with the logit loss gets close to the square loss in terms of NDCG@200 in both datasets. The proposed method with the logit loss outperforms BPR, and the relative improvements increase as the dimension grows. Compared to WRMF, the proposed methods are better even with the increase of dimension, especially in the MovieLens dataset. In the sparser Gowalla dataset, as the dimension increases, the relative improvements to WRMF algorithms gradually decrease. This shows the superiority of the proposed ranking-based implicit regularizer to the implicit regularizer in WRMF.

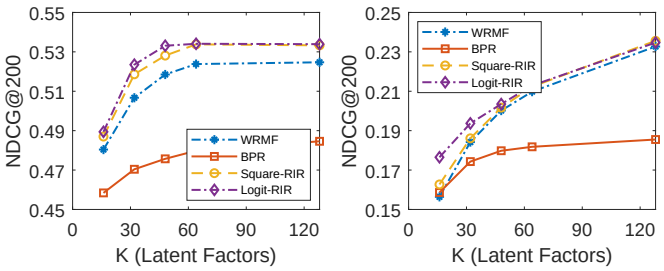
5.2.2 Effect of loss functions

As we exploit two kinds of loss functions and four normalization strategies in our framework, we conduct an ablation study to understand the effect of normalization strategies and loss functions in the four datasets. Alternating least square (ALS) is used for optimizing all the algorithms. "none", "avg", "sqrt" and "log" denotes $f(x) = 1$, $f(x) = \frac{1}{x}$, $f(x) = \frac{1}{\sqrt{x}}$ and $f(x) = \frac{1}{\log x}$, respectively.

Figure 2 summarizes the performance of NDCG@50 when the coefficient of the regularzier varies. The best parameter for different normalization strategies is different from each other. For the "avg" and "sqrt" strategy, α is much larger than the other two strategies and thus much easier to tune. Moreover, these two strategies show better recommendation performance. This indicates these two

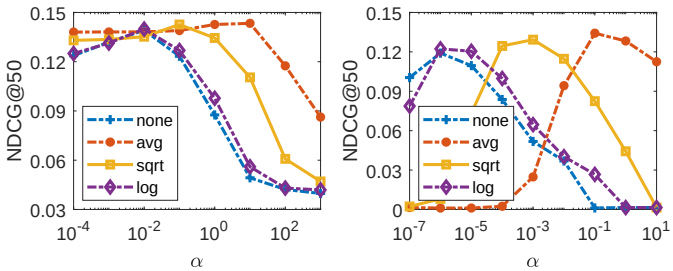
TABLE 2
Comparison with baselines.

