Improving the Accuracy of Top-N Recommendation using a Preference Model[☆]

Jongwuk Lee[a], Dongwon Lee[b,], Yeon-Chang Lee[c], Won-Seok Hwang[c], Sang-Wook Kim[c]

[a]*Hankuk University of Foreign Studies, Republic of Korea*
[b]*The Pennsylvania State University, PA, USA*
[c]*Hanyang University, Republic of Korea*

Abstract

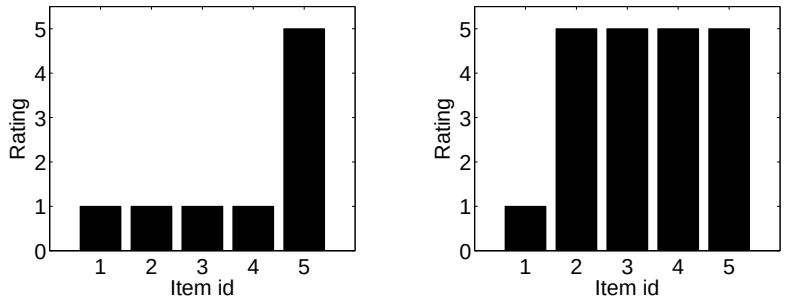
In this paper, we study the problem of retrieving a ranked list of *top-N items* to a target user in recommender systems. We first develop a novel *preference model* by distinguishing different rating patterns of users, and then apply it to existing *collaborative filtering* (*CF*) algorithms. Our preference model, which is inspired by a voting method, is well-suited for representing *qualitative* user preferences. In particular, it can be easily implemented with less than 100 lines of codes on top of existing CF algorithms such as user-based, item-based, and matrix-factorization-based algorithms. When our preference model is combined to three kinds of CF algorithms, experimental results demonstrate that the preference model can improve the accuracy of all existing CF algorithms such as ATOP and NDCG@25 by 3%–24% and 6%–98%, respectively.

1. Introduction

The goal of recommender systems (RS) [1] is to suggest appealing items (*e.g.*, movies, books, news, and music) to a target user by analyzing her prior prefer-

[☆]This work was supported by Hankuk University of Foreign Studies Research Fund of 2014 (Jongwuk Lee). His research was in part supported by NSF CNS-1422215 and Samsung 2015 GRO-175998 awards (Dongwon Lee). This work was partly supported by the National Research Foundation of Korea (NRF) grant funded by the Korean Government (MSIP) (NRF-2014R1A2A1A10054151) and by the Ministry of Science, ICT and Future Planning (MSIP), Korea, under the Information Technology Research Center (ITRC) support program (IITP-2015-H8501-15-1013) (Sang-Wook Kim).

Email addresses: julee@hufs.ac.kr (Jongwuk Lee), dongwon@psu.edu (Dongwon Lee), lyc0324@agape.hanyang.ac.kr (Yeon-Chang Lee), hws23@agape.hanyang.ac.kr (Won-Seok Hwang), wook@agape.hanyang.ac.kr (Sang-Wook Kim)



(a) user u (b) user v

Figure 1: The rating distribution of two users u and v

ences. As the number of applications using RS as a core component has increased rapidly, improving the quality of RS becomes a critically important problem. The quality of RS can be measured by various criteria such as accuracy, serendipity, diversity, relevance, and utility [29, 38]. In measuring the accuracy of RS, in particular, two approaches have prevailed: *rating prediction* and *top-N recommendation* [36]. A typical practice of rating prediction is to minimize prediction errors for all *unobserved* (or *missing*) user-item pairs. In contrast, top-*N* recommendation provides users with the most appealing top-*N* items. In this paper, we mainly focus on improving the accuracy of top-*N* recommendation.

Collaborative filtering (*CF*) [37] has been widely used as one of the most popular techniques in RS. Basically, it is based on users' past behavior such as explicit user ratings and implicit click logs. Using the similarity between users' behavior patterns, CF suggests the most appealing top-*N* items for each user. In this process, existing CF algorithms [6, 10, 11, 16, 30, 31] return top-*N* items with the highest ratings in terms of rating prediction, and ignore the *qualitative* order of items for top-*N* recommendation. Although some recent work [2, 5, 20, 32, 39] developed CF algorithms for optimizing top-*N* recommendation, they still have the limitations in identifying qualitative user preferences, *i.e.*, relative preferences between items with explicit ratings.

Specifically, two key issues in existing top-*N* recommendation are: (1) how to distinguish *user experience* between observed and missing ratings and (2) how to model *latent* user preferences from observed ratings. As the solution to (1), for instance, we may impute missing ratings as *zero* (*i.e.*, *negative user feedback*), as used in [5]. However, as to the solution to (2), it is non-trivial to distinguish relative preferences between observed ratings (*i.e.*, *positive user feedback*) as user ratings often have different distributions per user.

To motivate our preference model, we illustrate a scenario in which two users u and v have rated for the same set of five items $\{i_1, \ldots, i_5\}$ (Figure 1). Even if

two users gave the same rating of 5 to item i_5, the meaning of the rating can be interpreted differently depending on rating distributions. For example, suppose that both u and v gave 5 to i_5. Because v tends to give many ratings of 5, it is difficult to determine how satisfactory v was really with i_5. In contrast, because u rarely gives the rating of 5, it is plausible to assure that u is very satisfied with i_5. Therefore, we argue that u prefers i_5 more than v does.

Our proposed preference model, which is inspired by a *voting method*, aims to discern subtle latent user preferences by considering different rating distributions. While existing algorithms are mainly based on the quantitative values among items to represent user preferences, our model exploits the qualitative order of items. Our preference model is able to convert raw user ratings into latent user preferences. In our formal analysis, it can be derived from *maximum likelihood estimation*, which is consistent with existing work [14].

We then develop a family of CF algorithms by embedding the proposed preference model into existing CF algorithms. As an orthogonal way, our preference model can be easily combined with existing CF algorithms such as user-based neighborhood, item-based neighborhood, and matrix-factorization-based algorithms. Observed user ratings are converted to user preference scores and missing ratings are imputed as zero values. After this conversion is performed, existing CF algorithms are applied with the converted user-item matrix. The key advantages of our proposed algorithms are two-fold: a simple algorithm design and a sizable accuracy improvement in various kinds of CF algorithms.

In order to validate our proposed algorithms, we lastly conduct comprehensive experiments with extensive settings. Because user ratings tend to be biased to item popularity and rating positivity [5, 27, 35], we employ two testing item sets: *All Items* (*i.e.*, all items in a test item set) and *Long-tail Items* (*i.e.*, non-popular items with a few ratings) [5]. In case of *cold-start users* [21] who have rated only a few items, we also compare our proposed algorithms against existing CF algorithms.

To summarize, this paper makes the following contributions.

- We design a preference model in order to identify latent user preferences from user ratings. The idea of the preference model is intuitive to understand, and easy to implement, costing less than 100 lines of codes, on top of existing CF algorithms.

- We propose a family of CF algorithms by incorporating the proposed preference model into existing CF algorithms. Our preference model can be easily applied to existing CF algorithms.

- We evaluate our proposed CF algorithms applied to user-based, item-based, and matrix-factorization-based algorithms. Our proposed algorithms are validated over two test sets: *All Items* and *Long-tail Items*. It is found that

our proposed algorithms improve three kinds of CF algorithms by 3%–24% and 6%–98% for ATOP and NDCG@25.

This paper is organized as follows. In Section 2, we survey existing work for top-*N* recommendation. In Section 3, we discuss two categories of CF algorithms and their variants for top-*N* recommendation. In Section 4, we design a preference model and propose a family of CF algorithms using our preference model. In Section 5, we show detailed evaluation methodology. In Section 6, we report and analyze the experimental results of our proposed algorithms. In Section 7, we finally conclude our work.

2. Related Work

Collaborative filtering (*CF*) [37] is well-known as one of the most prevalent techniques in recommender systems (RS). (The broad survey for RS can be found in [1, 29].) Conceptually, it collects the past behavior of users, and makes rating predictions based on the similarity between users' behavior patterns. The underlying assumptions behind CF are as follows: (1) If users have similar behavior patterns in the past, they will also make similar behavior patterns in the future. (2) The users' behavior patterns are consistent over time. The user behavior is used to infer hidden user preferences and is usually represented by *explicit* (*e.g.*, user ratings) and *implicit* user actions (*e.g.*, clicks and query logs). In this paper, we assume that the user behavior is represented by explicit user ratings on items.

