Recommending User Generated Item Lists

Yidan Liu
Dept. of Computer Science,
Univ. of British Columbia
yidanliu@cs.ubc.ca

Min Xie
Dept. of Computer Science,
Univ. of British Columbia
minxie@cs.ubc.ca

Laks V.S. Lakshmanan
Dept. of Computer Science,
Univ. of British Columbia
laks@cs.ubc.ca

ABSTRACT

Existing recommender systems mostly focus on recommending individual items which users may be interested in. User-generated item lists on the other hand have become a popular feature in many applications. E.g., Goodreads provides users with an interface for creating and sharing interesting book lists. These user-generated item lists complement the main functionality of the corresponding application, and intuitively become an alternative way for users to browse and discover interesting items to be consumed. Unfortunately, existing recommender systems are not designed for recommending user-generated item lists. In this work, we study properties of these user-generated item lists and propose a Bayesian ranking model, called LIRE for recommending them. The proposed model takes into consideration users' previous interactions with both item lists and with individual items. Furthermore, we propose in LIRE a novel way of weighting items within item lists based on both position of items, and personalized list consumption pattern. Through extensive experiments on a real item list dataset from Goodreads, we demonstrate the effectiveness of our proposed LIRE model.

Categories and Subject Descriptors

H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval - Information Filtering

Keywords

List Recommendation; Collaborative Filtering

1. INTRODUCTION

Recommender systems have become extremely popular because of their wide application and success in domains such as E-commerce (Amazon), Music (iTunes), Movies (Netflix), and Apps (Apple App Store). E.g., back in 2006, Amazon already had 35% of purchases originating from recommended items [21].

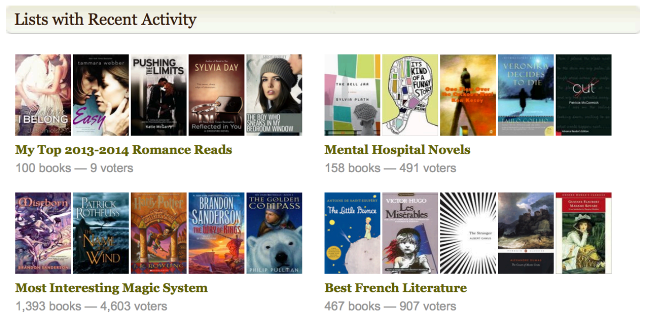
Lists with Recent Activity
My Top 2013-2014 Romance Reads
100 books — 9 voters
Mental Hospital Novels
158 books — 491 voters
Most Interesting Magic System
1,393 books — 4,603 voters
Best French Literature
467 books — 907 voters

Figure 1: User-generated book lists on GoodReads.

Existing recommender systems usually help users find potentially interesting items by means of *collaborative filtering*. Initial works on collaborative filtering mainly focused on *neighborhood-based methods* [29, 22], which essentially leverage similarity between items or between users as measured by heuristic functions such as *cosine* or *Pearson correlation*. These similarities are calculated based on the ratings provided by (received by) users (resp., items) in the past. More recent works on collaborative filtering mostly focus on *latent factor models*, which have become widely known from their success at the *Netflix Competition* [12, 17, 19, 20].

Although existing collaborative filtering works are effective at recommending individual items to users, the proposed solutions usually lack flexibility in supporting more complex recommendation scenarios [4]. For example, existing recommendation algorithms are not optimized for recommending *user-generated item lists*, which are very popular among various applications such as lists of books created by readers on GoodReads [2], lists of twitter users created by users on Twitter [1], and lists of products created by shoppers on Amazon [3].

User-generated item lists are usually deployed as a way to help users organize and share items consumed in the corresponding application. These user-generated item lists are usually made public by default, so in addition to the main functionality provided by the service (e.g., recommending as well as shopping or consuming items), these lists serve as an alternative way to engage users of the service. For users, exploring user-generated item lists is often a very effective way for browsing and identifying interesting items, as each list contains only a small subset of all items, and items in each list are usually manually organized by other users around a specific theme. E.g., users can find from a book list on French Literature interesting and rare books on this topic, which may not be recognized by the item recommendation engine or through keyword search interface as these books may not explicitly contain the keywords "French Literature". An example of user-generated book lists on GoodReads is shown in Figure 1.

However, because of the sheer volume of user-generated item lists, and the complexity of each list (which typically contains tens or even hundreds of items), it is extremely challenging for a standard item presentation interface to help users discover lists which he/she might find interesting. Thus users usually have to resort to keyword search to find interesting lists, which again is extremely challenging because of the large number of items within each list, and also the huge number of lists which might match the query.

In this work, motivated by recent study on automatic recommendation of items based on user's previous interaction with the underlying system, we study the problem of how to generalize item recommendation to incorporate lists, and propose algorithms which can automatically find relevant *item lists* for each user based on her/his previous interactions with both lists and items in the corresponding service. Intuitively, these recommended item lists complements existing item recommender system and serve as a way for users to explore items around a small set of themes which might match users' interest.

We note that in the domain of music, playlists have a similar structure as the user-generated item lists studied in this work, and several existing works have been devoted to developing recommendation algorithms for playlists [7, 9, 8]. However, a huge difference between playlists and the item lists studied in this work is that there usually exists a strong correlation between neighboring songs in a playlist: in most cases songs in a playlist have to be consumed sequentially. This property has been explored extensively by existing works such as [9], [7]. However, it is clear that for the item lists under our consideration, such as book lists, twitter user lists, and shopping lists, items do not necessarily have to be consumed sequentially. In fact, a user might well be interested in only one or a few items in a list and may consume them in no particular order.

