

Recommending User-Generated Item Lists

by

Yidan Liu

B.Eng., Computer Science, Beijing Institute of Technology, 2013

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE

in

The Faculty of Graduate and Postdoctoral Studies
(Computer Science)

THE UNIVERSITY OF BRITISH COLUMBIA
(Vancouver)

April 2016

© Yidan Liu 2016

Abstract

Nowadays, more and more websites are providing users with the functionality to create item lists. For example, in Yelp, users can create restaurant lists he/she likes. In Goodreads, users can create booklists consisting of books they like. 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. Existing recommender systems are not designed for recommending user-generated item lists directly. In this work, we study properties of these user-generated item lists and propose two different models for recommending user-generated lists.

We first model the problem as a recommendation problem and propose a Bayesian ranking model, called LIRE . 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 positions 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.

We then summarize the lessons learned from the recommendation model and model the problem as an optimization problem. We show that the list recommendation optimization can be reduced to Maximum k -Coverage Problem, which is an NP-hard problem. We derive three different problem variants and proposed algorithms for each of them. We then conduct experiments to compare their results. Lessons learned and possible application scenarios are also discussed in the hope of helping us and more people improve the work. Furthermore, we propose several potential directions for future work.

Preface

This thesis is a collaborative research work with my colleague Min Xie and my supervisor Laks V.S. Lakshamanan. The three of us together published the below listed paper [21], which subsumes the first chapter of this thesis.

1. Liu, Yidan, Min Xie, and Laks VS Lakshmanan. "Recommending user generated item lists." Proceedings of the 8th ACM Conference on Recommender systems. ACM, 2014.

Below lists the contribution of each co-author in detail.

Dr. Laks V.S. Lakshamanan, my supervisor, motivated the initial brainstorming of the problem studied in this thesis. He later involved in formulating the research problem, designing algorithms as well as experiments, and revising the paper before submission.

Min Xie, a PhD colleague of mine, has been very generous in helping me formulating the problem, designing algorithms to solve the problem, and conducting experiments to evaluate the solutions. Min provided me with ideas of how to model an optimization problem. He also involved in writing and revising the aforementioned paper.

I was in charge of summarizing all the discussions and formulating the ideas at the early stage of the project. Later I was involved in formulating the optimization problem and designing algorithms to solve the problem. I proposed three problems variants and designed algorithms to solve each of them. Also, I implemented the experiments conducted in the aforementioned paper and this thesis, including collecting data from websites, preprocessing, adapting the algorithms, and evaluation. In addition, I was involved in writing, reviewing and revising the published paper and this thesis.

Lemma 3.2.3 is derived from the chapter 2 of the below listed paper [11].

1. Cohen, Reuven, and Liran Katzir. "The generalized maximum coverage problem." *Information Processing Letters* 108.1 (2008): 15-22.

Table of Contents

Abstract	ii
Preface	iii
Table of Contents	v
List of Tables	vii
List of Figures	viii
Acknowledgements	ix
1 Introduction	1
2 Recommendation Model	4
2.1 Lists Properties	4
2.2 Problem Studied	5
2.3 Baseline Algorithms	5
2.3.1 Popularity-based Approach	5
2.3.2 Collaborative Filtering for Implicit Data	6
2.3.3 Markov-based Recommendation	8
2.4 Recommendation Model	9
2.4.1 User Preference over Individual Item List	9
2.4.2 Modeling Observed Data	11
2.4.3 Model Learning	12
2.5 Experiment Results	15
2.5.1 Goodreads Data Setup	15

Table of Contents

2.5.2	Granularity of Browsing	18
2.5.3	Convergence of LIRE	19
2.5.4	Quality Comparison	19
3	Optimization Model	21
3.1	Lessons Learned	21
3.2	Problem Studied and Application Scenarios	22
3.2.1	Preliminaries	22
3.2.2	Problem Definition	23
3.2.3	Problem Variants	23
3.2.4	Application Scenarios	27
3.3	Modelling Relevance	28
3.3.1	User Item Relevance	28
3.3.2	User List Relevance	29
3.4	Algorithms	30
3.5	Experiment Results	32
3.5.1	Data Setup	32
3.5.2	Topic Modeling	33
3.5.3	Ground Information	33
3.5.4	Similarity Measures	34
3.5.5	Evaluation Results	35
3.5.6	Results Analysis	41
4	Related Works	43
4.1	Playlist Recommendation	43
4.2	Travel Package Recommendation	43
4.3	Implicit Feedback Recommendation	44
4.4	Hybrid-model Recommendation	45
5	Conclusion	46
6	Future Works	47
	Bibliography	48

List of Tables

2.1	How β value affect the AUC.	19
3.1	Datasets statistics	33
3.2	Topic modeling statistics	33

List of Figures

1.1	Examples of user-generated lists from different websites. . . .	2
2.1	Statistics of Goodreads data w.r.t. lists and users, users and books	17
2.2	Statistics of Goodreads data w.r.t. lists and books	17
2.3	Convergence of LIRE when varying dimensionality of the latent factors.	19
2.4	Quality comparison for various algorithms.	20
3.1	List recommendation scenario.	22
3.2	Evaluation results on Goodreads data	37
3.3	Evaluation results on Askubuntu data	38
3.4	Evaluation results on Programmer data	39
3.5	Evaluation results on Unix data	40
3.6	Compare the results of with and without content information	41

Acknowledgements

I wish to express my sincere thanks to Prof. Laks V.S. Lakshmanan for his guidance during my Master’s study. His passion to research, attention to detail and hard work have been inspiring me during my exploration to the world of recommender systems.

I must also acknowledge my PhD colleague Min Xie for his assistance and support all along. Min has extensive knowledge in databases and recommender systems. He has given me a lot of suggestions on formalizing the problem as well as writing the thesis.

I would also like to thank my second reader, Prof. Giuseppe Carenini for his insightful comments and valuable suggestions. His comments on comparing the two models has given me a new perspective on my thesis work.

I would like to thank my fellow lab mates in Data Mining and Management Lab. They have given me a lot of valuable suggestions and encouragement during my Master’s study. Thanks also goes out to my friends Shanshan Chen and Yan Peng, who have been encouraging me along the way.

Most importantly, I would like to thank my family for the support and love they have given to me through my entire life. I must also acknowledge my fianc Yu Yan for his generosity and understanding.

Chapter 1

Introduction

With the development of web applications and services, people are provided with more opportunities as well as assistance to discover the items of their interest. These days, a lot of websites provide users with the service of creating item lists. More specifically, users can create lists on the site, adding items to the lists for future use. The types of items in the lists often depend on the web service. For example, in Youtube, users can create a list consist of music videos from the same genre. In Yelp, a user can create a restaurant list that he/she often visits. In Foursquare, users can create a list of places he/she would like to visit. These item lists are usually created with a certain theme and are often made public by default, which enables them to be shared with other users on the website.

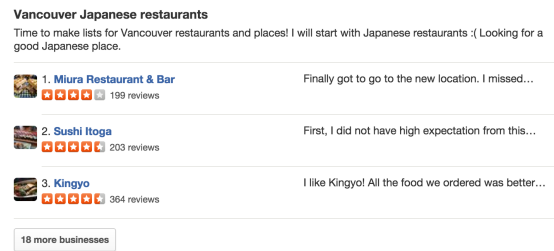
While conventional item recommendations have been well-studied, there are few works directly focusing on recommending item lists. In this work, we studied the properties of user-generated lists and proposed multiple models for picking the most relevant lists for each user.

The primary motivations for studying item lists are three-folded. Firstly, these user-generated item lists have become a trend these days. As more and more websites are providing users with such functionality, the popularity of item lists is self-evident.

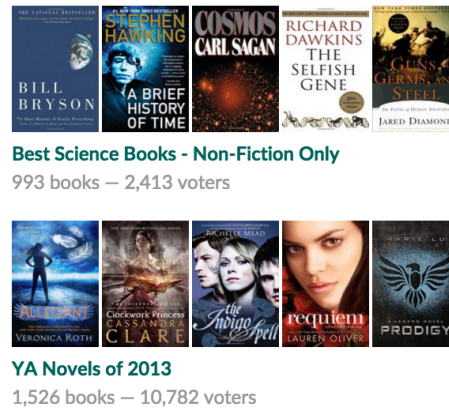
Second, the user-generated item lists can be indeed helpful for users to discover items of their interests. For websites like Amazon [2] or Yelp [3], the quantity of the items on the website can be considerably large. While users may be able to find more relevant items from a large number of items, it also becomes more challenging for them to search and identify their items of interest. The user-generated item lists provide an efficient way to manage similar items. Moreover, since these lists can be shared with other users, it

represents the wisdom of the crowds. With the help of item lists, users can narrow down their search and find items of their interest more efficiently.

Imagine that a user has found an item she likes and would like to discover more items similar to this one. Instead of exhaustively searching with keywords, she can just browse through the lists containing the current item and look for more relevant items. This way the search range can be quickly narrowed down. Below are some examples from different websites.



(a) Yelp list example



(b) Goodreads list example

Figure 1.1: Examples of user-generated lists from different websites.

Last but not the least, while conventional item recommendations have been well-studied, there are few works directly targeting at recommending user-generated item lists. 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 [6, 8, 9]. 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 [6, 9]. However, for the item lists in our case, 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. Therefore, the lack of research works on user-generated list recommendation further motivates us to investigate the user-generated item lists and make full use of them.