Dataset	Alg	NDCG@10	RECALL@10	NDCG@50	RECALL@50	NDCG@200	Recall@200
MovieLens	Square-RIR	0.4242 ± 0.0003	0.2223 ± 0.0003	0.4439 ± 0.0004	0.4817 ± 0.0006	0.5194 ± 0.0003	0.7154 ± 0.0006
	RankALS	0.2337 ± 0.0005	0.1283 ± 0.0005	0.3212 ± 0.0005	0.4155 ± 0.0003	0.4282 ± 0.0005	0.7151 ± 0.0005
	Logit-RIR	0.4193 ± 0.0002	0.2171 ± 0.0002	0.4422 ± 0.0002	0.4813 ± 0.0005	0.5236 ± 0.0002	0.7300 ± 0.0007
	BPR	0.3502 ± 0.0009	0.1797 ± 0.0008	0.3828 ± 0.0009	0.4339 ± 0.0009	0.4710 ± 0.0009	0.6991 ± 0.0003
	WRMF	0.4134 ± 0.0004	0.2133 ± 0.0005	0.4333 ± 0.0003	0.4688 ± 0.0003	0.5124 ± 0.0003	0.7113 ± 0.0005
	SQL_RANK	0.0443 ± 0.0003	0.0190 ± 0.0002	0.0501 ± 0.0003	0.0594 ± 0.0003	0.0737 ± 0.0002	0.1350 ± 0.0004
Yelp	Square-RIR	0.0480 ± 0.0004	0.0560 ± 0.0005	0.0810 ± 0.0004	0.1624 ± 0.0009	0.1252 ± 0.0002	0.3484 ± 0.0007
	RankALS	0.0202 ± 0.0002	0.0256 ± 0.0003	0.0453 ± 0.0002	0.1053 ± 0.0008	0.0897 ± 0.0001	0.2923 ± 0.0008
	Logit-RIR	0.0470 ± 0.0004	0.0541 ± 0.0005	0.0788 ± 0.0004	0.1571 ± 0.0005	0.1227 ± 0.0004	0.3411 ± 0.0008
	BPR	0.0302 ± 0.0005	0.0363 ± 0.0005	0.0555 ± 0.0005	0.1171 ± 0.0011	0.0919 ± 0.0006	0.2708 ± 0.0017
	WRMF	0.0460 ± 0.0004	0.0547 ± 0.0005	0.0794 ± 0.0004	0.1606 ± 0.0006	0.1238 ± 0.0002	0.3471 ± 0.0004
	SQL_RANK	0.0002 ± 0.0000	0.0003 ± 0.0000	0.0011 ± 0.0000	0.0026 ± 0.0001	0.0025 ± 0.0001	0.0086 ± 0.0003
Netflix	Square-RIR	0.3893 ± 0.0002	0.1401 ± 0.0001	0.3728 ± 0.0002	0.3473 ± 0.0004	0.4400 ± 0.0010	0.5994 ± 0.0003
	RankALS	0.1701 ± 0.0003	0.0607 ± 0.0002	0.2205 ± 0.0003	0.2451 ± 0.0004	0.3369 ± 0.0002	0.5677 ± 0.0002
	Logit-RIR	0.3861 ± 0.0002	0.1400 ± 0.0002	0.3730 ± 0.0002	0.3532 ± 0.0002	0.4421 ± 0.0002	0.6107 ± 0.0002
	BPR	0.2998 ± 0.0010	0.1043 ± 0.0004	0.3044 ± 0.0006	0.2959 ± 0.0006	0.3863 ± 0.0007	0.5745 ± 0.0008
	WRMF	0.3841 ± 0.0001	0.1369 ± 0.0001	0.3651 ± 0.0002	0.3378 ± 0.0004	0.4300 ± 0.0002	0.5842 ± 0.0002
	SQL_RANK	0.0120 ± 0.0120	0.0113 ± 0.0028	0.0600 ± 0.0017	0.0564 ± 0.0047	0.0954 ± 0.0048	0.1712 ± 0.0089
CiteULike	Square-RIR	0.1158 ± 0.0020	0.1240 ± 0.0025	0.1805 ± 0.0012	0.3238 ± 0.0004	0.2496 ± 0.0010	0.5797 ± 0.0010
	RankALS	0.0902 ± 0.0011	0.0954 ± 0.0006	0.1564 ± 0.0011	0.2946 ± 0.0015	0.2265 ± 0.0013	0.5566 ± 0.0024
	Logit-RIR	0.1127 ± 0.0022	0.1157 ± 0.0023	0.1712 ± 0.0021	0.3029 ± 0.0034	0.2374 ± 0.0020	0.5506 ± 0.0030
	BPR	0.0900 ± 0.0010	0.0916 ± 0.0010	0.1365 ± 0.0014	0.2418 ± 0.0035	0.1889 ± 0.0015	0.4385 ± 0.0033
	WRMF	0.1107 ± 0.0015	0.1172 ± 0.0008	0.1733 ± 0.0008	0.3077 ± 0.0020	0.2430 ± 0.0007	0.5670 ± 0.0032
	SQL_RANK	0.0017 ± 0.0003	0.0021 ± 0.0004	0.0022 ± 0.0004	0.0041 ± 0.0007	0.0035 ± 0.0004	0.0114 ± 0.0015
Gowalla	Square-RIR	0.0924 ± 0.0004	0.0949 ± 0.0005	0.1341 ± 0.0007	0.2249 ± 0.0015	0.1867 ± 0.0006	0.4306 ± 0.0010
	RankALS	0.0415 ± 0.0009	0.0513 ± 0.0006	0.0835 ± 0.0010	0.1776 ± 0.0012	0.1433 ± 0.0009	0.4115 ± 0.0006
	Logit-RIR	0.0999 ± 0.0005	0.0982 ± 0.0006	0.1408 ± 0.0005	0.2320 ± 0.0005	0.1951 ± 0.0004	0.4458 ± 0.0012
	BPR	0.0860 ± 0.0008	0.0866 ± 0.0008	0.1243 ± 0.0007	0.2103 ± 0.0008	0.1743 ± 0.0009	0.4083 ± 0.0013
	WRMF	0.0898 ± 0.0003	0.0932 ± 0.0004	0.1324 ± 0.0008	0.2240 ± 0.0015	0.1852 ± 0.0007	0.4303 ± 0.0011
	SQL_RANK	0.0003 ± 0.0001	0.0003 ± 0.0000	0.0011 ± 0.0001	0.0026 ± 0.0002	0.0071 ± 0.0000	0.0252 ± 0.0001
LastFM	Square-RIR	0.1724 ± 0.0003	0.1482 ± 0.0002	0.2616 ± 0.0002	0.3466 ± 0.0009	0.3368 ± 0.0003	0.5787 ± 0.0003
	RankALS	0.0530 ± 0.0010	0.0532 ± 0.0009	0.1282 ± 0.0002	0.2215 ± 0.0002	0.2285 ± 0.0001	0.5325 ± 0.0002
	Logit-RIR	0.1685 ± 0.0002	0.1452 ± 0.0010	0.2581 ± 0.0002	0.3450 ± 0.0001	0.3387 ± 0.0003	0.5944 ± 0.0004
	BPR	0.0636 ± 0.0002	0.0469 ± 0.0002	0.0934 ± 0.0003	0.1151 ± 0.0002	0.1356 ± 0.0003	0.2464 ± 0.0004
	WRMF	0.1580 ± 0.0002	0.1400 ± 0.0008	0.2516 ± 0.0002	0.3473 ± 0.0002	0.3302 ± 0.0002	0.5897 ± 0.0003
	SQL_RANK	0.0031 ± 0.0006	0.0041 ± 0.0008	0.0146 ± 0.0022	0.0288 ± 0.0050	0.0200 ± 0.0021	0.0469 ± 0.0048