To address the limitations of previous approaches in modeling users' item list preference, we propose a new *List Recommendation Model* called LIRE in this work. In LIRE , we take a user's item preference into consideration when modeling the user's item list preference, where item preferences can be learned from user's previous interaction with individual items. We note that a simple way of aggregating item preferences into list preferences may not work very well. Indeed, one challenge in modeling a user's interest in user-generated item lists is that on websites such as Goodreads, because of the way user interface is presented and the huge number items contained in each list, users usually see just the top items in a list first, and they may often stop browsing an item list when enough items have been explored or consumed. To take these facts into consideration, we consider in our model, ingredients which can be leveraged to weight items which are ranked at different positions, and we also learn users' browsing patterns by fitting a parameter that indicates roughly how many items need to be consumed before a user stops browsing an item list.

We make the following contributions: First, we propose a Bayesian-based ranking model LIRE for recommending user-generated item lists, which, unlike playlists, need not be consumed sequentially; Second, in the LIRE model, we consider the problem of how preferences over individual items can be properly aggregated to model user's interest in an item list; Third, through extensive experiments on a real item list dataset obtained from Goodreads, we show that our LIRE model significantly outperforms various previously proposed recommendation algorithms on the task of recommending user-generated item lists.

In the rest of this paper, we first define the item list recommendation problem in Section 2.1. Then in Section 2.2, we present some intuitive baseline algorithms which can be applied to the item list dataset to generate list recommendations. In Section 3, we introduce our proposed list recommendation model. Experimental results comparing different algorithms on Goodreads dataset are presented in Section 4. Related work is discussed in Section 5. Finally, we conclude the paper and present possible future directions in Section 6.

2. PROBLEM STUDIED AND BASELINE ALGORITHMS

2.1 Item List Recommendation

To formulate the item list recommendation problem, consider a set of users $U = \{u_1, \ldots, u_m\}$, a set of items $T = \{t_1, \ldots, t_n\}$, and a set of item lists $L = \{l_1, \ldots, l_q\}$, where each list $l_i \in L$ is composed of a subset of items from T.

Let the set of item lists which user u has shown interest in be denoted as L_u, $L_u \subseteq L$, where a list $l \in L_u$ can be a list which u has liked, voted, or commented on. Similar to many other recommendation problems such as those discussed in [12] and [27], typically user feedback on item lists is *implicit* as compared to explicit ratings which can be found on movie recommendation websites [20].

On most websites such as Goodreads which provide the functionality for creating and sharing item lists, each user u is also associated with a set of items T_u, namely the items which u has previously consumed/rated.

PROBLEM 1. **Item List Recommendation:** *Given T, U, L, for each user $u \in U$, based on u's previously consumed items T_u, and u's previously consumed item lists L_u, recommend to u the top-N item lists from $L \backslash L_u$ which u will be most interested in.*

2.2 Baseline Algorithms

2.2.1 Popularity-based Approach

For many a recommendation application be it item recommendations or item list recommendations, there is typically a long tail distribution w.r.t. the number of user ratings/interactions of each item. E.g., most of the items can observe only a few ratings, whereas only a few enjoy much attention from users. Thus this popularity bias becomes an important factor for explaining and understanding the underlying data, as was also observed in the Netflix competition [18].

One simple item list recommendation algorithm thus is to directly leverage the popularity bias and recommend item lists based on the popularity of each list. That is, we sort item lists w.r.t. the number of votes received, and recommend to every user the same set of top item lists which have received the highest number of votes. We note that this is very similar to how Goodreads website recommends item lists on every book detail page, i.e., every popular book is likely included in many book lists, and according to our observation, only two of the hottest book lists are shown on the detail page of every book.

We call above algorithm GLB (GLoBal), as the recommendation generated using the above approach is based on global statistics and is not personalized. We also consider the following two different personalized variants, among the baselines; these methods are in part motivated by the heuristic popularity-based approaches for generating playlists which have been demonstrated to have good performance [8].

The first alternative PPOP (Personalized POPularity) is based on the intuition that a user u may only be interested in item lists which contain items that u is familiar with. Thus instead of recommending the most popular item lists across all users, we recommend the most popular item lists which contain at least one item the user has interacted with before.

The second alternative PIF (Personalized Item Frequency) is based on the same intuition as PPOP that a user u is only interested in item lists which contain items that u is familiar with. But instead of ranking these candidate item lists by popularity, we rank these candidate item lists by how many items the list contains that the user has interacted before. The rationale for this heuristic is that the more books a list contains that the user is familiar with, the more interesting that list to the user. Ties in PIF are broken using popularity of the corresponding item list.

2.2.2 Collaborative Filtering for Implicit Data

Consider a matrix M^L of users' interest in each list, where each entry y_{ul} is 1 if u has voted list l before, and 0 otherwise. We could simply apply previously proposed collaborative filtering algorithms to this matrix M^L.

Typical collaborative filtering algorithms are based on latent factor models, which were made popular because of their success in the Netflix competition [20]. The nature of the datasets we consider is that user feedback is implicit, i.e., it tends to be in the form of votes. Thus, we adapt the recently proposed BPR method [27], demonstrated to be very effective on implicit feedback data, to item list recommendation.

In BPR [27], each user u is associated with a latent factor w_u, each item t_i is associated with a latent factor h_i, then similarly to other latent factor models, the rating $\hat{x}_{ui}$ of user u on item t_i can be predicted based on the dot product $w_u \cdot h_i$.