To model the list recommendation problem, we analyze the properties of the user-generated item lists and proposed two different frameworks: modelling as a recommendation problem and an optimization problem. We first 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. One challenge in modeling a user’s interest in 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 just see the top items in a list, and may stop browsing the list when enough items have been explored. Therefore, we weight items based on their positions in the list, and 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.

Then we summarize the lessons learned from LIRE and propose an optimization model. In this model, based on different ways of crediting repeated items, we propose three problem variants and design algorithms for each problem. We also take content information into consideration. Finally three algorithms’ performances are compared on real-world datasets.

In the rest of the thesis, we first model the problem as a recommendation problem in Chapter 2. In Chapter 3, we model the problem as an optimization problem. Related works are discussed and compared in Chapter 4.

Chapter 2

Recommendation Model

In this chapter, we will first give a brief overview of the list properties. Next, we will give a detailed definition of the problem and present our recommendation model.

2.1 Lists Properties

In this section, we will discuss in detail the properties of user-generated item lists, in the hope of helping readers to understand the type of lists discussed in our work.

First, the item lists we studied are generated by website users and are open to other users on the website. Users can view lists created by other users, as well as comment or edit the lists. Similarly, the recommendation output will also be the user-generated item lists.

Second, the item lists are composed by a set of consumable items. Users on the website can interact with both items and item lists. Items within a list are ranked in a particular order, either decided by the user or by metadata such as popularity or modified date.

Third, these lists usually have a specific theme, and items within the lists are relevant to this theme. Therefore, a list is more than a random set of items. Internal relationships exist between these items, as well as between the set of items and the list itself.

Finally, user feedback on item lists is *implicit*. 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 [19]. 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

[14, 25]. For the item lists in our case, we can only get boolean observations of whether a user has shown interest to a list or not, as compared to explicit ratings which can be found on movie recommendation websites.

2.2 Problem Studied

To formulate the item list recommendation problem, consider a set of users $U = \{u_1, \dots, u_m\}$, a set of items $T = \{t_1, \dots, t_n\}$, and a set of item lists $L = \{l_1, \dots, l_q\}$, 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. As is mentioned in 2.1, we only focus on implicit feedback.

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 $L \setminus L_u$ which u will be most interested in.*

2.3 Baseline Algorithms

2.3.1 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].

2.3. Baseline Algorithms

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.

2.3.2 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

2.3. Baseline Algorithms

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 [19]. 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 [25], demonstrated to be very effective on implicit feedback data, to item list recommendation.

In BPR [25], 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 [25], 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 [25], 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.

$$\begin{aligned}
 \ln P(\Theta \mid D_S) &\propto \ln P(D_S \mid \Theta) P(\Theta) \\
 &= \ln \prod_{(u, t_i, t_j) \in D_S} \sigma(\hat{x}_{ui} - \hat{x}_{uj}) p(\Theta) \\
 &= \sum_{(u, t_i, t_j) \in D_S} \ln \sigma(\hat{x}_{ui} - \hat{x}_{uj}) + \ln p(\Theta) \\
 &= \sum_{(u, t_i, t_j) \in D_S} \ln \sigma(\hat{x}_{ui} - \hat{x}_{uj}) - \lambda_\Theta \|\Theta\|^2 \quad (2.1)
 \end{aligned}$$

Here, \propto refers to direct proportionality. $\sigma = \frac{1}{1+e^{-x}}$ 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 [25], the optimization criteria of BPR is closely related to *Area Under the ROC Curve* (AUC) [12]. And above posterior in Equation 2.1 can be optimized through *Stochastic Gradient Descent* (SGD) which

has been demonstrated to be useful for learning various latent factor-based models [17].

2.3.3 Markov-based Recommendation

As we will discuss in Chapter 4, 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 of a playlist l , we consider the probability of each list being “generated” as a series of Markov transitions between consecutive songs.

$$P(l) = \prod_{a=1}^{|l|} P(l[a] \mid l[a-1]) \quad (2.2)$$

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 follow 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 $P(l[a] \mid l[a-1])$.

$$P(l[a] \mid l[a-1]) = \frac{e^{-\Delta(h_{l[a]}, h_{l[a-1]})}}{Z(l[a-1])} \quad (2.3)$$

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.

2.4 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 and lists into the same latent space. Since we also take items into consideration, we map users and items into the same latent space. So users, items, and lists are mapped 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 $g_j \in \mathbb{R}^k$.

2.4.1 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.

$$\hat{x}_{uj}^L = w_u g_j + \frac{1}{|l_j|} \sum_{t_i \in l_j} (w_u h_i) \quad (2.4)$$

where $|l|$ denotes the length of list l . In Equation (2.4), 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 (2.4) 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 2.5.1, 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 [15]. To capture this property, we adopt a function inspired by DCG (*Discounted Cumulative Gain*) [16], 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 i th item in list l_j , $1 \leq i \leq |l_j|$, and let h_i be the corresponding latent factor associated with t_i .

$$\hat{x}_{uj}^L = w_u g_j + C \left(w_u h_1 + \sum_{i=2}^{|l_j|} \frac{w_u h_i}{\log_2 i} \right) \quad (2.5)$$

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

2.4. Recommendation Model

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.

$$\hat{x}_{uj}^L = w_u g_j + C(w_u h_1 + \sum_{i=2}^{|l_j|} \mathbb{I}(i \leq \tau_u) \frac{w_u h_i}{\log_2 i}) \quad (2.6)$$

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 .

2.4.2 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* [28], 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 [25] 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_u^L and \succ_u^T 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.

$$\begin{aligned}
 \mathcal{P}(\Theta) &= \ln p(\Theta | \succ_u^L, \succ_u^T) = \ln p(\succ_u^L, \succ_u^T | \Theta) p(\Theta) \\
 &= \ln p(\succ_u^L | \Theta) p(\succ_u^T | \Theta) p(\Theta) \\
 &= \ln p(\succ_u^L | \Theta) + \ln p(\succ_u^T | \Theta) - \lambda_\Theta \|\Theta\|^2 \\
 &= \sum_{(u, l_i, l_j) \in D_S^L} \ln \sigma(\hat{x}_{uij}^L(\Theta)) + \sum_{(u, t_i, t_j) \in D_S^T} \ln \sigma(\hat{x}_{uij}^T(\Theta)) \\
 &\quad - \lambda_\Theta \|\Theta\|^2
 \end{aligned} \tag{2.7}$$

where D_S^L denotes the set of triples (u, l_i, l_j) for which $l_i \succ_u l_j$ holds, D_S^T 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 $\hat{x}_{uij}^L(\Theta)$. Following the BPR model, we decompose the estimator and define \hat{x}_{uij} as:

$$\hat{x}_{uij}^L(\Theta) := \hat{x}_{ui}^L(\Theta) - \hat{x}_{uj}^L(\Theta) \tag{2.8}$$

Similarly, the relationship between user u , item t_i , and item t_j is captured by $\hat{x}_{uij}^T(\Theta)$, which can be modeled as $\hat{x}_{ui}^T - \hat{x}_{uj}^T$.

2.4.3 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_S^T , until convergence of $\mathcal{P}(\Theta)^T = \ln p(\Theta \mid \succ_u^T)$; then we sample (u, l_i, l_j) from D_S^L , until convergence of $\mathcal{P}(\Theta)^L = \ln p(\Theta \mid \succ_u^L)$. 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 [25], thus in the following, we focus on how a gradient-based algorithm can be applied to $\mathcal{P}(\Theta)^L$.

Recall from Section 2.4.1 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 non-continuous 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. Since UNIFORM and DCG only differ from DCG-T with respect to the calculation of \hat{x}_{uj} , the similar gradients can be derived for them. To simplify the notation, we omit mentioning Θ for $\hat{x}_{uij}^T(\Theta)$ and $\hat{x}_{uij}^L(\Theta)$ if there is no ambiguity.

$$\begin{aligned}
 \frac{\partial \mathcal{P}(\Theta)^L}{\partial \Theta} &= \sum_{(u, l_i, l_j) \in D_S^L} \frac{\partial \ln \sigma(\hat{x}_{uij}^L)}{\partial \Theta} - \lambda_\Theta \frac{\partial \|\Theta\|^2}{\partial \Theta} \\
 &= \sum_{(u, l_i, l_j) \in D_S^L} \frac{\partial \ln \frac{1}{1 + e^{-\hat{x}_{uij}^L}}}{\partial \Theta} - \lambda_\Theta \Theta \\
 &= \sum_{(u, l_i, l_j) \in D_S^L} \frac{e^{-\hat{x}_{uij}^L}}{1 + e^{-\hat{x}_{uij}^L}} \frac{\partial \hat{x}_{uij}^L}{\partial \Theta} - \lambda_\Theta \Theta
 \end{aligned} \tag{2.9}$$

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

index into the corresponding latent factor.

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

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 $\mathcal{P}(\Theta)^L$.

$$\eta_u^* = \arg \max_{1 \leq \eta_u \leq \lceil |l_j|/\beta \rceil} \mathcal{P}(\Theta)^L \quad (2.10)$$

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($\beta, \Theta, |l|$)

```

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_S^T and update model parameters W, H through *Stochastic Gradient Ascent* until conver-

2.5. Experiment Results

gence. Then we sample (u, l_i, l_j) from D_S^L , 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($\Theta, D_S^L, D_S^T, \beta$)

```

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

2.5 Experiment Results

2.5.1 Goodreads Data Setup

To evaluate our proposed model LIRE and compare it with existing models for item and playlist recommendation, we collected data from Goodreads. Due to the time constraint, we obtained during one month a 10% sample of all user-generated book lists. 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.