The approaches for CF can be categorized as two models: neighborhood models and latent factor models [37]. First, the neighborhood models make predictions based on the similarity between users or items such as user-based algorithms [10] and item-based algorithms [31]. Second, the latent factor models learn hidden patterns from observed ratings by using matrix factorization techniques [11, 17]. In particular, *singular value decomposition* (*SVD*) is widely used as one of the well-established techniques [13, 15, 16, 30]. In this paper, we mainly handle CF algorithms with explicit user ratings.

Evaluation of top-*N* recommendation: Top-*N* recommendation provides users with a ranked set of N items [6], which is also involved to the "who rated what" problem [4]. To evaluate top-*N* recommendation, we have to take the characteristics of observed ratings into account. That is, ratings are distributed *not at random* [22, 23, 34]. The ratings are biased to *item popularity* [5, 35] and tend to be more preferred than missing ratings, *i.e.*, *rating positivity* [27]. In this paper, we thus employ two testing items such as *All Items* and *Long-tail Items*, and adopt various metrics for fair evaluation of top-*N* recommendation.

One-class collaborative filtering: When user ratings are represented by binary values, observed user ratings are simply set as one (positive feedback). Mean-

while, because missing ratings can be seen as a mixture of unknown and negative feedback, they need to be considered carefully. First, [26] introduced this problem to distinguish different semantics of missing ratings. In addition, [25] proposed an improved weight scheme to discern missing ratings. As the simplest way, we imputed all missing ratings as zero (*i.e.*, negative feedback). We compare our proposed algorithms with the one class collaborative filtering method, where missing values are considered negative feedback with uniform weights. In our future work, we will discuss an alternative method for handling missing ratings, *e.g.*, a imputation method [21] for missing ratings.

Ranking-oriented collaborative filtering: For top-*N* recommendation, it is important to consider the ranking of items. Weimer et al. [39] proposed *CoFiRank* that uses maximum margin matrix factorization to optimize the ranking of items. Liu et al. [20] developed *EigenRank* to decide the rankings of items using neighborhood-based methods and Hu [12] proposed a preference-relation-based similarity measure for multi-criteria dimensions. In addition, [2, 32] combined CF with *learning to rank* methods to optimize the ranking of items. Shi et al. [33] combined rating- and ranking-oriented algorithms with a linear combination function, and Lue et al. [19] extended probabilistic matrix factorization with list-wise preferences. Rendle et al. [28] proposed *Bayesian personalized ranking* that maximizes the likelihood of pair-wise preferences between observed and unobserved items in implicit datasets. In particular, [28] proposed a new objective function that aims to achieve higher accuracy for top-*N* recommendation. Ning and Karypis [24] developed a *sparse linear method* that learns a coefficient matrix of item similarity for top-*N* recommendation. Recently, [38] developed a hybrid approach to combining the content-based the and collaborative filtering method in which the user participates in the feedback process by ranking her interests in the user profiles. Liu et al. [18] adopted *boosted regression trees* to represent conditional user preferences in CF algorithms. In this paper, we propose an alternative user preference model, and combine it with a family of CF algorithms.

3. Collaborative Filtering Algorithms

In this section, we first introduce basic notations used throughout this paper. The domain consists of a set of m users, $\mathcal{U} = \{u_1, \dots, u_m\}$, and a set of n items, $\mathcal{I} = \{i_1, \dots, i_n\}$. All user-item pairs can be represented by an m-by-n matrix $\mathcal{R} = \mathcal{U} \times \mathcal{I}$, where an entry r_{ui} indicates the rating of user u to item i. If r_{ui} has been *observed* (or *known*), it is represented by a positive value in a specific range. Otherwise, r_{ui} is *empty*, implying a *missing* (or *unknown*) rating. Let $\mathcal{R}^+ \subseteq \mathcal{R}$ denote a subset of user-item pairs for which ratings are observed, *i.e.*, $\mathcal{R}^+ = \{r_{ui} \in \mathcal{R} | r_{ui} \text{ is observed}\}$. For the sake of representation, u and v indicate arbitrary users in $\mathcal{U}$, and i and j refer to arbitrary items in $\mathcal{I}$.

In the following, we explain existing CF algorithms. They can be categorized into: (1) *neighborhood-based models* and (2) *latent-factor-based models*.

3.1. Neighborhood Models

The neighborhood models are based on the similarity between either users or items. There are two major neighborhood algorithms, *i.e.*, *user-based* and *item-based* algorithms, depending on which sides are used. The user-based algorithms make predictions based on users' rating patterns similar to that of a target user u. Let $U(u;i)$ be a set of users who rate item i and have similar rating patterns to the target user u. The predicted rating $\hat{r}_{ui}$ of user u to item i is computed by:

$$\hat{r}_{ui} = b_{ui} + \frac{\sum_{v \in U(u;i)} (r_{vi} - b_{vi}) w_{uv}}{\sum_{v \in U(u;i)} w_{uv}} \tag{1}$$

where r_{vi} is a rating of user v to item i, b_{vi} is a biased rating value for v to i, and w_{uv} is the similarity weight between two users u and v. In other words, $\hat{r}_{ui}$ is calculated by a weighted combination of neighbors' residual ratings. Note that b_{ui} and w_{uv} can be different depending on the rating normalization scheme and the similarity weight function.

On the other hand, the item-based algorithms predict the rating for target item i of user u based on the rating patterns between items. Let $I(i;u)$ be a set of items that have rating patterns similar to that of i and have been rated by u. Let w_{ij} denote the similarity weight between two items i and j. Specifically, the predicted rating $\hat{r}_{ui}$ is calculated by:

$$\hat{r}_{ui} = b_{ui} + \frac{\sum_{j \in I(i;u)} (r_{uj} - b_{uj}) w_{ij}}{\sum_{j \in I(i;u)} w_{ij}} \tag{2}$$

We discuss how the neighborhood models can be adapted for top-*N* recommendation. It is observed that predicting the exact rating values is unnecessary for top-*N* recommendation. Instead, it is more important to distinguish the importance of items that are likely to be appealing to the target user. Toward this goal, items are ranked by ignoring the denominator used for rating normalization. By simply removing the denominators in Equations (1) and (2), it can only consider the magnitude of observed ratings. This modification can help discern the *user experience* between observed and missing ratings. Formally, the predicted score is computed by:

$$\hat{r}_{ui} = b_{ui} + \sum_{v \in U(u;i)} (r_{vi} - b_{vi}) w_{uv} \tag{3}$$

Similarly, the predicted score in the item-based algorithms is calculated by:

$$\hat{r}_{ui} = b_{ui} + \sum_{j \in I(i;u)} (r_{uj} - b_{uj}) w_{ij} \tag{4}$$

Note that $\hat{r}_{ui}$ in Equations (3) and (4) means the score for quantifying the importance of item i to user u. For top-*N* recommendation, it is observed that the non-normalized algorithms outperform conventional neighborhood algorithms that minimize prediction errors. The results are also consistently found in existing work [5].

3.2. Latent Factor Models

The latent factor model is an alterative way to infer hidden characteristics of rating patterns by using matrix factorization techniques. In this paper, we adopt *singular value decomposition* (*SVD*) as the well-established matrix factorization method [17]. A key idea of SVD is to factorize an m-by-n matrix $\mathcal{R}$ into an inner product of two low-rank matrices with dimension f. That is, one low-rank matrix is called an m-by-f *user-factor* matrix and the other is called an n-by-f *item-factor* matrix. Each user u is thus associated with an f-dimensional vector $p_u \in \mathbb{R}^f$, and each item i is involved with an f-dimensional vector $q_i \in \mathbb{R}^f$. Formally, the predicted rating $\hat{r}_{ui}$ is calculated by:

$$\hat{r}_{ui} = b_{ui} + p_u q_i^T \tag{5}$$

In this process, a key challenge is how to handle the presence of missing ratings. Recent work adopts an objective function that only minimizes prediction errors for observed ratings with regularization. Formally, the objective function is represented by:

$$\min_{p,q,b} \frac{1}{2} \sum_{r_{ui} \in \mathcal{R}^+} (r_{ui} - b_{ui} - p_u q_i^T)^2 + \frac{\lambda}{2}(||p_u||^2 + ||q_i||^2) \tag{6}$$

where λ is a parameter for regularization. In order to improve the accuracy of rating predictions, other information such as implicit user feedback [13] and temporal effects [15] can be used together in the objective function.