Because of the nature of the implicit dataset, similar to [27], we assume that items which have been voted by a user are preferred over other items. Thus, we let $t_i \succ_u t_j$ denote that user u prefers item t_i to item t_j. Following the notation introduced in [27], let D_S denote set of triples (u, t_i, t_j) such that $t_i \succ_u t_j$ holds. Then BPR formalizes a learning problem by maximizing the following log-posterior.

$$\begin{aligned}\ln P(\Theta \mid D_S) &\propto \ln P(D_S \mid \Theta)P(\Theta)\\ &= ln \prod_{(u,t_i,t_j)\in D_S} \sigma(\hat{x}_{ui} - \hat{x}_{uj})p(\Theta)\\ &= \sum_{(u,t_i,t_j)\in D_S} \ln \sigma(\hat{x}_{ui} - \hat{x}_{uj}) + \ln p(\Theta)\\ &= \sum_{(u,t_i,t_j)\in D_S} \ln \sigma(\hat{x}_{ui} - \hat{x}_{uj}) - \lambda_\Theta \|\Theta\|^2 \quad (1)\end{aligned}$$

where $\sigma = \frac{1}{1+e^{-x}}$ is the logistic sigmoid function, Θ denotes all the parameters (latent factors of users and items), and each latent factor is assumed to be following a zero mean Gaussian prior. Finally, λ_Θ is a model specific regularization parameter.

As discussed in [27], the optimization criteria of BPR is closely related to *Area Under the ROC Curve* (AUC) [11]. And above posterior in Equation 1 can be optimized through *Stochastic Gradient Descent* (SGD) which has been demonstrated to be useful for learning various latent factor-based models [17].

2.2.3 Markov-based Recommendation

As we will discuss in Section 5, the item list recommendation problem resembles the playlist recommendation problem studied before [8, 9]. One important difference is that with playlists, there is an inherent assumption that items in the list are meant to be consumed sequentially, which may not be relevant for user-generated item lists such as book lists or product lists. We adapt algorithms proposed for playlist recommendation for the sake of comparison with the algorithms we develop.

In the literature, one very recent and representative model-based approach for playlist recommendation is LME or *Latent Markov Embedding* [9, 10]. In LME, similar to other latent factor models, each item t_i is associated with a latent vector h_i. Given a playlist l, let each item in l at position a be denoted as $l[a]$. In order to model the sequential property a playlist l, we consider the probability of each list being "generated" as a series of Markov transitions between consecutive songs.

$$P(l) = \prod_{a=1}^{|l|} P(l[a] \mid l[a-1]) \qquad (2)$$

where the transition probability $P(l[a] \mid l[a-1])$ can be modeled as a function of the Euclidean distance $\Delta(h_{l[a]}, h_{l[a-1]})$ between the two songs under consideration. Let A denote the set of all songs which can following one specific song $l[a-1]$, we denote by $Z(l[a-1]) = \sum_{t\in A} e^{-\Delta(h_t, h_{l[a-1]})}$ the summation over transition probabilities w.r.t. all possible next song given $l[a-1]$. We can then use the following logistic model to estimate $P(l[a] \mid l[a-1])$.

$$P(l[a] \mid l[a-1]) = \frac{e^{-\Delta(h_{l[a]}, h_{l[a-1]})}}{Z(l[a-1])} \qquad (3)$$

Similar to BPR, an SGD-like algorithm can be leveraged to learn the latent factor model in LME by maximizing the likelihood of generating the training playlists. We note that the original model in [9, 10] is not personalized.

3. LIST RECOMMENDATION MODEL

In this section, we propose a *List Recommendation Model* called LIRE for the item list recommendation problem. Similar to previous latent factor-based models, we map users, items, and lists into a joint latent factor space of dimensionality k. Accordingly, each user u is associated with a vector $w_u \in \mathbb{R}^k$, each item t_i with a vector $h_i \in \mathbb{R}^k$, and each list l_j with a vector $g_j \in \mathbb{R}^k$.

3.1 User Preference over Individual Item List

Intuitively, given a user u and an item list l_j, u's interest in l_j can be captured using two major factors: (i) the overall intrinsic quality of the list l itself, and (ii) the user's aggregate interest in the items in l_j. The first factor can be modeled simply as an inner product between w_u and g_j, while the second factor involves aggregating user's preferences over multiple items within l_j.

A simple idea to model the relationship between u and items in l_j is to assume that each item in l_j has an equal influence on user u. Thus we could model the preference of u on l_j using the following equation.

$$\hat{x}^L_{uj} = w_u g_j + \frac{1}{|l_j|}\sum_{t_i \in l_j}(w_u h_i) \tag{4}$$

where $|l|$ denotes the length of list l. In Equation (4), the first component $w_u g_j$ models user's intrinsic interest in the list as a whole, and the second component models the relationship between u and items in l_j. We call above model UNIFORM. Note that omitting the second component in Equation (4) amounts to directly factorizing the user list interaction matrix M^L into user latent vectors and item list latent vectors and using that solely as the model for recommendation.

As discussed in Section 4.1, usually different items in the list have different influence on a user. E.g., items which are shown at the top of a list, or items which are shown on the first page of the list if pagings are enabled, obviously have more significant influence on the user. This effect is usually due to the way different items of a ranking list are shown on the user interface, and how users consume items in a list. Similar effect can also be observed in information retrieval [13]. To capture this property, we adopt a function inspired by DCG (*Discounted Cumulative Gain*) [15], in order to weight down items which are ranked low in a displayed item list or are shown in later pages in case of a paging enabled interface. Without any ambiguity, let $t_i = l_j[i]$ denote the ith item in list l_j, $1 \leq i \leq |l_j|$, and let h_i be the corresponding latent factor associated with t_i.

$$\hat{x}^L_{uj} = w_u g_j + C(w_u h_1 + \sum_{i=2}^{|l_j|}\frac{w_u h_i}{\log_2 i}) \tag{5}$$

where $C = 1/(1 + \sum_{i=2}^{|l_j|}\frac{1}{\log_2 i})$ serves as a normalization constant. We call above model DCG.

Although DCG is able to weight down items with lower position in the displayed list, given the huge number of items which may exist in an item list, items which are ranked low down in a list usually will not be examined by the user. E.g., one list named "Best Book Cover Art" has 4,775 books and it's unlikely a user will dig deep down such lists. Thus, incorporating these lower ranked items in predicting a user's preference over an item list may introduce lots of noise. In this work, to address this issue, we make the reasonable assumption that users browse an item list top down, and stop browsing when enough items have been seen. Note that this threshold number of items which indicates the number of items users consume before stopping, may vary among different users, thus needs to be personalized. Even for a specific user, this number may change from list to list, which may motivate us to associate one random variable per user and per list. However, given the sparsity of the dataset, such fine granularity threshold modeling might easily lead to overfitting. Thus we consider in this work a trade-off approach by introducing a personalized granularity controlling browsing threshold τ_u.