(a) MovieLens (b) Gowalla

Fig. 1. NDCG@200 and Recall@200 on the MovieLens and Gowalla.



(a) square loss (b) Logit Loss

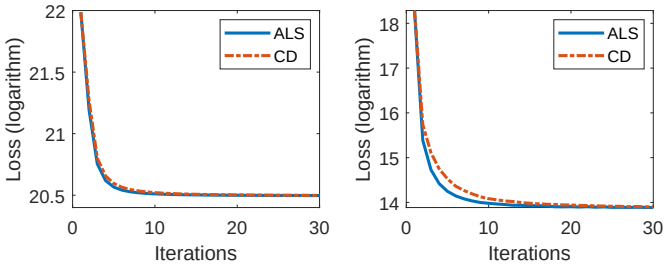
Fig. 2. Sensitivity with respect to the parameter α ($k = 32$ in the CiteULike dataset) with different loss functions and normalization strategies.

strategies strike a better balance between the ranking-based regularizer and the loss function.The "log" strategy is very similar to the "none" strategy in terms of performance and choice of coefficient α. This may be because it excessively shrinks x of $f(x)$ and approximates $f(x) = 1$.

5.2.3 *Effect of optimization methods*

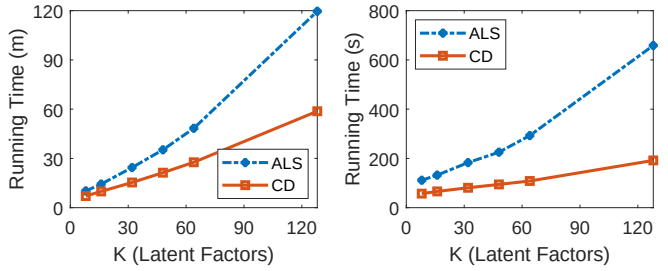
Figure 3 shows how loss values reduce in the CiteULike dataset when the loss functions are optimized with the alternating least square method and the coordinate descent method, respectively. We choose the normalization strategy $f(x) = 1$ and fine-tune the hyperparameters. In the figure, loss values are shown on the y-axis with a logarithmic scale. We observe that all methods converge within 30 iterations.