Although the objective function is effective for minimizing prediction errors, it does not obtain high accuracy for top-*N* recommendation [5]. This is because it does not differentiate between observed ratings and missing ratings. As a simplest way, all missing ratings are considered *negative* user feedback, and they are imputed as zero, *i.e.*, $\forall r_{ui} \in \mathcal{R}/\mathcal{R}^+ : r_{ui} = 0$. The imputation for missing ratings enables us to discern user experience between observed ratings and missing ratings. Furthermore, because this modification can form a complete m-by-n matrix $\mathcal{R}$, the conventional SVD method can be applied to $\mathcal{R}$.

$$\mathcal{R} \approx U \Sigma V^T \tag{7}$$

where a low-rank matrix approximation is used with dimension f. That is, U is an n-by-f orthonormal matrix, V is an m-by-f orthonormal matrix, and Σ is an

f-by-f diagonal matrix. The existing work [5] reports that the pure SVD using Equation (7) improves the accuracy of top-*N* recommendation.

4. Proposed Preference Model

Unlike the rating prediction that focuses on minimizing prediction errors of unrated items, top-*N* recommendation aims to identify a sorted list of N items that a user would prefer. The key challenges in top-*N* recommendation are two-fold: (1) how to distinguish the user experience between observed and missing ratings and (2) how to model user preferences for observed ratings. Basically, the observed and missing ratings can be represented as positive/negative user feedback, *i.e.*, 1 if the rating is observed and 0 otherwise. Because the observed ratings are at a different scale, we can exploit them for deciding the relative preferences of items. In the following, we propose a *preference model* in order to address the challenges for top-*N* recommendation.

4.1. Basic Intuition

Our main question for designing a preference model is: *how does it effectively derive latent user preferences from observed ratings?* One simple way is to adopt existing rating normalization schemes. Let $\mathcal{R}_u^+$ denote a set of observed ratings for target user u. For instance, *mean-centered* normalization can be used as one of the popular rating normalization schemes. After the average μ of $\mathcal{R}_u^+$ is computed, each rating r_{ui} in $\mathcal{R}_u^+$ is replaced by $r_{ui} - \mu$. In that case, because normalized ratings with negative values can be confused as negative feedback, it is difficult to distinguish user experience between observed and missing ratings. In addition, when the relative preferences between items can be quantified by computing *absolute distance* between normalized ratings, it does not discern different rating distributions for each user.

As depicted in Figure 1, it is important to distinguish different rating distributions of two users u and v, where they have rated for the same set of five items. When the mean-centered normalization is used, item i_1 rated by 1 can be interpreted as negative feedback. In addition, the difference of normalized ratings between i_1 and i_5 is equal for two users u and v, even if the rating distributions of users are vastly different. This is because the existing normalization schemes are based on *quantitative* ratings instead of *qualitative* ordering of items.

To address this problem, we adopt an alternative preference model, which is inspired by a voting method, *Borda count method* [7]. When items are sorted by the ascending order of ratings, the preference score $pref(r_{ui})$ of item i for target user u is computed by accumulating the number of items that are ranked lower than or equal to item i. Let $pref_{op}(r_{ui})$ denote the number of items in $\mathcal{R}_u^+$ that are satisfied with the condition for op, *i.e.*, $pref_{op}(r_{ui}) = |\{r_{uj} \in \mathcal{R}_u^+ | r_{ui} \ op \ r_{uj}\}|$,

where op can be replaced with any comparison operator, *e.g.*, $>$, $<$, and $=$. Specifically, there are two cases for computing the preference score.

1. Given user rating r_{ui}, we simply count the number of items that are ranked lower than i. Let $pref_{>}(r_{ui})$ is the preference score of r_{ui}, which is the number of items with lower rankings, *i.e.*, $pref_{>}(r_{ui}) = |\{r_{uj} \in \mathcal{R}_u^+ | r_{ui} > r_{uj}\}|$.
2. Let $pref_{=}(r_{ui})$ is the preference score of r_{ui}, which is the number of items with the same rankings, *i.e.*, $pref_{=}(r_{ui}) = |\{r_{uj} \in \mathcal{R}_u^+ | r_{ui} = r_{uj}\}|$.

By combining the two cases, the preference score of r_{ui} can be computed as a weighted linear combination, where $|\mathcal{R}_u^+|$ is used as a normalization parameter.

$$pref(r_{ui}) = \alpha \cdot \frac{pref_{>}(r_{ui})}{|\mathcal{R}_u^+|} + \beta \cdot \frac{pref_{=}(r_{ui})}{|\mathcal{R}_u^+|} \tag{8}$$

where α and β are weight parameters for $pref_{>}(r_{ui})$ and $pref_{=}(r_{ui})$, respectively. Using this equation, we can convert user ratings into qualitative user preference scores depending on different rating distributions.

We now explain how to compute the preference scores of items with categories. When a set of user ratings is aggregated into a set of rating categories, it is easier to handle the set of rating categories with a smaller size, *e.g.*, $r_{ui} \in \{1, \ldots, 5\}$. Let $\{C_1, \ldots, C_k\}$ be a set of rating categories in a specific range. Intuitively, the items in a high rating category are ranked higher than those in a low rating category. Suppose that r_{ui} belongs to category C, *i.e.*, $r_{ui} \in C$. To compute the preference score of r_{ui}, we simply count the number of items that are in low rating category C', *i.e.*, $C > C'$, and the number of items in the same category C. The preference score $pref(C)$ of C is computed as:

$$pref(C) = \alpha \cdot \sum_{C' \in \{C_1, \ldots C_k\}} \frac{pref_{>}(C, C')}{|\mathcal{R}_u^+|} + \beta \cdot \frac{pref_{=}(C)}{|\mathcal{R}_u^+|} \tag{9}$$

where $pref_{>}(C, C')$ is the number of items that belong to lower rating category C' and $pref_{=}(C)$ is the number of items that belong to C. Based on the formal analysis in Section 4.3, parameters α and β are set as 1.0 and 0.5, respectively.

EXAMPLE 1 (QUALITATIVE PREFERENCE). Consider two users u and v who have rated the same set of 10 items. Table 1 describes the observed ratings of two users, where the items are simply numbered in ascending order of ratings. Assuming the rating categories range in $\{1, \ldots, 5\}$, Table 2 shows the preference scores of rating categories for u and v. That is, we count the number of items for each category, and compute the preference score for each category by using Equation (9), where α = 1.0 and β = 0.5. After the preference scores are computed,

Table 1: Observed ratings for users u and v

	i_1	i_2	i_3	i_4	i_5	i_6	i_7	i_8	i_9	i_{10}
u	1	2	3	3	3	3	3	4	5	5
v	2	2	2	3	3	3	4	4	4	5

Table 2: Preference scores for rating categories of users u and v

	C_1	C_2	C_3	C_4	C_5
u	0.05	0.15	0.45	0.75	0.9
v	0.0	0.15	0.45	0.75	0.95

Table 3: Converting ratings to preference scores for users u and v

	i_1	i_2	i_3	i_4	i_5	i_6	i_7	i_8	i_9	i_{10}
u	0.05	0.15	0.45	0.45	0.45	0.45	0.45	0.75	0.9	0.9
v	0.15	0.15	0.15	0.45	0.45	0.45	0.75	0.75	0.75	0.95

user ratings are converted to the corresponding preference scores. Table 3 depicts the preference scores of items for users u and v. When the mean-center scheme is used, both users u and v have the same average value, *i.e.*, 3.2. The relative preference gap between two items i_4 and i_{10} is thus equally computed as 2 for both u and v. In contrast, because our preference model is based on the qualitative order of items, we can identify user preference scores in a more sophisticated manner. While u rated 5 for both i_9 and i_{10}, v rated 5 for only i_{10}. For i_{10}, the preference scores of u and v are thus different, *i.e.*, 0.9 and 0.95. In addition, the preference gap between i_4 and i_{10} is different as well, *i.e.*, 0.45 and 0.5. □

4.2. Applying Our Preference Model

Now, we discuss how to apply our proposed preference model to existing CF algorithms. First, the explicit ratings for each user are converted to corresponding preference scores. Based on this conversion, user-item matrix $\mathcal{R}$ is updated. Let $\mathcal{R}_{pref}$ denote a transformed matrix in which each entry represents a preference score. In this process, all missing ratings are imputed as zero, as used in existing work [5]. Then, any existing CF algorithms can easily be applied to $\mathcal{R}_{pref}$.

We first discuss how the preference model is combined with neighborhood-based algorithms. In computing the prediction score, a key difference is to use $pref(r_{ui})$ instead of $r_{ui} - b_{ui}$ in Equation (3). The predicted score in the user-based algorithms is computed as:

$$\hat{r}_{ui} = \sum_{v \in U(u;i)} pref(r_{vi}) w_{uv} \tag{10}$$

Similarly, the predicted score in the item-based algorithms is calculated as:

$$\hat{r}_{ui} = \sum_{j \in I(i;u)} pref(r_{uj}) w_{ij} \tag{11}$$

In both cases, $\hat{r}_{ui}$ indicates the importance of items for top-*N* recommendation.