We set $\tau_u = \beta \times \eta_u$. Here, $\eta_u \in \mathbb{Z}^+$ is a personalized discrete random variable with a positive integer as its value, β is a constant which is used to capture a coarse list browsing granularity. E.g., β can either be the number of items contained in a single page on the website, or can just be a constant such as 10 items, which captures a finer granularity. The value of β can be tuned according to the underlying data. In this work, using our Goodreads dataset, we set β to be 5 which leads to the best performance on our dataset.

Let $\mathbb{I}(i \leq \tau_u)$ be an indicator function which equals 1 if $i \leq \tau_u$ is true, and 0 otherwise. We could adapt DCG to the following model DCG-T.

$$\hat{x}^L_{uj} = w_u g_j + C(w_u h_1 + \sum_{i=2}^{|l_j|}\mathbb{I}(i \leq \tau_u)\frac{w_u h_i}{\log_2 i}) \tag{6}$$

where $C = 1/(1 + \sum_{i=2}^{|l_j|}\mathbb{I}(i \leq \tau_u)\frac{1}{\log_2 i})$ serves as a normalization constant. This model captures the idea that user u stops browsing beyond depth τ_u.

3.2 Modeling Observed Data

In general users' feedback data on item lists tends to be even more sparse than the feedback on items. This makes learning users' preference over item lists more challenging. Fortunately, for most applications in which users can create and share item lists, users also can and tend to interact with individual items. E.g., on Goodreads, users can vote for book lists which are generated by other users, and can also add books to their shelves, meaning they either have already read the books, or are interested in reading those books in the future. Thus in addition to the matrix M^L which captures interactions between users and lists, we also have the matrix M^T which captures interactions between users and items.

Motivated by recent effort on *Collective Matrix Factorization* [31], we consider that users share their preferences over items and lists. Thus we could potentially leverage information learnt from users' item preferences to help mitigate the sparsity issue when modeling users' list preferences. Considering the fact that the interactions between users and items or between users and item lists are often implicit, e.g., vote on Goodreads and subscription on Twitter, we adapt the framework of BPR [27] for deriving the optimization criteria for our item list recommendation problem. Let $\Theta = \{W, G, H, \tau\}$ be the set of parameters associated with the LIRE model. We assume Θ is associated with a zero-mean Gaussian prior. By assuming $\succ^L_u$ and $\succ^T_u$ are conditional independent given Θ, the log of the posterior of Θ given the observed user/item interactions and user/item list interactions can be calculated as follows.

$$\begin{aligned}\mathcal{P}(\Theta) &= \ln p(\Theta \mid \succ^L_u, \succ^T_u) = \ln p(\succ^L_u, \succ^T_u \mid \Theta)p(\Theta)\\ &= \ln p(\succ^L_u \mid \Theta)p(\succ^T_u \mid \Theta)p(\Theta)\\ &= \ln p(\succ^L_u \mid \Theta) + \ln p(\succ^T_u \mid \Theta) - \lambda_\Theta\|\Theta\|^2\\ &= \sum_{(u,l_i,l_j)\in D^L_s}\ln\sigma(\hat{x}^L_{uij}(\Theta)) + \sum_{(u,t_i,t_j)\in D^T_S}\ln\sigma(\hat{x}^T_{uij}(\Theta))\\ &\quad - \lambda_\Theta\|\Theta\|^2 \end{aligned} \tag{7}$$

where D^L_S denotes the set of triples (u, l_i, l_j) for which $l_i \succ_u l_j$ holds, D^T_S denotes the set of triples (u, t_i, t_j) for which $t_i \succ_u t_j$ holds, And λ_Θ are the model specific regularization parameters. The relationship between user u, list l_i, and list l_j is captured by $\hat{x}^L_{uij}(\Theta)$, which can be estimated as $\hat{x}^L_{ui} - \hat{x}^L_{uj}$. Similarly, the relationship between user u, item t_i, and item t_j is captured by $\hat{x}^T_{uij}(\Theta)$, which can be modeled as $\hat{x}^T_{ui} - \hat{x}^T_{uj}$.

3.3 Model Learning

Given $\mathcal{P}(\Theta)$, we use *Maximum-a-Posteriori* (MAP) to perform a point estimation of the parameters of the Lire model. Considering the fact that $\mathcal{P}(\Theta)$ is differentiable w.r.t. most parameters, one popular way to optimize $\mathcal{P}(\Theta)$ is through gradient-based algorithms. A naive approach to doing so would be to directly sample quadruples (l_i, l_j, t_i, t_j), where l_i, l_j are positive and negative item list instances for a user u, and t_i, t_j are positive and negative item instances. More precisely, user u prefers l_i to l_j and similarly t_i to t_j. However, this introduces a huge sampling space. Hence, we propose an alternating learning framework, where we first sample (u, t_i, t_j) from D^T_S, until convergence of $\mathcal{P}(\Theta)^T = \ln p(\Theta \mid \succ^T_u)$; then we sample (u, l_i, l_j) from D^L_S, until convergence of $\mathcal{P}(\Theta)^L = \ln p(\Theta \mid \succ^L_u)$. We iterate above process until the overall posterior converges. We note that the gradient of $\mathcal{P}(\Theta)^T$ w.r.t. Θ can be derived in a similar way as in [27], thus in the following, we focus on how a gradient-based algorithm can be applied to $\mathcal{P}(\Theta)^L$.