To prevent the user-list interaction data from being too sparse, 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 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 2.2 (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 2.2 (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 lists.

Similar statistics of the Goodreads dataset can be observed between users and lists, and also between users and books, as shown in Figure 2.1. 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.

Algorithms Compared

We compare our proposed LIRE with the following 5 baseline algorithms as discussed in Section 2.3: 1. GLB; 2. PPOP; 3. PIF; 4. BPR; 5. LME. For BPR and LME, we also used grid search on the whole dataset to find the best learning parameter settings. Finally, we consider two variations of LIRE – LIRE -UNIFORM with uniform item weighting, and LIRE -DCG-T

2.5. Experiment Results

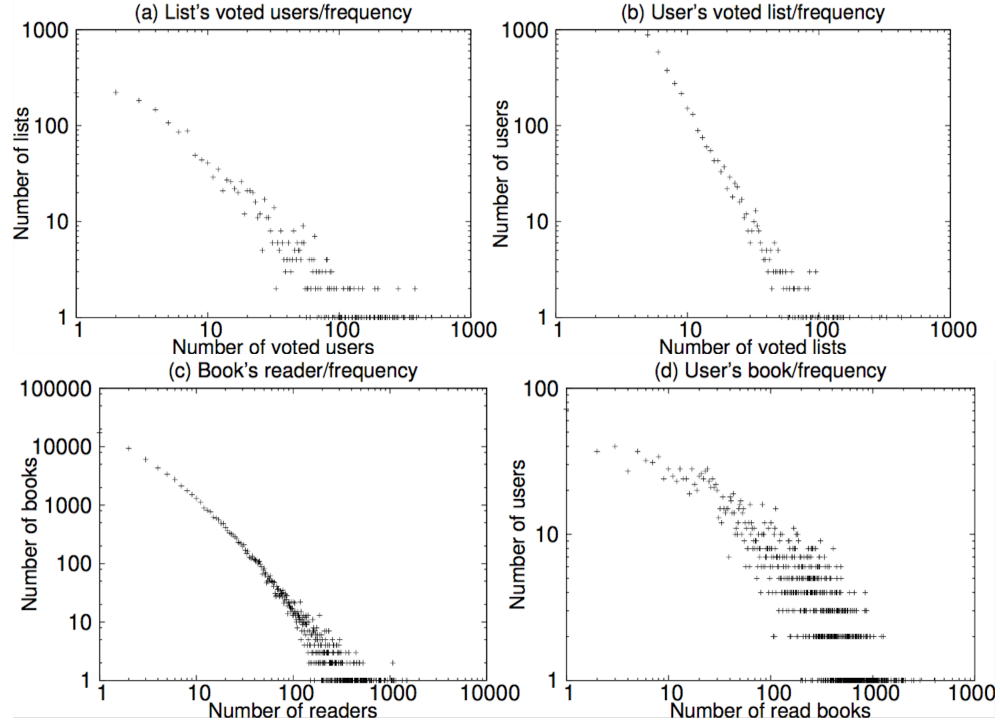


Figure 2.1: Statistics of Goodreads data w.r.t. lists and users, users and books

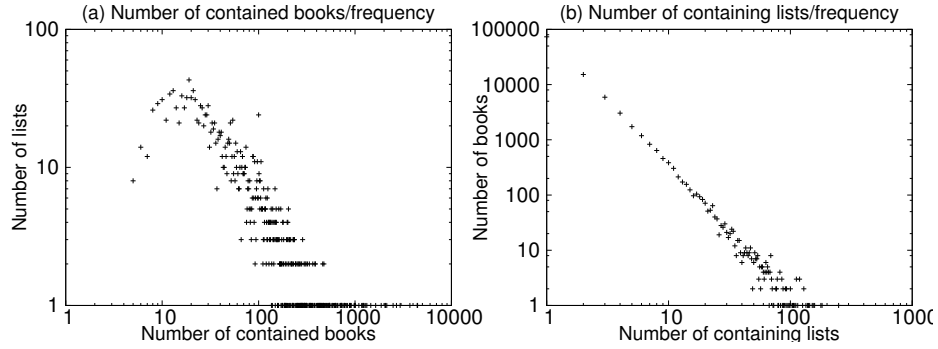


Figure 2.2: Statistics of Goodreads data w.r.t. lists and books

with item position-based weighting and personalized browsing threshold.

Performance Measure

AUC is a commonly used metric for testing the quality of rank orderings. Following [25], 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 [5]. 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.

2.5.2 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 2.4, 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 2.1. 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 since they are very similar. This result indicates that β in practice can be tuned based on the actual dataset.

2.5. Experiment Results

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

Table 2.1: How β value affect the AUC.

2.5.3 Convergence of LIRE

In Figure 2.3, we demonstrate the convergence behavior of the LIRE model. As can be found in Figure 2.3, our model converges fairly quickly (around 5 iterations). We note that this convergence result also holds for other dimensionality setting of the latent factors.

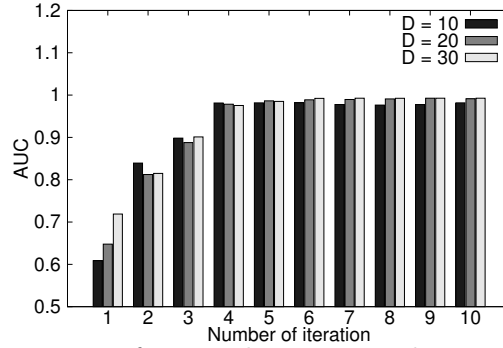


Figure 2.3: Convergence of LIRE when varying dimensionality of the latent factors.

2.5.4 Quality Comparison

Finally, we compare the performance of LIRE with other algorithms as presented in Section 2.3. From Figure 2.4, 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 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.

2.5. Experiment Results

For the two variations of LIRE, we can see from Figure 2.4, both of them perform better than the other algorithms. LIRE-DCG-T performs slightly better than LIRE-UNIFORM. We consider that there are two reasons for this: (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.

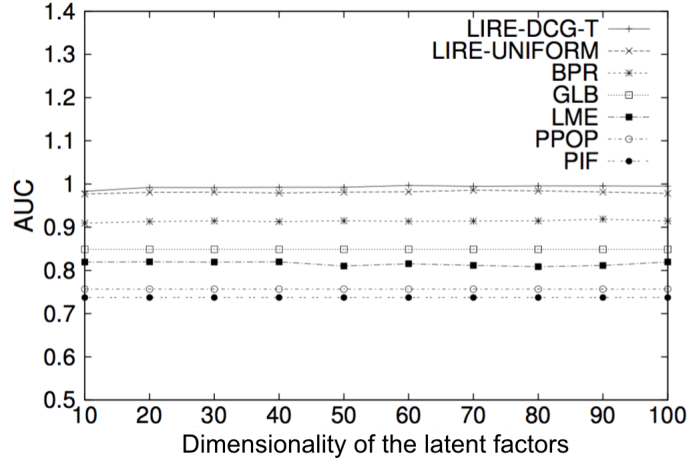


Figure 2.4: Quality comparison for various algorithms.

Chapter 3

Optimization Model

In this chapter, we modeled the list recommendation problem as an optimization problem. We will give a detailed explanation of the model and show how we achieved efficient list recommendation with only user-item interaction and content information.

3.1 Lessons Learned

Before going into the model detail, we first summarized the lessons we learned from recommendation model.

First, in the previous recommendation model, we only utilized the past interaction data, which is all implicit feedback. In fact, based on our observation, there is usually rich content associated with the items as well as item lists. This information has not been used in our studies.

Second, from our previous observation, user-list interactions are usually much sparser than user-item interactions. The reasons are 3-folded: 1) the number of items is much larger than the number of lists; 2) the activities that user can perform on an item are more diverse than the activities one can perform on a list. For example, in Goodreads, users can rate, comment on a book. They can also add books into their bookshelves. While for book lists, they can only comment a list or vote on a book list. 3) although many websites are providing the list functionality, not all users are aware of the functionality or have tried the functionality. Therefore, user-list interactions can be very sparse as compared to user-item interactions. In our previous recommendation model, we need both user-list interaction and user-item interaction to perform efficient recommendation, which might not perform well for those users who do not have sufficient user-list interactions yet.

If we can achieve effective list recommendation based only on user-item interactions, our model can also benefit more users, i.e., users who do not have sufficient past interactions with lists.

3.2 Problem Studied and Application Scenarios

3.2.1 Preliminaries

Consider the scenario where user u has interacted with several items in the past. Some of these items appear in one or more lists. We want to recommend to u the top- k most relevant lists based on his/her past interaction with items. Figure 3.1 shows the relationship between list, item and user.

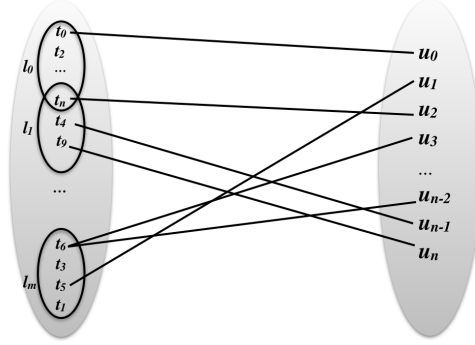


Figure 3.1: List recommendation scenario.