Next, we explain how to apply the preference model for latent factor based algorithms. For top-*N* recommendation, ranking-based models (*e.g.*, [39, 32, 2]) employ an objective function that minimizes the error of item ranking. However, because the models are mainly based on the ordering of pair-wise items, they may fail to reflect overall rating distributions. In addition, their computation overhead is prohibitevely high if all pair-wise preferences of items are considered. To address this problem, we first transform $\mathcal{R}$ into $\mathcal{R}_{pref}$ (as done in neighborhood-based algorithms), and then factorize $\mathcal{R}_{pref}$ using the conventional SVD method.

$$\mathcal{R}_{pref} \approx U \Sigma V^T \tag{12}$$

4.3. Formal Analysis

We now derive the preference model by using *maximum likelihood estimation*. For simplicity, suppose that a set of ratings is represented by a rating distribution with categories. Given a set of category $\{C_1, \ldots, C_k\}$, X_i denotes a set of items assigned to the corresponding rating category C_i. All items with ratings are associated with $\{X_1, \ldots, X_k\}$. We estimate a *preference score* θ_i for the corresponding rating category C_i. Let $\theta = \{\theta_1, \ldots, \theta_k\}$ denote probability scores that each category is preferred. When generating a ranked list L of observed items for user preferences, we identify a parameter θ that maximizes the likelihood function.

$$\theta^* = \arg\max_{\theta} p(L|\theta) \tag{13}$$

In that case, $p(L|\theta)$ can be replaced by the product of probability of generating the correct ranked list for any two items. That is, $p(L|\theta)$ is represented by:

$$\begin{aligned} p(L|\theta) \;=\; & \prod_{i>j} \left(\prod_{x \in X_i, x' \in X_j} p(x > x'|\theta) \right) \times \\ & \prod_{i} \left(\prod_{x \neq x' \in X_i} p(x > x'|\theta) \right) \end{aligned} \tag{14}$$

where $p(x > x'|\theta)$ is the probability that item x is preferred to item x', following binary distribution for $p(x)$:

$$p(x > x'|\theta) = p(x)(1 - p(x')) \tag{15}$$

Table 4: Detailed statistics of real-life datasets

	MovieLens 100K	MovieLens 1M
Density	6.30%	4.26%
Min. # of ratings of users	20	20
Max. # of ratings of users	737	2,314
Avg. # of ratings per user	106.04	165.59

where $p(x)$ is the probability that item x is preferred.

As discussed in Section 4.1, a set of user ratings is represented by a set of discrete rating categories $\{C_1, \ldots, C_k\}$. We can replace $p(X)$ with the probability $p(C_i)$ for rating category C_i, and combine Equations (14) with (15). Therefore, $p(L|\theta)$ can be computed by:

$$\begin{aligned} p(L|\theta) &= \prod_{C_i > C_j} \left(p(C_i)^{n(C_i)} (1 - p(C_j))^{n(C_j)} \right) \times \\ & \quad \prod_{C_i} \left(p(C_i)(1 - p(C_i)) \right)^{n(C_i)(n(C_i)-1)/2} \end{aligned} \tag{16}$$

where $n(C_i)$ is the number of items assigned to C_i.

By maximizing the log likelihood of $p(L|\theta)$, we can finally obtain the highest preference probability $\theta_i \in \theta$, which is derived as:

$$p(C_i) = \sum_{i>j} p(C_i > C_j) + \frac{1}{2} \times p(C_i = C_j) \tag{17}$$

where $p(C_i > C_j)$ is the probability that C_i is preferred to C_j, and $p(C_i = C_j)$ is the probability that C_i and C_j are preferred equally. In our experiments, we set α = 1.0 and β = 0.5 in Equation (9). It is found that the derivation for our preference model is consistent with existing work [14].

5. Evaluation Methodology

In this section, we first explain real-life datasets used for evaluation (Section 5.1), and then show various accuracy metrics for top-*N* recommendation (Section 5.2).

5.1. Datasets

We employ two real-life datasets of *MovieLens 100K* and *MovieLens 1M*. These are publicly available at *http://grouplens.org/datasets/movielens*. The MovieLens 100K dataset includes 943 users, 1,682 items, and 100,000 ratings, and the MovieLens 1M dataset includes 6,040 users, 3,952 items, and 1,000,209 ratings.

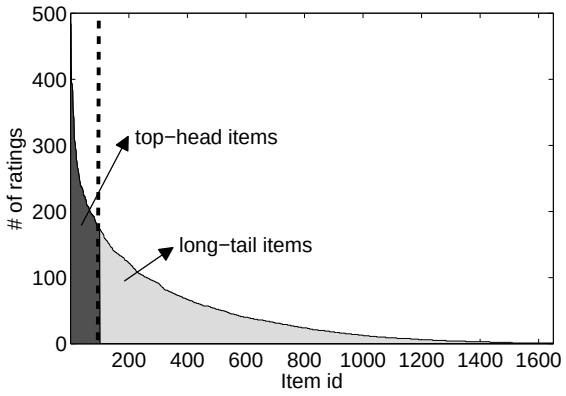


Figure 2: The popularity distribution of items in MovieLens 100K

The ratings take integer values ranging from 1 (worst) to 5 (best). Table 4 summarizes the statistics for the two datasets.

For fair evaluation, we run a 5-fold cross validation. The dataset is randomly split into two subsets, *i.e.*, 80% *training* and 20% *testing* data. Let $Tr \subset \mathcal{R}^+$ and $Te \subset \mathcal{R}^+$ denote training and testing subsets partitioned from $\mathcal{R}^+$, *i.e.*, $Tr \cap Te = \emptyset$, $Tr \cup Te = \mathcal{R}^+$. Each subset is used for executing and evaluating CF algorithms. Specifically, Tr_u and Te_u denote a set of items rated by target user u, *i.e.*, $Tr_u = \{i \in \mathcal{I} | r_{ui} \in Tr\}$ and $Te_u = \{i \in \mathcal{I} | r_{ui} \in Te\}$. In particular, the items with the highest ratings are taken as *relevant* items in Te_u. Let Te_u^+ denote a set of relevant items, *i.e.*, $Te_u^+ = \{i \in \mathcal{I} | r_{ui} \in Te_u, r_{ui} = 5\}$. That is, Te_u^+ is used as *ground truth* for measuring accuracy metrics. Meanwhile, the items with low ratings and unrated items are considered *non-relevant*. In other words, suggesting the most preferred items only is treated as an effective top-*N* recommendation.

In order to evaluate top-*N* recommendation, it is also important to select a set of candidate items that CF algorithms will rank. Let L_u denote a set of candidate items to be ranked for user u. Given L_u, the CF algorithms sort L_u by the descending order of prediction scores and generate a sorted list with top-*N* items. Depending on the selection of L_u, different top-*N* items can be created [3]. For a generalized setting, we form $L_u = \mathcal{I} - Tr_u$ as a set of whole items except for the items that have been rated by user u. In this paper, we call this setting *All Items*.

When using *All Items*, it is observed that the majority of ratings is biased to only a small fraction of popular items. Figure 2 depicts the distribution of items in MovieLens 100K, where Y-axis represents *popularity*, *i.e.*, the number of observed ratings. The items in X-axis are sorted by the descending order of popularity. In Figure 2, the top-100 popular items (*i.e.*, about 6%) have 29,931 ratings (*i.e.*, about 30%). In that case, because the top-*N* items tend to be biased to a few popular items, top-*N* recommendation may not bring much benefit for users.

Note that the popularity bias of rated items can be found in other datasets [5, 27].

To address this problem, we design a more elaborated setting in selecting L_u. Given a set $\mathcal{I}$ of items, it is partitioned into two subsets in terms of popularity. That is, most popular 100 items and remaining items are called *top-head* and *long-tail* items, respectively. Let Top denote a set of top-head items. In that case, we deliberately form L_u as long-tail items, *i.e.*, $L_u = (\mathcal{I} - Top) - Tr_u$ as shown in [5]. We call this setting *Long-tail Items*. The evaluation for long-tail items can be less biased toward the popular items. Compared to *All Items*, it can be a more difficult setting. For extensive evaluation, we will use both settings, *i.e.*, *All Items* and *Long-tail Items*.