Recall from Section 3.1 that during the learning process to maximize $\mathcal{P}(\Theta)^L$, the threshold parameter τ_u for every user takes on positive integers as values, thus $\mathcal{P}(\Theta)^L$ is noncontinous w.r.t. Θ. To solve this issue, we further decompose the maximization of $\mathcal{P}(\Theta)^L$ into the following two steps: first, fix τ_u then optimize the remaining model parameters; then fix the remaining model parameters and optimize τ_u.

Given a fixed τ_u, the following is the gradient of $\mathcal{P}(\Theta)^L$ w.r.t. the model parameter Θ for DCG-T. Similar gradients can be derived for UNIFORM and DCG and we suppress the details. To simplify the notation, we omit mentioning Θ for $\hat{x}^T_{uij}(\Theta)$ and $\hat{x}^L_{uij}(\Theta)$ if there is no ambiguity.

$$\begin{aligned} \frac{\partial \mathcal{P}(\Theta)^L}{\partial \Theta} &= \sum_{(u,l_i,l_j) \in D^L_s} \frac{\partial \ln \sigma(\hat{x}^L_{uij})}{\partial \Theta} - \lambda_\Theta \frac{\partial \|\Theta\|^2}{\partial \Theta} \\ &= \sum_{(u,l_i,l_j) \in D^L_s} \frac{e^{-\hat{x}^L_{uij}}}{1 + e^{-\hat{x}^L_{uij}}} \frac{\partial \hat{x}^L_{uij}}{\partial \Theta} - \lambda_\Theta \Theta \end{aligned} \quad (8)$$

Since $\partial \hat{x}^L_{uij}/\partial\Theta = \partial \hat{x}^L_{ui}/\partial\Theta - \partial \hat{x}^L_{uj}/\partial\Theta$, we list in the following equations how $\partial \hat{x}^L_{uj}/\partial\Theta$ can be derived for w, and g in DCG-T, where f indicates an index into the corresponding latent factor.

$$\frac{\partial \hat{x}^L_{uj}}{\partial \Theta} = \begin{cases} g_{jf} + C(h_{1f} + \sum_{i=2}^{|l_j|} \mathbb{I}(i \le \tau_u) \frac{h_{if}}{\log_2 i}) & \Theta = w_{uf} \\ w_{uf} & \Theta = g_{jf} \end{cases}$$

Given $\tau_u = \beta \times \eta_u$, η_u may have a small domain given the size $|l_j|$ of a list l_j. E.g., on Goodreads, when β is set to the number of items on a single web page, most lists have size less than 40 web pages. A simple idea thus is to enumerate all possible η_u and then find the optimal η^*_u which maximizes $\mathcal{P}(\Theta)^L$.

$$\eta^*_u = \underset{1 \le \eta_u \le \lceil |l_j|/\beta \rceil}{\arg\max} \; \mathcal{P}(\Theta)^L \quad (9)$$

However, when β is tuned to be at a finer granularity, the cost of above exhaustive search becomes prohibitive. Given the fact that $\mathcal{P}(\Theta)^L$ may not be monotone w.r.t. τ_u based on different possible values of the latent factors, we consider a local search algorithm LocalOpt (Algorithm 1) which finds a local maximum of τ_u.

Algorithm 1: LocalOpt(β,Θ,$|l|$)

```
1  $\eta^* \leftarrow$ A random integer between 1 and $\lceil |l|/\beta \rceil$;
2  $Q \leftarrow$ An empty candidate queue;
3  $Q$.add(($\max(\eta_u - 1, 1), left$));
4  $Q$.add(($\min(\eta_u + 1, \lceil |l|/\beta \rceil), right$));
5  while $Q$ is not empty do
6      ($\eta, direction$) $\leftarrow Q$.pop();
7      if $\mathcal{P}_{\eta^*}(\Theta)^L < \mathcal{P}_{\eta}(\Theta)^L$ then
8          $\eta^* \leftarrow \eta$;
9          if $direction = left$ and $\eta > 1$ then
10             $Q$.add(($\eta - 1, left$));
11         if $direction = right$ and $\eta < \lceil |l|/\beta \rceil$) then
12             $Q$.add(($\eta + 1, right$));
13 return $\eta^*$
```

The overall algorithm for learning Lire is listed in Algorithm 2. In each iteration of the main loop, we first sample (u, t_i, t_j) from D^T_S and update model parameters W, H through *Stochastic Gradient Ascent* until convergence. Then we sample (u, l_i, l_j) from D^L_S, and update model parameters W, G through *Stochastic Gradient Ascent* until convergence. Finally, we find the optimal τ through Algorithm LocalOpt for every user. We repeat above three steps until the overall model has converged or the maximum number of iterations has been reached.

Algorithm 2: Lire -Learn(Θ, D^L_S, D^T_S, β)

```
1  Random initialize $\Theta$;
2  repeat
3      repeat
4          Draw $(u, t_i, t_j)$ from $D^T_S$;
5          Update $W$, $H$ w.r.t. $\mathcal{P}(\Theta)^T$;
6      until convergence;
7      repeat
8          Draw $(u, l_i, l_j)$ from $D^L_S$;
9          Update $W$. $G$ w.r.t. $\mathcal{P}(\Theta)^L$;
10     until covergence;
11     foreach User $u$ do
12         $|l| \leftarrow$ Longest list size considered by $u$;
13         $\tau_u \leftarrow \beta\times$ LocalOpt($\beta, \Theta, |l|$);
14 until convergence or max-iter has been reached;
```

4. EXPERIMENT RESULTS

4.1 Goodreads Data Setup

To evaluate our proposed model Lire and compare it with existing models for item and playlist recommendation, we obtained during one month a 10% sample of all user-generated book lists on Goodreads. As can be found from the Goodreads website, there exist in total around 34,750 user-generated book lists, and our obtained sample includes 3,000 randomly selected book lists from the entire collection.