From Figure 3.1, we can see that users only have direct interactions with items. The relevance of a list to a user can be derived from the items contained in the list. For example, for u_2 in Figure 3.1, l_0 and l_1 will likely to be more close to u_2 's interest than l_m , since both of l_0 and l_1 contain u_2 's relevant item t_n . Moreover, note that t_n has a higher rank in l_1 compared with l_0 . Intuitively, an item which ranks high in a list is more representative of the list, as compared to items which have lower ranks. Therefore, l_1 is more relevance to u_2 than l_0 .

3.2.2 Problem Definition

In this section, we introduce our notation and problem formulation. Given a set of users U , a set of items T , a set of lists L , and each user's relevant (past interacted) items P_u , the goal is to find a subset of lists R_u , $R_u \subseteq L$, which maximizes the weighted coverage of u 's relevant items. In our problem, the relevant items associated with each user are provided as implicit feedbacks.

Problem 2 Coverage problem: *Given a set of items T , a set of lists L , a set of users U , an undirected bipartite graph $G = (L, U, E)$, where $E(U, L) \rightarrow \mathbb{R}^+$ is a weighted-coverage function, and a capacity constraint k . Each user is associated with a set of consumed items $P_u \subseteq T$, the goal is to find a set of lists R_u from L in G which satisfies the capacity constraint and maximizes the total value of weighted-coverage.*

3.2.3 Problem Variants

To solve the problem, we first need to define the coverage function. We now discuss the problem of finding a set of lists for an individual user. For a single user, the problem of finding a set of potential lists is very similar to the Maximum k -Coverage problem [13]. One difference between our problem and the weighted maximum coverage problem is that we generalize the item weights. In conventional weighted maximum coverage problem, the weight of an item will remain the same regardless of the list covering it. In our problem, since the rank of the item may change in different lists, the weight of the item will also change accordingly.

Moreover, in our case, lists may have overlapped items. We further derive three problem variants based on different ways of crediting the items which appear in multiple lists in the final selection. In the classical maximum k -coverage problem, each item would only be credited once. It makes no difference in which set the item receives credit since the weight of each item is the same for all the sets. However, in our problem, the weight of an item also depends on its rank within the list. Even if an item can receive credit

3.2. Problem Studied and Application Scenarios

for only once, we need to determine in which list should the item be credited.

We propose 3 problem variants with different crediting strategy.

Problem 3 IndCredit *Given a set of items T , a set of item lists L , a user u and his/her relevant items P_u , find k list from L , denoted as R_u , such that $Cov(R_u) = \sum_{l \in R_u} \sum_{t \in l} weight(u, t, l)$ is maximized.*

In the first problem variant, the weighted-coverage of a candidate set R_u is computed by summing up the relevance values of all the lists in R_u . Here, we will give full credit to an item for every appearance in R_u . In this case, the lists can be regarded as independent of each other.

Problem 4 MaxCredit *Given a set of items T , a set of item lists L , a user u and his/her relevant items P_u , find k list from L , denoted as R_u , such that $Cov(R_u) = \sum_{t \in P_u} \max_{l \in R_u} \{weight(u, t, l)\}$ is maximized.*

In the second problem variant, given a candidate set R_u , for each relevant item t , we only give credit to it within the list l where t receive the highest weight among all the lists in R_u , i.e., $l = \operatorname{argmax}_{l \in R_u} \{weight(u, t, l)\}$.

Problem 5 DecayCredit *Given a set of items T , a set of item lists L , a user u and his/her relevant items P_u , find k list from L , denoted as R_u , such that $Cov(R_u) = \sum_{l \in R_u} \sum_{t \in l} d(n, t)weight(u, t, l)$ is maximized, where n denotes the number of times t already been covered in R_u and $d(n, t)$ is a decay function.*

In the third problem variant, we allow an item to be credited for more than once, but with a decay credit. Consider a simple scenario where user u has 2 relevant items, t_1 and t_2 . In the candidate set of lists, t_1 appears in k lists, while t_2 only appears once. Apparently t_1 is more important and popular than t_2 . Therefore, we would lose this popularity information if we only credit t_1 once. In order to keep this popularity information, and also prevent t_1 from dominating other relevant items, we have proposed the DecayCredit variant. This problem variant aims to find a balance between

IndCredit and MaxCredit.

We now introduce the Maximum k -coverage problem, which will help derive the complexity of our problems.

Maximum k -coverage problem

The maximum k -coverage problem can be described as follows[13]:

INSTANCE: A universal set of elements U , an integer k , and a class \mathbb{R} of subsets of U . Each element $u \in U$ has an associated weight $w(u)$.

OPTIMIZATION PROBLEM: Select k subsets, A_1, A_2, \dots, A_k of U , where each subset selected is a member of \mathbb{R} , such that the weight of the elements in $\bigcup_{l=1}^k A_l$ is maximized.

Maximum k -coverage problem is known to be NP-hard[13]. The item list recommendation scenario is very similar to maximum k -coverage problem. Consider the item set T in our problem as the universal element set U in maximum k -coverage problem, and the list set L as the subsets U . In maximum k -coverage problem, the goal is to find k subsets that maximizes the total weight of items covered by k subsets. In our problem, the goal is to find k lists which also maximizes the total weight of items, w.r.t. the coverage function defined in each specific problem variant.

Lemma 3.2.1 *MaxCredit is NP-hard.*

From the definition of MaxCredit problem, it is easy to see that the Maximum k -coverage problem is a special case of MaxCredit: consider the case where the ranks of the same item among all the lists that cover it are the same, which indicates that the weights of the same item among all the lists covering it are also the same. In this case, the Maximum k -coverage problem has a solution if and only if the MaxCredit problem has a solution.

Lemma 3.2.2 *DecayCredit is NP-hard.*

3.2. Problem Studied and Application Scenarios

The Maximum k -coverage problem can also be reduced to DecayCredit problem: still consider the case where the ranks of the same item among all the lists covering it are the same. Also, define the decay function $d(n, t)$ as:

$$d(n, t) = \begin{cases} 1, & n = 0 \\ 0, & n \geq 1 \end{cases} \quad (3.1)$$

In this case, the weights of the same item among different lists are the same, and each item would only be credited for once. The Maximum k -coverage problem has a solution if and only if the DecayCredit problem has a solution.

Lemma 3.2.3 *MaxCredit problem has $(\frac{2e-1}{e-1})$ -approximation ratio.*

We now prove that by reducing the MaxCredit problem to Generalized Maximized Cover Problem[11], we can get a $(\frac{2e-1}{e-1})$ -approximation bound. The formal definition of GMCP is as follows: Given a set of items E , a set of bins B , and a cost constraint L , each item has a pair of profit/cost associated with it, which will differ depending on the set covering it, each set is also associated with a cost. The problem is to find a selection of bins and items under the cost constraint, and the profit is maximized. For the GMCP, let the weights of items be 0, and let the weights of lists be unit. Let budget L equals to k . Since the assignment function in GMCP assigns each item t to only one list, it must be the list where t receives the highest weight. Otherwise there must exist another assignment which achieves higher coverage than the current assignment. So the MaxCredit problem has a solution if and only if GMCP has a solution. In GMCP, they proposed a greedy algorithm which selects a subset of elements with the maximum residual density in each step. The selection for each greedy step is a α -approximation and the approximation bound for GMCP is $\frac{1+\alpha-\alpha \cdot e^{-\frac{1}{\alpha}}}{1-e^{-\frac{1}{\alpha}}}$ using the greedy algorithm[11]. In the MaxCredit problem, each greedy step is optimal since the sets of items are the lists, and we can get an optimal solution by traversing all the lists. Therefore, we can get a $\frac{2e-1}{e-1}$ approximation bound for MaxCredit problem.

3.2.4 Application Scenarios

In this section, we briefly discuss the application scenarios of our problem.

Capacity constraint The capacity constraint in our problem determines how many list recommendations each user receive. This conforms to the *Top-k* recommendation, where only k lists with the highest relevance will be recommended.

Novelty of recommendation We can further add *Novelty* into consideration by differentiating between "old" and "new" items and incorporate this into the utility function. Old items refer to items that the user has already interacted with and new items refer to the opposite. This is similar to the explore/exploit tradeoff and should also be personalized. For example, a user may have his/her own preference of how many new items/old items to be covered in the final recommendation lists R_u . This can be measured by a function $g(u, t)$, where u is the current user and t is the specific item we want to measure. The utility function can be represented as:

$$Cov(R_u) = \sum_{t \in P_u} \{max_{l \in R_u} \{weight(u, t, l)\} + g(u, t)\}. \quad (3.2)$$

One issue about equation 3.2 is that the model may get into the extreme of recommending too many new items or old items. To prevent this, we can replace the utility function with:

$$Cov(R_u) = F\{G_u(O, N), \sum_{t \in P_u} max_{l \in R_u} \{weight(u, t, l)\}\}. \quad (3.3)$$

Here, O is the set of old items relevant to user u covered by lists in R_u , and N is the set of new items relevant to u . $G_u(O, N)$ measures the overall novelty of the recommendation. F is an aggregation function which takes both novelty and coverage into consideration.

Diversity of recommendation Another commonly-used measurement for recommendation is *Diversity*. Considering two book lists l_1 and l_2 . l_1 is composed of all sci-fi books while l_2 is composed of books from various genres or topics. In this case, l_2 is more diverse than l_1 . Similar to novelty,

3.3. Modelling Relevance

users' preference to diversity should also be personalized. Diversity can be taken into consideration using:

$$Cov(R_u) = F\{D_u(R_u), \sum_{t \in P_u} max_{l \in R_u} \{weight(u, t, l)\}\}. \quad (3.4)$$

In 3.4, R_u is also the set of recommended lists to user u , and $D_u(R_u)$ measures the personalized diversity rewards of the recommendation set. F is an aggregation function which takes both diversity and coverage into consideration.