Lastly, we evaluate top-*N* recommendation for *cold-start users* [21] who have rated only a few items. Because it is more challenging to identify users' latent preferences using only a small number of ratings, top-*N* recommendation results become less accurate. In this paper, the cold-start user setting is simulated in MovieLens 100K. Specifically, two subsets of users are randomly chosen: 500 users as training users (300, 400, and 500 users, respectively), and 200 users as testing users. For each target user, we vary the number of items rated by a target user from 5, 10 and 20 (Given 5, Given 10 and Given 20, respectively), where the rated items are randomly selected. By default, the testing items are selected from *Long-tail Items*.

5.2. Metrics

There are several accuracy metrics to evaluate top-*N* recommendation. A difficulty for evaluating top-*N* recommendation is that the number of observed ratings is usually small and missing ratings are simply ignored [6]. In addition, because ratings are *not at random* [23, 22, 34], top-*N* results can affect item popularity [5, 35] and rating positivity [27]. To alleviate the problem, we adopt various metrics to complement their drawbacks. For each metric, we report the average of all users who have rated items in Te.

First, we employ traditional accuracy metrics such as *precision* and *recall* used in the IR community. Let N_u denote a sorted list of N items as the result of top-*N* recommendation. The precision and recall at N are computed by:

$$\text{P@}N = \frac{|Te_u^+ \cap N_u|}{N} \tag{18}$$

$$\text{R@}N = \frac{|Te_u^+ \cap N_u|}{|Te_u^+|} \tag{19}$$

where $|Te_u^+|$ is the number of relevant observed items for user u. The two metrics consider the relevance of the top-*N* items, but neglect their ranked positions in N_u.

Second, we employ *normalized discounted cumulative gain* (*NDCG*). The ranked position of items in N_u highly affects NDCG. Let y_k represent a binary variable for the k-th item i_k in N_u, *i.e.*, $y_k \in \{0, 1\}$. If $i_k \in Te_u^+$, y_k is set as 1. Otherwise, y_k is set as 0. In this case, NDCG@N is computed by:

$$\text{NDCG@}N = \frac{\text{DCG@}N}{\text{IDCG@}N} \tag{20}$$

In addition, DCG@N is computed by:

$$\text{DCG@}N = \sum_{k=1}^{N} \frac{2^{y_k} - 1}{\log_2(k+1)} \tag{21}$$

IDCG@N means an *ideal DCG* at N, *i.e.*, for every item i_k in N_u, y_k is set as 1. Note that, because large values of N are useless for top-*N* recommendation, N is set to $\{5, 10, 15, 20, 25\}$ for the three metrics.

Lastly, we employ an alternative measure that reflects all positions of items in L_u, *area under the recall curve*, also known as *ATOP* [34]. Because ATOP is calculated by the average rank of all relevant items in Te_u, it is more effective for capturing the overall distribution for the rankings of relevant items. Let rank(i_k) denote the ranking score of item i_k in L_u. If i_k is ranked highest, rank(i_k) has the largest ranking score, *i.e.*, rank$(i_k) = |L_u|$. Meanwhile, if i_k is ranked lowest, rank(i_k) has the smallest ranking score, *i.e.*, rank$(i_k) = 1$. In that case, rank(i_k) is normalized by Nrank$(i_k) = (\text{rank}(i_k) - 1)/(|L_u| - 1)$. Formally, ATOP is computed by:

$$\text{ATOP} = \frac{1}{|Te_u^+|} \sum_{i_k \in Te_u^+} \text{Nrank}(i_k) \tag{22}$$

Note that all four metrics are maximized by 1.0, *i.e.*, the higher they are, the more accurate top-*N* recommendation is.

6. Experiments

In this section, we first explain state-of-the-art collaborative filtering algorithms to compare our proposed algorithm (Section 6.1). We then report our experimental results (Section 6.2).

6.1. Competing Algorithms

As the baseline method, we first compare the existing decoupled model [14], called Decoupled. Because it has focused on predicting the ratings of items, it is unsuitable for recommending top-*N* items. However, it is effective for evaluating the key differences between Decoupled and our proposed algorithms using

the preference model. We also compare a non-personalized algorithm using *item popularity*, which is served as another baseline method. Specifically, it sorts items by the descending order of popularity, (*i.e.*, the number of ratings) and suggests most popular N items regardless of user ratings, called ItemPopularity. Despite its simple design, it shows even higher accuracy than existing CF algorithms for rating predictions (*i.e.*, UserKNN, ItemKNN, SVD, and SVD++) in *All Items* (as shown in Table 5) [5]. This is because it can directly capture the popularity bias of items.

We then discuss detailed implementations of the CF algorithms using our preference model and existing CF algorithms. Note that our implementations are based on MyMediaLite [9], which is a well-known open-source code for evaluating RS. The key advantage of our proposed method is that existing CF algorithms can be easily extended by incorporating our preference model. We apply our preference model to both neighborhood models and latent factor models.

The neighborhood models can be categorized into two approaches: user-based and item-based algorithms. Broadly, the neighborhood models consist of three key components: (1) rating normalization, (2) similarity weight computation, and (3) neighborhood selection. First, the biased rating scheme [16] is used for rating normalization. Next, in order to quantify the similarity weight, *binary cosine similarity* is used for both user-based and item-based algorithms. Lastly, k-nearest-neighbor filtering is used for neighbor selection in $U(u;i)$ and $I(i;u)$. That is, the neighbors with negative weights are filtered out, and the highest similarity weights are chosen as k neighbors. Empirically, we set $k = 80$ for both user-based and item-based algorithms.

We compare the following user-based neighborhood algorithms:

- UserKNN: user-based neighborhood algorithm [10] using Equation (1)
- NonnormalizedUserKNN: user-based neighborhood algorithm with non-normalized ratings [5] using Equation (3)
- PrefUserKNN: our proposed user-based neighborhood algorithm that incorporates the preference model using Equation (10)

Similarly, we compare the following item-based neighborhood algorithms:

- ItemKNN: item-based neighborhood algorithm [31] using Equation (2)
- NonnormalizedItemKNN: item-based neighborhood algorithm with non-normalized ratings [5] using Equation (4)
- PrefItemKNN: our proposed item-based neighborhood algorithm that incorporates the preference model using Equation (11)

We also explain how to implement latent-factor-based algorithms. Although there are various latent factor models, we focus on evaluating SVD-based algorithms and their variations [8]. The number of latent factors f is fixed as 50. In case of SVD and SVD++, we set a regularized parameter $\lambda = 0.015$. Specifically, we compare the following latent-factor-based algorithms:

- SVD: typical SVD-based algorithm [30] with regularization for the incomplete matrix using Equation (6)
- SVD++: state-of-the-art SVD-based algorithm [16] that shows the highest accuracy in terms of prediction errors, *i.e.*, root mean squired error (RMSE)
- PureSVD: SVD-based algorithm [5] for the complete matrix by filling missing ratings as zero using Equation (7)
- PrefPureSVD: our proposed SVD-based algorithm that incorporates the preference model using Equation (12)

Lastly, we compare our proposed algorithms with other latent-factor-based algorithms with implicit datasets. The ratings used in the algorithms are thus represented by binary data.

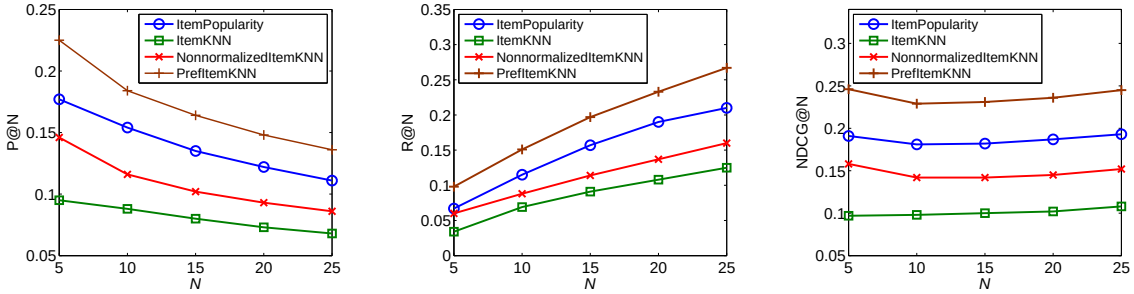
- WRMF: one class collaborative filtering [25, 26], where missing ratings are used as negative feedback with uniform weights
- BPRSLIM: sparse linear method [24] that leverages the objective function as Bayesian personalized ranking [28]
- BPRMF: matrix factorization using stochastic gradient method where the objective function is Bayesian personalized ranking for pair-wise preferences between observed and unobserved items in implicit datasets [28]

All experiments were conducted in Windows 7 running on Intel Core 2.67 GHz CPU with 8GB RAM. All algorithms were implemented with MyMediaLite [9]. For simplicity, the other parameters for each algorithm are set as default values provided in MyMediaLite.