We filter book lists which have fewer than 5 voters, and the resulting dataset contains 1,998 book lists. For the set of voters obtained from these 1,998 book lists, we again filter

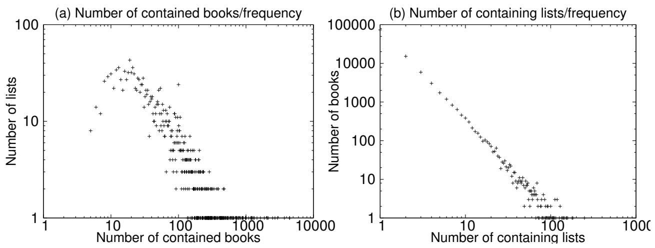

Figure 3: Statistics of Goodreads data w.r.t. lists and books

out those who have voted less than 5 times within the 1,998 book lists. The total number of unique users left at the end of this process is 3,425. To fit users' preference over individual items, we also obtained the set of books which have been added to these 3,425 users' book shelves. The total number of unique books is 105,030.

In Figure 3 (a), we group lists into different buckets based on how many books they contain, and plot the number of lists which belong to each bucket. It can be seen that the majority of the obtained book lists contain only a small number of books, but there does exist book list which contains 4,775 books. The average number of books per list is 109. Similarly, in Figure 3 (b), we group books into different buckets based on how many book lists they belong to, and plot the number of books belong to each bucket. Again, there is an obvious power law distribution between the number of lists a book belongs to and the number of books that belong to the same number of books.

Similar statistics of the Goodreads dataset can be observed between users and lists, and also between users and books, as shown in Figure 2. In a nutshell, the majority of users interact only with a few lists or books, while a few power users interact with a large number of lists or books.

From the obtained Goodreads dataset, we randomly sample 10% user/list interaction data as the test set, and the remaining data is treated as training set. To simplify the training process, we assume each parameter of the model is associated with the same regularization parameter λ. Learning rate γ in the Stochastic gradient descent step, and regularization parameter λ are selected based on grid search. In our experiments, we found that usually setting $\gamma = 0.1$ and $\lambda = 0.01$ leads to the best performance.

4.1.1 Algorithms Compared

We compare our proposed Lire with the following 5 baseline algorithms as discussed in Section 2.2: 1. GLB; 2. PPOP; 3. PIF; 4. BPR; 5. LME. For BPR and LME, we also used grid search to find the best learning parameter settings. Finally, we consider two variations of Lire – Lire -UNIFORM with uniform item weighting, and Lire -DCG-T with item position-based weighting and personalized browsing threshold.

4.1.2 Performance Measure

AUC is a commonly used metric for testing the quality of rank orderings. Following [27], we use the AUC metric described below to compare the proposed Lire model with the baseline algorithms as it has been demonstrated to be a good metric for evaluating Top-N recommendation tasks [6]. Suppose the set of users, positive book list instance (book lists voted by the user in the test dataset), negative book list instance (book lists which have not been voted either in the training set or testing dataset) in the test dataset are denoted as D_S^{test}. Then, the formula to compute AUC is given by:

$$\frac{1}{|D_S^{test}|} \sum_{(u,l_i,l_j) \in D_S^{test}} \mathbb{I}(\hat{x}_{ui} > \hat{x}_{uj})$$

where $\mathbb{I}(\hat{x}_{ui} > \hat{x}_{uj})$ is an indicator function which returns 1 if $\hat{x}_{ui} > \hat{x}_{uj}$ is true, and 0 otherwise.

4.2 Granularity of Browsing

We first test how the granularity parameter β of the browsing threshold τ can affect the performance of the Lire model. As discussed in Section 3, on one hand, when setting β to a small value, the predicted browsing threshold may lack flexibility in predicting user's browsing pattern across different lists. Whereas on the other hand, setting β to be a large value, we may incorporate unnecessary items into the prediction of user's interest in a specific list.

The above trade-off is confirmed by our results on Goodreads dataset as shown in Table 1. By fixing the dimensionality of D to be 10, the overall AUC on the test set increases when β is varied from 1 to 5, and AUC decreases when we further increase the value of β. We observed similar results for other D settings and suppress them for lack of space. This result indicates that β in practice can be tuned based on the actual dataset.

	β value				
	1	3	5	7	9
AUC	0.9778	0.9787	**0.9830**	0.9819	0.9800

Table 1: How β value affect the AUC.

4.3 Convergence of Lire

In Figure 4, we demonstrate the convergence behavior of the Lire model. As can be found in this figure, our model converges fairly quickly (around 5 iterations). We note that this convergence result also holds for other dimensionality setting of the latent factors.

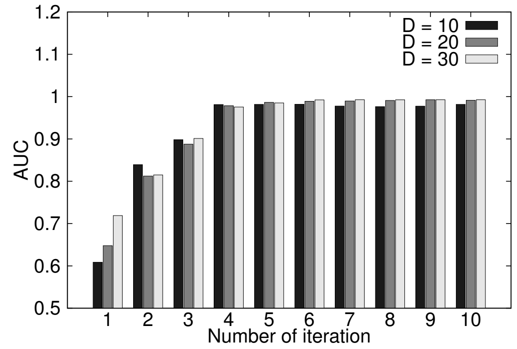

Figure 4: Convergence of Lire when varying dimensionality of the latent factors.