3.3 Modelling Relevance

3.3.1 User Item Relevance

To determine the relevance of each user and each item, we incorporated users' past interactions and items' content information.

We consider that user's past interacted items can directly reflect the user's taste and interest. The past interactions between users and items can be modeled as a 0-1 matrix, P . Each entry $P(u, t)$ in the matrix indicates whether or not user u has consumed item t in the past.

Apart from direct interaction information, items' content information can be used to represent the item features, and further help find relevant items. The content information refers to text information associated with each item, and is usually depend on the particular web service. For example, in Goodreads, every book has text information including title, author, abstract, publisher and so on. For other shopping websites, content information may include item type, brand and so on. Using classical topic modelling strategy, we are able to get the topic distribution of each item. For each user u , based on the user's past item interactions, we can derive the user's preference over difference topics.

Therefore, for user u , the relevance between u and each item t can be

3.3. Modelling Relevance

represented as follows.

$$rel(u, t) = \mathbb{1}_{R_u} t + Ct(t) \cdot Cu(u) \quad (3.5)$$

In 3.5, $\mathbb{1}_{R_u} t$ is an indicator function which equals to 1 if u has interacted with t and 0 otherwise. $Ct(t)$ denotes the content feature of t and $Cu(u)$ denotes the content feature of u . By performing a dot product between user content feature and item content feature, we can get the content relevance between the user and the item.

3.3.2 User List Relevance

For a specific user u , the relevance of a list l is computed over all the items in l . There are two factors to consider. First, the appearance of u 's relevant items within the list. If l contains an item t which is relevant to u , then the relevance between l and u will increase. Also, the more relevant t is to u , the relevance gain between l and u will be larger. Second, the rank of the relevant item within l . Ranking is a good measure to indicate how close the item is to the theme of the list. Consider two lists l_1 and l_2 , both containing an item t which the user u likes. Since l_1 and l_2 may have different theme, the importance/value of t within these two lists can also be different. If t is ranked higher in l_1 than l_2 , it is likely that t is closer to the theme of l_1 , which indicates l_1 will better fit the user's taste than l_2 . Thus, we also need to take the rank into consideration. By combining these two parts together using multiplication, we can get the weight of t to u within the context of l .

$$weight(u, t, l) = rel(u, t) * rank(t, l). \quad (3.6)$$

One simple way to aggregate the relevance of items into the relevance of the list is summation. Given each user, we first get the weight each item within candidate lists, then sum up the weights and get the relevance of l .

$$rel(u, l) = \sum_{t \in l} weight(u, t, l) \quad (3.7)$$

3.4 Algorithms

Based on the three problem variants proposed in the previous section, we now discuss the algorithms for these problems.

IndCredit Each relevant item can be covered repeatedly with full credit in R_u . In this case, each list $l \in R_u$ will be measured independently. The weighted-coverage of R_u can be represented as:

$$Cov(R_u) = \sum_{l \in R_u} \sum_{t \in l} weight(u, t, l) \quad (3.8)$$

Properties

- Additivity. $Cov(R) = Cov(R_1) + Cov(R_2)$, where $R_1 \cup R_2 = R$.

MaxCredit Each relevant item t can only be credited once in R_u , and the list covering it should be the list where t receives the highest weight among all the lists in R_u . This problem resembles *Generalized Maximum Coverage Problem*. (GMCP) [11]. The corresponding weighted-coverage of R_u can be represented as:

$$Cov(R_u) = \sum_{t \in P_u} \max_{l \in R_u} \{weight(u, t, l)\} \quad (3.9)$$

Properties

- Subadditivity. $Cov(R) \leq Cov(R_1) + Cov(R_2)$, where $R_1 \cup R_2 = R$.

We now present the greedy algorithm for MaxCredit. At each stage, select the list with the maximum coverage. When computing the coverage value for each list, residual item weight is used instead of the original item weight. This ensures that each item only receive credit in the list where the item has the highest rank.

$$weight(u, t, l) = \begin{cases} 0, & weight(u, t, l) < \max_{l' \in R_u} \{weight(u, t, l')\} \\ weight(u, t, l) - \max_{l' \in R_u} \{weight(u, t, l')\}, & otherwise \end{cases} \quad (3.10)$$

3.4. Algorithms

Using the above subroutine, the greedy algorithm for MaxCredit is as follows.

Algorithm 3: Greedy for MaxCredit

```

1  $L \leftarrow$  list set,  $T \leftarrow$  item set,  $P \leftarrow$  user's relevant items ;
2  $R \leftarrow$  empty candidate set,  $k \leftarrow$  the number of lists in the final candidate set
  ;
3 Compute the weighted coverage of items in  $P$  for all the list in  $L$  ;
4 for  $stage = 1, 2, \dots, k$  do
5    $l \leftarrow$  the list with the highest weighted coverage in  $L$ ;
6    $Q \leftarrow$  the set of relevant items in  $l$  ;
7    $R.add(l)$ ;
8    $L.remove(l)$ ;
9   for  $t \in Q$  do
10     $\lfloor$  Recompute the residual weight of  $t$  in all lists in  $L$  which contain  $t$  ;
11 return  $R$ .
```

DecayCredit The weight of each item will decay when covered multiple times by R_u . In this case, the lists in R_u are not longer independent to each other and R_u should be measured as a whole. We can use a decay function $d(n, t)$ to represent the decay, where n is the number of appearances of item t in R_u . For each item covered in the set, The corresponding weighted-coverage of R_u can be represented as:

$$Cov(R_u) = \sum_{l \in R_u} \sum_{t \in l} d(n, t) weight(u, t, l) \quad (3.11)$$

Properties

- Subadditivity. $Cov(R) \leq Cov(R_1) + Cov(R_2)$, where $R_1 \cup R_2 = R$.

Here is the greedy heuristic for picking R_u . Each time the algorithm picks the list with highest weighted-coverage of relevant items. A recomputation is needed at the end of each iteration in order to reflect the reduction of weights of the covered items.

3.5. Experiment Results

Algorithm 4: Greedy for DecayCredit

```
1  $L \leftarrow$  list set,  $T \leftarrow$  item set,  $P \leftarrow$  user's relevant items ;
2  $R \leftarrow$  empty candidate set,  $k \leftarrow$  the number of lists in the final candidate set
  ;
3 Compute the weighted coverage of items in  $P$  for all the list in  $L$  ;
4 for  $stage = 1, 2, \dots, k$  do
5    $l \leftarrow$  the highest weighted coverage in  $L$ ;
6    $Q \leftarrow$  the set of relevant items in  $l$  ;
7    $R.add(l)$ ;
8    $L.remove(l)$ ;
9   for  $t \in Q$  do
10     $\lfloor$  Recompute the weight of  $t$  in all lists in  $L$  which contain  $t$  ;
11 return  $R$ .
```

3.5 Experiment Results

3.5.1 Data Setup

For experiments, we used datasets from Goodreads and Stack Overflow. The Goodreads dataset was collected and used in our previous work. In Goodreads data, user can interact with a list by voting on the book list. Here, we also need the content information. So we did extra work to collect and process the title as well as the abstract of each book. We also removed items which we can not get the content information due to language reason.

Apart from Goodreads data, we also used Stack Overflow datasets. Stack Overflow is an online Question&Answer website. In Stack Overflow, users can post questions, answer questions, and vote for questions or answers. Each post contains one question and multiple answers. Each post also has one or more tags, which are edited by users and help to categorize the post. For Stack Overflow data, we regard each tag as a list and each post (including both the question and the answers) as a single item. A user can interact with an item/post by adding the question or the answers. The Stack Overflow datasets are downloaded from Stack Exchange Data Dump [1]. The data dump contains 315 small datasets of different areas. We selected three datasets: Programmer, Askubuntu, and Unix. Table 3.1 shows the statistical information of the datasets we use.

3.5. Experiment Results

Dataset	# of Users	# of Items	# of Lists
Goodreads	3,425	70,413	1,998
Askubuntu	18,061	167,765	2,409
Programmer	8,007	35,560	1,598
Unix	5,975	48,430	1,620

Table 3.1: Datasets statistics

3.5.2 Topic Modeling

We used **Latent Dirichlet Allocation** for getting content features. LDA is a three-level hierarchical Bayesian model which has been commonly used in topic modeling applications. In LDA, each document can be viewed as a mixture of multiple topics, and each word in the document is attributable to one of the topics. For Goodreads, we regard each book as a document. The title and abstract parts of the book compose the document. For Stack Overflow, each post is regarded as a document. The content of the question, as well as the answers, composes the document. For each dataset, we determine the number of topics using grid search on the whole dataset. Figure 3.2 shows the dataset information and parameters we used in experiments.

Dataset	# of Documents	# of Terms	# of Topics
Goodreads	70,413	399,002	200
Askubuntu	167,765	1,029,247	80
Programmer	35,560	561,257	80
Unix	48,430	674,316	80

Table 3.2: Topic modeling statistics

3.5.3 Ground Information

For Goodreads dataset, we used the known user-list interaction data as ground truth for evaluation. For each user, we compared the similarities between the set of lists that the user has interacted with and the set of lists that are recommended by the algorithm. The similarity is used for representing algorithm accuracy. In other words, the more similar the recommend

lists to the ground lists, we consider the corresponding algorithm to be more accurate.