According to this figure, we can see that the alternating least square method converges in fewer iterations than the coordinate descent method, particularly optimizing the logit loss. This is because the coordinate descent method updates one factor at a time while different factors are correlated with each other. Therefore, the coordinate descent method is much more efficient but requires more iterations for convergence. As analyzed before, the time cost of the coordinate descent method is K times less than the alternating least



(a) Square Loss (b) Logit Loss

Fig. 3. Convergence study of the proposed method on CiteUlike.



(a) MovieLens (b) CiteULike

Fig. 4. The running time of different optimization algorithms with same iterations when K varies in the MovieLens and CiteULike dataset.

square method. This is also evidenced in Figure 4. In addition, an interesting observation is that using the coordinate descent method to optimize the logit loss can converge to a smaller loss than the alternating least square method. A potential explanation is that in the coordinate descent method, the variational parameter $\lambda(\hat{\epsilon}_{uij})$ is recalculated in each iteration, so that the loss function is closer to the upper bound. This is also confirmed by using the coordinate descent method to optimize the square loss function converge to the approximately same loss to the alternating least square method, where there is no variational parameter.

In addition to loss values, we also report the recommendation performance in Table 3, where the square loss and the logit loss are optimized by these two optimization methods. Here, we utilize the "sqrt" normalization strategy and set the dimension to $K = 16$. From this table, we can observe that in the CiteUlike and Gowalla datasets, optimizing both loss functions by the coordinate descent method is slightly better than the alternating least square loss method. In the MovieLens and Yelp datasets, optimization based on the coordinate descent method only leads to improvements for the logit loss but not for the square loss. This is also explained in the coordinate descent method, the variational parameter is recalculated each time, so the loss function is better approximated by its upper bound.

6 Conclusion and Future Work

In this paper, we propose a novel ranking-based implicit regularizer and leverage them to improve the ranking-based one-class collaborative filtering algorithms, whose loss functions include the square loss and the logit loss. We develop an alternating least square method and a coordinate descent method to efficiently optimize the proposed objective functions. We then elaborate on the analysis of training time complexity, showing both of them scale linearly with data size, and the coordinate descent method is K times faster than the alternating least square method in each iteration. Finally, we extensively evaluate the proposed method with 6 real-world datasets and show that the proposed regularizer significantly improves the recommendation performance of the ranking-based one-class collaborative filtering algorithms, and outperforms the completing baselines. We also show that the coordinate descent method is more efficient and even yields slightly better recommendation performance than the alternating least square method, particularly for the logit loss.

In the future, we would like to investigate how to optimize the proposed objective functions with stochastic gradient descent and apply them for advanced neural network recommenders [7]. Moreover, we will also try to relax the hypothesis that prediction scores of uninteracted items should not deviate a lot from each other. For example, we may consider only restricting score proximity among uninteracted items with similar popularity.

TABLE 3
Comparisons between different optimization methods.

Dataset	Loss	Opt	NDCG@50	Recall@50
MovieLens	Square Loss	ALS	0.4202	0.4542
		CD	0.4071	0.4391
	Logit Loss	ALS	0.4066	0.4441
		CD	0.4099	0.4468
Yelp	Square Loss	ALS	0.0665	0.1391
		CD	0.0661	0.1386
	Logit Loss	ALS	0.0681	0.1377
		CD	0.0684	0.1384
CiteULike	Square Loss	ALS	0.1419	0.2588
		CD	0.1447	0.2557
	Logit Loss	ALS	0.1316	0.2349
		CD	0.1352	0.2355
Gowalla	Square Loss	ALS	0.1105	0.1967
		CD	0.1114	0.1976
	Logit Loss	ALS	0.1208	0.2004
		CD	0.1215	0.2079

Acknowledgments

The work was supported by grants from the National Natural Science Foundation of China (Grant No. 61976198, 62022077, 61972069, 61836007, 61832017), the Fundamental Research Funds for the Central Universities, and Sichuan Science and Technology Program under Grant 2020JDTD0007.

References

[1] Y. Hu, Y. Koren, and C. Volinsky, "Collaborative filtering for implicit feedback datasets," in *Proceedings of ICDM'08*. IEEE, 2008, pp. 263–272.

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

[3] S. Rendle, C. Freudenthaler, Z. Gantner, and L. Schmidt-Thieme, "Bpr: Bayesian personalized ranking from implicit feedback," in *Proceedings of UAI'09*. AUAI Press, 2009, pp. 452–461.