4.4 Quality Comparison

Finally, we compare the performance of Lire with other algorithms as presented in Section 2.2. From Figure 5, we can readily observe that the three baseline algorithms GLB, PPOP, PIF do not perform nearly as well as BPR and Lire . We note that this is in contrast with the result on playlist dataset, where popularity-based methods, and especially personalized popularity-based methods excel [8]. Also the Markov-based approach LME does not perform as

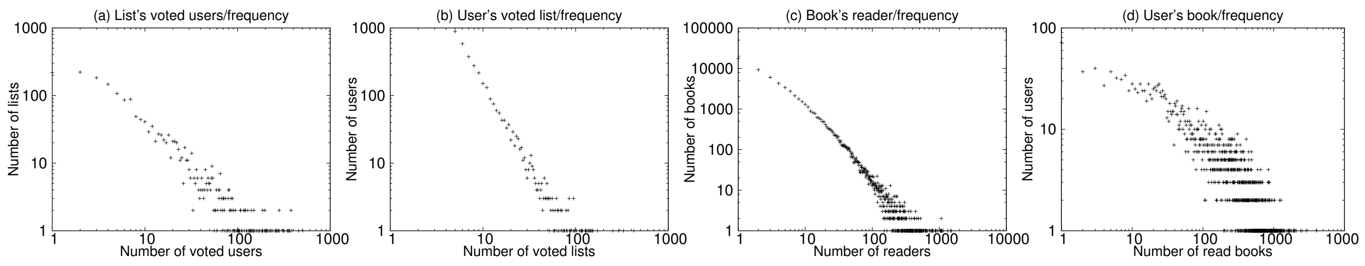

Figure 2: Statistics of Goodreads data w.r.t. lists and users, users and books

well as BPR and LIRE . We claim that the reason for this is that user-generated lists such as book lists on Goodreads do not have the sequential consumption property normally assumed by playlist modeling works.

For the two variations of LIRE , we can see from Figure 5, both of them perform better than the other algorithms. However, LIRE -DCG-T is better than LIRE -UNIFORM, for two reasons: (i) LIRE -UNIFORM may aggregate more than the necessary number of items' preference when modeling a user's interest in an item list, and (ii) it ignores the fact that preference over items positioned higher up in the list may have a larger impact on the user's preference for the list. In addition, because LIRE -UNIFORM needs to visit all items in an item list during training, training of LIRE -UNIFORM is much slower compared with LIRE -DCG-T.

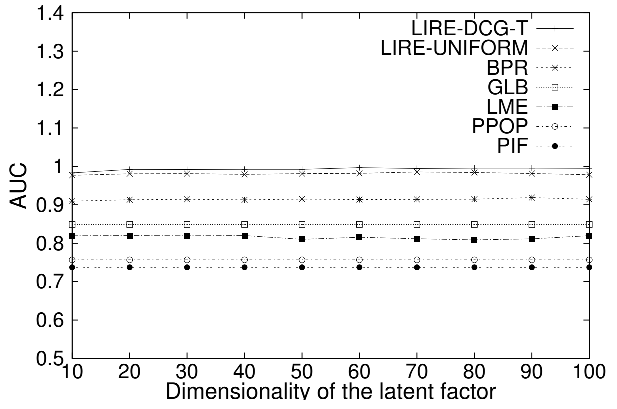

Figure 5: Quality comparison for various algorithms.

5. RELATED WORK

Latent factor models for recommendation have generated significant amount of attention due to their famous success at the Netflix competition [20]. In latent factor models, each observed rating in a recommender system can be explained as a dot product between two learned latent factors corresponding to the underlying user and item. Recently, latent factor models have been extended extensively to incorporate additional features such as time [19], context [16], and content features of items [5].

Many existing works on recommender system focus on explicit rating datasets in which ratings can take values from a small domain such as 1 to 5. However, for many recommendation scenarios, feedbacks from users take an implicit form, such as whether a user purchased a product, liked a feed, clicked on an item, etc. These implicit data settings create a huge challenging for existing latent factor models, and researchers have recently proposed various models for modeling implicit datasets. One notable work in this direction is BPR [27], in which the authors proposed to solve the implicit data problem through a general Bayesian ranking model, which can learn users' preferences over items by sampling positive and negative/missing entries from the user rating matrix. Another work which proposes to model implicit dataset through ranking is CLiMF [30], which instead of optimizing AUC as in BPR, considers *Mean Reciprocal Rank* as the optimization criterion. In [24], the authors argue that reasoning about preferences between individual items might be too fine a granularity, and propose to lift item preferences to preferences over sets of items. The focus of this work is still on item recommendation, and when aggregating item preferences, the proposed model does not consider the position of an item, and the defined item sets are hidden from users thus there is no need to model how users might view items within a set. Other works on implicit datasets include [12], in which the authors propose a different way of modeling implicit datasets by weighting the importance of each observed rating through heuristic functions. In [25], the authors propose a more complex model which models the relationship between the original implicit dataset (which can be considered as a bipartite graph between users and items) and random bipartite graphs which captures potential items which users have considered before actually consuming items in the original dataset.

User-generated item lists are similar to playlists as explored in previous works such as [9] and [7]. An important observation made by these works is that neighboring items in a playlist are usually correlated with each other and items in a playlist usually have to be consumed sequentially. E.g., in LME [9], the proposed model assumes that transitions between neighboring items in a playlist satisfy the Markovian property, i.e., next song in a playlist depend only on the current song which is playing. Similarly, in [7], the authors consider that each song in a playlist is closely related to other songs which are played within the same short time window. An interesting survey of algorithms for modeling playlists is presented in [8].

The proposed item list recommendation problem is also related to recent effort on package recommendation [26, 28, 32]. However, most of these works focus on handling hard constraints specified by users as opposed to learning user's preferences over item lists from previous feedback. In [23], the authors propose a *Probability Matrix Factorization*-based approach to model user's preferences over travel packages, where each package is composed of a set of places of interest. The focus of this work is on modeling the cost of a package, whereas in our work we focus on how item preferences can be aggregated to model user's preferences over item lists.

Finally, the way we model interactions between users and item lists, and also interactions between users and individual items is closely related to recent efforts on *collective matrix* *factorization* [31, 14], which has become a popular way of modeling scenarios with heterogeneous types of interactions through sharing latent factors.