For Stack Overflow datasets, there is no direct user-list interaction data available for evaluation. We consider that the ground information can be regarded as personalized global information. Therefore, if we can derive such personalized global information from the data, we can use it as the ground truth. Since we have the user-post interaction and the post-tag relation, we can derive a user-tag relation that for each user u . In this user-tag relation, for each user u , tags are ordered by the number of times u has interacted with the tag through directly interacting with posts. In the experiments, we will use this user-tag relation as the ground information to evaluate the recommendation.

3.5.4 Similarity Measures

One crucial subroutine of the evaluation process is to compare the similarity between the recommended lists and the ground lists. We applied multiple measures for calculating the similarity.

Jaccard The Jaccard similarity measure is a commonly-used metric for comparing the similarity of different sets. It is defined as the size of the intersection divided by the size of the union of the two sets. The value of Jaccard varies from 0.0 to 1.0, with 1.0 representing two same sets of items.

$$J(A, B) = \frac{A \cap B}{A \cup B} \quad (3.12)$$

In our experiments, intersection is computed as the set of lists which appear in both the recommendation result as well as the ground information.

Nested Jaccard Because each recommendation is actually a list, which consists of multiple items, it might not be enough just comparing their IDs. For example, it is possible that there are two lists which contain the same set of items, but with different list IDs. In this case, merely comparing the list IDs will not give the accurate result. Therefore, we proposed a Nested Jaccard measurement for evaluating the similarity between the recommended list and the ground list. In Nested Jaccard, two lists are determined as the

3.5. Experiment Results

same lists if they both contain a certain number of items.

Topic We also proposed topic measure for comparing recommended list and ground list. For each list, we can calculate its topic feature by summing up topic features of the items in the list. Then we can calculate the cosine similarity of recommended list and ground list. The cosine similarity is calculated as:

$$similarity = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\| \|\mathbf{B}\|} \quad (3.13)$$

\mathbf{A} and \mathbf{B} are the feature vectors to be compared. The similarity value is bounded in 0.0 to 1.0, and higher value indicates higher similarity.

NDCG In both Jaccard and nested Jaccard, the order of recommended lists is not considered. In our experiments, the order is crucial for Stack Overflow data, since the ground information is personalized tag ranking. Therefore, we further choose **Normalized Discounted Cumulative Gain** for evaluating the recommendation accuracy. NDCG measures the performance of a recommendation based on the graded relevance of the recommended elements. It varies from 0.0 to 1.0, with 1.0 representing the ideal ranking of the elements. The discounted cumulative gain can be calculated as:

$$DCG_k = \sum_{i=1}^k \frac{2^{rel_i} - 1}{\log_2(i + 1)} \quad (3.14)$$

where k represents the number of recommended elements. The normalized DCG is calculated as:

$$nDCG_k = \frac{DCG_k}{IDCG_k} \quad (3.15)$$

where $IDCG_k$ is the idea DCG for a given set of elements.

3.5.5 Evaluation Results

We first show the evaluation results for Goodreads dataset, where Jaccard, Nested Jaccard, and Topic metrics are applied. For each user u , we compare the similarity between the ground lists P_u and the recommended lists R_u . For the results shown below, x-axis represents the similarity value between

3.5. Experiment Results

recommendation lists and ground lists, and y-axis represents the number of users that falls into each similarity range. For example, for user u , if the similarity value between u 's ground lists and u 's recommended lists is 0.45, then the number of users for similarity 0.4 will increment by 1.

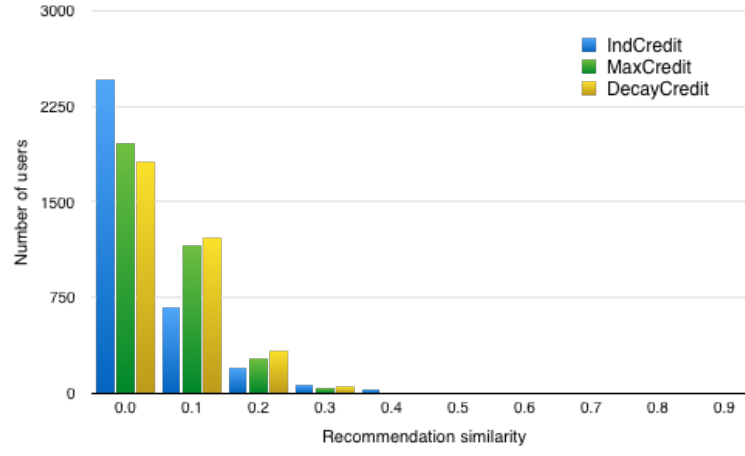
Figure 3.2 shows the evaluation results on Goodreads dataset, using Jaccard, Nested Jaccard, and Topic measures. From Figure 3.2, we can see that the recommendation results from MaxCredit and DecayCredit are more similar to the ground truth as compared to the recommendation result from IndCredit.

Then we show the evaluation results for Stack Overflow datasets. For these datasets, we applied NDCG and Topic to measure the list similarity.

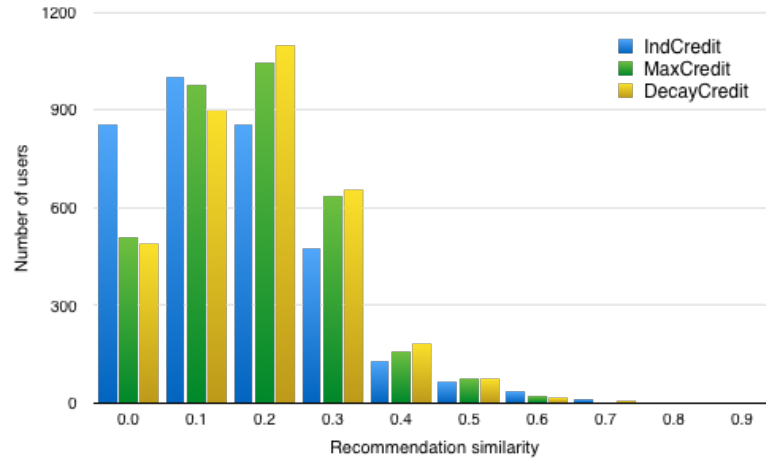
Figure 3.3, Figure 3.4 and Figure 3.5 shows the evaluation result on three datasets from Stack Overflow. From the figures we can see that the performances of three algorithms are very close. DecayCredit has higher accuracy in terms of list similarity.

Next we compare the evaluation results for both with content information and without content information. Figure 3.6 shows the comparison results of with/without topic feature on Goodreads data. It is easy to see that the algorithms with content information perform better than algorithms without content information.

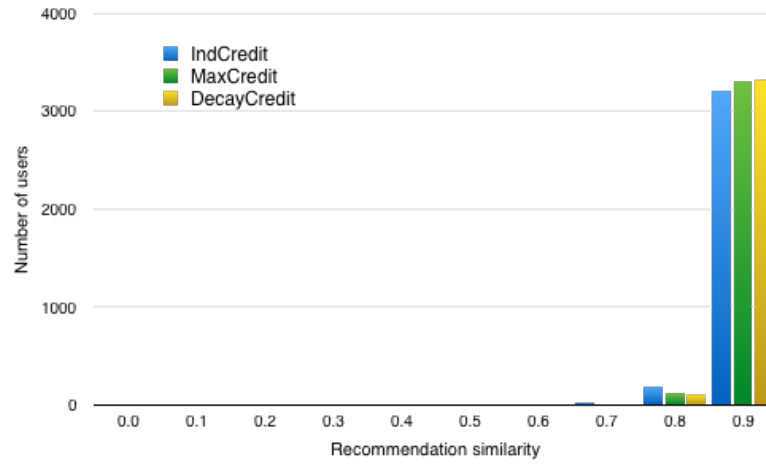
3.5. Experiment Results



(a) Recommendation similarity using Jaccard



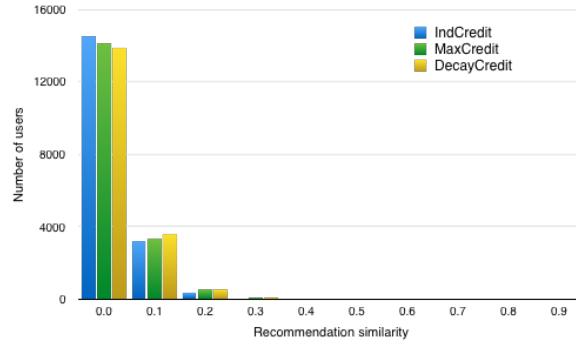
(b) Recommendation similarity using Nested Jaccard



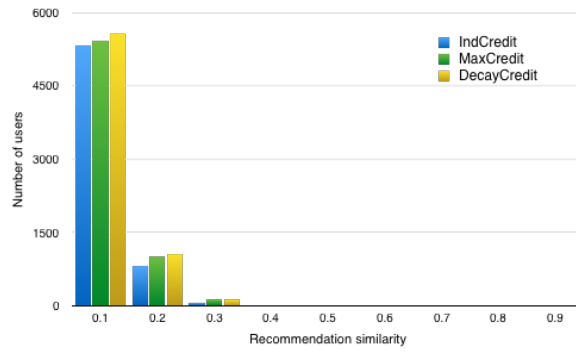
(c) Recommendation similarity using Topic