[4] R. Pan and M. Scholz, "Mind the gaps: weighting the unknown in large-scale one-class collaborative filtering," in *Proceedings of KDD'09*. ACM, 2009, pp. 667–676.

[5] H.-F. Yu, M. Bilenko, and C.-J. Lin, "Selection of negative samples for one-class matrix factorization," in *Proceedings of SDM'17*. SIAM, 2017, pp. 363–371.

[6] I. Bayer, X. He, B. Kanagal, and S. Rendle, "A generic coordinate descent framework for learning from implicit feedback," in *Proceedings of WWW'17*, 2017, pp. 1341–1350.

[7] W. Krichene, N. Mayoraz, S. Rendle, L. Zhang, X. Yi, L. Hong, E. Chi, and J. Anderson, "Efficient training on very large corpora via gramian estimation," *arXiv preprint arXiv:1807.07187*, 2018.

[8] G. Takács and D. Tikk, "Alternating least squares for personalized ranking," in *Proceedings of RecSys'12*. ACM, 2012, pp. 83–90.

[9] J. Weston, S. Bengio, and N. Usunier, "Large scale image annotation: learning to rank with joint word-image embeddings," *Machine learning*, vol. 81, no. 1, pp. 21–35, 2010.

[10] C.-K. Hsieh, L. Yang, Y. Cui, T.-Y. Lin, S. Belongie, and D. Estrin, "Collaborative metric learning," in *Proceedings of WWW'17*, 2017, pp. 193–201.

[11] J. Chen, D. Lian, and K. Zheng, "Improving one-class collaborative filtering via ranking-based implicit regularizer," in *Proceedings of AAAI'19*, vol. 33, 2019, pp. 37–44.

[12] T. Jaakkola and M. Jordan, "A variational approach to bayesian logistic regression models and their extensions," in *Sixth International Workshop on Artificial Intelligence and Statistics*, vol. 82, 1997.

[13] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl, "Item-based collaborative filtering recommendation algorithms," in *Proceedings of WWW'01*. ACM, 2001, pp. 285–295.

[14] Y. Koren, "Factor in the neighbors: Scalable and accurate collaborative filtering," *ACM Transactions on Knowledge Discovery From Data*, vol. 4, no. 1, pp. 1–24, 2010.

[15] X. Ning and G. Karypis, "Slim: Sparse linear methods for top-n recommender systems," in *Proceedings of ICDM'11*. IEEE, 2011, pp. 497–506.

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

[17] R. Salakhutdinov and A. Mnih, "Bayesian probabilistic matrix factorization using markov chain monte carlo," in *Proceedings of ICML'08*. ACM, 2008, pp. 880–887.

[18] Y. Koren, "Factorization meets the neighborhood: a multifaceted collaborative filtering model," in *Proceedings of KDD'08*. ACM, 2008, pp. 426–434.

[19] S. Rendle, "Factorization machines with libfm," *ACM Trans. Intell. Syst. Tech.*, vol. 3, no. 3, p. 57, 2012.

[20] D. Agarwal and B.-C. Chen, "Regression-based latent factor models," in *Proceedings of KDD'09*. ACM, 2009, pp. 19–28.

[21] R. J. Little and D. B. Rubin, *Statistical analysis with missing data*. John Wiley & Sons, 2019, vol. 793.

[22] H. Steck, "Training and testing of recommender systems on data missing not at random," in *Proceedings of KDD'10*. ACM, 2010, pp. 713–722.

[23] R. Devooght, N. Kourtellis, and A. Mantrach, "Dynamic matrix factorization with priors on unknown values," in *Proceedings of KDD'15*. ACM, 2015, pp. 189–198.

[24] T. Schnabel, A. Swaminathan, A. Singh, N. Chandak, and T. Joachims, "Recommendations as treatments: Debiasing learning and evaluation," in *Proceedings of ICML'16*, 2016, pp. 1670–1679.

[25] X. Wang, R. Zhang, Y. Sun, and J. Qi, "Doubly robust joint learning for recommendation on data missing not at random," in *Proceedings of ICML'19*, 2019, pp. 6638–6647.