6.2. Experimental Results

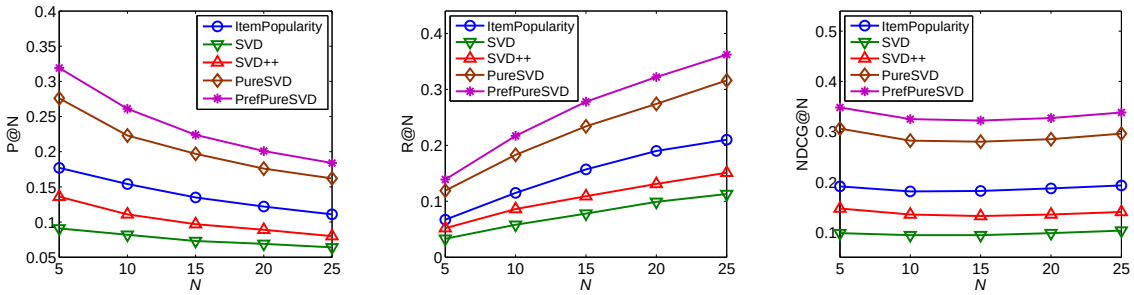
We evaluate our proposed algorithms in extensive experimental settings. Specifically, our empirical study is performed in order to answer the following questions:

1. Do our proposed algorithms using the preference model outperform existing algorithms in both of *All Items* and *Long-tail Items*?



(a) P@N (b) R@N (c) NDCG@N

Figure 3: Comparison results of item-based algorithms in *All Items* over varying N (MovieLens 1M)



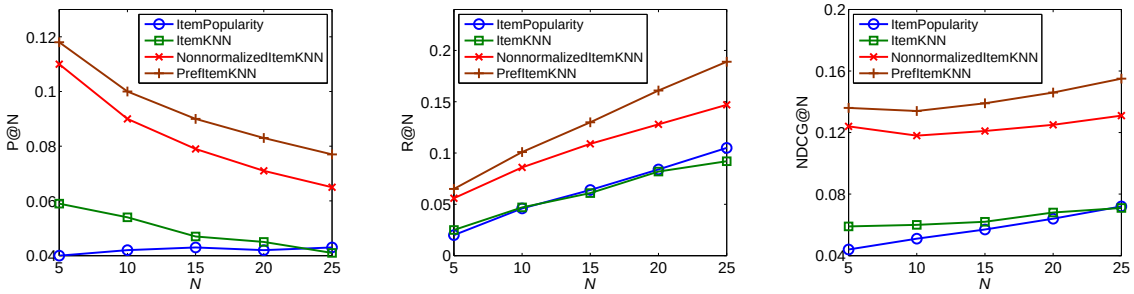
(a) P@N (b) R@N (c) NDCG@N

Figure 4: Comparison results of SVD-based algorithms in *All Items* over varying N (MovieLens 1M)

2. Do our proposed algorithms also outperform the existing algorithms in the cold-start user setting?
3. Which one of our proposed algorithms show the highest accuracy in extensive experimental settings?
4. Do our proposed algorithms show a consistent improvement for the four metrics?

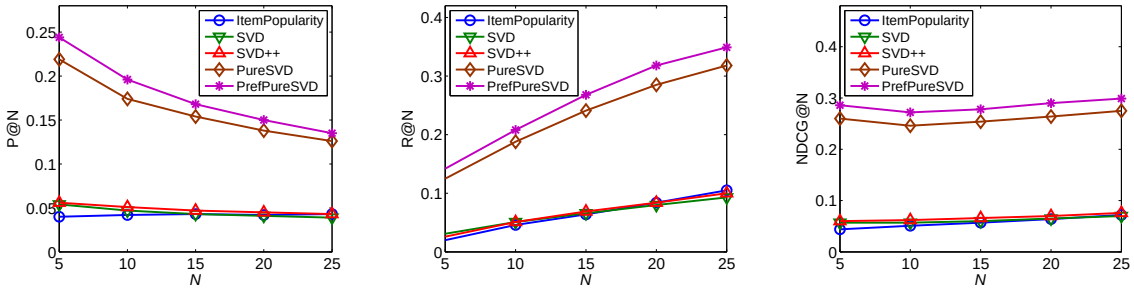
6.2.1. Evaluation for All Items

Figure 3 reports the accuracy of item-based neighborhood algorithms over *All Items*. For all algorithms, P@N decreases with N, R@N increases with N, and NDCG@N tends to stay constant with N. (For simplicity, we later report the experimental results for Decoupled that shows the worst accuracy.) We found three key observations. First, for the three metrics, PrefItemKNN shows the highest accuracy regardless of N. Specifically, PrefItemKNN improves NonnormalizedItemKNN by 0.093 (*i.e.*, 61%) in terms of NDCG@25. Second, NonnormalizedItemKNN shows higher accuracy than ItemKNN as observed in



(a) P@N (b) R@N (c) NDCG@N

Figure 5: Comparison results of item-based algorithms in *Long-tail Items* over varying N (MovieLens 1M)



(a) P@N (b) R@N (c) NDCG@N

Figure 6: Comparison results of SVD-based algorithms in *Long-tail Items* over varying N (MovieLens 1M)

existing work [5]. Third, because of popularity bias of items, ItemPopularity outperforms item-based algorithms for the three metrics. (In the setting of *Long-tail Items*, however, we found an opposite result.) We omit to report the results for user-based algorithms because they show a similar tendency with item-based algorithms.

Figure 4 reports the accuracy of SVD-based algorithms over *All Items*. It is clear that PrefPureSVD outperforms all existing SVD-based algorithms. For NDCG@25, PrefPureSVD improves PureSVD by 0.042 (*i.e.*, 14%). In addition, ItemPopularity shows higher accuracy than SVD and SVD++ for the three metrics. Unlike the item-based algorithms, the normalized variant PureSVD of SVD shows higher accuracy than ItemPopularity.

6.2.2. Evaluation for Long-tail Items

Figure 5 depicts the accuracy comparison of item-based algorithms over *Long-tail Items*. In particular, there are two key observations. First, PrefItemKNN still shows the highest accuracy among all item-based algorithms for the three

metrics. PrefItemKNN apparently outperforms NonnormalizedItemKNN by 0.024 (*i.e.*, 18%) for NDCG@25. Second, NonnormalizedUserKNN outperforms ItemPopularity for the three metrics, unlike the results in *All Items*.

Figure 6 depicts the accuracy of SVD-based algorithms. It is apparent that PrefPureSVD outperforms all existing SVD-based algorithms for the three metrics. Similar to *All Items*, PureSVD is higher than SVD and SVD++ in accuracy. Meanwhile, ItemPopularity and SVD++ are comparable in *Long-tail Items*.

In summary, Tables 5 and 6 report comparison results for all algorithms (including Decoupled, WRMF, BPRSLIM, and BPRMF) in MovieLens 100K and MovieLens 1M, respectively. Note that, in both tables, gray color rows indicate our proposed algorithms and bold font signifies the highest accuracy compared to previous baseline algorithms. Besides, the percent in parentheses is a relative improvement ratio of our proposed algorithms using the preference model, compared to the base CF algorithms: NonnormalizedUserKNN, NonnormalizedItemKNN, and PureSVD. For all metrics, our proposed algorithms outperform the existing algorithms in both settings. In both datasets, our proposed algorithms consistently outperform all existing algorithms. Compared to PrefUserKNN and PrefItemKNN, PrefPureSVD shows the highest accuracy. In addition, PrefPureSVD shows comparable and better accuracy than other latent-factor-based methods using ranking models. In particular, PrefPureSVD always outperforms the existing algorithms in terms of recall@25 and NDCG@25.

6.2.3. Evaluation for Cold-Start Users

Tables 7 and 8 describe comparison results for user-based and SVD-based algorithms for ATOP and NDCG@25. Note that, since the data setting is based on cold-start users, item-based algorithms cannot be applied. (In case of cold-start items, it is found that the item-based algorithms show similar results with user-based algorithms.) As the number of training users and the number of items rated for testing increases, ATOP and NDCG@25 increase. Regardless of parameter settings, our proposed algorithms PrefUserKNN and PrefPureSVD outperform all existing algorithms. PrefUserKNN and PrefPureSVD show comparable results in both metrics. When the number of training users and the number of given ratings is small, PrefUserKNN is slightly better than PrefPureSVD. Meanwhile, as the parameter values increase, PrefPureSVD shows higher accuracy than PrefUserKNN. (Because our proposed algorithms show consistently the highest accuracy, we skip the results for other metrics.) Similar to *All Items* and *Long-tail Items* settings, PrefPureSVD always outperforms existing latent-factor-based algorithms in terms of the ATOP, and it shows a comparable accuracy in terms of the NDCG@25.