6. CONCLUSION

In this paper, we motivated the problem of recommending user-generated item lists and proposed a novel model called Lire for this purpose. Existing works in recommender system usually focus on item recommendation, thus do not consider how user's preference over item lists can be decomposed into preferences over individual items within the list. Though playlist recommendation considers item lists in a specific setting, the proposed model cannot be applied to other item list settings as studied in this work, due to the fact that items in these lists do not need to be consumed sequentially. In this work, we identify that there exist multiple challenges on how presentation of an item list in an application such as Goodreads might impact a user's preference over item lists. And based on our observations from Goodreads book list dataset, we propose to model users' preferences over item lists by aggregating users' preferences over individual items of a list, we capture how users perceive an item list by weighting items within a list based on position, and we also consider how many items a user might consume before deciding whether to like this list or not. Through extensive experiments we demonstrate that our proposed model has a better performance compared with many existing recommendation algorithms.

There exist multiple directions for future work: first, we can study how temporal information from the dataset can be leveraged to help better model a user's preference over item lists. E.g., we could check whether liking a list will result in a subsequent consumption of an item within the same list. Second, many applications which have user-generated item lists involve social network. This raises the question how social influence can be modeled in Lire . Finally, it's interesting to ask how contents of lists and items can be incorporated into Lire through methods such as [5].

7. ACKNOWLEDGEMENT

This work was supported in part by the Institute for Computing, Information and Cognitive Systems (ICICS) at UBC.

8. REFERENCES

[1] https://support.twitter.com/articles/76460-using-twitter-lists.

[2] https://www.goodreads.com/list.

[3] http://www.amazon.ca/gp/help/customer/display.html?nodeId=1197914.

[4] G. Adomavicius and A. 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.

[5] D. Agarwal and B.-C. Chen. Regression-based latent factor models. In *KDD*, pages 19–28, 2009.

[6] A. Ahmed, B. Kanagal, S. Pandey, V. Josifovski, L. G. Pueyo, and J. Yuan. Latent factor models with additive and hierarchically-smoothed user preferences. In *WSDM*, pages 385–394, 2013.

[7] N. Aizenberg, Y. Koren, and O. Somekh. Build your own music recommender by modeling internet radio streams. In *WWW*, pages 1–10, 2012.

[8] G. Bonnin and D. Jannach. Evaluating the quality of generated playlists based on hand-crafted samples. In *ISMIR*, pages 263–268, 2013.

[9] S. Chen, J. L. Moore, D. Turnbull, and T. Joachims. Playlist prediction via metric embedding. In *KDD*, pages 714–722, 2012.

[10] S. Chen, J. Xu, and T. Joachims. Multi-space probabilistic sequence modeling. In *KDD*, pages 865–873, 2013.

[11] A. Herschtal and B. Raskutti. Optimising area under the roc curve using gradient descent. In *ICML*, 2004.

[12] Y. Hu, Y. Koren, and C. Volinsky. Collaborative filtering for implicit feedback datasets. In *ICDM*, pages 263–272, 2008.

[13] S. B. Huffman and M. Hochster. How well does result relevance predict session satisfaction? In *SIGIR*, pages 567–574, 2007.

[14] M. Jamali and L. Lakshmanan. Heteromf: recommendation in heterogeneous information networks using context dependent factor models. In *WWW*, pages 643–654, 2013.

[15] K. Järvelin and J. Kekäläinen. Cumulated gain-based evaluation of ir techniques. *ACM Trans. Inf. Syst.*, 20(4):422–446, 2002.

[16] A. Karatzoglou, X. Amatriain, L. Baltrunas, and N. Oliver. Multiverse recommendation: N-dimensional tensor factorization for context-aware collaborative filtering. In *RecSys*, pages 79–86, 2010.

[17] Y. Koren. Factorization meets the neighborhood: A multifaceted collaborative filtering model. In *KDD*, pages 426–434, 2008.

[18] Y. Koren. The bellkor solution to the netflix grand prize, 2009.

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

[20] Y. Koren, R. M. Bell, and C. Volinsky. Matrix factorization techniques for recommender systems. *IEEE Computer*, 42(8):30–37, 2009.

[21] P. Lamere and S. Green. Project aura: recommendation for the rest of us. In *Sun JavaOne Conference*, 2008.

[22] G. Linden, B. Smith, and J. York. Amazon.com recommendations: Item-to-item collaborative filtering. *IEEE Internet Computing*, 7(1):76–80, 2003.

[23] Q. Liu, Y. Ge, Z. Li, E. Chen, and H. Xiong. Personalized travel package recommendation. In *ICDM*, 2011.

[24] W. Pan and L. Chen. Cofiset: Collaborative filtering via learning pairwise preferences over item-sets. In *SDM*, pages 180–188, 2013.

[25] U. Paquet and N. Koenigstein. One-class collaborative filtering with random graphs. In *WWW*, pages 999–1008, 2013.

[26] A. G. Parameswaran and H. Garcia-Molina. Recommendations with prerequisites. In *RecSys*, pages 353–356, 2009.

[27] S. Rendle, C. Freudenthaler, Z. Gantner, and L. Schmidt-Thieme. Bpr: Bayesian personalized ranking from implicit feedback. In *UAI*, pages 452–461, 2009.

[28] S. B. Roy, S. Amer-Yahia, A. Chawla, G. Das, and C. Yu. Constructing and exploring composite items. In *SIGMOD*, pages 843–854, 2010.

[29] B. M. Sarwar, G. Karypis, J. A. Konstan, and J. Riedl. Item-based collaborative filtering recommendation algorithms. In *WWW*, pages 285–295, 2001.

[30] 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 *RecSys*, pages 139–146, 2012.

[31] A. P. Singh and G. J. Gordon. Relational learning via collective matrix factorization. In *KDD*, pages 650–658, 2008.

[32] M. Xie, L. V. Lakshmanan, and P. T. Wood. Breaking out of the box of recommendations: From items to packages. In *RecSys*, pages 151–158. ACM, 2010.