Figure 3.2: Evaluation results on Goodreads data

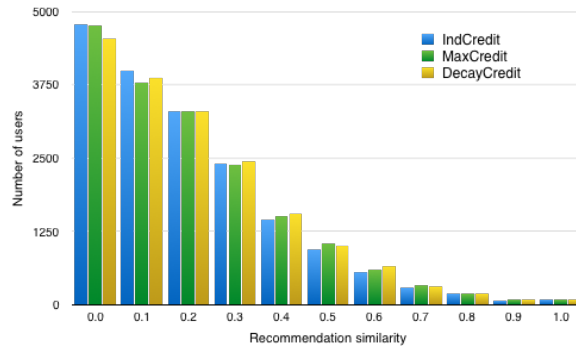
3.5. Experiment Results



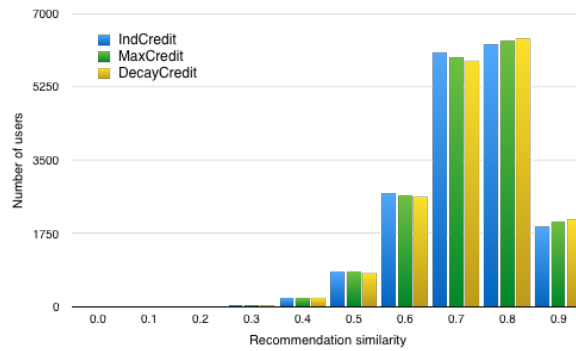
(a) Recommendation similarity using Jaccard



(b) Recommendation similarity using Nested Jaccard



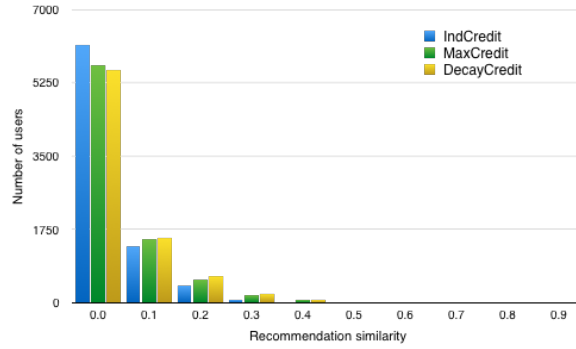
(c) Recommendation similarity using NDCG



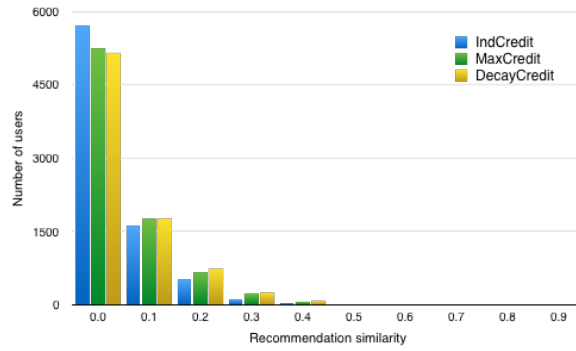
(d) Recommendation similarity using Topic

Figure 3.3: Evaluation results on Askubuntu data

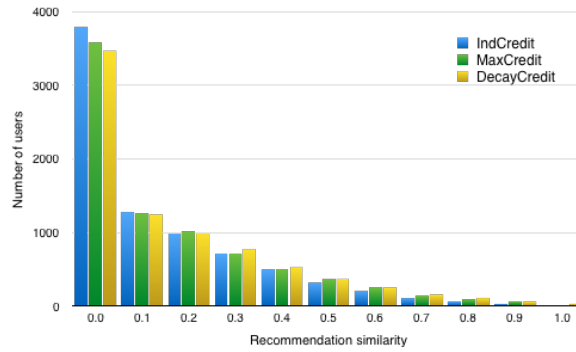
3.5. Experiment Results



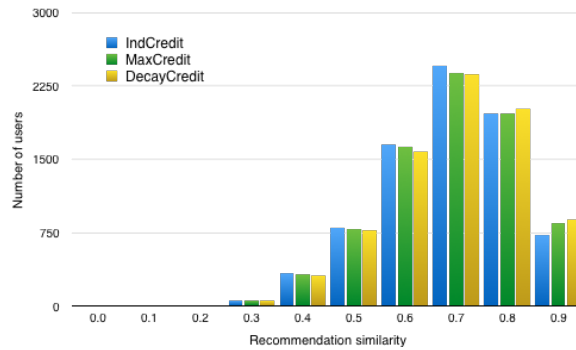
(a) Recommendation similarity using Jaccard



(b) Recommendation similarity using Nested Jaccard



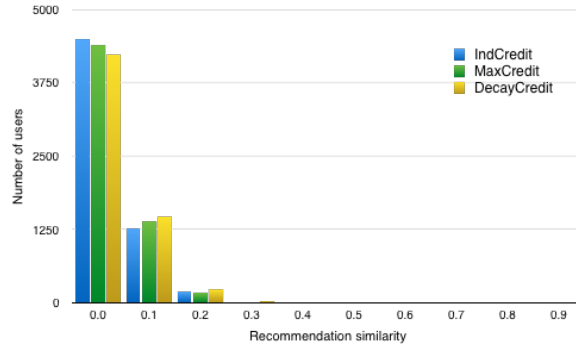
(c) Recommendation similarity using NDCG



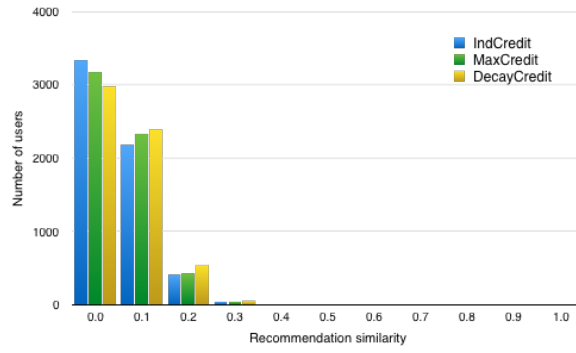
(d) Recommendation similarity using Topic

Figure 3.4: Evaluation results on Programmer data

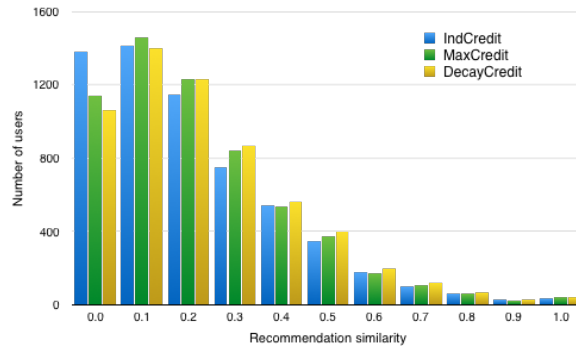
3.5. Experiment Results



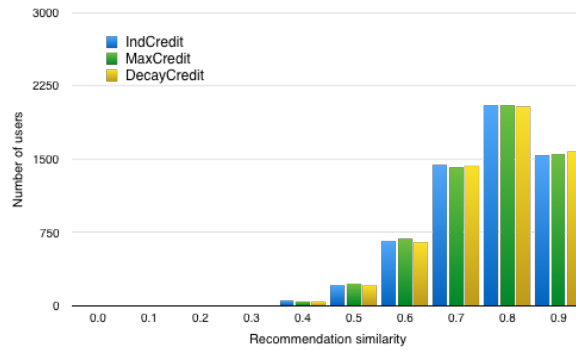
(a) Recommendation similarity using Jaccard



(b) Recommendation similarity using Nested Jaccard



(c) Recommendation similarity using NDCG



(d) Recommendation similarity using Topic

Figure 3.5: Evaluation results on Unix data

3.5. Experiment Results

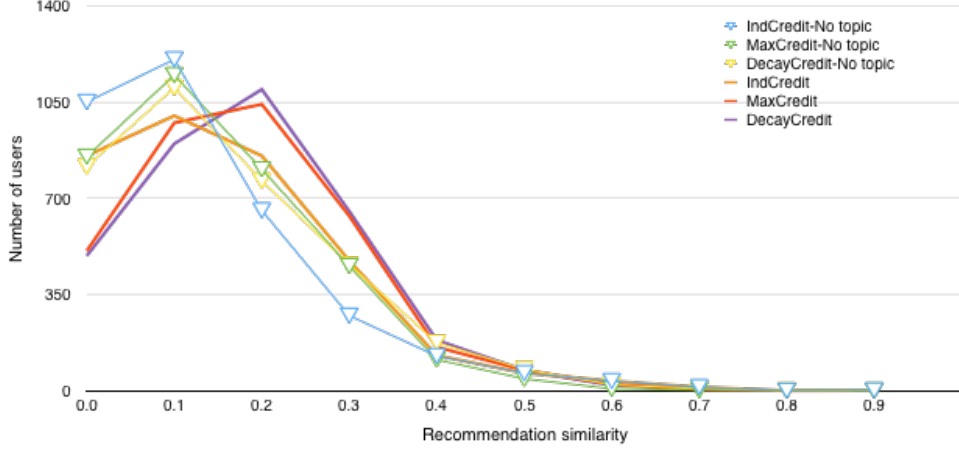


Figure 3.6: Compare the results of with and without content information

3.5.6 Results Analysis

From the evaluation results shown above, we have the following observations.

1. MaxCredit and DecayCredit achieve higher recommendation accuracies than IndCredit. From the evaluation results, we can see that the lists recommended by MaxCredit and DecayCredit adopt higher similarity to the ground lists. This conclusion holds when different similarity measure are applied. The reason for this could be the dominance of popular items. Recall that in IndCredit, items can receive full credits for multiple times. There are a small set of items which appear frequently in lists and have been interacted by many users. If we give the item full credit in every list it appears, these popular items will be likely to dominating other items that the user has interacted.