Table 5: Evaluation of the accuracy of top-*N* recommendation (MovieLens 100K)

Algorithms	*All Items*			
	ATOP	P@25	R@25	NDCG@25
Decoupled	0.831	0.022	0.151	0.087
ItemPopularity	0.890	0.054	0.280	0.174
UserKNN	0.791	0.006	0.030	0.011
NonnormalizedUserKNN	0.786	0.060	0.280	0.191
PrefUserKNN	0.927 (18%)	0.081 (35%)	0.440 (57%)	0.297 (56%)
ItemKNN	0.812	0.042	0.184	0.106
NonnormalizedItemKNN	0.749	0.046	0.205	0.137
PrefItemKNN	0.925 (24%)	0.077 (67%)	0.411 (101%)	0.272 (98%)
SVD	0.752	0.037	0.166	0.102
SVD++	0.803	0.046	0.215	0.124
PureSVD	0.888	0.078	0.425	0.272
PrefPureSVD	**0.930 (5%)**	**0.097 (24%)**	**0.500 (18%)**	**0.339 (25%)**
WRMF	0.942	0.092	0.490	0.320
BPRSLIM	0.915	0.066	0.383	0.238
BPRMF	0.942	0.083	0.449	0.282

Algorithms	*Long-tail Items*			
	ATOP	P@25	R@25	NDCG@25
Decoupled	0.785	0.007	0.066	0.033
ItemPopularity	0.855	0.022	0.145	0.069
UserKNN	0.752	0.002	0.014	0.004
NonnormalizedUserKNN	0.761	0.031	0.203	0.113
PrefUserKNN	0.904 (19%)	0.044 (42%)	0.334 (65%)	0.188(66%)
ItemKNN	0.774	0.025	0.158	0.081
NonnormalizedItemKNN	0.760	0.031	0.201	0.120
PrefItemKNN	0.900 (18%)	0.041 (32%)	0.309 (54%)	0.175(46%)
SVD	0.727	0.022	0.142	0.074
SVD++	0.750	0.026	0.169	0.087
PureSVD	0.879	0.053	0.394	0.226
PrefPureSVD	**0.908 (3%)**	**0.059 (11%)**	**0.424 (8%)**	**0.245 (8%)**
WRMF	0.924	0.057	0.418	0.243
BPRSLIM	0.889	0.039	0.312	0.166
BPRMF	0.924	0.050	0.368	0.193

Table 6: Evaluation of the accuracy of top-N recommendation (MovieLens 1M)

Algorithms	*All Items*			
	ATOP	P@25	R@25	NDCG@25
Decoupled	0.836	0.052	0.092	0.080
ItemPopularity	0.898	0.111	0.210	0.193
UserKNN	0.839	0.030	0.058	0.033
NonnormalizedUserKNN	0.802	0.099	0.187	0.180
PrefUserKNN	0.929 (16%)	0.139 (40%)	0.287 (54%)	0.269 (50%)
ItemKNN	0.845	0.068	0.125	0.108
NonnormalizedItemKNN	0.756	0.086	0.160	0.152
PrefItemKNN	0.919 (22%)	0.136 (58%)	0.267 (67%)	0.245 (61%)
SVD	0.818	0.064	0.113	0.103
SVD++	0.842	0.080	0.151	0.140
PureSVD	0.869	0.162	0.316	0.296
PrefPureSVD	**0.931 (7%)**	**0.184 (14%)**	**0.362 (15%)**	**0.338 (14%)**
WRMF	0.940	0.201	0.319	0.324
BPRSLIM	0.909	0.136	0.229	0.224
BPRMF	0.937	0.185	0.288	0.296

Algorithms	*Long-tail Items*			
	ATOP	P@25	R@25	NDCG@25
Decoupled	0.798	0.021	0.052	0.037
ItemPopularity	0.867	0.043	0.105	0.072
UserKNN	0.800	0.009	0.019	0.011
NonnormalizedUserKNN	0.761	0.052	0.122	0.098
PrefUserKNN	0.912 (20%)	0.079 (52%)	0.200 (64%)	0.165 (68%)
ItemKNN	0.813	0.041	0.092	0.071
NonnormalizedItemKNN	0.767	0.065	0.147	0.131
PrefItemKNN	0.900 (17%)	0.077 (19%)	0.189 (29%)	0.155(18%)
SVD	0.790	0.039	0.093	0.070
SVD++	0.796	0.043	0.100	0.076
PureSVD	0.899	0.126	0.318	0.275
PrefPureSVD	**0.929 (3%)**	**0.135 (7%)**	**0.349 (10%)**	**0.299 (9%)**
WRMF	0.929	0.134	0.251	0.230
BPRSLIM	0.889	0.089	0.181	0.157
BPRMF	0.923	0.117	0.219	0.197

Table 7: ATOP comparison for cold-start user evaluation (MovieLens 100K)

Algorithms	# of training users = 300		
	Given 5	Given 10	Given 20
Decoupled	0.755	0.761	0.765
ItemPopularity	0.826	0.834	0.846
UserKNN	0.714	0.717	0.728
NonnormalizedUserKNN	0.734	0.738	0.745
PrefUserKNN	**0.872 (19%)**	**0.879 (19%)**	0.888 (19%)
SVD	0.61	0.645	0.684
SVD++	0.721	0.725	0.73
PureSVD	0.723	0.772	0.792
PrefPureSVD	0.854 (18%)	0.878 (14%)	**0.890 (12%)**
WRMF	0.810	0.847	0.863
BPRSLIM	0.705	0.766	0.807
BPRMF	0.814	0.831	0.847

Algorithms	# of training users = 400		
	Given 5	Given 10	Given 20
Decoupled	0.760	0.759	0.770
ItemPopularity	0.831	0.837	0.850
UserKNN	0.722	0.725	0.736
NonnormalizedUserKNN	0.744	0.745	0.756
PrefUserKNN	**0.877 (18%)**	0.882 (18%)	0.891 (18%)
SVD	0.59	0.605	0.64
SVD++	0.7	0.697	0.7
PureSVD	0.74	0.782	0.817
PrefPureSVD	0.863 (17%)	**0.884 (13%)**	**0.902 (10%)**
WRMF	0.822	0.835	0.881
BPRSLIM	0.722	0.777	0.826
BPRMF	0.831	0.844	0.857

Algorithms	# of training users = 500		
	Given 5	Given 10	Given 20
Decoupled	0.769	0.767	0.781
ItemPopularity	0.834	0.838	0.852
UserKNN	0.736	0.738	0.749
NonnormalizedUserKNN	0.750	0.754	0.765
PrefUserKNN	**0.880 (17%)**	0.885 (17%)	0.895 (17%)
SVD	0.572	0.617	0.634
SVD++	0.727	0.724	0.734
PureSVD	0.734	0.782	0.822
PrefPureSVD	0.872 (19%)	**0.893 (14%)**	**0.914 (11%)**
WRMF	0.808	0.841	0.878
BPRSLIM	0.722	0.784	0.832
BPRMF	0.842	0.857	0.879

Table 8: NDCG@25 comparison for cold-start user evaluation (MovieLens 100K)

Algorithms	# of training users = 300		
	Given 5	Given 10	Given 20
Decoupled	0.042	0.041	0.032
ItemPopularity	0.061	0.061	0.066
UserKNN	0.004	0.003	0.002
NonnormalizedUserKNN	0.08	0.091	0.086
PrefUserKNN	**0.142 (78%)**	0.149 (64%)	0.149 (73%)
SVD	0.045	0.053	0.065
SVD++	0.073	0.076	0.069
PureSVD	0.079	0.114	0.145
PrefPureSVD	**0.142 (80%)**	**0.151 (33%)**	**0.168 (16%)**
WRMF	0.133	0.147	0.171
BPRSLIM	0.065	0.099	0.109
BPRMF	0.090	0.102	0.114

Algorithms	# of training users = 400		
	Given 5	Given 10	Given 20
Decoupled	0.032	0.032	0.030
ItemPopularity	0.066	0.069	0.074
UserKNN	0.003	0.003	0.002
NonnormalizedUserKNN	0.094	0.099	0.100
PrefUserKNN	**0.155 (65%)**	**0.159 (61%)**	0.153 (53%)
SVD	0.033	0.040	0.050
SVD++	0.070	0.068	0.070
PureSVD	0.083	0.114	0.157
PrefPureSVD	0.143 (72%)	0.153 (34%)	**0.167 (6%)**
WRMF	0.136	0.160	0.178
BPRSLIM	0.074	0.106	0.125
BPRMF	0.089	0.105	0.123