[26] D. Lian, C. Zhao, X. Xie, G. Sun, E. Chen, and Y. Rui, "Geomf: joint geographical modeling and matrix factorization for point-of-interest recommendation," in *Proceedings of KDD'14*. ACM, 2014, pp. 831–840.

[27] D. Lian, K. Zheng, Y. Ge, L. Cao, E. Chen, and X. Xie, "Geomf++ scalable location recommendation via joint geographical modeling and matrix factorization," *ACM Transactions on Information Systems*, vol. 36, no. 3, pp. 1–29, 2018.

[28] D. Lian, Y. Ge, F. Zhang, N. J. Yuan, X. Xie, T. Zhou, and Y. Rui, "Scalable content-aware collaborative filtering for location recommendation," *IEEE Transactions on Knowledge and Data Engineering*, 2018.

[29] 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," in *Proceedings of RecSys'12*, 2012, pp. 139–146.

[30] S. Rendle and C. Freudenthaler, "Improving pairwise learning for item recommendation from implicit feedback," in *Proceedings of WSDM'14*. ACM, 2014, pp. 273–282.

[31] W. Zhang, T. Chen, J. Wang, and Y. Yu, "Optimizing top-n collaborative filtering via dynamic negative item sampling," in *Proceedings of SIGIR'13*. ACM, 2013, pp. 785–788.

[32] D. Lian, Q. Liu, and E. Chen, "Personalized ranking with importance sampling," in *Proceedings of The Web Conference 2020*, 2020, pp. 1093–1103.

[33] B. Jin, D. Lian, Z. Liu, Q. Liu, J. Ma, X. Xie, and E. Chen, "Sampling-decomposable generative adversarial recommender," *Advances in Neural Information Processing Systems*, vol. 33, 2020.

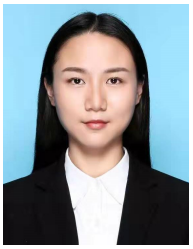
[34] D. Lian, R. Liu, Y. Ge, K. Zheng, X. Xie, and L. Cao, "Discrete content-aware matrix factorization," in *Proceedings of KDD'17*, 2017, pp. 325–334.

[35] C. Wang and D. M. Blei, "Collaborative topic modeling for recommending scientific articles," in *Proceedings of KDD'11*. ACM, 2011, pp. 448–456.

[36] áOscar Celma, *Music Recommendation and Discovery: The Long Tail, Long Fail, and Long Play in the Digital Music Space*. Springer, 2010.

[37] L. Wu, C.-J. Hsieh, and J. Sharpnack, "SQL-rank: A listwise approach to collaborative ranking," in *Proceedings of ICML'18*, vol. 80, 2018, pp. 5315–5324.

Defu Lian is a professor in the School of Computer Science and Technology, University of Science and Technology of China (USTC), Hefei. He received the B.E. and Ph.D. degrees in computer science from University of Science and Technology of China (USTC) in 2009 and 2014, respectively. His research interest includes spatial data mining, recommender systems, and learning to hash.



Jin Chen is a PhD student in University of Electronic Science and Technology of China (UESTC). She received the BE degrees in University of Electronic Science and Technology of China (UESTC). Her main research interest includes recommender system, autoML, and reinforcement learning.

Kai Zheng is a Professor of Computer Science with University of Electronic Science and Technology of China. He received his PhD degree in Computer Science from The University of Queensland in 2012. He has been working in the area of spatial-temporal databases, uncertain databases, social-media analysis, inmemory computing and blockchain technologies. He has published over 140 papers in prestigious journals and conferences in data management field such as SIGMOD, ICDE, VLDB Journal.

Enhong Chen (SM'07) received the PhD degree from USTC. He is a professor and vice dean of the School of Computer Science, USTC. His general area of research includes data mining and machine learning, social network analysis, and recommender systems. He was on program committees of numerous conferences including KDD, ICDM, and SDM. His research is supported by the National Science Foundation for Distinguished Young Scholars of China.

Xiaofang Zhou is a Chair Professor at the Department of Computer Science and Engineering (CSE), Hong Kong University of Science and Technology (HKUST). He received his BSc and MSc degrees from Nanjing University in 1984 and 1987 respectively, and PhD in Computer Science from University of Queensland (UQ) in 1994. His research focus is to find effective and efficient solutions for managing, integrating, and analysing large-scale complex data for business, scientific and personal applications.