Imagine a simple case where user u has interacted with t_0 , t_1 and t_2 . Both t_0 and t_1 appear in list l_0 and l_1 , while t_2 only appears in l_2 . If we can only recommend two lists to u , l_0 and l_1 are more likely to be recommended by IndCredit than l_2 since the former two both contain two relevant items. In this recommendation, t_2 will be left out and the u 's preference is not accurately represented. If we use MaxCredit or DecayCredit, in the first step, we would still pick one of l_0 and l_1 .

3.5. Experiment Results

Suppose we pick l_1 , then l_0 's marginal coverage will decrease and we have more possibility to select l_2 .

2. Recommended lists and ground lists are similar in topics. From the evaluation result, recommended lists and ground lists are very similar when using Topic as similarity measure. This indicates that although the recommended lists generated by the algorithms are not exactly the same as ground lists, they fall into similar topics and the recommendation can be potential fits for future consumption.
3. The difference between three algorithms' performances is smaller in Stack Overflow datasets than in Goodreads dataset. From the evaluation results of Stack Overflow dataset, the performances of three algorithms are very close. Although DecayCredit usually has the best performance, the difference is very small. We consider the reason of the small difference is that in Stack Overflow, the number of lists containing a certain item is small. In our model, each post is an item and the associated tags are the lists containing the item. Usually the number of tags for one post is around three. Therefore, for a item appearing in multiple lists, the credit it receives would not change too much between different algorithms.

Chapter 4

Related Works

We referred to related research works from several categories.

4.1 Playlist Recommendation

User-generated item lists are similar to playlists as explored in previous works such as [9] and [6]. An important observation made by these works is that neighbouring 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 neighbouring 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 [6], 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 modelling playlists is presented in [8]. This sequential assumption differs our work from the playlist recommendation works.

4.2 Travel Package Recommendation

The proposed item list recommendation problem is also related to recent effort on package recommendation [24, 26, 29]. Although a travel package is presented in a list structure, travel package recommendation is different from our work in several aspects. The first one is the heterogeneity of items. Although both travel package and list consist of multiple items, the items in a travel package are much more heterogeneous than the items in the list in our problem. For example, in a travel package, the items can be flight, hotel, tours and so on. However, in the list, the items are of the same

type, such as books, pins, or places, which depends on the services provided by the website. The second major difference is that 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 [20], 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 their work is on modelling the cost of a package. In our work, we focus on how item preferences can be aggregated to model user’s preferences over item lists.

4.3 Implicit Feedback Recommendation

Implicit data settings create a huge challenge for existing latent factor models. In particular, there is little information on what users dislike [14]. Researchers have recently proposed various models for modelling implicit datasets. One notable work in this direction is BPR [25], 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 [27], which instead of optimizing AUC as in BPR, considers *Mean Reciprocal Rank* as the optimization criterion. In [22], the authors argue that reasoning about preferences between individual items might be too fine-grained, 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 [14], in which the authors propose a different way of modelling implicit datasets by weighting the importance of each observed rating through heuristic functions. In [23], 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 capture potential items which users have considered before actually consuming items in the original dataset.

4.4 Hybrid-model Recommendation

Content-based and collaborative filtering models are both very popular in recommender systems. The collaborative filtering methods usually suffer from cold-start issue and popularity bias. The content-based methods require the content to be encoded as meaningful features and are usually easy to overfit [7]. This brings in a third type of method, which is the hybrid approaches. There are some recommendation works about combining matrix factorization with regression, i.e., using both implicit features and explicit features [4, 30]. In [30], the authors applied a simple addition to combine the two parts. In [4], the model learns a transition matrix to transform explicit features to latent factors. In our work, we used implicit features when modeling list recommendation as a recommendation problem, and used explicit features when modeling it as an optimization problem. A potential future work direction would be to combine implicit features with explicit features.

Chapter 5

Conclusion

In this work, we motivate the problem of recommending user-generated item lists and proposed two different models to solve the problem. Existing works in recommender system usually focus on item recommendation. Thus, they do not consider how user’s preference over item lists can be decomposed into preferences over individual items within the list.

In this work, we propose two models to capture users’ preference over item lists. Based on our observations of user-generate item lists, we first propose a novel model LIRE which models users’ preferences over item lists by aggregating users’ preferences over individual items within a list. We capture how users perceive an item list by weighting items within a list based on position. We also take into consideration the number of 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.

Next, we summarize the lessons we have learned from the LIRE and propose an optimization model for solving the problem. Based on different ways of rewarding repeatedly-appeared items, we specify three problem variants, presenting hardness proof and algorithms for each problem variant. We further conduct experiments on four datasets collected from two websites and show the effectiveness of our optimization model.

Chapter 6

Future Works

In this chapter, we propose potential future works for list recommendation 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 can check whether liking a list will result in subsequent interactions with the items in a list.

Second, many applications that provide list functionality also involve social network. Friendship between users often indicates similar preferences. This further raises the question that how social influence can be incorporated into our current models to facilitate list recommendation.

Third, it's interesting to think about whether we can generate explanation for our recommendation. Explanation can be useful for users to understand why certain recommendations are generated. Specifically in our problem, explanation is particularly important for lists with larger size. For example, when a long list is recommended to the user, it would be impossible for the user to browse over the entire list trying to find interesting items. If we can give the user an explanation, such as "this list is recommended to you because we think you might be interested in these items in the list" along with the relevant items, the recommendation would be more effective.

Bibliography

- [1] <https://archive.org/details/stackexchange>, April 2016.
- [2] <http://www.amazon.ca/gp/help/customer/display.html?nodeId=1197914>, April 2016.
- [3] http://www.yelp.com/list_search, March 2016.
- [4] D. Agarwal and B.-C. Chen. Regression-based latent factor models. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '09, pages 19–28, New York, NY, USA, 2009. ACM.
- [5] 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.
- [6] N. Aizenberg, Y. Koren, and O. Somekh. Build your own music recommender by modeling internet radio streams. In *Proceedings of the 21st International Conference on World Wide Web*, WWW '12, pages 1–10, New York, NY, USA, 2012. ACM.
- [7] X. Amatriain. The recommender problem revisited. In *Proceedings of the 8th ACM Conference on Recommender Systems*, RecSys '14, pages 397–398, New York, NY, USA, 2014. ACM.
- [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 *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '12, pages 714–722, New York, NY, USA, 2012. ACM.

- [10] S. Chen, J. Xu, and T. Joachims. Multi-space probabilistic sequence modeling. In *KDD*, pages 865–873, 2013.
- [11] R. Cohen and L. Katzir. The generalized maximum coverage problem. *Information Processing Letters*, 108(1):15–22, 2008.
- [12] A. Herschtal and B. Raskutti. Optimising area under the roc curve using gradient descent. In *ICML*, 2004.
- [13] D. S. Hochbaum and A. Pathria. Analysis of the greedy approach in problems of maximum k-coverage. *Naval Research Logistics*, 45(6):615–627, 1998.
- [14] Y. Hu, Y. Koren, and C. Volinsky. Collaborative filtering for implicit feedback datasets. In *Proceedings of the 2008 Eighth IEEE International Conference on Data Mining, ICDM '08*, pages 263–272, Washington, DC, USA, 2008. IEEE Computer Society.
- [15] S. B. Huffman and M. Hochster. How well does result relevance predict session satisfaction? In *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 567–574. ACM, 2007.
- [16] K. Järvelin and J. Kekäläinen. Cumulated gain-based evaluation of ir techniques. *ACM Trans. Inf. Syst.*, 20(4):422–446, 2002.
- [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. *Netflix prize documentation*, 81, 2009.
- [19] Y. Koren, R. Bell, and C. Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37, 2009.
- [20] Q. Liu, Y. Ge, Z. Li, E. Chen, and H. Xiong. Personalized travel package recommendation. In *Proceedings of the 2011 IEEE 11th International Conference on Data Mining, ICDM '11*, pages 407–416, Washington, DC, USA, 2011. IEEE Computer Society.
- [21] Y. Liu, M. Xie, and L. V. Lakshmanan. Recommending user generated item lists. In *Proceedings of the 8th ACM Conference on Recommender systems*, pages 185–192. ACM, 2014.

- [22] W. Pan and L. Chen. Cofiset: Collaborative filtering via learning pairwise preferences over item-sets. In *SDM*, pages 180–188, 2013.
- [23] U. Paquet and N. Koenigstein. One-class collaborative filtering with random graphs. In *WWW*, pages 999–1008, 2013.
- [24] A. G. Parameswaran and H. Garcia-Molina. Recommendations with prerequisites. In *RecSys*, pages 353–356, 2009.
- [25] S. Rendle, C. Freudenthaler, Z. Gantner, and L. Schmidt-Thieme. Bpr: Bayesian personalized ranking from implicit feedback. In *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence, UAI '09*, pages 452–461, Arlington, Virginia, United States, 2009. AUAI Press.
- [26] S. B. Roy, S. Amer-Yahia, A. Chawla, G. Das, and C. Yu. Constructing and exploring composite items. In *SIGMOD*, pages 843–854, 2010.
- [27] 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.
- [28] A. P. Singh and G. J. Gordon. Relational learning via collective matrix factorization. In *KDD*, pages 650–658, 2008.
- [29] 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.
- [30] W. Zhang, J. Wang, and W. Feng. Combining latent factor model with location features for event-based group recommendation. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 910–918. ACM, 2013.