Algorithms	# of training users = 500		
	Given 5	Given 10	Given 20
Decoupled	0.028	0.028	0.023
ItemPopularity	0.068	0.062	0.070
UserKNN	0.004	0.004	0.004
NonnormalizedUserKNN	0.094	0.097	0.102
PrefUserKNN	**0.152 (62%)**	**0.154 (59%)**	0.160 (57%)
SVD	0.029	0.036	0.043
SVD++	0.082	0.080	0.080
PureSVD	0.078	0.109	0.148
PrefPureSVD	0.140 (80%)	0.148 (36%)	**0.173 (17%)**
WRMF	0.130	0.151	0.174
BPRSLIM	0.067	0.095	0.125
BPRMF	0.097	0.100	0.124

7. Conclusion

In this paper, we have studied how to improve the accuracy of top-N recommendation. To address this problem, two key challenges arise: (1) how to distinguish user experience between observed and missing ratings and (2) how to infer latent user preference for observed ratings. We first designed a preference model based on the qualitative order of items. We then proposed a family of CF algorithms that combine the preference model with existing CF algorithms: user-based neighborhood, item-based neighborhood, and matrix-factorization-based algorithms. The empirical study showed that our proposed algorithms improved the existing algorithms by 3%–24%, 7%–67%, 8%–101%, and 6%–98% for ATOP, P@25, R@25, and NDCG@25, respectively.

In future work, we plan to consider two directions to further improve the accuracy of our proposed algorithms. First, we considered all missing ratings as negative feedback. Inspired by one class collaborative filtering [25, 26], we plan to distinguish the weight of missing ratings based on the confidence of negative feedback. We hope that such a fine-grained distinction scheme for negative feedback can contribute to the improvement of accuracy. Second, we plan to combine our proposed preference model with a pair-wise ranking model of items in Bayesian personalized ranking [28]. We expect that such an idea can represent relative preferences more effectively.

References

[1] Gediminas Adomavicius and Alexander Tuzhilin. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE Trans. Knowl. Data Eng.*, 17(6):734–749, 2005.

[2] Suhrid Balakrishnan and Sumit Chopra. Collaborative ranking. In *WSDM*, pages 143–152, 2012.

[3] Alejandro Bellogín, Pablo Castells, and Iván Cantador. Precision-oriented evaluation of recommender systems: an algorithmic comparison. In *RecSys*, pages 333–336, 2011.

[4] András A. Benczúr and Miklós Kurucz. Who rated what? a recommender system benchmark winner report. *ERCIM News*, 2008(73), 2008.

[5] Paolo Cremonesi, Yehuda Koren, and Roberto Turrin. Performance of recommender algorithms on top-N recommendation tasks. In *RecSys*, pages 39–46, 2010.

[6] Mukund Deshpande and George Karypis. Item-based top-N recommendation algorithms. *ACM Trans. Inf. Syst.*, 22(1):143–177, 2004.

[7] Peter Emerson. The original borda count and partial voting. *Social Choice and Welfare*, 40(2):353–358, 2013.

[8] Simon Funk. Netflix update: Try this at home, 2006.

[9] Zeno Gantner, Steffen Rendle, Christoph Freudenthaler, and Lars Schmidt-Thieme. Mymedialite: a free recommender system library. In *RecSys*, pages 305–308, 2011.

[10] Jonathan L. Herlocker, Joseph A. Konstan, Al Borchers, and John Riedl. An algorithmic framework for performing collaborative filtering. In *SIGIR*, pages 230–237, 1999.

[11] Thomas Hofmann. Latent semantic models for collaborative filtering. *ACM Trans. Inf. Syst.*, 22(1):89–115, 2004.

[12] Yi-Chung Hu. Recommendation using neighborhood methods with preference-relation-based similarity. *Inf. Sci.*, 284:18–30, 2014.

[13] Yifan Hu, Yehuda Koren, and Chris Volinsky. Collaborative filtering for implicit feedback datasets. In *ICDM*, pages 263–272, 2008.

[14] Rong Jin, Luo Si, ChengXiang Zhai, and James P. Callan. Collaborative filtering with decoupled models for preferences and ratings. In *CIKM*, pages 309–316, 2003.

[15] Yehuda Koren. Collaborative filtering with temporal dynamics. In *KDD*, pages 447–456, 2009.

[16] Yehuda Koren. Factor in the neighbors: Scalable and accurate collaborative filtering. *TKDD*, 4(1), 2010.

[17] Yehuda Koren, Robert M. Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *IEEE Computer*, 42(8):30–37, 2009.

[18] Juntao Liu, Chenhong Sui, Dewei Deng, Junwei Wang, Bin Feng, Wenyu Liu, and Caihua Wu. Representing conditional preference by boosted regression trees for recommendation. *Inf. Sci.*, 327:1–20, 2016.

[19] Juntao Liu, Caihua Wu, Yi Xiong, and Wenyu Liu. List-wise probabilistic matrix factorization for recommendation. *Inf. Sci.*, 278:434–447, 2014.

[20] Nathan Nan Liu and Qiang Yang. Eigenrank: a ranking-oriented approach to collaborative filtering. In *SIGIR*, pages 83–90, 2008.

[21] Hao Ma, Irwin King, and Michael R. Lyu. Effective missing data prediction for collaborative filtering. In *SIGIR*, pages 39–46, 2007.

[22] Benjamin M. Marlin and Richard S. Zemel. Collaborative prediction and ranking with non-random missing data. In *RecSys*, pages 5–12, 2009.

[23] Benjamin M. Marlin, Richard S. Zemel, Sam T. Roweis, and Malcolm Slaney. Collaborative filtering and the missing at random assumption. In *UAI*, pages 267–275, 2007.

[24] Xia Ning and George Karypis. SLIM: sparse linear methods for top-n recommender systems. In *ICDM*, pages 497–506, 2011.

[25] Rong Pan and Martin Scholz. Mind the gaps: weighting the unknown in large-scale one-class collaborative filtering. In *KDD*, pages 667–676, 2009.

[26] Rong Pan, Yunhong Zhou, Bin Cao, Nathan Nan Liu, Rajan M. Lukose, Martin Scholz, and Qiang Yang. One-class collaborative filtering. In *ICDM*, pages 502–511, 2008.

[27] Bruno Pradel, Nicolas Usunier, and Patrick Gallinari. Ranking with non-random missing ratings: influence of popularity and positivity on evaluation metrics. In *RecSys*, pages 147–154, 2012.

[28] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. BPR: bayesian personalized ranking from implicit feedback. In *UAI*, pages 452–461, 2009.

[29] Francesco Ricci, Lior Rokach, Bracha Shapira, and Paul B. Kantor. *Recommender Systems Handbook*. Springer, 2011.

[30] Ruslan Salakhutdinov and Andriy Mnih. Probabilistic matrix factorization. In *NIPS*, pages 1257–1264, 2007.

[31] Badrul M. Sarwar, George Karypis, Joseph A. Konstan, and John Riedl. Item-based collaborative filtering recommendation algorithms. In *WWW*, pages 285–295, 2001.

[32] Yue Shi, Martha Larson, and Alan Hanjalic. List-wise learning to rank with matrix factorization for collaborative filtering. In *RecSys*, pages 269–272, 2010.

[33] Yue Shi, Martha Larson, and Alan Hanjalic. Unifying rating-oriented and ranking-oriented collaborative filtering for improved recommendation. *Inf. Sci.*, 229:29–39, 2013.

[34] Harald Steck. Training and testing of recommender systems on data missing not at random. In *KDD*, pages 713–722, 2010.

[35] Harald Steck. Item popularity and recommendation accuracy. In *RecSys*, pages 125–132, 2011.

[36] Harald Steck. Evaluation of recommendations: rating-prediction and ranking. In *RecSys*, pages 213–220, 2013.

[37] Xiaoyuan Su and Taghi M. Khoshgoftaar. A survey of collaborative filtering techniques. *Adv. Artificial Intellegence*, 2009.

[38] Álvaro Tejeda-Lorente, Carlos Porcel, Eduardo Peis, Rosa Sanz, and Enrique Herrera-Viedma. A quality based recommender system to disseminate information in a university digital library. *Inf. Sci.*, 261:52–69, 2014.

[39] Markus Weimer, Alexandros Karatzoglou, Quoc V. Le, and Alexander J. Smola. COFI RANK - maximum margin matrix factorization for collaborative ranking. In *NIPS*, pages 1593–1600, 2007.